EGZAMIN MATURALNY

INFORMATYKA

Poziom rozszerzony

ZBIÓR ZADAŃ

Materiały pomocnicze dla uczniów i nauczycieli

Centralna Komisja Egzaminacyjna
2015

Publikacja opracowana przez zespół koordynowany przez **Renatę Świrko** działający w ramach projektu *Budowa banków zadań* realizowanego przez Centralną Komisję Egzaminacyjną pod kierunkiem Janiny Grzegorek.

Autorzy
dr Lech Duraj
dr Ewa Kołczyk
Agata Kordas-Łata
Beata Laszkiewicz
Michał Malarski
dr Rafał Nowak
Rita Pluta
Dorota Roman-Jurdzińska

Komentatorzy
prof. dr hab. Krzysztof Diks
prof. dr hab. Krzysztof Loryś
Romualda Laskowska
Joanna Śmigielska

Opracowanie redakcyjne
Jakub Pochrybniak

Redaktor naczelny
Julia Konkołowicz-Pniewska

Zbiory zadań opracowano w ramach projektu *Budowa banków zadań*,
Działanie 3.2 Rozwój systemu egzaminów zewnętrznych,
Priorytet III Wysoka jakość systemu oświaty,
Program Operacyjny Kapitał Ludzki

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

CENTRALNA
KOMISJA
EGZAMINACYJNA

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Spis treści

Wprowadzenie

Prezentowany zbiór zadań z informatyki z rozwiązaniami adresowany jest przede wszystkim do uczniów szkół ponadgimnazjalnych przygotowujących się do egzaminu maturalnego z informatyki w formule obowiązującej od 2015 r.. Zbiór został przygotowany tak, aby można było z niego korzystać zarówno podczas samodzielnej pracy w domu, jak również na lekcjach informatyki pod kierunkiem nauczyciela. W zbiorze znajduje się 110 wiązek zadań, ilustrujących wszystkie rodzaje zagadnień stosowanych na egzaminie maturalnym (w tym zadania pojedyncze oraz zadania składające się z kilku poleceń).

Zbiór składa się z kilku rozdziałów. Rozdział pierwszy i drugi to treści zadań. Zostały one przygotowane w taki sposób, jak są prezentowane zadania na egzaminie maturalnym: rozdział pierwszy to zadania rozwiązywane bez użycia komputera (arkusz I), rozdział drugi to zadania praktyczne — takie, których rozwiązanie wymaga zastosowania znanych narzędzi informatycznych (arkusz II). Rozdział drugi został poprzedzony obszernym wstępem teoretycznym, opisującym sposoby importowania danych do bazy danych, arkusza kalkulacyjnego oraz wczytywania i wypisywania danych w zadaniach programistycznych.

W rozdziale trzecim umieszczono omówienie sposobów rozwiązania zadań, przydatne szczególnie tym uczniom, którzy potrzebują podpowiedzi. Czwarty rozdział zawiera odpowiedzi do wszystkich zadań, a piąty — najistotniejsze sprawdzane zadaniami wymagania ogólne i szczegółowe z *Podstawy programowej*.

Zakres treści i umiejętności sprawdzanych zadaniami jest zgodny z zapisem w podstawie programowej informatyki dla III etapu edukacyjnego (gimnazjum) i IV (szkoła ponadgimnazjalna). Zadania zgrupowano w ośmiu działach tematycznych będących ilustracją wymagań szczegółowych podstawy programowej przedmiotu *Informatyka*. Są to: analiza algorytmów, tworzenie algorytmów, praktyka w teorii, test z ogólnej wiedzy informatycznej, algorytmy w praktyce, symulacja, przetwarzanie i tworzenie informacji, bazy danych.

W zbiorze znajdują się zadania wymagające udzielenia krótkiej odpowiedzi (słowo, kilka słów, zdanie, kilka zdań), zadania reprezentujące różne typy zadań zamkniętych, np.: zadania wyboru wielokrotnego, z luką, zadania typu: prawda-fałsz, a także wiele zadań praktycznych — do rozwiązania z użyciem poznanych narzędzi informatycznych.

Aby ułatwić uczniom przygotowującym się samodzielnie do egzaminu maturalnego korzystanie ze zbioru zadań, w rozdziale trzecim umieszczono komentarze do zadań i dokładne opisy rozwiązań. Komentarze zawierają szczegółowe wskazówki przydatne do rozwiązania zadania, opisy zastosowanych algorytmów, fragmenty kodów programów oraz uwagi, które mogą stanowić podstawę do przemyśleń i samodzielnej pracy nad zadaniami. Z uwagi na rosnącą liczbę zdających, którzy wybierają na egzaminie maturalnym język programowania C/C++, rozwiązania zadań programistycznych przedstawiono w tym właśnie języku.

Mamy nadzieję, że proponowany zbiór zadań będzie pomocny uczniom w przygotowaniu się do egzaminu maturalnego z informatyki, a nauczycielom w monitorowaniu zgodności przebiegu procesu nauczania z obowiązującą *Podstawą programową* przedmiotu informatyka.

Autorzy

1. Zadania rozwiązywane bez użycia komputera

1.1. Analiza algorytmów

Zadanie 1.

Wiązka zadań *Ciągi rekurencyjne*

Dana jest następująca funkcja rekurencyjna:

```
funkcja wynik( i )
    jeżeli i < 3
        zwróć 1 i zakończ;
    w przeciwnym razie
        jeżeli i mod 2 = 0
            zwróć wynik(i – 3) + wynik(i – 1) + 1
        w przeciwnym razie
            zwróć wynik(i – 1) mod 7
```

Uwaga: Operator mod oznacza resztę z dzielenia.

1.1.

Uzupełnij poniższą tabelę:

i	**wynik(*i*)**
2	1
3	
4	
5	
6	
7	
8	

1.2.

Wykonaniem elementarnym nazywać będziemy wykonanie *wynik*(0), *wynik*(1) lub *wynik*(2). Natomiast *złożonością elementarną wynik*(*i*) nazywamy liczbę *wykonań elementarnych* będących efektem uruchomienia *wynik*(*i*). Złożoność elementarną *wynik*(*i*) oznaczamy przez *E*(*i*).

Na przykład złożoność elementarna *wynik*(4) wynosi *E*(4) = 2, ponieważ wykonując *wynik*(4), wywołamy *wynik*(3) i *wynik*(1) (wykonanie elementarne), a z kolei przy wykonaniu *wynik*(3) wywołamy *wynik*(2) (drugie wykonanie elementarne).

Uzupełnij poniższą tabelę:

i	***E*(*i*)**
0	1
3	1
5	
7	
9	
10	

Okazuje się, że $E(i)$ można opisać rekurencyjnym wyrażeniem, którego niekompletną postać podajemy poniżej. Uzupełnij brakujące miejsca tak, aby $E(i)$ dawało poprawną złożoność elementarną *wynik*(*i*) dla każdego całkowitego nieujemnego *i*.

E(0) = E(1) = E(2) = 1
E(*i*) = E(.....) + E(.....) dla parzystego $i > 2$
E(*i*) = E(.....) dla nieparzystego $i > 2$

1.3.

Naszym celem jest wyznaczenie największej liczby spośród wartości funkcji *wynik*(0), *wynik*(1),…,*wynik*(1000) bez konieczności rekurencyjnego wyznaczania kolejnych wartości. Poniżej prezentujemy niekompletny algorytm realizujący to zadanie.

```
W[0] ← 1
W[1] ← 1
W[2] ← 1
max_wart ← 1
 dla i = 3, 4, …, 1 000 wykonuj
        jeżeli i mod 2 = 0
              W[i] ← .....
        w przeciwnym razie
              W[i] ←.....
        jeżeli W[i] > max_wart
                    .....
 zwróć max_wart
```

Uzupełnij brakujące miejsca w algorytmie tak, aby zwracał on największą liczbę spośród *wynik*(0), *wynik*(1),…,*wynik*(1000).

Komentarz do zadania

1.1.

Do rozwiązania tego zadania stosujemy definicję rekurencyjną *wynik*(*i*), wynikającą wprost z podanego pseudokodu:

wynik(0) = *wynik*(1) = *wynik*(2)=1
wynik(*i*) = *wynik*(*i* – 3) + *wynik*(*i* – 1) + 1 dla parzystych $i>2$
wynik(*i*) = *wynik*(*i* – 1) mod 7 dla nieparzystych $i>2$

A zatem:

- *wynik*(3) = *wynik*(2) mod 7 = 1 mod 7 = 1
- *wynik*(4) = *wynik*(3) + *wynik*(1) + 1 = 1 + 1 + 1 = 3

- *wynik*(5) = *wynik*(4) mod 7 = 3 mod 7 = 3
- *wynik*(6) = *wynik*(5) + *wynik*(3) + 1 = 3 + 1 + 1 = 5
- *wynik*(7) = *wynik*(6) mod 7 = 5 mod 7 = 5
- *wynik*(8) = *wynik*(7) + *wynik*(5) + 1 = 5 + 3 + 1 = 9

Zwróćmy uwagę, że w powyższym rozwiązaniu dla kolejnych argumentów (większych niż 2) funkcja *wynik* jest wywoływana wielokrotnie. Nie musimy jej jednak wielokrotnie wyznaczać, np. raz obliczony *wynik*(3) możemy wykorzystać przy każdym odwołaniu, bez ponownego obliczania.

1.2.

Najpierw rozwiążemy drugą część zadania, czyli podamy zależność rekurencyjną, określającą złożoność elementarną. Na podstawie tej zależności łatwo rozwiążemy pierwszą część zadania.

Ponieważ wykonania *wynik*(0), *wynik*(1) oraz *wynik*(2) kończą się natychmiastowym zwróceniem wyniku, każdemu z nich odpowiada dokładnie jedno wykonanie elementarne. To daje nam warunek brzegowy rekurencji:

$$E(0) = E(1) = E(2) = 1.$$

Natomiast dla każdego $i>2$ sytuacja jest zależna od parzystości liczby i:

- Dla parzystych $i>2$ wykonanie *wynik*(i) pociąga za sobą wykonanie *wynik*(i–1) oraz *wynik*(i–3). Zgodnie z przyjętymi oznaczeniami wykonanie *wynik*(i–1) wymaga $E(i–1)$ wykonań elementarnych. Analogicznie *wynik*(i–3) wymaga $E(i–3)$ wykonań elementarnych. Zatem *wynik*(i) pociąga za sobą $E(i–1) + E(i–3)$ wykonań elementarnych. Stąd:

  $$E(i) = E(i–1) + E(i–3) \text{ dla parzystego } i>2.$$

- Z kolei wykonanie *wynik*(i) dla nieparzystego $i > 2$ powoduje tylko wykonanie *wynik*(i–1). Stąd:

  $$E(i) = E(i–1) \text{ dla nieparzystego } i>2.$$

Ostatecznie:

$$E(i) = \begin{cases} 1, & dla \quad i \in \{0,1,2\}, \\ E(i-3)+E(i-1), & dla \quad parzystego \quad i>2, \\ E(i-1), & dla \quad nieparzystego \quad i>2, \end{cases}$$

Znając rekurencyjną formułę dla wyznaczania $E(i)$, możemy wypełnić podaną w zadaniu tabelkę, stosując tę formułę dla kolejnych i=3,4, …,10, podobnie jak w zadaniu 1 robiliśmy to dla funkcji *wynik*. Drobna różnica polega jedynie na tym, że w zadaniu 1 formuła zapisana była w innej postaci, a mianowicie w postaci pseudokodu.

1.3.

Naturalnym sposobem uniknięcia kolejnych wywołań rekurencyjnych jest tablicowanie wyników, co zostało zasugerowane w podanym niekompletnym algorytmie. Jeśli wartości *wynik*(0), *wynik*(1),…,*wynik*(i – 1) przechowywane są w komórkach $W[0]$, $W[1]$,…,$W[i – 1]$ tablicy W, to wartość *wynik*(i) możemy wyznaczyć jako $W[i–3]+W[i–1]+1$ dla i parzystych oraz $W[i]$ mod 7 dla i nieparzystych. Aby wyznaczoną wartość zapamiętać w odpowiedniej ko-

mórce tablicy *W*, użyjemy powyższych wyrażeń do uzupełnienia instrukcji podstawienia w podanym algorytmie.

Zmienna *max_wart*, jak wskazuje nazwa, może być użyta do przechowywania największej wyznaczonej dotychczas wartości funkcji *wynik*, zatem aktualizujemy ją zawsze, gdy *W*[i]>*max_wart*. Poniżej prezentujemy kompletne rozwiązanie:

```
W[0] ← 1
W[1] ← 1
W[2] ← 1
max_wart ← 1
 dla i = 3, 4, …, 1 000 wykonuj
        jeżeli i mod 2 = 0
              W[i] ← W[i – 1]+W[i – 3]+1
        w przeciwnym razie
              W[i] ← W[i] ←W[i – 1] mod 7
        jeżeli W[i] > max_wart
              W[i] ← max_wart
 zwróć max_wart
```

Rozwiązanie

1.1.

i	**wynik(*i*)**
2	1
3	1
4	3
5	3
6	5
7	5
8	9

1.2.

Poprawne wartości *E*(*i*) dla argumentów podanych w tabeli

i	**E(*i*)**
0	1
3	1
5	2
7	3
9	5
10	8

Poprawna definicja rekurencyjna:

E(0) = *E*(1) = *E*(2) = 1
E(*i*) = *E*(*i* – 1) + *E*(*i* – 3) dla parzystego *i* > 2
E(*i*) = *E*(*i* – 1) dla nieparzystego *i* > 2

1.3.

```
W[0] ← 1
W[1] ← 1
W[2] ← 1
max_wart ← 1
 dla i = 3, 4, …, 1000 wykonuj
        jeżeli i mod 2 = 0
                W[i] ← W[i – 1]+W[i – 3]+1
        w przeciwnym razie
                W[i] ← W[i – 1] mod 7
        jeżeli W[i] > max_wart
                W[i] ← max_wart
 zwróć max_wart
```

Zadanie 2.

Wiązka zadań *Ułamki dwójkowe*

W systemach pozycyjnych o podstawie innej niż 10 można zapisywać nie tylko liczby całkowite, ale również rzeczywiste z pewną dokładnością. Na przykład w systemie dwójkowym cyfry po przecinku odpowiadają kolejnym potęgom 1/2 (jednej drugiej). Cyfra 1 na pierwszym miejscu po przecinku odpowiada 1/2, na drugim miejscu — 1/4, na trzecim — 1/8 i tak dalej.

Na przykład $(0{,}101)_2 = 1/2 + 1/8 = 5/8 = 0{,}625_{10}$. Podobnie jak w systemie dziesiętnym nie każda liczba daje się zapisać w ten sposób dokładnie — na przykład liczba 1/3 nie ma skończonego rozwinięcia w systemie dwójkowym (ani też w dziesiętnym). Można jednak stosunkowo łatwo wyznaczyć zadaną liczbę początkowych cyfr po przecinku dla każdej liczby rzeczywistej.

Następujący algorytm przyjmuje na wejściu liczbę rzeczywistą *x* należącą do przedziału [0, 1) oraz dodatnią liczbę całkowitą *k* i wypisuje *k* pierwszych cyfr liczby *x* w zapisie dwójkowym. Przeanalizuj algorytm i odpowiedz na podane pytania.

Dane:
x — liczba rzeczywista, $0 \le x < 1$,
k — liczba całkowita dodatnia.
Wynik:
zapis dwójkowy liczby *x* do *k*-tego miejsca po przecinku.

```
funkcja binarny(x, k)
        wypisz „0,”
        y ← x
        dla i=1, 2, ..., k wykonuj
(*)             jeżeli y ≥ 1/2
                        wypisz „1”
                w przeciwnym razie
                        wypisz „0”
                y ← y * 2
                jeżeli y ≥ 1
                        y ← y – 1
```

2.1.

Podaj liczbę wypisaną przez algorytm dla $x = 0.6$, $k = 5$ oraz wartości zmiennej y przy każdorazowym wykonaniu wiersza oznaczonego (*).

Kolejne wykonanie (*)	**Wartość zmiennej** *y*
1	
2	
3	
4	
5	

Liczba wypisana przez algorytm: .....

2.2.

Podaj przykład liczby *x,* dla której po wykonaniu funkcji *binarny(x,4)* zmienna *y* ma wartość 0, a po wykonaniu funkcji *binarny(x, 3)* zmienna *y* nie jest równa 0.

2.3.

W systemie trójkowym używa się cyfr 0, 1 i 2. Cyfra 1 na pierwszym miejscu po kropce oznacza 1/3, zaś 2 oznacza 2/3. Na drugim miejscu są to odpowiednio 1/9 i 2/9, na trzecim — 1/27 i 2/27 i tak dalej, z kolejnymi potęgami trójki w mianownikach.

Poniżej podany jest algorytm wypisujący dla zadanej liczby rzeczywistej x z przedziału [0,1) oraz liczby całkowitej dodatniej k pierwsze k cyfry zapisu x w systemie trójkowym. Uzupełnij luki tak, aby algorytm działał prawidłowo.

```
funkcja trójkowy(x, k)
    wypisz „0,”
    y ← x
    dla i = 1, 2, ..., k wykonuj
        jeżeli y ≥ 2/3
            wypisz „2”
        jeżeli .....
            wypisz „1”
        jeżeli .....
            wypisz „0”
        y = y * 3
        jeżeli y≥2
            .....
        jeżeli y≥1
            .....
```

Komentarz do zadania

2.1.

Algorytm zaczyna od wypisania zera i przecinka dziesiętnego. Następnie zaczyna się główna pętla: w pierwszej iteracji $y = 0{,}6$, a zatem pierwszą po przecinku cyfrą jest 1. Mnożymy y przez 2 i jeśli przekroczy 1, odejmujemy 1. Ponieważ $0{,}6 \cdot 2 = 1{,}2$, to po tej iteracji zmienna y

przybierze wartość 0,2. W kolejnym obrocie pętli wypiszemy cyfrę 0 (jako że 0,2 < 0,5), po czym podwoimy *y*, otrzymując 0,4. Kontynuując w ten sposób działania, dojdziemy do odpowiedzi takiej jak wzorcowa.

Można przy okazji zauważyć, że po czwartej iteracji *y* znowu przybierze wartość 0,6, a zatem dalsze kroki algorytmu, gdybyśmy wykonali ich więcej, byłyby identyczne z pierwszymi czterema. Widać stąd, że $(0,6)_2 = 0,1001100110011...$

2.2.

Wartość zmiennej *y* równa zero oznacza po prostu, że już wszystkie dalsze cyfry dwójkowe liczby *x* są zerami, innymi słowy, że rozwinięcie dwójkowe *x* się skończyło.

W zadaniu pytamy zatem o liczbę, która po trzech iteracjach ma jeszcze dalsze niezerowe cyfry dwójkowe do wypisania, ale po czterech już nie ma. Musi to więc być liczba, która w rozwinięciu dwójkowym ma dokładnie cztery cyfry po przecinku. Najmniejszą taką liczbą jest $(0,0001)_2$, czyli 1/16 = 0,0625. Innymi możliwymi odpowiedziami są na przykład $(0,0011)_2 = 3/16 = 0,1875$ albo $(0,1111)_2 = 15/16 = 0,9375$.

2.3.

Algorytm *binarny(x, k)* opiera się na następującym pomyśle: jeśli liczba *x* jest nie mniejsza od 1/2, to jej pierwszą cyfrą dwójkową po przecinku musi być 1. Jeśli teraz pomnożymy liczbę *x* przez 2, to odpowiada to przesunięciu całego rozwinięcia o jedno miejsce w lewo. Druga po przecinku cyfra liczby *x* to pierwsza cyfra po przecinku *2x*, czyli możemy ją wyznaczyć podobnie jak poprzednio: sprawdzając, czy jest większa lub równa 1/2. Oczywiście musimy najpierw pominąć stojącą przed przecinkiem część całkowitą — odejmujemy zatem 1, jeśli liczba przekroczyła 1 w czasie mnożenia.

Bardzo podobną technikę stosujemy w algorytmie *trojkowy(x, k)*. Tutaj pierwsza cyfra po przecinku powinna być równa 2, jeśli liczba *x* jest nie mniejsza od 2/3, 1 — jeśli *x* należy do przedziału [1/3, 2/3), a 0 — jeśli *x* jest mniejsze od 1/3. Następnie dokonujemy przesunięcia w lewo, mnożąc liczbę przez 3, po czym usuwamy z niej część całkowitą. Istotną różnicą jest to, że teraz część całkowita liczby może wynosić 0, 1 lub 2, zatem musimy odjąć 1 lub 2, zależnie od potrzeby:

```
dla i=1, 2, ..., k wykonuj
        jeżeli y ≥ 2/3
                wypisz „2”
        jeżeli y ≥ 1/3 oraz y < 2/3
                wypisz „1”
        jeżeli y < 1/3
                wypisz „0”
        y ← y * 3
        jeżeli y≥2
                y ← y – 2
        jeżeli y≥1
                y ← y – 1
```

Zamiast ostatnich czterech wierszy — odpowiadających pomijaniu części całkowitej — można było w oryginalnym algorytmie napisać tak:

jeżeli y≥2
    y ← y – 1
jeżeli y≥1
    y ← y – 1

Jeśli liczba będzie większa niż 2, algorytm najpierw odejmie 1, a następnie zauważy, że liczba wciąż jest większa niż 1, i odejmie znów jedynkę. Warto jeszcze wspomnieć, że gdyby luki nie narzucały konstrukcji algorytmu, można byłoby użyć krótszego zapisu:

dopóki y≥1 **wykonuj**
    y ← y – 1

Miałoby to ten sam efekt: odrzucenie z liczby *y* jej części całkowitej.

Zadanie 3.

Wiązka zadań *Ciekawe mnożenia*

Dana jest następująca funkcja rekurencyjna:

Dane:

x — liczba całkowita,

n — dodatnia liczba całkowita.

```
funkcja F(x, n)
     jeżeli n = 1
          podaj wynik x i zakończ
     w przeciwnym razie
          jeżeli n mod 3 = 0
               k ← F(x, n div 3)
(*)            podaj wynik k*k*k i zakończ
          w przeciwnym razie
(**)           podaj wynik x*F(x, n-1) i zakończ
```

Uwaga: „div” jest operatorem dzielenia całkowitego.

3.1.

Podaj wszystkie wywołania rekurencyjne funkcji F oraz obliczany po każdym wywołaniu wynik, jeśli na początku wywołamy F(2, 10).

wywołanie	wynik
F(2, 10)	
F(;)	

3.2.

Uzupełnij tabelę o brakujące elementy:

x	***n***	wynik **F(*x*, *n*)**
2	2	4
2	3	
3		81
	5	32
2		256
	10	1024

3.3.

Uzupełnij tabelę, podając łączną liczbę mnożeń wykonanych w wierszach oznaczonych (*) i (**) po wywołaniu F dla podanych argumentów *x* i *n*:

x	***n***	Liczba operacji mnożenia
2	2	1
2	3	
3	4	
4	7	
4	8	
4	9	

3.4.

Podaj, która z poniższych funkcji określa liczbę wszystkich operacji mnożenia wykonywanych przez powyższy algorytm dla argumentu *n* będącego potęgą trójki ($n = 3^m$ dla pewnego nieujemnego *m*):

- $lmnozen(n) = n \operatorname{div} 2$
- $lmnozen(n) = \log_2 n$
- $lmnozen(n) = 2 \cdot \log_3 n$
- $lmnozen(n) = 1 + \sqrt{n}$

Zadanie 4.

Wiązka zadań *Silniowy system pozycyjny*

Pojęcie *silni* dla liczb naturalnych większych od zera definiuje się następująco:

$$n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (n-1) \cdot n$$

Silniowy system pozycyjny to pozycyjny sposób zapisu liczb naturalnych, w którym mnożniki dla kolejnych pozycji są definiowane przez silnie kolejnych liczb naturalnych, tzn.

$$(x)_! = (x_n x_{n-1}\, x_{n-2} \ldots x_2 x_1)_! = x_n \cdot n! + x_{n-1} \cdot (n-1)! + \cdots + x_2 \cdot 2! + x_1 \cdot 1!$$

W systemie silniowym współczynnik x_i, który odpowiada mnożnikowi $i!$, spełnia zależność $0 \leq x_i \leq i$.

Zapis każdej liczby w silniowym systemie pozycyjnym jest jednoznaczny, tzn. każdą liczbę naturalną można zapisać tylko w jeden sposób i każdą liczbę naturalną można zapisać dokładnie w jeden sposób.

Uwaga: W poniższych zadaniach będziemy mieć do czynienia tylko z takimi liczbami, dla których współczynniki x_i spełniają zależność $0 \leq x_i \leq 9$.

Przykład

$$(1220)_! = 1 \cdot 4! + 2 \cdot 3! + 2 \cdot 2! + 0 \cdot 1! = 24 + 12 + 4 + 0 = 40.$$

4.1.

Uzupełnij tabelę. Zamień zapis liczby w systemie silniowym na jej zapis w systemie dziesiętnym.

liczba w systemie silniowym	**liczba w systemie dziesiętnym**
$(310)_!$	
$(2011)_!$	
$(54211)_!$	

4.2.

Podaj zapis w systemie silniowym największej liczby, jaką można w tym systemie zapisać na pięciu pozycjach.

4.3.

Zamiana zapisu liczby w systemie dziesiętnym na zapis w systemie silniowym może przebiegać według następującego schematu: Szukamy największej liczby *k*, której silnia nie przekracza liczby *x*. Pierwsza jej cyfra to wynik dzielenia całkowitego *x* przez *k*!. Kolejne cyfry zapisu silniowego (zaczynając od cyfr najbardziej znaczących) otrzymujemy przez wyznaczanie wyników dzielenia liczby *x* przez (*k* – 1)!, (*k* –2)!, ..., 2!, 1!. Po wyznaczeniu cyfry x_i, odpowiadającej współczynnikowi *i*!, zmniejszamy wartość *x* o liczbę odpowiadającą cyfrze x_i, czyli $x_i \cdot i!$. Oznacza to, że *x* przyjmuje wartość *x* mod *k*!.

Przykład

x	***k***	***x* div *k*!**	***x* mod *k*!**
1548	6	2	**108**
108	5	0	108
108	4	4	12
12	3	2	0
0	2	0	0
0	1	0	0

Liczba dziesiętna 1548 w zapisie silniowym: $(204200)_!$

Wykonaj zamianę liczby 5489 z systemu dziesiętnego na silniowy zgodnie z opisanym powyżej algorytmem. Uzupełnij poniższą tabelkę oraz podaj zapis silniowy liczby 5489.

x	***k***	***x* div *k*!**	***x* mod *k*!**
5489			

Liczba dziesiętna 5489 w zapisie silniowym: .**.....**

4.4.

Poniżej przedstawiono algorytm z lukami, który zamienia zapis liczb z systemu dziesiętnego na system silniowy. Uzupełnij luki w tym algorytmie.

Specyfikacja

Dane:

x — liczba całkowita dodatnia zapisana w systemie dziesiętnym,

Wynik:

s — napis reprezentujący liczbę *x* zapisaną w systemie silniowym.

silnia ← 1
k ←1
dopóki (*silnia* < *x*) **wykonuj**
    k ← *k* + 1
    silnia← *silnia** *k*
jeżeli **.....**
    silnia ← *silnia* div *k*
    k ← *k* – 1
s ←” ”
dopóki (*k*>0) **wykonuj**
    cyfra ← **.....**
    s ← *s* ◦ tekst (*cyfra*)
    x ← **.....**
    silnia ← **.....**
    k ←*k* – 1

Uwaga
tekst (*x*) oznacza funkcję zamieniającą liczbę *x* na jej zapis tekstowy
” ” oznacza napis pusty
u ◦ *v* oznacza sklejenie dwóch napisów: *u* oraz *v*

Zadanie 5.

Wiązka zadań *Sortowanie przez wstawianie na dwa sposoby*

Sortowanie przez wstawianie polega na powtarzaniu operacji wstawiania elementu do już uporządkowanego ciągu. Aby znaleźć w uporządkowanym ciągu miejsce, w które należy wstawić nowy element, można stosować różne strategie. Poniższy algorytm znajduje to miejsce metodą wyszukiwania binarnego.

Specyfikacja

Dane:

n — liczba naturalna oznaczająca długość ciągu,

A[1..n] — ciąg liczb całkowitych zapisanych w tablicy

Wynik:

A[1..n] — tablica liczb całkowitych, w której liczby zostały ustawione w porządku niemalejącym

Algorytm

dla $j = n - 1, n - 2, \ldots, 1$ **wykonuj**
  $x \leftarrow A[j]$
  $p \leftarrow j$
  $k \leftarrow n + 1$
  dopóki $k - p > 1$ **wykonuj**
    $i \leftarrow (p + k)$ **div** 2
    jeżeli $x \leq A[i]$
      $k \leftarrow i$
    w przeciwnym razie
      $p \leftarrow i$
  dla $i = j, j + 1, \ldots, p - 1$ **wykonuj**
    $A[i] \leftarrow A[i + 1]$
  $A[p] \leftarrow x$

5.1.

Przeanalizuj działanie powyższego algorytmu dla ciągu 12, 4, 3, 10, 5, 11 o długości $n = 6$ i podaj, ile razy zostaną wykonane instrukcje $k \leftarrow i$ i $p \leftarrow i$ dla kolejnych wartości j zamieszczonych w tabeli.

Wartość *j*	**Liczba wykonań**	
	k* ← *i	***p* ← *i***
5		
4		
3		
2		
1		

5.2.

Uzupełnij luki w poniższym algorytmie sortowania przez wstawianie tak, aby znajdowanie miejsca na kolejny wstawiany element było realizowane metodą wyszukiwania liniowego.

Specyfikacja

Dane:

n — liczba naturalna oznaczająca długość ciągu, A[1..n] — ciąg liczb całkowitych zapisanych w tablicy.

Wynik:

A[1..n] — tablica liczb całkowitych, w której liczby zostały ustawione w porządku niemalejącym.

Algorytm:

dla $j = n - 1, n - 2, \ldots, 1$ **wykonuj**
  $x \leftarrow \ldots\ldots\ldots$
  $i \leftarrow \ldots\ldots\ldots$
  dopóki $(i \leq n)$ **i** $(x >$ A[i]) **wykonuj**
    $A[i - 1] \leftarrow A[i]$
    $i \leftarrow i + 1$
  $\ldots\ldots \leftarrow x$

5.3.

Porównaj dwa algorytmy sortowania przez wstawianie: taki, w którym miejsce dla wstawianego elementu jest znajdowane metodą wyszukiwania binarnego, i taki, w którym jest ono znajdowane metodą wyszukiwania liniowego. Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe (P), a które jest fałszywe (F).

Oba algorytmy dla ciągu 12, 4, 3, 10, 5, 11 wykonują

	P	F
jednakową liczbę porównań między elementami ciągu liczb.		
jednakową liczbę przesunięć elementów w tablicy.		
tyle samo powtórzeń pętli zewnętrznej w algorytmie.		
jednakową liczbę instrukcji podstawienia wartości do zmiennej *x*.		

Zadanie 6.

Wiązka zadań *Od szczegółu do ogółu*

Rozważmy następujący algorytm:

Dane:

k — liczba naturalna,

A[1...2^k] — tablica liczb całkowitych.

Algorytm 1:

```
    n ← 1
    dla i=1,2,…,k wykonuj
        n ← 2·n
    s ← 1
    dopóki s<n wykonuj
        j ← 1
        dopóki j<n wykonuj
(*)         jeżeli A[j]>A[j+s]
(**)            zamień(A[j],A[j+s])
            j ← j+2·s
        s ← 2·s
    zwróć A[1]
```

Uwaga: Funkcja *zamień*(*A*[*j*],*A*[*j*+*s*]) zamienia miejscami wartości *A*[*j*] oraz *A*[*j*+*s*].

6.1.

Prześledź działanie algorytmu 1 dla podanych poniżej wartości *k* i początkowych zawartości tablicy *A*. W każdym wierszu poniższej tabeli wpisz końcową zawartość tablicy *A*.

k	Początkowa zawartość tablicy A[1...2^k]	Końcowa zawartość tablicy A[1...2^k]
2	[4, 3, 1, 2]	[1, 4, 3, 2]
2	[2, 3, 4, 1]	
3	[1, 2, 3, 4, 5, 6, 7, 8]	
3	[8, 7, 6, 5, 4, 3, 2, 1]	
3	[4, 5, 6, 1, 8, 3, 2, 4]	

6.2.

Wskaż, które z poniższych zdań są prawdziwe (P), a które fałszywe (F), wstawiając znak X w odpowiedniej kolumnie:

	P	F
Po zakończeniu działania algorytmu 1 komórka $A[2^k]$ zawiera największą z liczb $A[1],\ldots,A[2^k]$.		
Po zakończeniu działania algorytmu 1 spełniona jest nierówność $A[i] \leq A[i+1]$ dla każdego i, takiego że $1 \leq i \leq 2^k$.		
Po zakończeniu działania algorytmu 1 komórka $A[1]$ zawiera najmniejszą z liczb $A[1],\ldots,A[2^k]$.		

6.3.

Wskaż, które z poniższych zdań są prawdziwe (P), a które fałszywe (F), wstawiając znak X w odpowiedniej kolumnie. Przyjmij, że $n=2^k$ oraz $k>1$:

	P	F
Instrukcja jeżeli w wierszu (*) jest wykonywana mniej niż $2n$ razy.		
Instrukcja jeżeli w wierszu (*) jest wykonywana mniej niż $n/2$ razy.		
Możliwe jest dobranie takiej początkowej zawartości $A[1..2^k]$, że instrukcja zamiany z wiersza (**) nie zostanie wykonana ani razu.		
Możliwe jest dobranie takiej początkowej zawartości $A[1..2^k]$, że instrukcja zamiany z wiersza (**) zostanie wykonana co najmniej $2n^2$ razy.		

6.4.

Rozważmy poniższy algorytm podobny do **algorytmu 1**.

Wejście: k — liczba naturalna,
$A[1...2^k]$ — tablica liczb całkowitych.

Algorytm 2:

```
    n ← 1
    dla i=1,2,…,k wykonuj
        n ← 2·n
    s ← 1
    dopóki s<n wykonuj
        j ← 1
        dopóki j<n wykonuj
(*)         jeżeli A[j]>A[j+1]
                zamień(A[j],A[j+1])
            j ← j+1
        s ← s+1
    zwróć A[1], A[2],…,A[n]
```

Uwaga: Funkcja *zamień*(*A*[*j*],*A*[*j*+1]) zamienia miejscami wartości *A*[*j*] oraz *A*[*j*+1].

Uzupełnij luki w poniższych zdaniach. Przyjmij $n=2^k$ oraz $k>1$.

Po zakończeniu działania algorytmu 2 element *A*[*i*] jest .....
niż element *A*[*i*+1] dla każdego *i* większego od .....
oraz mniejszego od.....

Wiersz (*) **algorytmu 2** wykonywany będzie w przebiegu algorytmu

- ..... niż n razy,
- ..... niż n^2 razy.

Zadanie 7.

Wiązka zadań *Scalanie*

Rozważmy następujący algorytm, który jako dane przyjmuje tablicę *n*-elementową, gdzie *n* jest potęgą dwójki:

Dane: tablica liczb rzeczywistych T[1..*n*], gdzie $n = 2^m$, a *m* jest liczbą całkowitą nieujemną

```
funkcja uporządkuj(T[1..n]):
    jeżeli n=1
          zwróć T[1..n] i zakończ
    k ← n/2
    A[1..k] ← uporządkuj(T[1 .. k])
    B[1..k] ← uporządkuj(T[k+1 .. n])
    zwróć scal(A, B) i zakończ
```

Funkcja *scal*(*A*,*B*) dla danych dwóch tablic o rozmiarze *k* zwraca tablicę o rozmiarze *2k*, powstałą przez połączenie tablic *A* i *B* w sposób uporządkowany, tj. od elementu najmniejszego do największego. Na przykład dla tablic *A* = [4,6,18,22] i *B* = [1,3,10,15] wywołanie *scal*(*A*,*B*) zwróci tablicę [1,3,4,6,10,15,18,22].

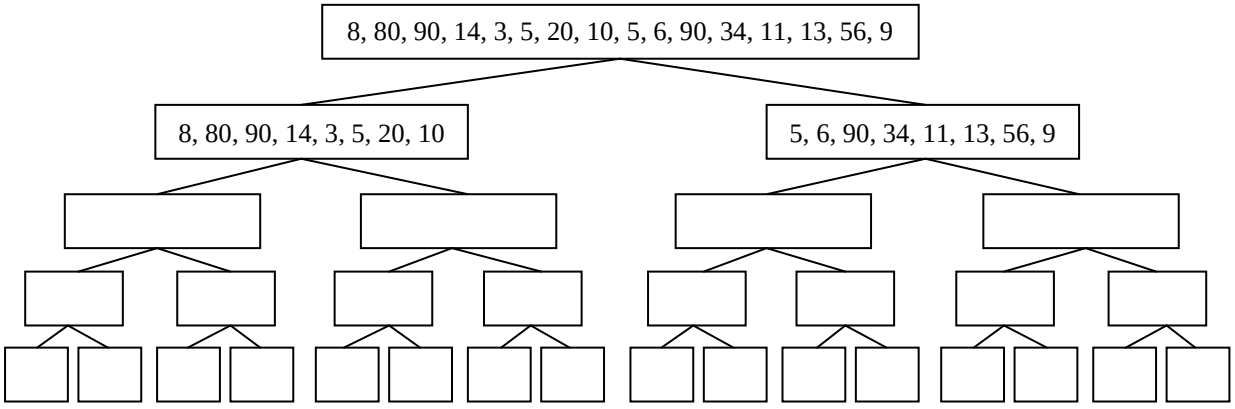
7.1.

Spośród danych tablic wybierz te, które są zgodne ze specyfikacją algorytmu, a następnie podaj dla nich wynik jego działania.

- *T* = [15, 11],
- *T* = [1, 3, 8],
- *T* = [8, 4, 2, 1],
- *T* = [10, 15, 1 , 6, 9, 2, 5, 90].

7.2.

Funkcja *uporządkuj* jest funkcją rekurencyjną. Uzupełnij poniższe drzewo wywołań rekurencyjnych dla danej tablicy *T* = [8, 80, 90, 14, 3, 5, 20, 10, 5, 6, 90, 34, 11, 13, 56, 9].



7.3.

Załóżmy, że wywołanie procedury *scal*(*A*,*B*) dla dwóch tablic o długości *k* wykonuje *2k*–1 kosztownych operacji (porównywania liczb). Podaj liczbę kosztownych operacji, jaka zostanie wykonana przez funkcję *uporządkuj* dla tablicy

$$T[1..16] = [\ 8, 80, 90, 14, 3, 5, 20, 10, 5, 6, 90, 34, 11, 13, 56, 9\ ].$$

Zadanie 8.

Wiązka zadań *Dwa ciągi*

Niech *A*[1..*n*] i *B*[1..*n*] będą uporządkowanymi rosnąco tablicami liczb całkowitych i niech *x* będzie liczbą całkowitą. Rozważmy następujący algorytm:

```
    i←1
    j←n
    dopóki  (i ≤ n oraz j > 0) wykonuj

    dopóki (i ≤ n  oraz  j > 0)wykonuj
(*)     jeżeli A[i] + B[j] = x
            podaj wynik PRAWDA i zakończ algorytm
        w przeciwnym razie
                jeżeli A[i] + B[j] < x
                        i← i + 1
                w przeciwnym razie
                        j← j – 1
    podaj wynik FAŁSZ
```

8.1.

Uzupełnij poniższą tabelę. Podaj wynik działania algorytmu oraz liczbę porównań wykonanych w wierszu oznaczonym (*).

Tablica *A*	**Tablica *B***	***x***	**Wynik działania algorytmu**	**Liczba porównań w kroku (*)**
3, 5, 12, 17	8, 10, 13, 14	21		
4, 6, 8, 10	5, 7, 9, 11	13		

8.2.

Przeanalizuj działanie zaprezentowanego algorytmu i uzupełnij poniższą specyfikację.

Dane:

n — dodatnia liczba całkowita
A[1..*n*], *B*[1..*n*] — *n*-elementowe tablice liczb całkowitych, posortowane rosnąco
x — liczba całkowita

Wynik:

PRAWDA, gdy .....

FAŁSZ, gdy .....

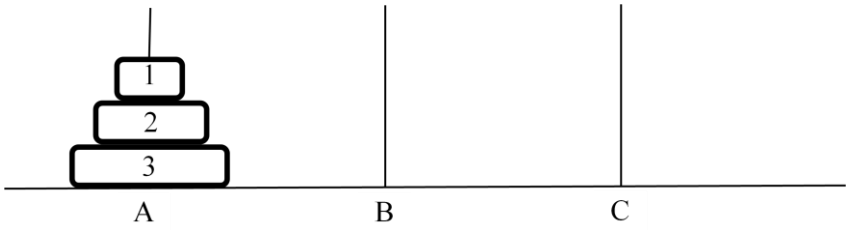
8.3.

Podaj przykład pięcioelementowych tablic *A* i *B*, dla których przy $x = 20$ algorytm wykona sześć porównań w wierszu (*), a wynikiem będzie *PRAWDA*.

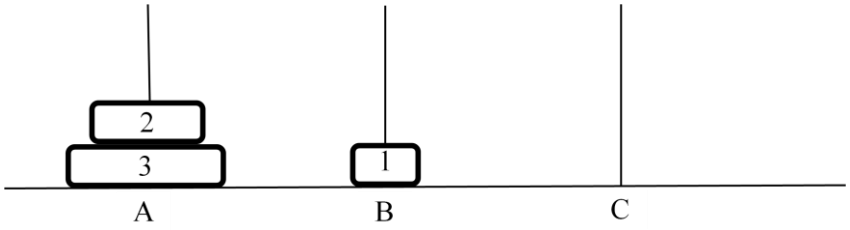
Zadanie 9.

Wiązka zadań *Wieże z Hanoi*

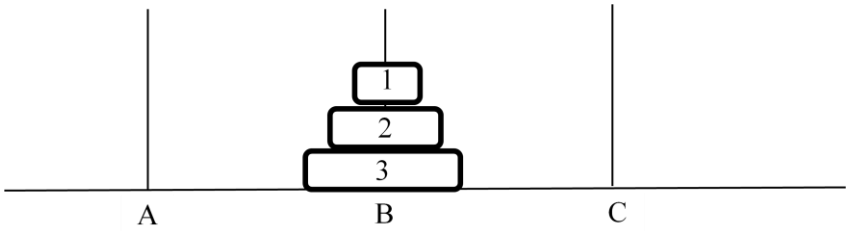
W problemie wież z Hanoi mamy trzy pręty oznaczone A, B i C oraz n okrągłych krążków o średnicach odpowiednio 1, 2, …, n. Na początku wszystkie krążki nałożone są na pręt A, w kolejności od największego do najmniejszego (największy na dole, najmniejszy na górze). Układ ten (dla n=3) został przedstawiony na poniższym rysunku.



Zgodnie z regułami problemu krążki można przekładać między prętami. W jednym ruchu możliwe jest przełożenie krążka znajdującego się na szczycie jednego z prętów na szczyt innego pręta, pod warunkiem że nie kładziemy przekładanego krążka na krążek mniejszy od niego. Na przykład na poniższym rysunku krążek 2 możemy przełożyć z pręta A na pręt C, natomiast niemożliwe jest przełożenie go na pręt B.



Zadanie polega na przełożeniu wszystkich krążków z pręta A na pręt B, przy czym można korzystać z pomocniczego pręta C. Na poniższym rysunku przedstawiono efekt końcowy.



Problem wież z Hanoi można rozwiązać za pomocą algorytmu rekurencyjnego. W algorytmie pręty: startowy, docelowy i pomocniczy, podane są jako parametry wejściowe, odpowiednio *x*, *y* i *z*. Algorytm polega na tym, że najpierw przenosimy $n - 1$ krążków na pręt pomocniczy *z*, potem największy krążek zostaje przeniesiony na pręt docelowy *y*, a na koniec $n - 1$ krążków zostaje przeniesionych z pręta pomocniczego *z* na pręt docelowy *y*, przy czym pręt startowy *x* traktowany jest jako pomocniczy.

Algorytm

Specyfikacja

Dane:

n — liczba całkowita dodatnia,

x — nazwa pręta startowego,

y — nazwa pręta docelowego,

z — nazwa pręta pomocniczego.

Wynik:

ciąg ruchów opisujący rozwiązanie problemu wież z Hanoi z n krążkami, w którym na początku wszystkie krążki znajdują się na pręcie x, a na końcu mają znaleźć się na pręcie y, zaś pomocniczym prętem jest z.

Uwaga: Pojedynczy ruch zapisujemy za pomocą znaku $\Rightarrow$. Na przykład C $\Rightarrow$ B oznacza przeniesienie krążka z pręta C na pręt B.

```
funkcja wieże(n, x, y, z)
    jeżeli n=1
        wypisz x ⇒ y
    w przeciwnym razie
        wieże(n – 1, x, z, y)
        wypisz x ⇒ y
        wieże(n – 1, z, y, x)
```

Przykład

Wywołanie *wieże*(2, A, B, C) spowoduje dwa wywołania rekurencyjne: *wieże*(1, A, C, B) oraz wieże(1, C, B, A). Ciąg ruchów utworzony przez *wieże*(2, A, B, C) ma postać:

$$A \Rightarrow C, A \Rightarrow B, C \Rightarrow B,$$

gdzie podkreślone ruchy są utworzone przez rekurencyjne wywołania *wieże*(1, A, C, B) oraz *wieże*(1, C, B, A).

9.1.

Podaj wszystkie wywołania rekurencyjne funkcji *wieże* (wraz z ich parametrami), do których dojdzie w wyniku wywołania *wieże*(3, A, B, C). Odpowiedź podaj w poniższej tabeli, uzupełniając parametry wszystkich wywołań rekurencyjnych.

n	*x*	*y*	*z*
3	A	B	C
2	A	C	B
1	A	B	C
1			
2			
1			
1			

9.2.

Prześledź działanie *wieże*(3, A, B, C) i uzupełnij poniżej wygenerowany ciąg ruchów:

A⇒B; A⇒C; ……

9.3.

Niech H(*n*) oznacza liczbę ruchów wykonanych przez podany algorytm dla *n* krążków. Zauważ, że rozwiązanie problemu dla *n*>1 krążków wymaga jednego ruchu oraz dwukrotnego rozwiązania problemu dla *n* – 1 krążków. W oparciu o tę obserwację uzupełnij poniższą tabelę.

n	H(*n*)
1	1
2	3
3	
4	
5	
7	
10	

Podaj ogólny wzór określający liczbę ruchów dla *n* krążków:

H(*n*)= .....

9.4.

Poniżej prezentujemy nierekurencyjne rozwiązanie problemu wież z Hanoi.

Specyfikacja

Dane:

n — liczba całkowita dodatnia,

Wynik:

ciąg ruchów opisujący rozwiązanie problemu wież z Hanoi z n krążkami, w którym na początku wszystkie krążki znajdują się na pręcie A, a na końcu powinny się znaleźć na pręcie B.

Algorytm:

(1) **dopóki** (pręt A jest niepusty lub pręt C jest niepusty) **wykonuj**
(2) **jeżeli** n jest parzyste:
(3) **przenieś** krążek nr 1 o jedną pozycję w lewo
(4) **w przeciwnym razie**
(5) **przenieś** krążek nr 1 o jedną pozycję w prawo
(6) **przenieś** krążek między prętami, na których nie ma krążka nr 1

W powyższym algorytmie przeniesienie krążka nr 1 o jedną pozycję w prawo oznacza wykonanie jednego z ruchów A⇒B, B⇒C lub C⇒A, tak aby krążek nr 1 został przeniesiony na inny pręt. Analogicznie przeniesienie krążka w lewo oznacza wybranie jednego z ruchów A⇒C, B⇒A lub C⇒B, tak aby krążek nr 1 został przeniesiony na inny pręt.

Ruch w kroku (6) powyższego algorytmu jest określony jednoznacznie, gdyż dopuszczalne jest tylko położenie mniejszego krążka na większym, a nie odwrotnie.

Przykład

Dla *n*=3 powyższy algorytm wykona następujący ciąg ruchów:

A⇒B; A⇒C; B⇒C; A⇒B; C⇒A; C⇒B; A⇒B,

gdzie ruchy podkreślone przenoszą krążek nr 1 o jedną pozycję w prawo.

Wypisz ciąg ruchów, który poda powyższy algorytm dla *n*=4.

Zadanie 10.

Wiązka zadań *Dzielenie wielomianu*

Rozważamy wielomiany $P(x)$ stopnia $n-1$, gdzie n jest potęgą dwójki:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}, \qquad n = 2^m.$$

Do obliczania wartości $P(x)$ można zastosować technikę „dziel i zwyciężaj" w następujący sposób: dzielimy postać ogólną wielomianu $P(x)$ na dwie części:

$$P(x) = \underbrace{a_0 + a_1 x + \cdots + a_{k-1} x^{k-1}}_{\text{cz. I}} + \underbrace{a_k x^k + a_{k+1} x^{k+1} + \cdots + a_{n-1} x^{n-1}}_{\text{cz. II}},$$

gdzie $k = n/2$. Jeśli w drugiej części wydzielimy czynnik x^k, to otrzymamy równość

(*) $$P(x) = A(x) + B(x) \cdot x^k,$$

gdzie $A(x)$ i $B(x)$ są wielomianami stopnia $k-1$:

$$A(x) = a_{k-1} x^{k-1} + \cdots + a_2 x^2 + a_1 x + a_0,$$
$$B(x) = a_{n-1} x^{k-1} + \cdots + a_{k+2} x^2 + a_{k+1} x + a_k,$$

W ten sposób problem obliczenia wartości $P(x)$ dzielimy na dwa podproblemy — obliczenia $A(x)$ i obliczenia $B(x)$, przy czym każdy z nich ma rozmiar o połowę mniejszy. Na przykład aby obliczyć wartość wielomianu ($n = 8$)

$$P(x) = -8 + 7x + 6x^2 - x^3 + 4x^4 + 5x^5 - 2x^6 + 3x^7,$$

obliczamy $k = n/2 = 4$, $A(x) = -8 + 7x + 6x^2 - x^3$ oraz $B(x) = 4 + 5x - 2x^2 + 3x^3$. Następnie korzystamy ze wzoru $P(x) = A(x) + B(x) \cdot x^4$.

Ponieważ do pełnej realizacji algorytmu we wzorze (*) potrzebne jest obliczenie wartości x^k, więc najpierw obliczamy wszystkie potęgi $x, x^2, x^4, \ldots, x^{2^{m-1}}$ i zapamiętujemy je w tablicy Z[0 .. m-1], np. za pomocą poniższej procedury.

Obliczanie tablicy $Z[0..m\text{-}1]$:

Dane:

n — liczba całkowita postaci 2^m, gdzie m jest liczbą całkowitą nieujemną,

x — liczba rzeczywista.

Wynik:

tablica $Z[0 .. m\text{-}1]$, dla której $Z[j] = x^{2^j}$

$Z[0] \leftarrow x$
$m \leftarrow 1$
$w \leftarrow 2$
dopóki $w < n$ **wykonuj**
    $Z[m] \leftarrow Z[m-1] \cdot Z[m-1]$
    $m \leftarrow m + 1$
    $w \leftarrow 2 \cdot w$

Mając tablicę Z, obliczamy wartość $P(x)$ za pomocą funkcji rekurencyjnej *F*.

Dane:

tablica liczb rzeczywistych $T[0\,..\,n-1]$ *ze współczynnikami* $a_0, a_1, \ldots, a_{n-1}$, *gdzie* $n = 2^m$, *a m jest liczbą całkowitą nieujemną;*

tablica Z[0 .. m-1], dla której $Z[j] = x^{2^j}$.

Wynik:

wartość $a_{n-1}x^{n-1} + \cdots + a_2x^2 + a_1x + a_0$.

funkcja F($T[0\,..\,n-1]$):
  jeżeli *n*=1
    zwróć T[0] **i zakończ**
  $k \leftarrow n/2$
  $A[0\,..\,k-1] \leftarrow T[0\,..\,k-1]$
  $B[0\,..\,k-1] \leftarrow T[k\,..\,n-1]$
  zwróć $\mathrm{F}(A) + \mathrm{F}(B) \cdot Z[m-1]$ **i zakończ**

Prześledźmy na przykład obliczenie wartości wielomianu

$$P(x) = -8 + 7x + 6x^2 - x^3 + 4x^4 + 5x^5 - 2x^6 + 3x^7 \qquad (n = 2^m, \qquad m = 3)$$

dla *x*=3.

W pierwszym kroku algorytm obliczy tablicę $Z[0\,..\,2]$:

$$Z[0] = 3, \qquad Z[1] = 3^2, \qquad Z[2] = 3^4 = 81.$$

Następnie algorytm obliczy *F*([-8,7,6,-1,4,5,-2,3]), tzn.

F([-8,7,6,-1,4,5,-2,3]) = *F*([-8,7,6,-1]) + *F*([4,5,-2,3])*Z[2]

Obliczanie *F*([4,5,-2,3]) i *F*([-8,7,6,-1]) odbywa się w analogiczny sposób, tzn.

F([-8,7,6,-1]) = *F*([-8,7])+ *F*([6,-1])*Z[1]
= F([-8])+F([7])*Z[0]+(F([6])+F([-1])*Z[0])*Z[1]
= (-8)+7*Z[0] + (6+(-1)*Z[0])*Z[1]
= (-8)+7*3+(6+(-1)*3)*9 = 40,

F([4,5,-2,3]) = *F*([4,5])+*F*([-2,3])*Z[1]
= *F*([4])+*F*([5])*Z[0]+(*F*([-2])+*F*([3])*Z[0])*Z[1]
= 4+5*Z[0]+((-2)+3*Z[0])*Z[1]
= 4+5*3+((-2)+3*3)*9=82.

Zatem

$$F([-8,7,6,-1,4,5,-2,3]) = F([-8,7,6,-1])+F([4,5,-2,3])*Z[2] = 40+82*81 = 6682.$$

Można zauważyć, że podczas obliczania *F*([-8,7,6,-1,4,5,-2,3]) łącznie zostanie wykonanych 7 mnożeń:

- 3 mnożenia podczas obliczania *F*([-8,7,6,-1])=40,
- 3 mnożenia podczas obliczania *F*([4,5,-2,3])=82,
- 1 mnożenie przy obliczaniu *F*([-8,7,6,-1])+ *F*([4,5,-2,3])*Z[2] = 40+82*Z[2].

Liczba wszystkich wywołań rekurencyjnych jest równa 15. Oto wszystkie wywołania funkcji *F*:

- *F*([-8,7,6,-1,4,5,-2,3])
  - *F*([-8,7,6,-1])
    - *F*([-8,7])
      - *F*([-8])
      - *F*([7])
    - *F*([6,-1])
      - *F*([6])
      - *F*([-1])
  - *F*([4,5,-2,3])
    - *F*([4,5])
      - *F*([4])
      - *F*([5])
    - *F*([-2,3])
      - *F*([-2])
      - *F*([3])

10.1.

Podaj, jakie wartości zostaną obliczone w tablicy *Z*[0 .. *m*-1] przy obliczaniu $P(2)$, gdzie

$$P(x) = 9 + x - 6x^3 + 2x^7 - 3x^{14} + 5x^{15}.$$

10.2.

Oblicz łączną liczbę operacji mnożenia, jaka zostanie wykonana przez funkcję *F*(T) dla *n*-elementowej tablicy T.

Uzupełnij poniższą tabelkę:

n	**Liczba operacji mnożenia**
1	0
2	
4	
8	
16	
1024	

Uzupełnij poniższy wzór rekurencyjny dla *f*(*n*) — liczby operacji mnożenia wykonywanych przez funkcję *F* dla tablicy *n*-elementowej:

$$f(n) = \ldots\ldots\ldots \cdot f(n/2) + \ldots\ldots\ldots\ldots \quad \text{dla } n > 1$$

10.3.

Wypisz wszystkie wywołania funkcji *F* podczas obliczania

$$P(x) = 9 + x - 6x^3 + 10x^5 + 2x^7 + 4x^9 - x^{10} - 3x^{14} + 5x^{15}.$$

Podaj wzór ogólny na $g(n)$ — liczbę wszystkich wywołań funkcji *F* dla wielomianu $P(x)$ stopnia $n-1$, gdzie n jest potęgą dwójki. Uzupełnij poniższy wzór:

$$g(n) = \text{.....}$$

10.4.

Do obliczania wartości wielomianu $P(x)$

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1}x^{n-1} \qquad (n = 3^m),$$

a więc gdy n jest potęgą trójki, można zastosować podział na trzy części

$$P(x) = \underbrace{a_0 + \cdots + a_{k-1}x^{k-1}}_{\text{cz. I}} + \underbrace{a_k x^k + \cdots + a_{2k-1}x^{2k-1}}_{\text{cz. II}} + \underbrace{a_{2k}x^{2k} + \cdots + a_{n-1}x^{n-1}}_{\text{cz. III}},$$

gdzie $k = n/3$. Wówczas uzyskujemy

$$P(x) = A(x) + B(x) \cdot x^k + C(x) \cdot x^{2k} = A(x) + (B(x) + C(x) \cdot x^k) \cdot x^k,$$

gdzie $A(x), B(x), C(x)$ są wielomianami, w których liczba współczynników jest trzykrotnie mniejsza niż w wyjściowym wielomianie $P(x)$.

Wzorując się na funkcji *F*, możemy skonstruować rekurencyjną funkcję *G*(T[0 .. *n*-1]), obliczającą wartość $P(x)$ przez podział tablicy *T* na trzy równe części, o ile $n > 1$. Po obliczeniu trzech wyników dla każdej z części, powiedzmy $A(x), B(x), C(x)$, funkcja oblicza swój wynik, wykonując dodatkowo dwa mnożenia:

- jedno mnożenie przy obliczaniu $C(x) \cdot x^k$.
- jedno mnożenie przy obliczaniu $(B(x) + C(x) \cdot x^k) \cdot x^k$.

Podobnie jak poprzednio pomijamy problem obliczania potęg x^k dla $k = 3^j$ ($j \geq 0$), tzn. przyjmujemy, że potęgi te są przechowywane w pewnej tablicy *Z*[0 .. *m*-1], a więc ich obliczanie nie jest wliczane do kosztu obliczeniowego funkcji *G*.

W ten sposób na przykład dla *n*=3 funkcja *G* wykona dokładnie 2 mnożenia, a dla *n*=9 funkcja wykona łącznie 8 mnożeń:

- po 2 mnożenia dla każdej z trzech części $A(x), B(x), C(x)$,
- jedno mnożenie przy obliczaniu $C(x) \cdot x^k$.
- jedno mnożenie przy obliczaniu $(B(x) + C(x) \cdot x^k) \cdot x^k$.

Uzupełnij poniższą tabelkę, podając liczbę mnożeń, jaka zostanie wykonana przez funkcję *G* dla *n*-elementowej tablicy współczynników T.

n	**Liczba operacji mnożenia**
3	2
9	8
27	
81	
243	

Zadanie 11.

Wiązka zadań *Odwrotna notacja polska*

Dla przetwarzania przez komputer wygodnym sposobem zapisu wyrażeń arytmetycznych jest tzw. odwrotna notacja polska (ONP). Zapis w ONP wyrażenia W nazywamy *postacią ONP* i oznaczamy ją ONP(W). W ONP operator (dodawania, odejmowania, mnożenia, dzielenia) umieszczamy za jego argumentami, np. 2+5 zapisujemy jako 2 5 +. Dokładniej, postać ONP dla wyrażenia definiujemy rekurencyjnie w następujący sposób:

1. Jeżeli W jest liczbą, to jego postać ONP jest równa W.
2. Jeżeli W ma postać W_1 *op* W_2, gdzie *op* jest operatorem, a W_1 i W_2 wyrażeniami, to ONP(W) jest równe ONP(W_1) ONP(W_2) *op*.

Przykład

$W = W_1$ *op* W_2	W_1	W_2	*op*	ONP(W)
1 + 2	1	2	+	1 2 +
5 – 7	5	7	–	5 7 –
3 ∗ (5 – 7)	3	5 – 7	∗	3 5 7 – ∗
(1 + 2) + (3 ∗ (5 – 7))	1 + 2	3 ∗ (5 – 7)	+	1 2 + 3 5 7 – ∗ +

Zauważmy, że dla W= (1 + 2) + (3 ∗ (5 – 7)) wartość ONP(W) uzyskujemy z połączenia ONP(1 + 2) = 1 2 +, ONP(3 ∗ (5 – 7)) = 3 5 7 – ∗ oraz znaku dodawania +.

11.1.

Uzupełnij poniższą tabelę, podając dla każdego wyrażenia z pierwszej kolumny jego podwyrażenia, łączący je operator oraz postać ONP tego wyrażenia.

$W = W_1$ *x* W_2	W_1	W_2	*op*	ONP(W)
4 + 3	4	3	+	4 3 +
(4 + 3) ∗ 2				
5 ∗ (7 – 6)				
((4 + 3) ∗ 2) – (5 ∗ (7 – 6))				

11.2.

Postać ONP wyrażeń, choć dla ludzi mało czytelna, ma własności bardzo przydatne dla analizy komputerowej. W ONP nie są potrzebne nawiasy, a do wyznaczania wartości wyrażenia można zastosować prosty algorytm podany poniżej.

Specyfikacja

Dane:

n — liczba całkowita dodatnia,

$X = X[1 \ldots n]$ — wyrażenie w ONP, gdzie $X[i]$ dla $1 \le i \le n$ jest liczbą lub znakiem ze zbioru $\{+, -, *\}$.

Wynik:

wartość wyrażenia X.

Algorytm:

```
k ← 1
dla i=1,2,…,n wykonuj
    jeżeli X[i] jest liczbą
        T[k] ← X[i]
    jeżeli X[i]∈{+, –, ∗}
        b ← T[k – 1]
        a ← T[k – 2]
        k ← k – 2
        jeżeli X[i] = `+`
            T[k] ← a + b
        jeżeli X[i] = `–`
            T[k] ← a – b
        jeżeli X[i] = `∗`
            T[k] ← a ∗ b
    k ← k + 1
zwróć T[1]
```

Prześledź działanie podanego algorytmu dla wyrażenia X = 9 7 + 3 ∗ 5 4 – 2 ∗ –, czyli dla n=11 oraz następujących wartości $X[1],.., X[11]$:

i	1	2	3	4	5	6	7	8	9	10	11
$X[i]$	9	7	+	3	∗	5	4	–	2	∗	–

Uzupełnij poniższą tabelę:

i	Wartość zmiennej k po i-tym przebiegu pętli	Zawartość tablicy $T[1..k\text{-}1]$ po i-tym przebiegu pętli
1	2	9
2	3	9, 7
4	3	16, 3
5		
6		
10		
11		

11.3.

Poniższy algorytm sprawdza, czy podany na wejściu ciąg liczb i operatorów jest poprawnym wyrażeniem w ONP.

Specyfikacja

Dane:

n — liczba całkowita dodatnia

$X = X[1..n]$ — ciąg elementów, z których każdy jest liczbą lub znakiem ze zbioru $\{+, -, *\}$.

Wynik:

Tak — jeśli X jest poprawnym wyrażeniem w ONP,

Nie — w przeciwnym przypadku.

Algorytm:

```
licznik ← 0
dla i=1,2,…,n wykonuj
    jeżeli X[i] jest liczbą
        licznik ← licznik + 1
    jeżeli X[i]∈{+, –, *}
        jeżeli licznik < 2
                zwróć „Nie” i zakończ działanie
        w przeciwnym razie
                licznik ← licznik – 1
jeżeli licznik ≠ 1
    zwróć „Nie”
w przeciwnym razie
    zwróć „Tak”
```

Oceń, które z podanych poniżej napisów są wyrażeniami zapisanymi poprawnie w ONP, wpisując słowa TAK lub NIE w trzeciej kolumnie poniższej tabeli. W drugiej kolumnie podaj wartości zmiennej *licznik* po zakończeniu działaniu algorytmu dla poszczególnych napisów.

Napis	Wartość zmiennej *licznik* po zakończeniu algorytmu	Czy poprawne wyrażenie w ONP?
1 2 + *	1	NIE
1 2 + 3 4 – 5 * 7 8 + 9		
1 2 3 4 5 + + + +		
1 2 3 4 5 + + + + + +		
1 2 3 4 5 + + + + +		
1 2 + 2 3 – 3 4 * 4 5 + – – – –		
1 2 + 2 3 – 3 4 * 4 5 + – – – – –		
1 2 + 3 4 – 5 * 7 8 + 9 + + +		

11.4.

W poniższych wyrażeniach przyjmujemy, że $op_1, \ldots, op_{10}$ to znaki ze zbioru $\{+, -, *\}$. Podaj postać ONP poniższych wyrażeń.

X: (((((((((1 op_1 2) op_2 3) op_3 4) op_4 5) op_5 6) op_6 7) op_7 8) op_8 9) op_9 10)
ONP(X):.....
.....
Y: (1 op_1 (2 op_2 (3 op_3 (4 op_4 (5 op_5 (6 op_6 (7 op_7 (8 op_8 (9 op_9 10)))))))))
ONP(Y):.....
.....

Zadanie 12.

Wiązka zadań *Szyfr Beauforta*

Każdej z wielkich liter angielskiego alfabetu przyporządkowany jest kod ASCII, będący ośmiobitową liczbą naturalną. Poniższa tabela przedstawia kody ASCII dla liter od A do Z:

65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Dany jest następujący algorytm szyfrujący:

Dane:

n — liczba naturalna, wielkość tablicy *Tekst*[]

Tekst[] — tablica zawierająca kody ASCII liter tekstu do zaszyfrowania, elementy tablicy są numerowane od 0 do n-1

k — nieujemna liczba całkowita

Wynik:

Szyfrogram[] — tablica o długości *n* zawierająca kody ASCII liter zaszyfrowanego tekstu

```
     i ← 0
     dopóki (i<n) wykonuj
(*)      Szyfrogram[i] ← (90-Tekst[i]+k) mod 26 + 65
         i ← i+1
         k ← k+i
```

12.1.

Uzupełnij poniższą tabelę wartościami zmiennych *i* oraz *k* w wierszu oznaczonym (*) w kolejnych iteracjach algorytmu dla *n* równego 7:

Lp	*i*	*k*
1	0	20
2		
3		
4		
5		
6		

12.2.

Ile dla przedstawionego algorytmu jest możliwych początkowych wartości *k*, dających w efekcie różne szyfrogramy? Odpowiedź krótko uzasadnij.

12.3.

Uzupełnij poniższą tabelę. W wierszu pierwszym zaszyfruj przedstawionym w treści zadania algorytmem podany tekst jawny, w wierszu drugim odszyfruj szyfrogram zakodowany tym algorytmem.

k	*Tekst*[]	*Szyfrogram*[]
1	MAPA	
14		DAN

12.4.

Używając pseudokodu lub wybranego języka programowania, napisz algorytm dekodujący szyfrogram.

Dane:

n — liczba naturalna, wielkość tablicy *Tekst*[]

Szyfrogram[] — tablica o długości *n* zawierająca kody ASCII liter zaszyfrowanego tekstu

k — nieujemna liczba całkowita

Wynik:

Tekst[] — tablica zawierająca kody ASCII liter odszyfrowanego tekstu, elementy tablicy są numerowane od 0 do n-1

Zadanie 13.

Wiązka zadań *Zbiór punktów*

Na płaszczyźnie kartezjańskiej danych jest *n* punktów: P_1, P_2,..., P_n oraz jeszcze jeden punkt A.

Współrzędne punktów zapisane są w tablicach *PX[1..n]*, *PY[1..n]* — współrzędne punktu P_i dane są jako para (*PX[i]*, *PY[i]*) dla *i* = 1, 2, ..., n. Współrzędne punktu A to (*ax*, *ay*). Przeanalizuj poniższy algorytm i wykonaj poniższe polecenia.

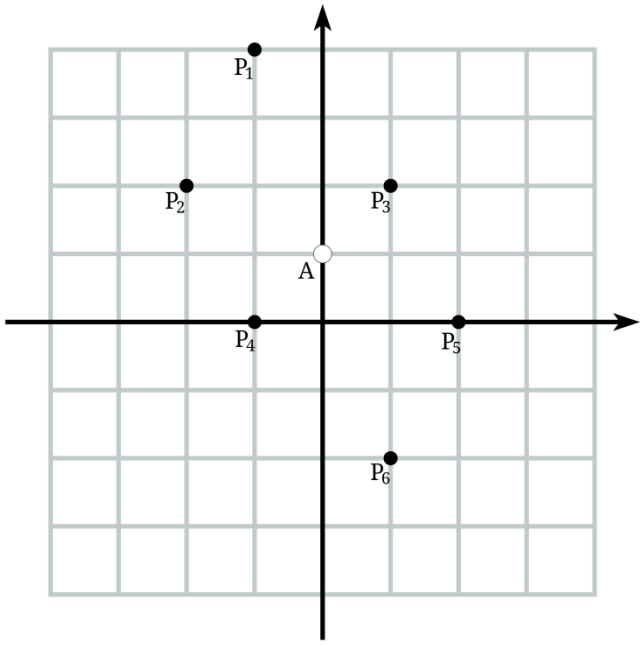
Dane:
PX[1..n], *PY[1..n]* — tablice liczb całkowitych zawierające współrzędne kolejnych punktów P_1, P_2,..., P_n
ax, *ay* — współrzędne punktu A.
Wynik:
odpowiedź TAK lub NIE.

```
funkcja sprawdz(PX[1..n], PY[1..n],ax,ay)
    dla i = 1, 2, ..., n wykonuj
        roznicax ← PX[i] – ax
        roznicay ← PY[i] – ay
        drugix ← ax – roznicax
```

```
        drugiy ← ay – roznicay
        znalezione ← fałsz
        dla k = 1, 2, ..., n wykonuj
                jeżeli drugix = PX[k] oraz drugiy = PY[k]
                        znalezione ← prawda
        jeżeli znalezione = fałsz
                wypisz NIE i zakończ wykonywanie algorytmu
wypisz TAK
```

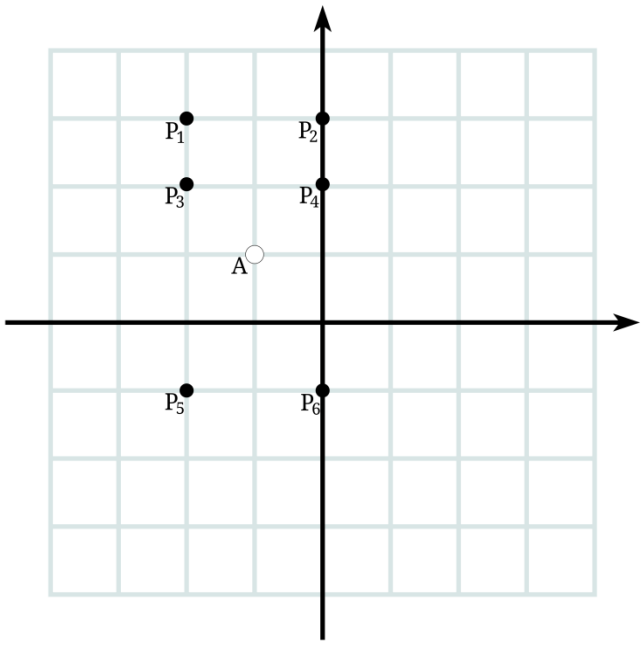
13.1.

Dla poniższych zbiorów punktów znajdź odpowiedź, jaką otrzymamy po wykonaniu powyższego algorytmu:



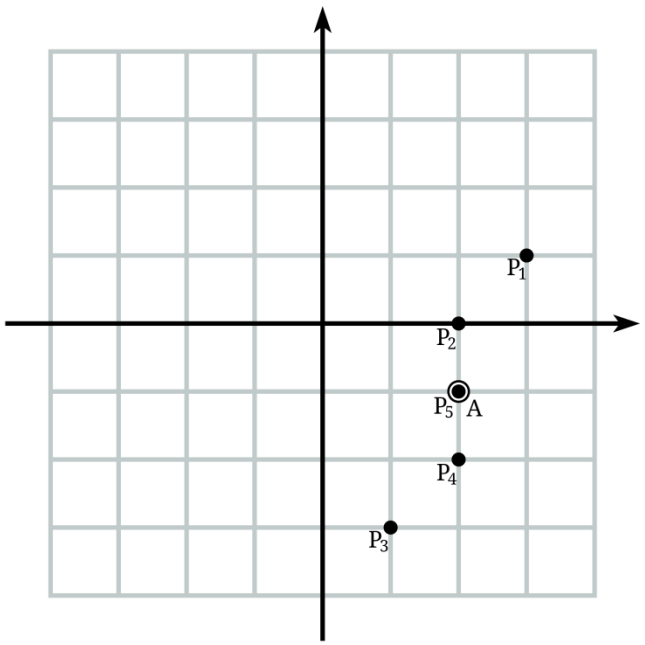
P_1 (-1,4)
P_2 (-2,2)
P_3 (1,2)
P_4 (-1,0)
P_5 (2,0)
P_6 (1,-2)
A (0, 1)

Odpowiedź algorytmu: .....



P_1 (-2,3)
P_2 (0,3)
P_3 (-2,2)
P_4 (0,2)
P_5 (-2,-1)
P_6 (0,-1)
A (-1,1)

Odpowiedź algorytmu: .....



P_1 (3,1)
P_2 (2,0)
P_3 (1,-3)
P_4 (2,-2)
P_5 (2,-1)
A (2,-1)

Odpowiedź algorytmu: .....

13.2.

Dane są punkty:

P_1 (3,–1)
P_2 (0,2)
P_3 (1,4)
P_4 (1,3)
P_5 (4,0)

oraz punkty P_6 i A o nieznanych współrzędnych. Wiadomo, że algorytm odpowiada TAK dla $n = 6$ oraz punktów P_1,..., P_6 i A. Podaj współrzędne punktów P_6 i A.

Zadanie 14.

Wiązka zadań *Działka*

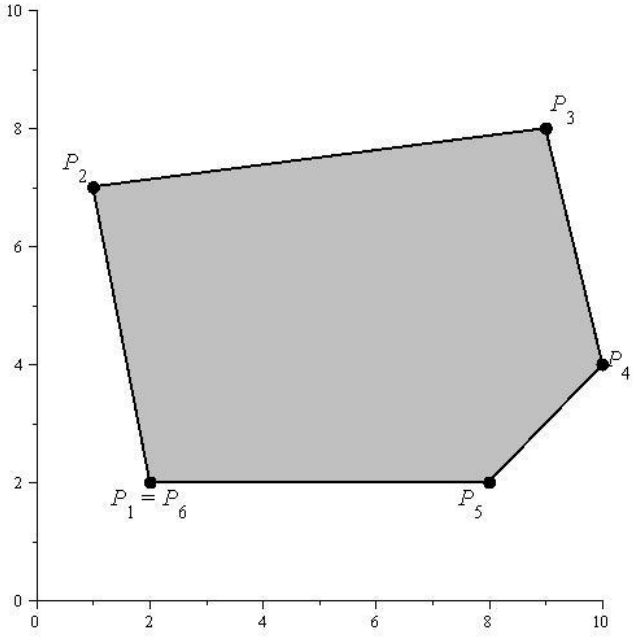
Pan G jest geodetą. Jego zadaniem jest wyznaczenie pola powierzchni działki budowlanej o kształcie pewnego wielokąta wypukłego. Aby wyznaczyć pole powierzchni działki, pan G obchodzi ją dookoła, tak aby działkę mieć zawsze po swojej prawej stronie. Podczas pomiaru zapisuje współrzędne kolejnych wierzchołków wielokąta, przy czym:

1. rozpoczyna obchód w punkcie $P_1 = [x_1, y_1]$;
2. współrzędne kolejno odwiedzonych wierzchołków oznacza:
$$P_2 = [x_2, y_2],\ P_3 = [x_3, y_3], \dots,\ P_n = [x_n, y_n];$$
3. kończy obchód w punkcie początkowym, tzn. przyjmuje $P_{n+1} = P_1$.

Przykład

Poniższy obrazek przedstawia działkę ($n = 5$) oraz kolejne punkty pomiaru zanotowane przez geodetę:

$P_1 = [2,2]$, $P_2 = [1,7]$, $P_3 = [9,8]$, $P_4 = [10,4]$, $P_5 = [8,2]$, $P_6 = P_1$.



Następnie pan G wyznacza pole działki za pomocą wzoru

$$pole = \frac{1}{2}\sum_{i=1}^{n}(x_{i+1} - x_{i-1}) \cdot y_i,$$

gdzie x_0 przyjmuje jako x_n.

Obliczanie pola z podanego wzoru realizowane jest według następującego algorytmu:

Dane:

n — liczba całkowita, $n \geq 0$,

$x_1, y_1; x_2, y_2; \ldots; x_n, y_n$ — współrzędne punktów pomiaru.

Wynik:

pole działki

Algorytm:

$x_0 \leftarrow x_n$
$x_{n+1} \leftarrow x_1$
$y_{n+1} \leftarrow y_1$
$pole = 0$
dla $i = 1,2,\ldots,n$ **wykonuj**
    $pole \leftarrow pole + (x_{i+1} - x_{i-1}) \cdot y_i$
zwróć $\frac{pole}{2}$ **i zakończ**

14.1.

Przeanalizuj działanie powyższego algorytmu dla przykładu z rysunku i uzupełnij poniższą tabelkę.

i	x_i	y_i	$x_{i+1} - x_{i-1}$	$(x_{i+1} - x_{i-1}) \cdot y_i$
1	2	2		
2	1	7		
3	9	8		
4	10	4		
5	8	2		

Podaj obliczone pole działki: .....

14.2.

Niech punkty $A = [x_1, y_1],\ B = [x_2, y_2],\ C = [x_3, y_3]$ będą wierzchołkami pewnego trójkąta, podanymi w kolejności zgodnej z ruchem wskazówek zegara. Bazując na metodzie geodety, podaj wzór na pole trójkąta ABC.

14.3.

Wyraź wzorem liczbę operacji mnożenia oraz łączną liczbę operacji dodawania i odejmowania współrzędnych punktów, jaka zostanie wykonana przez podany algorytm podczas obliczania pola powierzchni działki o kształcie wielokąta n-wierzchołkowego.

Uzupełnij poniższą tabelkę:

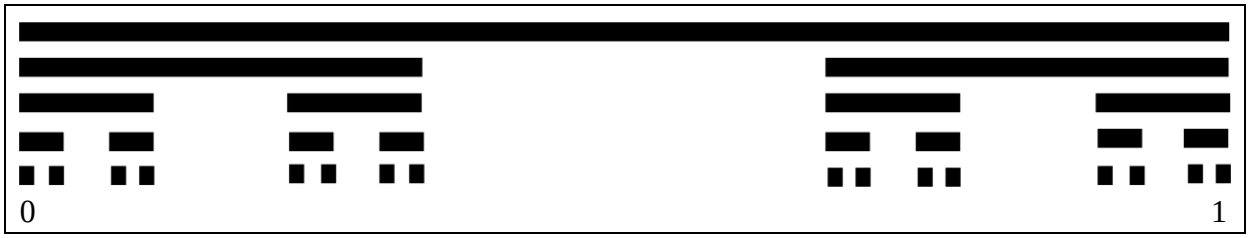
Działanie	**Liczba operacji**
dodawanie i odejmowanie	
mnożenie	

Zadanie 15.

Wiązka zadań *Zbiór Cantora*

Zbiór (fraktal) Cantora rzędu 0 jest równy odcinkowi jednostkowemu [0; 1]. Zbiór Cantora rzędu 1 uzyskujemy, dzieląc odcinek jednostkowy [0; 1] na trzy równe części i usuwając odcinek środkowy (pozostawiając zaś jego końce). Składa się on zatem z dwóch fragmentów odcinka jednostkowego: [0; 1/3], [2/3; 1].

Ogólnie zbiór Cantora rzędu *n*+1 tworzymy ze zbioru Cantora rzędu *n* w następujący sposób: każdy odcinek zbioru Cantora rzędu *n* dzielimy na trzy równe części i usuwamy odcinek środkowy, pozostawiając jego końce. Zgodnie z tą regułą zbiór Cantora rzędu 2 składa się z odcinków: [0; 1/9], [2/9; 3/9], [6/9; 7/9] i [8/9; 1].



Rysunek. Geometryczna ilustracja zbiorów Cantora rzędu 0, 1, 2, 3 i 4

15.1.

Uzupełnij poniższą tabelę, podając liczbę odcinków w zbiorach Cantora podanych rzędów.

n	Liczba odcinków w zbiorze Cantora rzędu *n*
0	1
1	2
2	4
3	8
5	
6	
9	
10	

Podaj ogólny wzór określający C(*n*), liczbę odcinków w zbiorze Cantora rzędu *n*:

C(*n*)=……..

15.2.

Zauważ, że każdy odcinek w zbiorze Cantora ustalonego rzędu ma tę samą długość. Uzupełnij poniższą tabelę, podając długość jednego odcinka w zbiorach Cantora podanych rzędów.

n	Długość jednego odcinka w zbiorze Cantora rzędu *n*
0	1
1	1/3
2	1/9
3	1/27
4	
5	
6	
7	

Podaj ogólny wzór określający długość D(*n*) jednego odcinka w zbiorze Cantora rzędu *n*:

D(*n*)=……..

15.3.

Uzupełnij poniższą listę odcinków zbioru Cantora rzędu 3, których końce zapisane są jako ułamki zwykłe **nieskracalne**:

[0; 1/27], [2/27; 1/9], ...
.....

Końce odcinków zbiorów Cantora można opisać w zwarty i regularny sposób w systemie trójkowym. W szczególności odcinki zbioru Cantora rzędu 1 zapisane w systemie trójkowym to

[0; 0,1], [0,2; 1],

natomiast odcinki zbioru Cantora rzędu 2 zapisane w systemie trójkowym to

[0; 0,01], [0,02; 0,1], [0,2; 0,21] i [0,22; 1].

Podaj poniżej odcinki zbiorów Cantora rzędu 3 zapisane w systemie o podstawie 3 (trójkowym):

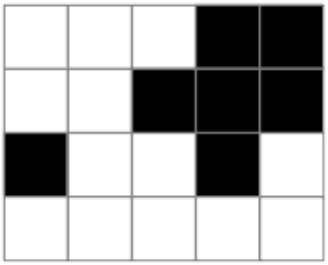
[0; 0,001], ...
.....

Zadanie 16.

Wiązka zadań *Odległość na planszy*

Dana jest prostokątna plansza o rozmiarach *m* na *n* (*m* wierszy, *n* kolumn), podzielona na *m·n* pól — jednostkowych kwadratów. Przez *pole (i, j)* rozumiemy pole, które jest w i-tym wierszu oraz j-tej kolumnie. Wiersze numerujemy kolejno od 1 do m, z góry do dołu, a kolumny kolejno od 1 do n,od lewej do prawej. **Dwa pola uważamy za *sąsiednie*, jeśli mają wspólny bok.**

Niektóre pola tej planszy zostały wycięte — na rysunku oznaczone są na czarno.



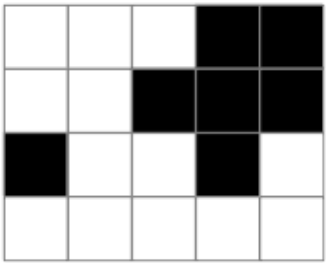
Poniżej podany jest algorytm, który tworzy tablicę *D[1..m, 1..n]* (o *m* wierszach oraz *n* kolumnach), odpowiadającą planszy, a następnie wpisuje liczby całkowite do komórek tablicy odpowiadających niewyciętym (białym) polom na planszy. Każda wpisana liczba mierzy w pewien sposób odległość odpowiedniego pola od lewego górnego rogu planszy (pola (1,1) — załóż, że to pole nigdy nie jest wycięte). Przeanalizuj algorytm i rozwiąż podanej niżej zadania.

```
dla każdego niewyciętego pola (i,j)
        D[i, j] ← -1

D[1, 1] ← 0
k ← 0
powtarzaj:
        jeśli nie ma takich pól (x,y), że D[x, y] = k
                zakończ algorytmu
        dla każdego pola (x,y) takiego, że D[x, y] = k wykonaj
                dla każdego pola (a,b) sąsiadującego z (x,y) wykonaj
                        jeśli (a,b) nie jest wycięte oraz D[a, b] = -1
                                D[a, b] ← k+1
        k ← k+1
```

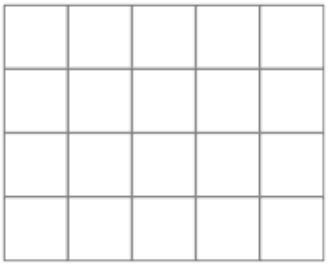
16.1.

Dla podanej poniżej planszy wyznacz wartości, które znajdą się w tablicy D po zakończeniu wykonywania algorytmu. Wpisz te wartości w odpowiednie pola planszy:



16.2.

Zaprojektuj taką planszę o rozmiarach $4 \cdot 5$, dla której w wynikowej tablicy D pojawi się liczba 10. Odpowiedź podaj, kolorując na poniższym rysunku pewne pola na czarno oraz wpisując liczbę „10” w odpowiednim polu:



16.3.

Zaprojektuj taką planszę (wybranych przez siebie rozmiarów), dla której w wynikowej tablicy D po zakończeniu wykonywania algorytmu w pewnej komórce pozostanie liczba –1. Narysuj taką planszę i wpisz -1 na wszystkich białych polach, na których ta wartość pozostanie do końca działania algorytmu.

Zadanie 17.

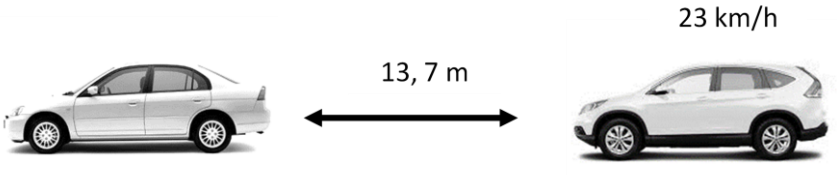
Wiązka zadań *System hamowania*

Nowoczesne samochody mają wbudowany komputer z systemem wspomagającym prowadzenie pojazdu. Jedną z jego funkcji jest aktywacja systemu hamowania w momencie, gdy samochód znajdzie się zbyt blisko pojazdu znajdującego się przed nim.

Automatyczne hamowanie wykorzystuje czujniki, które mierzą prędkość samochodu oraz jego odległość od pojazdu znajdującego się przed nim. Reakcja systemu zależy od prędkości samochodu, w którym jest on uruchomiony. W zadaniach ograniczamy się do analizy działania systemu przy małych prędkościach samochodu.

Przyjmujemy, że automatyczne uruchomienie hamowania następuje, jeśli odległość od najbliższego pojazdu jest **mniejsza niż** 15 metrów oraz prędkość samochodu, w którym działa system, wynosi **mniej niż** 30 km/h (zobacz poniższy rysunek). Po automatycznym uruchomieniu hamowania prędkość samochodu będzie spadać, a hamulce będą włączone do momentu osiągnięcia przez samochód prędkości 0 km/h, niezależnie od tego, jak będzie się zmieniać odległość od najbliższego pojazdu.

W analizie działania systemu hamowania przyjmuje się, że samochód nie przekracza prędkości 350 km/h.



Działanie systemu automatycznego hamowania zostało poniżej błędnie opisane w pseudokodzie.

odczytaj *prędkość samochodu*;
odczytaj *odległość od najbliższego pojazdu*;
jeżeli *prędkość samochodu* < 30 **lub** *odległość od najbliższego pojazdu* < 15:
    dopóki *prędkość samochodu* = 100 **wykonuj**
        wyślij *polecenie hamowania* do układu hamulcowego;
        odczytaj *prędkość samochodu*

17.1.

W powyższym pseudokodzie występują dwa błędy powodujące niezgodność z podanym wcześniej opisem. Znajdź i popraw oba błędy.

Błędny zapis w pseudokodzie	Poprawny zapis w pseudokodzie

17.2.

Przyjmujemy, że system hamowania odczytuje prędkość samochodu i odległość od najbliższego pojazdu na tyle często, że sąsiednie pomiary mogą się różnić co najwyżej o odpowiednio 1 km/h i 1 m (pomiary prędkości podawane są w km/h, w zaokrągleniu do wartości całkowitych; pomiary odległości podawane są w m, również w zaokrągleniu do wartości całkowitych). Oceń, czy podane w poszczególnych wierszach poniższej tabeli dane testowe są danymi standardowym (niezmieniające stanu sytemu hamowania), brzegowymi (wartości, które mogą powodować włączenie lub wyłączenie systemu hamowania) czy też danymi niezgodnymi ze specyfikacją.

Dane testowe	Typ danych testowych
prędkość samochodu — 29 km/h, odległość między samochodami — 1 m	**brzegowe**
prędkość samochodu — 400 km/h, *odległość od najbliższego pojazdu — 0 m*	
prędkość samochodu — 13 km/h, *odległość od najbliższego pojazdu — 8 m*	
prędkość samochodu — 45 km/h, *odległość od najbliższego pojazdu — 17 m*	
prędkość samochodu — 0 km/h, *odległość od najbliższego pojazdu — 17 m*	

17.3.

Wskaż, które z poniższych dokończeń zdań są prawdziwe (P), a które fałszywe (F), wstawiając znak X w odpowiedniej kolumnie.

Celem testowania systemu dla różnych typów danych jest

	P	F
upewnienie się, że system poradzi sobie z odczytem parametrów z czujników i tempo jego reakcji pozwoli uniknąć zderzenia.		
zwiększenie precyzji pomiarów prędkości i odległości.		
zmniejszenie długości kodu programu.		
sprawdzenie, czy system działa zgodnie z podaną specyfikacją.		

17.4.

Uzupełnij poniższą tabelę, wpisując w ostatniej kolumnie słowo *włączony, wyłączony* (gdy automatyczne hamowanie będzie przy danych odczytach włączone/wyłączone niezależnie od wartości poprzedniego pomiaru) lub *nieustalony* (gdy aktywność automatycznego hamowania zależy zarówno od bieżącego, jak i od poprzedniego pomiaru).

Prędkość samochodu	**Odległość między samochodami**	**Stan automatycznego hamowania**
poniżej 30 km/h	poniżej 15m	włączony
poniżej 30 km/h	równa15m	
równa 30 km/h	poniżej 15m	
równa 30 km/h	równa15m	
powyżej 30 km/h	dowolna	

1.2. Tworzenie algorytmów

Zadanie 18.

Wiązka zadań *Największy wspólny dzielnik*

Opisana poniżej funkcja wyznacza największy wspólny dzielnik (NWD) dwóch liczb całkowitych dodatnich.

Specyfikacja

Dane**:**

liczby całkowite dodatnie a i b.

Wynik**:**

Największy wspólny dzielnik liczb a i b.

funkcja *NWD* (*a*, *b*)
    dopóki $b \neq 0$ **wykonuj**
        $r \leftarrow a \bmod b$ (*)
        $a \leftarrow b$
        $b \leftarrow r$
    zwróć *a* **i zakończ**

18.1.

Przeanalizuj działanie funkcji *NWD* i dla wskazanych argumentów podaj liczbę wykonań operacji modulo:

a	*b*	liczba operacji mod (*)
25	15	
116	324	
762	282	

18.2.

Wykorzystując funkcję *NWD* oraz następującą zależność:

$$\text{NWD}(a_1, a_2, \ldots, a_n) = \text{NWD}\,(\text{NWD}\,(a_1, a_2, \ldots, a_{n-1}), a_n),$$

możemy wyznaczyć największy wspólny dzielnik *n* liczb całkowitych dodatnich $a_1, a_2, \ldots, a_n$.

Przykład

$$\text{NWD}(15, 24, 60) = \text{NWD}\,(\text{NWD}\,(15,24), 60) = \text{NWD}\,(3,60) = 3.$$

Uzupełnij poniższą tabelkę i dla wskazanych *n* liczb całkowitych dodatnich $a_1, a_2, \ldots, a_n$ oblicz ich największy wspólny dzielnik.

$a_1, a_2, \ldots, a_n$	$\text{NWD}(a_1, a_2, \ldots, a_n)$
36, 24, 72, 150	
119, 187, 323, 527, 731	
121, 330, 990, 1331, 110, 225	

18.3.

Napisz algorytm, który korzystając z funkcji *NWD* (*a*, *b*), wyznaczy największy wspólny dzielnik ciągu liczb $a_1, a_2, \ldots, a_n$.

Dane:

$a_1, a_2, \ldots, a_n$ — liczby całkowite dodatnie.

Wynik:

Największy wspólny dzielnik liczb $a_1, a_2, \ldots, a_n$.

Komentarz do zadania

18.1.

Dla pierwszego przykładu: $a = 25$, $b = 15$, kolejne reszty r są równe: 10, 5, 0, zatem wykonywane są 3 operacje a mod b. Dla $a = 116$, $b = 324$ kolejne reszty r są równe: 116, 92, 24, 20, 4, 0, stąd w pętli wykonywanych jest 6 operacji a mod b. Podobnie jest dla $a = 762$, $b = 282$: zmienna r przyjmuje kolejno następujące wartości: 198, 84, 30, 24, 6, 0, stąd wykonywanych jest 6 operacji a mod b.

18.2.

W tym zadaniu należy przeanalizować obliczanie NWD dla n liczb z wykorzystaniem funkcji napisanej w treści zadania:

NWD (36, 24, 72, 150) =
NWD (NWD (36, 24, 72), 150) =
NWD (NWD (NWD (36, 24), 72), 150) =
NWD(NWD (12, 72), 150) =
NWD(12, 150) = 6

NWD(119, 187, 323, 527, 731) =
NWD (NWD (119, 187, 323, 527), 731) =
NWD (NWD (NWD (17, 323), 527), 731) =
NWD (NWD (17, 527), 731) =
NWD (17, 731) = 17

NWD (121, 330, 990, 1331, 110, 225) =
NWD(NWD(121, 330, 990, 1331, 110), 225) =
NWD(NWD(NWD (121, 330, 990, 1331), 110), 225) =
NWD(NWD(NWD (NWD(121, 330, 990), 1331), 110), 225) =
NWD(NWD(NWD (NWD(NWD(121, 330), 990), 1331), 110), 225) =
NWD(NWD (NWD (NWD (11, 990), 1331), 110), 225) =
NWD(NWD (NWD ((11, 1331), 110), 225) =
NWD(NWD (11, 110), 225) =
NWD(11, 225) = 1

18.3.

Algorytm utworzymy w oparciu o tożsamość:

NWD (a_1, a_2, ..., a_n) = NWD (NWD (a_1, a_2, …, a_{n-1}), a_n),

podaną w zadaniu drugim.

Za największy wspólny dzielnik obieramy liczbę a_1, a następnie iteracyjnie obliczamy największy wspólny dzielnik zapamiętanego wyniku oraz liczby a_i dla kolejnych i = 2, 3, 4, ..., n. Prowadzi to do następującego rozwiązania zadania:

$w \leftarrow a_1$
dla i = 2, 3, 4, ..., n **wykonuj**
  $w \leftarrow NWD\ (w, a_i)$
zwróć w **i zakończ**

Podane rozwiązanie jest poprawne także wtedy, gdy n jest równe 1.

Rozwiązanie

18.1.

a	*b*	liczba operacji mod
25	15	3
116	324	6
762	282	6

18.2.

$a_1, a_2, \ldots, a_n$	NWD$(a_1, a_2, \ldots, a_n)$
36, 24, 72, 150, 114	6
119, 187, 323, 527, 731	17
121, 330, 990, 1331, 110, 225	1

18.3.

Przykładowa poprawna odpowiedź:

$w \leftarrow a_1$
dla i = 2, 3, 4, ..., n **wykonuj**
  $w \leftarrow NWD\ (w, a_i)$
zwróć *w* **i zakończ**

Zadanie 19.

Wiązka zadań *Zakupy międzyplanetarne*

Mieszkańcy galaktyki Różnoliczbowo zamieszkują 9 planet: Liczbowo$_2$, Liczbowo$_3$, …, Liczbowo$_{10}$. Na każdej planecie Liczbowo$_i$ jej mieszkańcy posługują się systemem liczbowym o podstawie *i*. Na każdej planecie wszystkie ceny są liczbami naturalnymi.

19.1.

Mieszkańcy czterech sąsiadujących planet: Liczbowo$_2$, Liczbowo$_4$, Liczbowo$_8$ oraz Liczbowo$_{10}$ często podróżują pomiędzy tymi planetami i kupują różne towary. W poniższej tabeli znajdują się ceny wybranych towarów zakupionych przez jedną osobę na różnych planetach. Uzupełnij tabelę, przeliczając **podane** ceny na systemy liczbowe wszystkich czterech planet.

Towar	**Cena towaru zapisana w systemie liczbowym planety**			
	Liczbowo$_2$	**Liczbowo$_4$**	**Liczbowo$_8$**	**Liczbowo$_{10}$**
Kozaki	10111011			
Płaszcz			724	
Skuter				1458

19.2.

Na różnych planetach ten sam towar może mieć różną cenę, na przykład cena ciasta kokosowego na planecie Liczbowo$_{10}$ wynosi 38_{10}, zaś na planecie Liczbowo$_8$ jego cena wynosi 55_8 (równą 45_{10}).

Mieszkańcy planety Liczbowo$_{10}$ są bardzo oszczędni i przed zakupami porównują ceny towarów na wybranych planetach. Uzupełnij w poniższej tabeli relacje (>, <, =) pomiędzy poszczególnymi cenami.

Liczbowo$_x$	Relacja	Liczbowo$_y$
110000100_2	>	556_8
3123_4		1747_8
110_{10}		11010_3
266_9		110100_3
110111101_2		674_8

19.3.

Sprzedawcy sklepów planety Liczbowo$_i$ wyliczają wartość zakupów klientów, sumując ceny zakupionych towarów w systemie obowiązującym na ich planecie; stosują przy tym metodę dodawania pisemnego.

Przykłady

Dodawanie w systemie o podstawie 2:

$$\begin{array}{cccccccc} & & & 1 & 0 & 1 & 0 \\ & & 1 & 0 & 0 & 1 & 1 \\ & & & 1 & 1 & 0 & 1 \\ +_2 & & 1 & 1 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Dodawanie w systemie o podstawie 4:

$$\begin{array}{cccc} & & 2 & 2 \\ & 1 & 0 & 3 \\ & & 3 & 1 \\ +_4 & 1 & 2 & 1 \\ \hline 1 & 0 & 0 & 3 \end{array}$$

Podsumuj rachunki pana Dwójkowskiego (z planety Liczbowo$_2$) oraz pana Czwórkowskiego (z planety Liczbowo$_4$).

Rachunek pana Dwójkowskiego:

$$\begin{array}{cccccccc} & & & & 1 & 0 & 1 & 1 \\ & & & 1 & 0 & 0 & 1 & 0 \\ & & 1 & 1 & 0 & 1 & 1 & 1 \\ & & & 1 & 1 & 0 & 1 & 0 \\ & & & 1 & 1 & 0 & 0 & 1 \\ +_2 & & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline \text{SUMA} & & \ldots\ldots & & & & & \end{array}$$

Rachunek pana Czwórkowskiego:

		3	3
	1	0	2
	3	1	1
	1	3	1
	1	2	3
$+_4$	3	0	1

SUMA …….

Podaj różnicę wartości obu rachunków w systemie obowiązującym na planecie Liczbowo_{10}.

Różnica rachunków w systemie obowiązującym na planecie Liczbowo_{10} wynosi: ……..

19.4.

Właściciele sieci sklepów ulokowanych na wszystkich planetach galaktyki postanowili dostarczyć do sklepów kalkulatory, które będą dodawały ceny w opisany powyżej sposób. Aby pomóc właścicielom, podaj algorytm dodawania (w postaci pseudokodu lub w języku programowania), który dla dwóch liczb a i b zapisanych w systemie o podstawie $p \in [2,9]$ wyznacza i wypisuje wartość sumy $a +_p b$ zapisaną w systemie o podstawie p zgodnie z poniższą specyfikacją. Twój algorytm **nie może** dokonywać zamiany liczb a i b na inny system pozycyjny.

Specyfikacja

Dane:

p — podstawa systemu, liczba naturalna z przedziału [2,9],

n — liczba cyfr liczb naturalnych a, $b \leq 255$ (przyjmujemy, że krótsza liczba jest uzupełniona z lewej strony zerami, tak aby obie liczby miały taką samą długość),

A[n], A[n–1],..., A[1] – kolejne cyfry liczby a zapisanej w systemie o podstawie p,

B[n], B[n–1],..., B[1] – kolejne cyfry liczby b zapisanej w systemie o podstawie p,

Wyniki:

wartość liczby $c = a +_p b$ zapisana w systemie o podstawie p w postaci ciągu cyfr C[n+1], C[n],…,C[1].

Przykład

Dla następujących danych:

$p = 4$

$n = 4$,

Liczba $a = 3122_4$

Liczba $b = 21_4$

Wynikiem jest liczba $c = 3203_4$

Zawartość tablic A, B, C:

i	5	4	3	2	1
A[i]		3	1	2	2
B[i]		0	0	2	1
C[i]	0	3	2	0	3

Komentarz do zadania

19.1.

Powszechnie znane są algorytmy konwersji liczby z zapisu dziesiętnego na zapis w systemie pozycyjnym o podstawie $s < 10$ i odwrotnie: z zapisu w systemie o podstawie s na system dziesiętny. Aby uniknąć wielokrotnego wykonywania takich konwersji, skorzystamy z zależności między reprezentacjami liczb w systemie o podstawie 2, podstawie $4 = 2 \cdot 2$ oraz podstawie $8 = 2 \cdot 2 \cdot 2$. Skoncentrujmy się najpierw na systemach o podstawach 2 i 4:

- reprezentację liczby w systemie czwórkowym można uzyskać z jej reprezentacji w systemie binarnym (czyli o podstawie 2), wybierając od końca pary cyfr i zamieniając je na ich czwórkowe reprezentacje;
- reprezentację liczby w systemie binarnym można uzyskać z jej reprezentacji w systemie czwórkowym, zamieniając każdą cyfrę czwórkową na jej dwucyfrową reprezentację binarną.

Ponieważ $8 = 2^3$, analogiczna własność zachodzi dla konwersji między systemem binarnym a systemem ósemkowym, z tą różnicą, że zamiast bloków 2 cyfr rozważamy bloki o długości 3. Korzystając z powyższych obserwacji, uzyskujemy rozwiązanie:

Towar	Cena towaru zapisana w systemie liczbowym planety			
	Liczbowo_2	Liczbowo_4	Liczbowo_8	Liczbowo_{10}
Kozaki	10111011	2323	273	187
Płaszcz	111010100	13110	724	468
Skuter	10110110010	112302	2662	1458

Dokładniej, dla $10111011_{(2)}$ uzyskujemy:

- reprezentację czwórkową: dzieląc 10111011 na bloki 10, 11, 10, 11 i zapisując w systemie o podstawie 4 wartość każdego bloku: 2, 3, 2 i 3;

1	0	1	1	1	0	1	1
10		11		10		11	
2		3		2		3	

- reprezentację ósemkową: dzieląc 10111011 na bloki 10, 111, 011 i zapisując w systemie o podstawie 4 wartość każdego bloku: 2, 7 i 3.

1	0	1	1	1	0	1	1
010		111			011		
2		7			3		

Z kolei z $724_{(8)}$ uzyskujemy:

- reprezentację binarną: zamieniając każdą cyfrę ósemkową na jej 3-cyfrową reprezentację binarną, czyli 111, 010, 100; daje to reprezentację 111010100;
- reprezentację czwórkową: dzieląc binarną reprezentację 111010100 na bloki 01, 11, 01, 01 i 00, zapisując w systemie o podstawie 4 wartość każdego bloku: 1, 3, 1, 1 i 0.

Znając zasady konwersji między systemami o podstawach 2, 4 i 8, reprezentację dziesiętną możemy uzyskać za pomocą standardowego algorytmu zamiany liczby z systemu o podstawie p różnej od 10 (np. $p = 2$) na system dziesiętny. Analogicznie, znając reprezentację dziesiętną,

wystarczy znaleźć jej reprezentację w jednym z pozostałych systemów, a potem zastosować omówione powyżej reguły konwersji między systemami o podstawach 2, 4 i 8.

19.2.

Używając standardowych algorytmów zamiany z systemu niedziesiętnego na dziesiętny, możemy wszystkie liczby przekształcić na postać dziesiętną i wówczas porównać. Metoda ta wymaga dość dużo obliczeń, dlatego pokażemy sposób wymagający mniej pracy. Podobnie jak w rozwiązaniu zadania 1 polegać on będzie na wykorzystaniu faktu, że dwie cyfry w systemie o podstawie *s* odpowiadają jednej cyfrze w systemie o podstawie s^2, podobnie trzy cyfry w systemie o podstawie *s* odpowiadają jednej cyfrze w systemie o podstawie s^3):

$556_8=101101110_2 < 110000100_2$
$3123_4=11011011_2$, $1747_8=001111100111_2$, czyli $3123_4<1747_8$
$266_9=022020_3$, czyli $266_9<110100_3$
$674_8=110111100_2$, czyli $6748<1101111012$

Jedyny przykład wymagający konwersji na system dziesiętny to porównanie 110_{10} i $11010_3=81_{10}+27_{10}+3_{10}=111_{10}$.

19.3.

Możemy wszystkie liczby w rachunku zamienić na system dziesiętny i je dodawać. Inne rozwiązanie polega na zastosowaniu metody dodawania pisemnego przeprowadzonego na reprezentacji binarnej i czwórkowej. Pokażemy je dla par liczb z rachunku pana Dwójkowskiego. Najpierw dodamy dwie pierwsze liczby.

przeniesienie:		0	0	1	0	
			1	**0**	**1**	**1**
	$+_2$	**1**	**0**	**0**	**1**	**0**
SUMA:		**1**	**1**	**1**	**0**	**1**

Następnie do wyniku dodamy trzecią liczbę

przeniesienie:	1	1	1	1	1	1	
			1	**1**	**1**	**0**	**1**
$+_2$		**1**	**1**	**0**	**1**	**1**	**1**
SUMA:	**1**	**0**	**1**	**0**	**1**	**0**	**0**

Dodając następnie liczby: 11010_2, 11001_2 i 101011_2, uzyskamy wynik 10110010_2.

Analogicznie postępujemy dla reprezentacji czwórkowych. Suma dwóch pierwszych liczb to:

przeniesienie:			1	1	
				3	**3**
	$+_4$		**1**	**0**	**2**
SUMA:			**2**	**0**	**1**

Dodając następnie liczby: 311_4, 131_4, 123_4 i 301_4, uzyskamy wynik 2333_4.

Następnie zamieniamy obie obliczone sumy na system dziesiętny:

$10110010_2=2_{10}+16_{10}+32_{10}+128_{10}=178_{10}$ oraz $2333_4=3_{10}+3_{10}\cdot 4_{10}+3\cdot 16_{10}+2\cdot 64_{10}=191_{10}$.

A zatem różnica wartości rachunków wynosi 13_{10}.

19.4.

Zadanie można rozwiązać, implementując zasadę dodawania pisemnego, której przykłady podaliśmy w rozwiązaniu zadania 3. Zgodnie z tą zasadą dodajemy odpowiadające sobie cyfry i przeniesienia, zaczynając od prawej strony. Przyjmujemy przy tym, że ostatnie przeniesienie $R[1]$ (na skrajnie prawej pozycji 1) jest równe zero. Ponadto wiemy, że:

- i-ta cyfra wyniku jest równa $(A[i] + B[i] + R[i]) \bmod p$, gdzie $R[i]$ to i-te przeniesienie,
- $(i + 1)$-sze przeniesienie $R[i + 1]$ jest równe $(A[i] + B[i] + R[i]) \operatorname{div} p$,

gdzie mod i div oznaczają odpowiednio operacje reszty z dzielenia i wyniku dzielenia całkowitego (tzn. zaokrąglenia dokładnego wyniku dzielenia do liczby całkowitej w dół). Musimy jednak uwzględnić, że wynik może być o jedną cyfrę dłuższy od dodawanych liczb. Dlatego przyjmiemy, że wynik ma n+1 cyfr i najbardziej znaczącą cyfrę wyniku zapiszemy na pozycji n+1.

```
i ← 1
R[1] ← 0
dopóki i < n+1 wykonuj
        c ← A[i] + B[i] + R[i]
        C[i] ← c mod p
        R[i + 1] ← c div p
        i ← i + 1
C[n+1] ← R[n+1]
```

W modelu odpowiedzi zamieściliśmy trochę inne rozwiązanie:

- zamiast tablicy przeniesień R przechowujemy tylko aktualne przeniesienie, w zmiennej r;
- pokazujemy tam, jak można uniknąć stosowania operatorów mod i div, korzystamy przy tym z tego, że $A[i]+B[i]$ jest zawsze nie większe niż $2p – 2$, a reszta $R[i]$ jest równa 0 lub 1.

Zadanie 20.

Wiązka zadań *Liczba narcystyczna*

Niech dana będzie liczba naturalna x, której zapis dziesiętny ma n cyfr:

$$x = a_{n-1}10^{n-1} + a_{(n-2)}10^{n-2} + \cdots + a_1 10 + a_0 \qquad (a_{n-1} \neq 0).$$

Powiemy, że liczba x jest *narcystyczna*, jeśli suma jej cyfr podniesionych do potęgi n-tej jest równa x, tzn.

$$a_{n-1}^n + a_{n-2}^n + \cdots + a_1^n + a_0^n = x.$$

Na przykład liczba 1634 jest narcystyczna, ponieważ

$$1^4 + 6^4 + 3^4 + 4^4 = 1634.$$

Powiemy, że liczba x jest *B-narcystyczna*, jeśli jej zapis w systemie o podstawie B ma n cyfr, których suma n-tych potęg jest równa x, tzn.

$$x = a_{n-1}B^{n-1} + a_{(n-2)}B^{n-2} + \cdots + a_1 B + a_0 \quad \text{oraz} \quad a_{n-1}^n + a_{n-2}^n + \cdots + a_1^n + a_0^n = x.$$

Na przykład liczba 289 jest 5-narcystyczna, ponieważ

$$289 = (2124)_5 = 2 \cdot 5^3 + 1 \cdot 5^2 + 2 \cdot 5 + 4 \text{ oraz } 289 = 2^4 + 1^4 + 2^4 + 4^4.$$

20.1.

Uzupełnij brakujące cyfry tak, aby powstałe liczby były liczbami narcystycznymi.

3□1

40□

54□48

20.2.

Sprawdź, które z poniższych liczb (podanych w systemie dziesiętnym) są B-narcystyczne dla podanych wartości B. Uzupełnij poniższą tabelkę, wpisując *prawda* lub *fałsz* w zależności od tego, czy liczba jest B-narcystyczna, czy też nie.

x	B	*prawda/fałsz*
3433	6	
4890	5	
8956	3	
15345	2	

20.3.

Napisz algorytm (w pseudokodzie lub wybranym języku programowania), który sprawdza, czy dana liczba x jest B-narcystyczna.

Dane:

x — liczba całkowita, $x \geq 0$,

B — liczba całkowita, $B \geq 2$.

Wynik:

TAK, jeśli liczba x jest B-narcystyczna, NIE — w przeciwnym przypadku.

Zadanie 21.

Wiązka zadań *Szybkie podnoszenie do potęgi*

W algorytmach szybkiego potęgowania można wykorzystać binarną reprezentację wykładnika dla obliczenia wartości x^k, gdzie k jest liczbą naturalną, $k \neq 0$, zaś x jest liczbą rzeczywistą. Przyjmijmy, że binarnym rozwinięciem wykładnika k jest ciąg $(k_n k_{n-1} k_{n-2} \ldots k_2 k_1 k_0)_2$.

Jedna z metod wyznaczania x^k polega na obliczaniu potęg liczby x dla wykładników o binarnych reprezentacjach:

k_n,

$k_n k_{n-1}$,

$k_n k_{n-1} k_{n-2}$,

…,

$k_n k_{n-1} k_{n-2} \ldots k_1$,

$k_n k_{n-1} k_{n-2} \ldots k_1 k_0$,

Inaczej mówiąc, uwzględniamy coraz dłuższe fragmenty ciągu $(k_n k_{n-1} k_{n-2} \ldots k_2 k_1 k_0)_2$. W pierwszym kroku przyjmujemy, że wynik jest równy *x*, gdyż $k_n = 1$. Znając wartość *x* do potęgi o binarnym zapisie $(k_n k_{n-1} k_{n-2} \ldots k_i)_2$, możemy łatwo wyliczyć *x* do potęgi o binarnym zapisie $(k_n k_{n-1} k_{n-2} \ldots k_i k_{i-1})_2$: podnosimy dotychczasowy wynik do kwadratu (do czego wystarczy jedno mnożenie). Jeśli $k_{i-1} = 1$, dodatkowo mnożymy uzyskany wynik przez *x*.

Przykład

Niech $k = 13 = (1101)_2$. Kolejne wyznaczane w naszym algorytmie potęgi to:

$$x^1,\ x^3 = x^2 x,\ x^6 = (x^3)^2,\ x^{13} = (x^6)^2 x,$$

zaś liczba wykonanych mnożeń jest równa 5 (zauważ, że aby obliczyć x^3, musisz najpierw obliczyć x^2, a aby obliczyć x^2, musisz wykonać jedno mnożenie: $x^2 = x \cdot x$).

21.1.

Korzystając z przedstawienia wykładnika w postaci binarnej, podaj kolejne potęgi liczby *x* wyznaczane powyższą metodą przy obliczaniu x^{38}.

21.2.

Uzupełnij tabelkę. Oblicz, ile mnożeń wykonywanych jest dla kolejnych wykładników.

k	**reprezentacja binarna *k***	**liczba mnożeń**
4	100	2
5	101	3
6		
7		
8		
15		
16		
22		
32		

21.3.

W wybranej przez siebie notacji (lista kroków, pseudokod, język programowania) napisz algorytm, który dla zadanej binarnej reprezentacji liczby naturalnej *k*, $k \neq 0$, oraz rzeczywistej liczby *x* oblicza wartość x^k zgodnie z metodą opisaną na początku zadania.

Specyfikacja

Dane:

x — liczba rzeczywista,

n — liczba całkowita nieujemna,

$k_n k_{n-1} k_{n-2} \ldots k_1 k_0$ — ciąg tworzący binarną reprezentację wykładnika *k*,

Wynik:

liczba rzeczywista $p = x^k = \underbrace{x \cdot x \cdot \ldots \cdot x}_{k\ razy}$

Zadanie 22.

Wiązka zadań *Schemat Hornera*

Schemat Hornera jest bardzo efektywną metodą obliczania wartości wielomianu

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0,$$

gdzie dane liczby rzeczywiste $a_0, a_1, \ldots, a_n$ nazywamy *współczynnikami*, a liczba całkowita $n \geq 0$ oznacza *stopień* wielomianu.

Schemat bazuje na zależności

$$P(x) = x(a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_2 x^1 + a_1) + a_0 = x \cdot Q(x) + a_0,$$

gdzie

$$Q(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \cdots + a_2 x^1 + a_1.$$

Stąd otrzymujemy następujący schemat obliczania wartości $P(x)$:

Dane:

n — liczba całkowita, $n \geq 0$,

x — liczba rzeczywista,

$a_0, a_1, \ldots, a_n$ — liczby rzeczywiste.

Wynik:

wartość $P(x)$

Algorytm (schemat Hornera):

$w \leftarrow a_n$
dla $k = n-1, n-2, \ldots, 0$ **wykonuj**
(*) $\quad w \leftarrow x \cdot w + a_k$
zwróć w **i zakończ**

22.1.

Uzupełnij poniższą tabelkę, podając wartości danych, jakie należy przyjąć w powyższym schemacie, aby wyznaczyć wartość $P(6)$ dla wielomianu

$$P(x) = 10x^5 - 13x^4 + x^3 + 2x^2 - 8x + 7.$$

Dane	Wartości
liczba naturalna n	
liczba rzeczywista x	
liczby rzeczywiste $a_0, a_1, \ldots, a_n$	

22.2.

Uzupełnij poniższą tabelkę, wyrażając wzorem liczbę operacji mnożenia i dodawania, jaka zostanie wykonana przez schemat Hornera (w wierszu oznaczonej przez (*)) dla danego wielomianu

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0.$$

Działanie	Liczba operacji
Dodawanie	
Mnożenie	

22.3.

Przeanalizuj działanie schematu Hornera podczas obliczania wartości $P(2)$ dla wielomianu

$$P(x) = 4x^6 - 3x^5 + 2x^3 - 5x^2 + 7x + 9.$$

W poniższej tabeli wpisz wartości *w* obliczane przez algorytm w linii (*)

	Wartość *w*
$k = 5$	
$k = 4$	
$k = 3$	
$k = 2$	
$k = 1$	
$k = 0$	

Podaj wynik, jaki zwróci algorytm: .....

22.4.

Wielomianem parzystym nazywamy wielomian stopnia $2n$ postaci

$$R(x) = a_n x^{2n} + a_{n-1} x^{2n-2} + \cdots + a_2 x^4 + a_1 x^2 + a_0,$$

tzn. taki, w którym występują tylko parzyste potęgi zmiennej x.

Bazując na schemacie Hornera, napisz algorytm o poniższej specyfikacji (w pseudokodzie lub wybranym języku programowania), który oblicza wartość parzystego wielomianu $R(x)$.

Dane:
n — liczba całkowita, $n \geq 0$,
x — liczba rzeczywista,
$a_0, a_1, \ldots, a_n$ — liczby rzeczywiste.
Wynik: wartość $R(x)$.

Przy ocenie rozwiązania będzie brana pod uwagę liczba operacji mnożenia i dodawania wykonywanych przez algorytm.

Zadanie 23.

Wiązka zadań *Obliczanie całkowitego pierwiastka kwadratowego*

Całkowity pierwiastek kwadratowy z liczby naturalnej *x* jest największą liczbą naturalną *p*, która spełnia nierówność $p^2 \leq x$. Poniższy algorytm służy do obliczania tej wartości przybliżonej.

Specyfikacja

Dane:

x — liczba naturalna

Wynik:

p — liczba naturalna spełniająca warunek $p^2 \leq x$ i $(p+1)^2 > x$

Algorytm

$i \leftarrow 0$
$a \leftarrow 0$
$r \leftarrow 1$
dopóki $a \leq x$ **wykonuj**
    $i \leftarrow i + 1$
    $a \leftarrow a + r$
    $r \leftarrow r + 2$
zwróć $i - 1$ **i zakończ**

23.1.

Niech $x = 21$. Przeanalizuj działanie powyższego algorytmu i uzupełnij wartości zmiennych a i r dla kolejnych wartości i podanych w tabeli.

Wartość i	Wartość a	Wartość r
0		
1		
2		
3		
4		
5		

23.2.

Zdecyduj, które z poniższych zdań są w odniesieniu do opisanego algorytmu prawdziwe (**P**), a które fałszywe (**F**). **Zaznacz znakiem X** odpowiednią rubrykę w tabeli .

	P	F
Konstruuje ciąg kwadratów kolejnych liczb naturalnych.		
Znajduje dokładną wartość pierwiastka z liczby x.		
Oblicza kolejne nieparzyste liczby naturalne.		
Wykonuje dokładnie tyle iteracji pętli, ile wynosi pierwiastek całkowity z liczby x.		

23.3.

Napisz algorytm obliczania całkowitego pierwiastka z liczby naturalnej, który wykorzystuje następującą zależność rekurencyjną definiującą ciąg x_n zbieżny do pierwiastka kwadratowego z liczby x:

$$\begin{cases} x_0 = \frac{x}{2} \\ x_{n+1} = \frac{1}{2}\left(x_n + \frac{x}{x_n}\right) \end{cases}$$

Specyfikacja

Dane:

x — liczba naturalna

Wynik:

p — liczba naturalna spełniająca warunek $p^2 \leq x$ i $(p+1)^2 > x$

Zadanie 24.

Wiązka zadań *Poszukiwania*

Dana jest liczba naturalna $n > 0$ oraz uporządkowana tablica liczb całkowitych $T[1..n]$. Rozważmy następującą funkcję F dla trzech argumentów p, k, e, które są liczbami całkowitymi dodatnimi:

```
funkcja F (p, k, e)
    jeżeli (k = p)
        jeżeli (T [p] > e)
            zwróć p i zakończ
        w przeciwnym razie
            zwróć p + 1  i zakończ
    w przeciwnym razie
        s ← (p+k) div 2
        jeżeli T [s] > e
            zwróć F (p, s, e)
        w przeciwnym razie
            zwróć F (s+1, k,  e)
```

24.1.

Przeanalizuj działanie funkcji F i uzupełnij poniższą tabelkę dla $p = 1$, $k = 5$, $e = 10$:

T	F (p, k, e)
[3, 4, 6, 8, 9]	
[15, 16, 18, 22, 24]	
[2, 10, 16, 24, 26]	
[1, 3, 10, 10, 18]	

24.2.

Zdecyduj, które z dokończeń podanego niżej zdania czynią z niego zdanie prawdziwe. Zaznacz to **znakiem X** w odpowiednich miejscach tabeli.

Funkcja F wykorzystuje

metodę zachłanną.	
strategię „dziel i zwyciężaj”.	
programowanie dynamiczne.	

24.3.

Zdecyduj, które z dokończeń podanego niżej zdania czynią z niego zdanie prawdziwe. Zaznacz to **znakiem X** w odpowiednich miejscach tabeli.

Liczba wywołań rekurencyjnych funkcji F dla początkowych wartości $p = 1$ oraz $k = n$, będącej potęgą dwójki, jest w najgorszym przypadku równa

n div 2.	
$\sqrt{n}$.	
$\log_2 n$.	

24.4.

Napisz algorytm, który korzystając z funkcji F, obliczy, ile jest liczb należących do przedziału [*a*, *b*] w uporządkowanej niemalejąco tablicy *T* [1..*n*], składającej się z liczb całkowitych dodatnich.

Specyfikacja

Dane**:**

n — liczba całkowita dodatnia

T [1..*n*] — uporządkowana tablica *n* liczb całkowitych, złożona z różnych liczb całkowitych dodatnich, taka że $T[1] \leq T[2] \leq \ldots \leq T[n]$

a — liczba całkowita dodatnia

b — liczba całkowita dodatnia

Wynik:

w — liczba elementów tablicy *T* [1..*n*] należących do przedziału [*a*, *b*].

Zadanie 25.

Wiązka zadań *Palindromy*

Palindrom to słowo lub zdanie, które czytane od lewej do prawej i od prawej do lewej brzmi tak samo. W zdaniu, które jest palindromem, ignorujemy odstępy między słowami. Ważne jest tylko, że kolejność liter przy czytaniu od lewej do prawej i od prawej do lewej jest taka sama. Dla potrzeb zadania przyjmujemy, że używamy tylko małych liter alfabetu łacińskiego. Poniżej podajemy przykłady słów i zdań, które są palindromami:

```
kajak
anna
elf ukladal kufle
trafili popili fart
ewo zeby tu byly buty bezowe
```

Poniższy algorytm sprawdza, czy podane na wejściu słowo jest palindromem**.**

Specyfikacja

Dane:

S[1..*d*] — słowo zapisane w tablicy znaków, gdzie *d*>1 oznacza liczbę znaków w słowie

Wynik:

TAK — gdy słowo *S* jest palindromem, NIE — w przeciwnym przypadku

```
    Algorytm:
     d ← długość(S)
     i ← d div 2
(*)  dopóki (i > 0) i (S[i] = S[d – i+1]) wykonuj
          i ← i – 1
     jeżeli i = 0
               zwróć TAK i zakończ,
     w przeciwnym razie
               zwróć NIE i zakończ
```

Uwaga: **div** oznacza operator dzielenia całkowitego, a wartością funkcji *długość* jest liczba znaków w słowie.

25.1.

Podaj przykład słowa o długości 9, niebędącego palindromem, dla którego powyższy algorytm wykonuje największą możliwą liczbę powtórzeń w pętli oznaczonej (*).

25.2.

Napisz algorytm, który sprawdza, czy zdanie jest palindromem.

Specyfikacja
Dane:

Zdanie[1..*d*] — zdanie o długości *d* znaków zapisane w tablicy znaków, składające się z małych liter alfabetu łacińskiego i spacji, w tym co najmniej jednej litery.

Wynik:

TAK, gdy *Zdanie* jest palindromem, NIE — w przeciwnym razie

Zadanie 26.

Wiązka zadań *Podobieństwo słów*

Słowo *X* nazywać będziemy *k–podrzędnym* względem słowa *Y*, jeśli **wszystkie litery występujące w *X* występują również w *Y*** oraz w słowie *Y* występuje dokładnie ***k* różnych liter**, które **nie** występują w słowie *X*.

Słowo *X* jest podrzędne względem słowa *Y*, gdy *X* jest *k*-podrzędne względem *Y* dla jakiegoś $k \geq 0$.

Przykład

Słowo *X=ABCAB* jest 1-podrzędne względem słowa *Y=BAACD* (w słowie *X* występują litery A, B, C; w słowie *Y* występują litery A, B, C, D). Podobnie słowo *X=ABCAB* jest 0-podrzędne względem *BAC* oraz nie jest podrzędne względem słów *ABDAB* i *ABAB* (litera *C* występuje w słowie X, a nie występuje w słowach *ABDAB* i *ABAB*). Zamieniając słowa rolami, możemy stwierdzić, że słowo *ABDAB* nie jest podrzędne względem słowa *ABCAB*, a słowo *ABAB* jest 1-podrzędne względem *ABCAB*.

Uwaga

W poniższych zadaniach przyjmujemy, że w słowach mogą występować tylko litery *A, B, C, D, E, F, G, H, I, J*. Ponadto w algorytmach dostępna jest funkcja *dlugosc*, która zwraca długość słowa będącego jej argumentem, oraz funkcja *kod* o wartościach podanych w poniższej tabeli:

litera	*A*	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*
kod(litera)	1	2	3	4	5	6	7	8	9	10

26.1.

Uzupełnij poniższą tabelę, wpisując w kolumnie Podrzędność słowo NIE, jeśli słowo *X* nie jest podrzędne względem słowa *Y*, a w przeciwnym wypadku — liczbę *k* taką, że *X* jest *k*-podrzędne względem *Y*.

Słowo *X*	Słowo *Y*	Podrzędność
HHGGFFEEDDCCBBAA	*ABCDEFGH*	
DCBADCBA	*FGHABCJD*	
ABCDE	*ABCCBAE*	
AAAAA	*AA*	
ABA	*ACA*	
ACEGJ	*ABCDEFGHJ*	

26.2.

Dany jest następujący **algorytm A**:

dla *i*=1,2,…,10 **wykonuj**
    Czyjest[*i*] ← fałsz
d ← *dlugosc*(*Y*)
dla *i*=1,2,…,*d* **wykonuj**
    lit ← *Y*[*i*]
    Czyjest[kod(*lit*)] ← prawda
d ← *dlugosc*(*X*)
czyp ← prawda
dla *i*=1,2,…,*d* **wykonuj**
    lit ← *X*[*i*]
    czyp ← *czyp* **i** *Czyjest*[kod(*lit*)]
jeżeli *czyp*=prawda
    zwróć 1
w przeciwnym razie
    zwróć 0

Podaj wynik działania powyższego algorytmu dla wartości *X* i *Y* z poniższej tabeli:

X	*Y*	wynik algorytmu A
HHGGFFEEDDCCBBAA	*ABCDEFGH*	
DCBADCBA	*FGHABCJD*	
ABCDE	*ABCCBA*	
AAAAA	*AA*	
AA	*AAAAA*	
ACEGJ	*ABCDEFGH*	

Uzupełnij podaną poniżej specyfikację algorytmu A.

Specyfikacja

Dane: X, Y — słowa, w których występują tylko litery ze zbioru {*A, B, C, D, E, F, G, H, I, J*}

Wynik: .....
.....

26.3.

Uzupełnij brakujące fragmenty poniższego algorytmu B, tak aby realizował on podaną specyfikację.

Specyfikacja

Dane:

X, Y — słowa, w których występują tylko litery ze zbioru {*A,B,C,D,E,F,G,H,I,J*}

Wynik:

k — liczba całkowita, taka że

— *X* jest *k*-podrzędne względem *Y*, jeśli $k \geq 0$,

— *X* nie jest podrzędne względem *Y*, gdy $k = -1$.

Algorytm B:

dla *i*=1,2,…,10 **wykonuj**
    Czy_x[*i*] ← fałsz
    Czy_y[*i*] ← fałsz
dx ← *dlugosc*(*X*)
dla *i*=1,2,…,*dx* **wykonuj**
    lit ← *X*[*i*]
    Czy_x[kod(*lit*)] ← prawda
dy ← *dlugosc*(*Y*)
dla *i*=1,2,…,*dy* **wykonuj**
    lit ← *Y*[*i*]
    Czy_y[kod(*lit*)] ← prawda
k ← 0
dla *i*=1,2,…,10 **wykonuj**
    jeżeli *Czy_y*[*i*]=prawda **oraz** *Czy_x*[*i*]= .....
        k← .....
    jeżeli *Czy_y*[*i*]=..... **oraz** *Czy_x*[*i*]=prawda
        zwróć –1 i **zakończ**
zwróć *k*

26.4.

Słowa *X* i *Y* nazywać będziemy *równoważnymi*, jeśli każda litera występuje tyle samo razy w słowie *X* i w słowie *Y*.

Przykład

Słowa *ABCA* i *BCAA* są równoważne, natomiast *ABCA* nie jest równoważne ze słowami *ABCC*, *BABB* i *ABCAD*.

Podaj algorytm, który sprawdza, czy dwa podane słowa są równoważne. Twój algorytm powinien realizować następującą specyfikację:

Specyfikacja

Dane:

X, *Y* — słowa, w których występują tylko litery ze zbioru {*A, B, C, D, E, F, G, H, I, J*}

Wynik:

1 — gdy *X* i *Y* są równoważne, 0 — w przeciwnym razie.

Zadanie 27.

Wiązka zadań *Dopasowanie z błędem*

W podanym *tekście*, złożonym z małych liter alfabetu łacińskiego, wyszukujemy słowo zwane *wzorcem*. Celem jest znalezienie takiego fragmentu tekstu, który jest albo dokładnie równy wzorcowi, albo jest mu równy z jednym błędem, czyli różni się od niego najwyżej jedną literą.

Na przykład wzorzec *para* można znaleźć w tekście *parawan* albo *aparat*, a z jednym błędem — w tekście *opera* albo *spadanie*. Będziemy zakładać, że tekst jest co najmniej tak samo długi jak wzorzec, a wzorzec składa się z co najmniej dwóch liter.

27.1.

Dla podanych wzorców i tekstów podaj, czy wzorzec występuje w tekście dokładnie, z jednym błędem, czy też w ogóle w nim nie występuje. Do tabeli wpisz odpowiednio „dokładnie", „z błędem" lub „nie".

Wzorzec	**Tekst**	**W jaki sposób wzorzec występuje w tekście?**
para	*opera*	z błędem
para	*aparat*	dokładnie
kran	*karawana*	
sport	*bezspornie*	
ryt	*zakryty*	
sofa	*solanka*	

27.2.

Może się zdarzyć, że wzorzec występuje w tekście więcej niż raz: na przykład w słowie *rabarbar* wzorzec *bar* występuje dwukrotnie. Wystąpienia mogą się częściowo nakładać: wzorzec *issi* występuje dwukrotnie w *mississippi*.

Podaj przykład tekstu o długości 8 oraz wzorca o długości 4, dla których wzorzec występuje w tekście (dokładnie) przynajmniej trzykrotnie.

27.3.

Podaj algorytm (w pseudokodzie lub wybranym języku programowania), który dla danego wzorca i danego tekstu, rozstrzygnie, czy wzorzec występuje w tekście (dokładnie lub z błędem).

Algorytm powinien wypisywać TAK, jeśli wzorzec występuje, NIE — w przeciwnym wypadku.

Dane:

dodatnie liczby całkowite m i n, $n \geq m$

wzorzec[1..m], *tekst[1..n]*, napisy złożone z małych liter alfabetu łacińskiego

Wynik:

słowo „TAK”, jeśli wzorzec występuje w tekście (dokładnie lub z błędem), zaś słowo „NIE”, jeśli nie występuje.

Zadanie 28.

Wiązka zadań *Słabe palindromy*

Niech $W[i,j]$ oznacza część słowa W, zaczynającą się na i-tej literze, a kończącej na j-tej literze W. Na przykład dla słowa $W = ABCDEF$ mamy $W[3,5] = CDE$. Z kolei $W[i]$ oznaczać będzie i-tą literę słowa W, czyli $W[i]=W[i,i]$.

Uwaga: W treści zadania nie będziemy odróżniać pojedynczych znaków od całych napisów, cojest charakterystyczne dla niektórych języków programowania.

Niech W będzie złożonym z liter A i B słowem o długości m. Słowo W nazywamy *słabym A-palindromem*, jeśli spełnia ono jeden z dwóch warunków:

1. W = „A”, czyli W jest słowem złożonym jedynie z litery A;
2. długość słowa m jest liczbą parzystą oraz spełnione są jednocześnie poniższe warunki:
   a. $W[1] = W[m]$
   b. $W[1,m/2]$ lub $W[m/2+1,m]$ jest słabym A-palindromem.

Przykłady

1. Jedynym dwuliterowym słabym A-palindromem jest *AA*.
2. Słowa $W_1=AAAA$ oraz $W_2=AABA$ są słabymi A-palindromami, oba spełniają warunek 2a (gdyż $W_1[1]=W_1[4]=A$ oraz $W_2[1]=W_2[4]=A$). Ponadto $W_1[1,2]=W_2[1,2]=AA$ jest także słabym A-palindromem, co zapewnia spełnienie warunku 2b zarówno dla W_1 jak i W_2.
3. Słowo $W=AAAAAA$ nie jest słabym A-palindromem, ponieważ nie jest spełniony warunek 2b. Słowo W ma długość m=6, co oznacza, że słowa $W[1,m/2]$ oraz $W[m/2+1,m]$ mają długość nieparzystą równą 3. W rezultacie żadne ze słów: $W[1,m/2]= AAA$, $W[m/2+1,m]= AAA$ nie jest słabym A-palindromem.
4. Słowo *AAABBAAA* nie jest słabym A-palindromem, natomiast *AABABBBA* jest słabym A-palindromem.

28.1.

Uzupełnij poniższą tabelę, ustalając, które słowa spełniają warunki 2a i 2b. oraz które są słabymi A-palindromami. W przypadku słabych A-palindromów w kolumnie *Uzasadnienie* wskaż tę połowę słowa, która zapewnia spełnienie warunku 2b (jeśli obie połowy zapewniają spełnienie 2b, to wystarczy wskazać tylko jedną). Dla słów, które nie są słabymi A-palindromami, kolumnę *Uzasadnienie* zostaw pustą.

Słowo	*Czy jest spełniony warunek*		*Czy słowo jest słabym A-palindromem?*	*Uzasadnienie*
	2a?	*2b?*		
AABAABAA	tak	tak	tak	AABA
AAABBAAA	tak	nie	nie	
AAABBAAB	nie	nie	nie	
AAAABBBB				
ABBBABAA				
ABAAAABA				
AAAAAAAAAA				

28.2.

Rozważmy mające różną długość słabe A-palindromy, w których występuje najmniej liter *A*. Wiemy, że jest tylko jeden słaby A-palindrom jednoliterowy równy *A* oraz jeden słaby A-palindrom dwuliterowy równy *AA*. Nietrudno sprawdzić, że wszystkie czteroliterowe słabe A-palindromy to: *AAAA, ABAA, AABA*. Zatem najmniejsza liczba liter A w słabym A-palindromie o długości 1 jest równa 1, najmniejsza liczba liter A w słabym A-palindromie o długości 2 jest równa 2, a najmniejsza liczba liter A w słabym A-palindromie o długości 4 jest równa 3.

Uzupełnij puste pola w poniższym zestawieniu:

m	*najmniejsza liczba liter A słabego A-palindromu o długości m*	*słaby A-palindrom o długości m i najmniejszej liczbie liter A*
2	2	AA
4	3	
8		

Uzupełnij puste pola w poniższej tabeli, podając najmniejszą liczbę liter A w słabych A-palindromach o podanych poniżej długościach, uzupełniając puste pola w poniższej tabeli.

m	*najmniejsza liczba liter A słabego A-palindromu o długości m*
16	
32	
2^{10}	
2^{20}	

28.3.

Podaj algorytm sprawdzający, czy podane na wejściu słowo *W* jest słabym A-palindromem. Twój algorytm powinien działać zgodnie z następującą specyfikacją:

Specyfikacja

Dane:

słowo *W*

Wynik:

„Tak”, gdy *w* jest słabym A-palindromem,

„Nie”, gdy w nie jest słabym A-palindromem.

Zadanie 29.

Wiązka zadań *Alfabet kulkowy*

W Kulkolandii do zapisu wiadomości używa się kulek o dwóch kolorach, czarnym i białym. Każda litera alfabetu jest zapisywana za pomocą pewnej liczby kulek odpowiednio uporządkowanych tak, jak to przedstawiono w poniższej tabeli. Zauważ, że zapis każdej litery kończy się dwiema czarnymi kulkami:

A	●●	I	○○●○●●	R	●○○○○●●
B	○●●	J	○●○○●●	S	●○○●○●●
C	○○●●	K	●○○○●●	T	●○●○○●●
D	●○●●	L	●○●○●●	U	○○○○○○●●
E	○○○●●	M	○○○○○●●	W	○○○○●○●●
F	○●○●●	N	○○○●○●●	X	○○○●○○●●
G	●○○●●	O	○○●○○●●	Y	○○●○○○●●
H	○○○○●●	P	○●○○○●●	Z	○●○○○○●●

29.1.

Odczytaj poniższą wiadomość zapisaną alfabetem kulkowym:

○○○○○●●●●●○●○○●●○○○○○○●●●○○○○●●●●

29.2.

Do dyspozycji masz trzy operacje:

- *pusty(ciąg)*, która inicjuje pusty ciąg kulek;
- *pobierz(ciąg),* która usuwa pierwszą kulkę z niepustego ciągu kulek i podaje ją jako wynik;
- *dołącz(ciąg, kulka)*, która do podanego ciągu dołącza na końcu wskazaną kulkę.

Zapisz algorytm (w pseudokodzie lub w wybranym języku programowania), który z danego ciągu kulek *wiadomosc* usuwa początkowy fragment odpowiadający pierwszej literze i zapisuje go jako ciąg kulek w zmiennej *litera*.

Specyfikacja

Dane:

wiadomosc — ciąg kulek z zakodowanym napisem

Wynik:

litera — ciąg kulek odpowiadający pierwszej literze z zakodowanego napisu

wiadomosc — ciąg kulek pozostałych po pobraniu kulek odpowiadających pierwszej literze

Uwaga: połączenie wynikowych *litera* i *wiadomosc* daje w rezultacie wejściową *wiadomosc*

29.3.

Do dyspozycji masz operacje:

- *pusty(tekst)*, która inicjuje pusty tekst;
- *pobierz_litere(ciąg)*, która usuwa z niepustego ciągu kulkowego z zapisaną wiadomością kulki odpowiadające pierwszej literze, a jako wynik podaje literę

odpowiadającą usuniętym kulkom;
- *dopisz(tekst, litera)*, która do podanego tekstu (może on być pusty) dołącza na końcu wskazaną literę;
- *czy_sa_kulki(ciąg)*, podającą wartość PRAWDA, gdy w podanym ciągu znajduje się przynajmniej jedna kulka, a wartość FAŁSZ, gdy w podanym ciągu nie ma żadnej kulki.

Zapisz algorytm (w pseudokodzie lub w wybranym języku programowania), który dekoduje napis zapisany kulkami.

Specyfikacja
Dane:
ciąg — ciąg kulek z zakodowaną wiadomością
Wynik:
wiadomosc — tekst z dekodowaną wiadomością

Zadanie 30.

Wiązka zadań *Szyfrowanie*

Szyfrowanie z kluczem k (k — liczba całkowita większa lub równa 0) polega na zastąpieniu każdego znaku wiadomości jawnej innym znakiem o pozycji przesuniętej w alfabecie o k znaków względem znaku szyfrowanego. Przy szyfrowaniu znaku należy postępować w sposób cykliczny, tzn. po osiągnięciu końcowego znaku alfabetu należy powrócić na jego początek. Rozważamy tylko wielkie litery alfabetu angielskiego, tj. litery o kodach dziesiętnych ASCII od 65 (dla znaku A) do 90 (dla znaku Z).

30.1.

Mając do dyspozycji funkcję *kod*(x), która dla danego znaku x podaje dziesiętny kod ASCII tego znaku (np. *kod*(A) = 65), oraz funkcję *znak*(y), która dla danego dziesiętnego kodu ASCII podaje znak odpowiadający temu kodowi (np. *znak*(77) = M), napisz funkcję *szyfruj*(zn,k), szyfrującą znak *zn* szyfrem z kluczem k.

Przykład działania funkcji: *szyfruj*(M,327) = B.

30.2.

Aby utrudnić proces deszyfrowania, postanowiono dla każdego znaku w słowie zastosować inny klucz. Kluczem znaku w słowie będzie numer jego pozycji w tym słowie.

Przykład: w słowie BIT, dla znaku B mamy k=1, dla I — k=2, a dla T — k=3. Zaszyfrowane słowo BIT to CKW.

Uzupełnij poniższą tabelę. Zaszyfruj opisanym sposobem podane słowo jawne oraz odszyfruj słowo zaszyfrowane.

Słowo jawne	**Słowo zaszyfrowane**
INFORMATYKA	
	LQPTZZLZ

30.3.

Dane jest słowo *S* zaszyfrowane sposobem opisanym w zadaniu 2. Podaj algorytm (w postaci pseudokodu lub kodu wybranego języka programowania) deszyfrujący słowo *S*.

Dane:

- n — liczba znaków w słowie S ($n>0$)
- $S[1..n]$ — zaszyfrowane słowo

Wynik:

- $J[1..n]$ — słowo jawne, które po zaszyfrowaniu daje słowo S

Zadanie 31.

Wiązka zadań *Szyfr skokowy*

Szyfrem skokowym z kluczem $k>0$ nazywać będziemy kodowanie tekstu opisane przez poniższy algorytm, w którym danymi na wejściu są: k — liczba całkowita dodatnia oraz W — tekst.

Algorytm:

$n \leftarrow dlugosc(W)$
$m \leftarrow n$ div k
jeżeli n mod $k \neq 0$
    $m \leftarrow m+1$
dla $i=1,2,\ldots,m$ **wykonuj**
    $j \leftarrow i$
    dopóki $j \leq n$ **wykonuj**
        wypisz $W[j]$
        $j \leftarrow j+m$

Uwaga: Przyjmujemy, że:

- dostępna jest funkcja *dlugosc*, która zwraca długość słowa będącego jej argumentem;
- kolejne znaki słowa W o długości n oznaczamy przez $W[1]$, $W[2]$,…, $W[n]$.

Przykład

Dla $k=3$ i tekstu SZYFROWANIE algorytm wypisze tekst SRNZOIYWEFA.

31.1.

Dla poniższych wartości k i W podaj tekst wypisany przez algorytm:

- $k=3$, W=ZADANIE1JESTŁATWE

  Tekst wypisany przez algorytm:.....

- $k=4$, W=ZADANIE1JESTPROSTE

  Tekst wypisany przez algorytm:.....

31.2.

Podaj teksty W, dla których powyższy algorytm wypisze na wyjściu podane wartości:

- $k=3$, W= .....

Tekst wypisany przez algorytm: UDOMEWIKAEOĆMD

- *k*=4, *W*= .....

  Tekst wypisany przez algorytm: DRJTOZEBES

31.3.

Podaj algorytm deszyfrowania tekstu zakodowanego szyfrem skokowym zgodny z poniższą specyfikacją:

Specyfikacja

Dane:

k — liczba całkowita dodatnia

X — tekst

Wynik:

W — tekst, którego szyfr skokowy z kluczem *k* jest równy *W*.

Przykład

Dla *k*=3 i *X*=SRNZOIYWEFA algorytm powinien zwrócić na wyjściu tekst *W*=SZYFROWANIE.

31.4.

Szyfrem mieszanym z kluczem *k*>0 nazywać będziemy następujący sposób kodowania tekstu *W* o długości *n*:

1. Dzielimy tekst *W* na *k* części tak, że wszystkie części poza ostatnią mają tę samą długość *m* równą zaokrągleniu w górę liczby *n* / *k*. Ostatnia część ma długość mniejszą lub równą *m*.
2. Tak uzyskane części wpisujemy w kolejnych wierszach prostokątnej tabeli o *k* wierszach i m liczbie kolumn.
3. Wypisujemy kolejne kolumny, zaczynając od pierwszej, zgodnie z następującą zasadą: kolumny o nieparzystym numerze wypisujemy z góry na dół, a kolumny o parzystym numerze z dołu do góry.

Przykład

Dla *k*=3 i tekstu *W*=SZYFROWANIE uzyskamy tabelę:

S	Z	Y	F
R	O	W	A
N	I	E	

a zaszyfrowana tekst ma postać SRNIOZYWEAF.

Podaj algorytm szyfrowania tekstu zakodowanego szyfrem mieszanym zgodny z poniższą specyfikacją:

Specyfikacja

Dane:

k — liczba całkowita dodatnia

W — tekst

Wynik:

X — szyfr mieszany tekstu *W* z kluczem *k*.

Zadanie 32.

Wiązka zadań *Kompresja*

Dany jest napis złożony z małych liter alfabetu angielskiego, który kompresujemy w następujący sposób: jeżeli w napisie występują kolejno po sobie dwa identyczne ciągi kolejnych znaków, na przykład *piripiri*, to możemy zastąpić je jednokrotnym wystąpieniem tego ciągu, ujętym w nawiasy okrągłe, np. *(piri)*.

Na przykład *xyzxyz* można zapisać jako *(xyz)*, zaś *rabarbar* jako *ra(bar)*.

32.1.

Niektóre fragmenty napisów znajdujących się w tabelce zostały skompresowane podaną powyżej metodą. Uzupełnij tabelkę, odtwarzając oryginalne napisy.

Napis skompresowany	**Napis oryginalny**
a(cd)a	acdcda
(pur)owy	
(z)(zz)	
(ab)a(abcd)	

32.2.

Podaj algorytm (w pseudokodzie lub wybranym języku programowania), który mając zapisany w tablicy pewien napis, sprawdzi czy jest on (w całości) dwukrotnym powtórzeniem pewnego fragmentu, a jeśli tak, wypisze skompresowany napis w postaci *(fragment)*. Jeśli napis wejściowy nie jest powtórzeniem, na wyjście nie należy nic wypisywać.

Dane:

- liczba całkowita *n>0* oraz tablica *napis[1..n]*, zawierająca napis złożony z małych liter alfabetu angielskiego

Wynik:

- jeżeli napis wejściowy jest dwukrotnym powtórzeniem tego samego fragmentu, wynikiem są kolejne znaki napisu skompresowanego.

32.3.

Dany jest napis, w którym niektóre fragmenty zostały skompresowane podaną powyżej metodą. Podaj algorytm (w pseudokodzie lub wybranym języku programowania), który mając na wejściu skompresowany napis zapisany w tablicy, wypisze na wyjście kolejne litery napisu oryginalnego.

Dane**:**

dodatnia liczba całkowita n oraz tablica *napis[1..n]*, zawierająca pewien napis skompresowany metodą opisaną wcześniej.

Wynik:

kolejne znaki oryginalnego napisu, którego skompresowana wersja jest w tablicy *napis.*

Uwaga: W napisie skompresowanym nie ma zagnieżdżeń nawiasów, czyli wewnątrz pary nawiasów nie wystąpią inne nawiasy.

Zadanie 33.

Wiązka zadań *Wędrówka po planszy*

Wędrowiec podróżuje po kwadratowej planszy rozmiaru $n \times n$. Swoją wędrówkę rozpoczyna na dowolnym polu **pierwszej kolumny** planszy, a na koniec powinien dotrzeć do **ostatniej kolumny**. Będąc w kolumnie $j<n$, wędrowiec może w jednym ruchu przenieść się do kolumny $j+1$ (nie może przenieść się do żadnej innej kolumny). Wartość każdego pola planszy jest liczbą całkowitą. Wartość pola w i-tym wierszu i j-tej kolumnie na planszy A oznaczać będziemy przez $A[i, j]$.

Rozważamy 3 typy wędrowca:

- skaczący, który z pola w kolumnie $j<n$ może przeskoczyć na dowolne pole w kolumnie $j+1$;
- spadający, który z pola $A[i, j]$ możne przenieść się tylko na pola $A[k, j+1]$ takie, że $i \le k \le n$;
- chodzący, który z pola $A[i, j]$ może przeskoczyć tylko na pola $A[k, j+1]$ takie, że $k=i$ lub $k=i-1$ lub $k=i+1$ oraz $1 \le k \le n$.

Przykład

Rozważmy planszę rozmiaru 10×10. Jeżeli bieżącą pozycją wędrowca jest pole $A[3,4]$, to

- **wędrowiec skaczący** może w jednym ruchu przenieść się do pól $A[1,5]$, $A[2,5],\ldots,A[10,5]$,
- **wędrowiec spadający** może przenieść się do pól $A[3,5]$, $A[4,5],\ldots,A[10,5]$,
- **wędrowiec chodzący** może przenieść się do pól $A[2,5]$, $A[3,5]$, $A[4,5]$.

Rozważmy następujący algorytm, opisujący trasę jednego z typów wędrowców, dla poniższych danych:

Dane: n — liczba naturalna większa od 1;
A — tablica rozmiaru $n \times n$ wypełniona liczbami całkowitymi.

Uwaga: W poniższym algorytmie $B[0...n+1,1..n]$ jest tablicą, której wiersze mają numery $0,1,\ldots,n,n+1$, a kolumny $1,2,\ldots,n-1, n$.

Algorytm:

```
dla i=1,2,…,n wykonuj
        jeżeli A[i,1]>0
                B[i,1] ← 1
        w przeciwnym razie
                B[i,1] ← 0
dla j=2,3,…,n wykonuj
        B[0,j] ← 0
        B[n+1,j] ← 0
        dla i=1,2,…,n wykonuj
                jeżeli A[i,j] ≤0
                        B[i,j] ← 0
                w przeciwnym razie
                        jeżeli B[i – 1, j – 1]=1 lub B[i, j – 1]=1 lub B[i + 1, j – 1]=1
                                B[i,j] ← 1
                        w przeciwnym razie
                                B[i,j] ← 0
d ← 0
dla i=1,2,…,n wykonuj
        jeżeli B[i,n]=1
                d ← 1
zwróć d
```

33.1.

Rozważmy działanie algorytmu dla n=5 oraz następującej zawartości tablicy A:

	1	2	3	4	5
1	–2	–1	4	7	8
2	–3	2	3	–10	–2
3	1	–4	–1	5	–5
4	–2	–1	–2	–3	9
5	–1	–5	1	–4	1

Podaj końcową zawartość kolumn 1,2,…,5 **tablicy B** oraz wartość zwracaną przez algorytm_1 dla powyższych danych.

Tablica B:

	1	2	3	4	5
0					
1					
2					
3					
4					
5					
6					

Wartość zwracana przez algorytm: .....

33.2.

Uzupełnij specyfikację podanego powyżej algorytmu.

Specyfikacja

Dane:

n — liczba naturalna większa niż 1

A — plansza rozmiaru $n \times n$ wypełniona liczbami całkowitymi.

Wynik:

1 — jeśli istnieje trasa wędrowca typu .....,

i prowadząca tylko przez pola o wartościach ..... .

0 — w przeciwnym przypadku

33.3.

Wartością trasy wędrowca nazywamy sumę liczb zapisanych na polach planszy, które wędrowiec odwiedza w trakcie trasy. Podaj algorytm zgodny z poniższą specyfikacją.

Specyfikacja

Dane:

n — liczba naturalna większa niż 1

A — plansza rozmiaru $n \times n$ wypełniona liczbami całkowitymi.

Wynik:

Największa wartość trasy wędrowca typu skaczącego zaczynającej się w pierwszej kolumnie i kończącej się w ostatniej kolumnie.

Przykład

Dla n=5 oraz zawartości planszy podanej w zadaniu 1 algorytm powinien zwrócić wartość 23; trasa wędrowca o największej wartości prowadzi przez pola $A[3,1]$, $A[2,2]$, $A[1,3]$, $A[1,4]$ i $A[4,5]$.

33.4.

Podaj algorytm zgodny z poniższą specyfikacją.

Specyfikacja

Dane:

n — liczba naturalna większa niż 1

A — plansza rozmiaru $n \times n$ wypełniona liczbami całkowitymi.

Wynik:

1 — jeśli istnieje trasa wędrowca spadającego zaczynająca się w pierwszej kolumnie, kończąca się w ostatniej kolumnie i przechodząca wyłącznie przez pola o wartościach nieujemnych;

0 — w przeciwnym przypadku.

Przykład

Dla *n*=5 oraz zawartości planszy podanej w zadaniu 1 algorytm powinien zwrócić wartość 0, gdyż trasa spełniająca podane warunki musi zaczynać się w polu *A*[3,1], z którego można przejść tylko do pól ujemnych w drugiej kolumnie. Dla *n*=5 i poniższej zawartości planszy algorytm powinien zwrócić wartość 1; trasa prowadząca tylko przez pola dodatnie może przechodzić np. przez pola: *A*[1,1], *A*[2,2], *A*[2,3], *A*[3,4] i *A*[4,5].

	1	2	3	4	5
1	2	–1	4	7	8
2	–3	2	3	–10	–2
3	1	–4	–1	5	–5
4	–2	–1	–2	3	9
5	–1	–5	–6	–4	1

1.3. Praktyka w teorii

Zadanie 34.

Bajtek i Bituś poznali starożytny chiński sposób testowania pierwszości liczby naturalnej *n*:

liczba *n* musi spełniać równanie

$$2^n \bmod n = 2,$$

gdzie *mod* oznacza operator dzielenia modulo, czyli resztę z dzielenia całkowitego.

Uwaga: Starożytny chiński sposób bywa zawodny, istnieją liczby, które spełniają to równanie, a nie są pierwsze; natomiast jeśli równanie nie zachodzi, to *n* jest na pewno złożona.

Chłopcy postanowili przetestować chińską metodą kolejne liczby n>=2, przeprowadzając obliczenia tylko na takich liczbach, które można reprezentować bez znaku na 8 bitach. co znaczy, że na każdym etapie obliczeń stosowali tylko liczby całkowite z zakresu od 0 do 255.

Bajtek obliczał potęgi 2^n dla kolejnych wartości *n* i dla obliczonej wartości wyznaczał resztę z dzielenia przez *n*.

Bituś skorzystał z praw arytmetyki modularnej. W każdym z kolejnych kroków obliczania potęgi, wynik iloczynu brał modulo *n*. Na przykład dla *n*=3 obliczenia wykonywał następująco:

$$2^3 \bmod 3 = ((2*2) \bmod 3)*2 \bmod 3 = 1*2 \bmod 3 = 2.$$

Który z nich mógł przetestować pierwszość liczb w większym zakresie?

Podaj możliwie największą wartość *n*, dla której możliwe było przeprowadzenie testu pierwszości metodą każdego z chłopców.

W obliczeniach Bajtka n_{max} = ……

W obliczeniach Bitusia n_{max} = ……

Komentarz do zadania

Największą liczbą, jaką można zapisać na 8 bitach, jest 255. Jest to największa wartość, jaka może wystąpić w obliczeniach cząstkowych i wynikowych bez ryzyka błędów obliczeń.

W obliczeniach Bajtka największą liczbą będzie 2^n. Aby nie przekroczyła ona wartości 255, *n* nie może być większe od 7: 2^7=128 mieści się w 8 bitach, ale 2^8=256 już się nie zmieści.

W obliczeniach Bitusia *n* nie może być większe niż 128, ponieważ wówczas

$2^7 = 128 = 128 \bmod n$, a 2*128 = 256, czego nie można zapisać na 8 bitach.

Zauważ, że mądry wybór algorytmu stwarza, mimo ograniczeń sprzętowych komputera, możliwość realizacji obliczeń w znacznie większym zakresie wartości niż metoda naiwnie najprostsza.

Rozwiązanie

W obliczeniach Bajtka n_{max} = 7.

W obliczeniach Bitusia n_{max} = 128.

Zadanie 35.

Jeden ze sposobów zapisania niezerowej liczby rzeczywistej *x* w pamięci komputera polega na:

1) przedstawieniu tej liczby w postaci iloczynu trzech liczb:

   $(-1)^S$ — przy czym *S* jest równe 0 lub 1 i nazywa się *znakiem*,

   M — *mantysy*, która jest liczbą z przedziału [1;2),

   2^C — przy czym *C* jest liczbą całkowitą zwaną *cechą*,

$$x = (-1)^S * M * 2^C.$$

2) zapisaniu w pamięci oddzielnie: znaku, mantysy i cechy.

Przykład

$$5.5 = 2.75*2^1 = \underbrace{1.375}_{\text{mantysa}}*2^{\overset{\text{cecha}}{②}}$$

Przyjmijmy, że każdą liczbę rzeczywistą zapisujemy na **8 bitach**.

Najstarszy bit (pierwszy z lewej) jest **bitem znaku**: 0 oznacza liczbę dodatnią, 1 — ujemną.

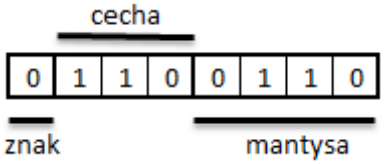
Następne 3 bity reprezentują **cechę —** liczbę całkowitą z zakresu od –4 do 3, reprezentowaną przez trzy cyfry binarne następująco:

-4	000
-3	001
-2	010
-1	011

0	100
1	101
2	110
3	111

Do zapisu **mantysy** pozostają ostatnie 4 bity. Część całkowita mantysy będzie zawsze równa 1, więc wystarczy zapisać tylko jej część ułamkową. Ułamek przedstawiamy w postaci binarnej i zapisujemy cztery pierwsze cyfry rozwinięcia, o wagach: 2^{-1}, 2^{-2}, 2^{-3} i 2^{-4}.

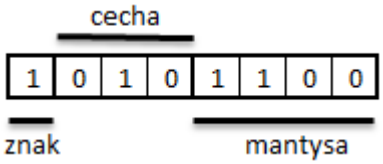
Przykład



Bit znaku ma wartość 0, więc liczba jest dodatnia
$C = 110_2 - 4 = 6 - 4 = 2$
$M = \mathbf{1}.0110_2 = 1 + 0.25 + 0.125 = 1.375$
$x = M*2^C = 1.375*2^2 = \mathbf{5.5}$

35a.

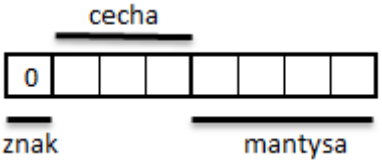
Oceń, czy podane poniżej zdania dotyczące liczby rzeczywistej zapisanej obok w postaci binarnej są prawdziwe, czy fałszywe, stawiając znak X w odpowiedniej kolumnie poniższej tabeli:



		P	F
A	Liczba jest dodatnia.		
B	Cecha ma wartość dziesiętną równą -2.		
C	Mantysa ma wartość dziesiętną równą 0.75.		
D	Liczba ma wartość dziesiętną równą -0.4375.		

35b.

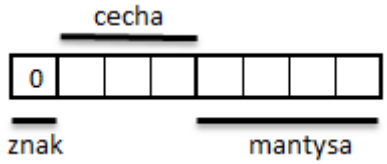
Jaką największą liczbę rzeczywistą *x* można zapisać, wykorzystując 8 bajtów w sposób opisany wyżej? Wypełnij bity reprezentacji binarnej tej liczby i podaj jej wartość dziesiętną.



C = ……
M = ……
x = ……

35c.

Jaką najmniejszą liczbę nieujemną można zapisać w opisanej wyżej reprezentacji 8-bitowej? Wypełnij bity jej reprezentacji binarnej i podaj jej wartość dziesiętną.



C = …….
M = …….
x = ……...

Komentarz do zadania

Zauważ, że w przyjętej reprezentacji:

- można zapisać jedynie wybrane liczby z przedziału od –15.5 do 15.5,
- nie da się zapisać liczby dokładnie równej zero,
- kolejne dwie liczby mogą różnić się od siebie nie mniej niż o 0.125.

W praktyce liczby rzeczywiste zapisywane są na większej liczbie bitów: 32 (w pojedynczej precyzji) lub 64 (w podwójnej precyzji). Szerokości cechy i mantysy są więc dużo większe. Nie zmienia to jednak istoty problemu: zakres wartości liczbowych możliwych do zapisania jest ograniczony, a wielu wartości należących do tego zakresu nie da się dokładnie zapisać.

Zadanie 36.

Dane są tabele `Uczniowie` i `Oceny`

Uczniowie:

Id_u	Imie	Nazwisko
1	Marek	Wolski
2	Hanna	Wieczorek
3	Irena	Iwanicka
4	Wojciech	Moscicki
5	Janina	Idzik
6	Arkadiusz	Czarnecki

Oceny

Id_u	Wynik
1	2
1	3
2	5
2	4
3	5
4	2
4	5
5	2
6	4
6	3

```
SELECT Uczniowie.Nazwisko
FROM Uczniowie
JOIN Oceny ON Uczniowie.Id_u = Oceny.Id_u
GROUP BY Uczniowie.Nazwisko
HAVING Avg(Oceny.Wynik)>=4.5;
```

Zaznacz nazwiska, które pojawią się w wyniku powyższego zapytania.

	T	N
Wolski		
Wieczorek		
Iwanicka		
Moscicki		
Idzik		
Czarnecki		

Zadanie 37.

Dana jest tabela `Produkty`:

Id	**Nazwa**	**Cena**
1	banany	3,20
2	jabłka	2,60
3	winogrona	7,20
4	gruszki	4,10
5	orzechy	5,30
6	pomarańcze	4,80
7	maliny	6,60
8	śliwki	4,10

```
SELECT Nazwa
FROM Produkty
ORDER BY Cena DESC;
```

Wśród pierwszych trzech wierszy wyniku dla powyższego zapytania pojawią się:

	T	N
gruszki		
orzechy		
pomarańcze		
maliny		
śliwki		

Zadanie 38.

Dana jest tabela `Agenci`:

ID_agenta	Obszar_dzialania
A001	Katowice
A002	Katowice
A003	Warszawa
A004	Warszawa
A005	Kraków
A006	Kraków
A007	Katowice

```
SELECT obszar_dzialania, COUNT(*)
FROM Agenci
GROUP BY obszar_dzialania;
```

Wynik powyższego zapytania to:

	T	N
Katowice 1 Warszawa 2 Kraków 3		
Katowice 3 Kraków 2 Warszawa 2		
Katowice A001 Kraków A005 Warszawa A003		
A00* 3		

Zadanie 39.

Wiązka zadań *Praktyka w teorii — Grafika*

Na stronie internetowej aquaparku postanowiono umieścić zdjęcia zachęcające do odwiedzin tego miejsca. Zarządcy aquaparku zależy, aby ze strony internetowej mogły korzystać także osoby mające jedynie dostęp do łącza internetowego niskiej prędkości (np. 1 Mbit/s). Jedno ze zdjęć o nazwie `aquapark1` zostało przekształcone za pomocą programu graficznego i zapisane jako `aquapark2`. W tabeli przedstawiono oba zdjęcia wraz z ich parametrami.

aquapark1	
typ obrazu:	jpeg
data:	14.08.2014
rozmiar:	3072 x 2304
głębia w bitach:	24bits
rozmiar pliku:	1,17 MB

aquapark2	
typ obrazu:	jpeg
data:	14.08.2014
rozmiar:	1024 x 768
głębia w bitach:	8bits
rozmiar pliku:	195 KB

39.1.

Który obraz powinien zostać dodany do galerii zdjęć na stronie aquaparku? Uzasadnij swój wybór.

39.2.

Dział promocji aquaparku zamierza przygotować do druku folder, w którym umieści różne zdjęcia. Wybierz z poniższej listy format pliku graficznego, który zapewni najlepszą jakość druku. Uzasadnij swój wybór.

TIFF	JPEG	GIF

39.3.

Ile pamięci zajmie bitmapa 1024 x 768 pikseli, jeśli zapisano ją w systemie RGB, przeznaczając na każdą składową 8 bitów? Wynik podaj w kilobajtach.

Zadanie 40.

Wiązka zadań *Praktyka w teorii*

Dla następujących zdań **zaznacz znakiem X**, która odpowiedź jest prawdziwa (P), a która jest fałszywa (F).

40.

W kolumnach `A`, `B`, `C`, `D` arkusza kalkulacyjnego w wierszach od 1 do 200 są wpisane oceny uczniów w postaci liczb całkowitych z przedziału <1,6> z następujących przedmiotów: język polski, język angielski, geografia i biologia.

Przykład

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
1	1	4	5	6					
2	3	3	1	5					
3	2	5	2	2					
4	5	2	5	3					
……									
200	6	2	5	3					

Aby w poszczególnych kolumnach `A`, `B`, `C`, `D` obliczyć liczbę ocen bardzo dobrych (o wartości liczbowej 5) i umieścić wyniki w komórkach `G1:J1`, można w komórce `G1` zastosować następującą formułę i skopiować ją do pozostałych komórek:

	P	F
`LICZ.JEŻELI(B$1:B$200;"=5").`		
`LICZ.JEŻELI(A$1:A$200;"=5").`		
`LICZ.JEŻELI($A$1:$A$200;"=5").`		
`LICZ.JEŻELI(A1:A200;"=5").`		

41.

Numer PESEL ma 11 cyfr. Dla osób urodzonych w latach 2000-2009 pierwszą cyfrą numeru PESEL jest zero. Aby dane w pliku tekstowym, zawierające kolumnę z numerem PESEL, zostały poprawnie zaimportowane do arkusza kalkulacyjnego (jako ciągi jedenastu znaków), należy podczas importu nadać tej kolumnie format:

	P	F
tekstowy.		
liczbowy.		
ogólny, a po zaimportowaniu zmienić go na format tekstowy.		
ogólny, a po zaimportowaniu zmienić go na specjalny format o nazwie „Numer PESEL”.		

Zadanie 42.

Wiązka zadań *Podział tablicy*

Rozważamy następujący algorytm.

Dane:

tablica liczb naturalnych $T[1..n]$

Algorytm:

```
        x ← T[1]
        i ← 0
        j ← n+1
        wykonuj
            wykonuj
              j ← j-1
(*)         dopóki T[j] > x
            wykonuj
              i ← i+1
(**)        dopóki T[i] < x
            jeżeli  i < j
(***)           zamień(T[i], T[j])
            w przeciwnym razie
                zakończ
```

Uwaga: funkcja *zamień*(T[i], T[j]) zamienia miejscami wartości T[i] oraz T[j].

42.1.

Przeanalizuj działanie algorytmu i podaj łączną liczbę operacji porównania, jakie zostaną wykonane w wierszach oznaczonych (*) i (**) dla danych zapisanych w poniższej tabeli:

Tablica T	**Liczba operacji porównania wykonanych w wierszu oznaczonym (*)**	**Liczba operacji porównania wykonanych w wierszu oznaczonym (**)**
4, 2, 5, 8, 1, 9, 7, 6, 3		
5, 4, 3, 2, 1, 6, 7, 8, 9, 10		
1, 2, 3, ... , 100		
100, 99, 98, ... , 1		

42.2.

Przeanalizuj działanie algorytmu i podaj łączną liczbę operacji zamiany, jakie zostaną wykonane w wierszu oznaczonym (***) dla danych zapisanych w poniższej tabeli:

Tablica T	**Liczba operacji zamiany wykonanych w wierszu oznaczonym (***)**
4, 2, 5, 8, 1, 9, 7, 6, 3	
5, 4, 3, 2, 1, 6, 7, 8, 9, 10	
1, 2, 3, ... , 100	
100, 99, 98, ... , 1	

Zadanie 43.

Wiązka zadań *Smartfon*

Rozważmy następujące aplikacje:

- serwis pogodowy,
- katalog książek biblioteki szkolnej,
- elektroniczny dziennik lekcyjny,
- edytor tekstu,
- serwis informacji turystycznej Wrocławia,
- wyszukiwanie optymalnej trasy samochodowej,
- arkusz kalkulacyjny,
- kompilator języka programowania,
- system komputerowego składu tekstu.

43.1.

Wskaż wśród powyższych aplikacji dwie, dla których przydatna jest informacja o położeniu geograficznym użytkowników korzystających z urządzeń mobilnych (np. smartfonów). Dla każdej z wybranych aplikacji opisz w jednym lub dwóch zdaniach sposób wykorzystania danych o lokalizacji użytkowników.

43.2.

Wskaż wśród powyższych aplikacji co najmniej dwie, których przydatność jest istotnie ograniczona w sytuacji, gdy użytkownik ma jedynie dostęp do urządzenia mobilnego (np. smartfon, tablet). Dla każdej z wybranych aplikacji uzasadnij swój wybór w co najwyżej dwóch zdaniach.

43.3.

Zdjęcia wykonywane na smartfonie Franka gromadzone są na jego koncie w chmurze obliczeniowej, o ile smartfon jest podłączony do Internetu. W przypadku braku dostępu do Internetu zdjęcia gromadzone są w pamięci podręcznej. Przy obecnych ustawieniach smartfonu często występuje problem braku miejsca w pamięci podręcznej, uniemożliwiający robienie dużej liczby zdjęć. Franek chciałby zmienić ustawienia tak, aby możliwe było zachowanie w pamięci większej liczby zdjęć.

Wśród podanych niżej sposobów wskaż te, które mogą pomóc w rozwiązaniu problemu Franka (zaznacz znakiem X która odpowiedź jest prawdziwa, a która jest fałszywa):

	P	F
zmiana formatu zapisu zdjęć na mapę bitową		
zmniejszenie rozdzielczości zapisywanych zdjęć		
zastosowanie kodów korekcji CRC, opartych na bitach parzystości		
obniżenie jakości kompresji zdjęć (w formacie JPEG)		

43.4.

Franek założył konto w banku i zamierza korzystać z bankowości internetowej. Wśród poniższych funkcjonalności wskaż te, które mogą służyć zabezpieczaniu usług bankowości internetowej przed nieuprawnionym dostępem (zaznacz znakiem X która odpowiedź jest prawdziwa, a która jest fałszywa):

	P	F
protokół transferu plików FTP (File Transfer Protocol).		
protokół SSL (Secure Socket Layer).		
uwierzytelnianie użytkownika przy pomocy hasła lub PIN.		
kompresja dysku twardego.		
hasła jednorazowe generowane przez układy kryptograficzne i dostarczane kanałami informacyjnymi alternatywnymi dla Internetu.		

1.4. Test z ogólnej wiedzy informatycznej

Zadanie 44.

Zdecyduj, które z dokończeń podanego niżej zdania czynią z niego zdanie prawdziwe (P), a które fałszywe (F). Zaznacz to **znakiem X** w odpowiednich miejscach tabeli.

Liczba 100110010_2

	P	F
jest dwa razy większa od liczby $10011001_{2.}$		
jest dwa razy mniejsza od liczby $1001100100_{2.}$		
jest większa niż $512_{10.}$		
jest mniejsza niż 472_8.		

Komentarz do zadania

Zadanie można rozwiązać na 2 sposoby. Jednym z nich jest zamiana zadanej liczby z systemu binarnego na dziesiętny i sprawdzenie, które odpowiedzi są poprawne:

$$100110010_2 = 1 \cdot 2^8 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^1 = 256 + 32 + 16 + 2 = 306\,.$$

Odpowiedź na pytanie pierwsze jest prawdziwa, ponieważ

$$10011001_2 = 1 \cdot 2^7 + 1 * 2^4 + 1 \cdot 2^3 + 1 \cdot 2^0 = 153.$$

Odpowiedź na pytanie drugie jest prawdziwa, ponieważ

$$1001100100_2 = 1 \cdot 2^9 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^2 = 612.$$

Odpowiedź na pytanie trzecie jest fałszywa, ponieważ liczba 306 jest mniejsza niż 512. W przypadku odpowiedzi na czwarte pytanie można zamienić liczbę zapisaną w systemie ósemkowym na system dziesiętny: $472_8 = 4 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0 = 314$, zatem odpowiedź jest prawdziwa.

Przedstawione rozumowanie wymaga jednak poświęcenia na obliczenia sporej ilości czasu. Zadanie można rozwiązać znacznie szybciej, bez zamiany zapisu zadanej liczby z systemu binarnego na dziesiętny.

Po pierwsze, wystarczy wiedzieć, że mnożenie liczby binarnych przez 2^i jest jednoznaczne z dopisywaniem i zer z prawej strony reprezentacji binarnej, zaś dzielenie liczb binarnych przez 2^i jest jednoznaczne z usuwaniem i najmniej znaczących bitów. W naszym przypadku $i = 1$, więc wystarczy:

- usunąć najmniej znaczący bit liczby w celu uzyskania odpowiedzi na pytanie pierwsze: $100110010_2 \rightarrow 10011001_2$,
- dopisać jedno zero z prawej strony liczby w celu uzyskania odpowiedzi na pytanie drugie: $100110010_2 \rightarrow 1001100100_2$

W pytaniu trzecim pojawia się liczba $512_{10} = 2^9 = 100000000_2 > 100110010_2$, zatem odpowiedź jest fałszywa.

W przypadku pytania czwartego wystarczy liczbę binarną rozdzielić na grupy 3-bitowe, idąc od strony prawej ku lewej. Jeśli w ostatniej grupie jest mniej bitów, to brakujące bity uzupełniamy zerami. Następnie każdą z grup bitowych zastępujemy odpowiednią cyfrą ósemkową. W wyniku tego otrzymujemy liczbę ósemkową o identycznej wartości jak wyjściowa liczba binarna: $100110010_2 \rightarrow 100\ 110\ 010_2 \rightarrow 462_8 < 472_8$ (co daje odpowiedź prawdziwą).

Rozwiązanie

PPFP

Zadanie 45.

Wskaż elementy, które są niezbędne do uruchomienia komputera i załadowania systemu operacyjnego.

	P	F
procesor		
twardy dysk		
pamięć operacyjna		
monitor		

Komentarz do zadania

Procesor jest centralną jednostką komputera, która wykonuje wszystkie obliczenia i steruje wykonywaniem instrukcji. Komputer nie może działać bez procesora.

Pamięć operacyjna jest konieczna, aby przechowywać dane, w tym podstawową część (jądro) systemu operacyjnego — komputer bez pamięci nie jest w stanie działać.

Komputer może działać bez twardego dysku: możliwe jest uruchomienie systemu operacyjnego z płyty CD lub pamięci typu *flash*, a nawet przez sieć lokalną, przy zdalnym użyciu dysku znajdującego się w innym komputerze.

Komputer może bez żadnych przeszkód działać bez monitora. Często na przykład w ten sposób pracują długo działające serwery, które wygodniej obsługiwać zdalnie, logując się na nie z innego komputera.

Zadanie 46.

Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe, a które jest fałszywe.

	P	F
System operacyjny przydziela zadaniom czas pracy procesora.		
System operacyjny używa zawsze tego samego systemu plików dla wszystkich urządzeń.		
W skład systemu operacyjnego wchodzi zawsze graficzny interfejs użytkownika.		
System operacyjny przydziela uruchamianym aplikacjom pamięć operacyjną.		

Zadanie 47.

Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe (P), a które jest fałszywe (F).

System plików NTFS

	P	F
nie jest obsługiwany przez system Linux.		
przechowuje informację o rozmiarze, dacie utworzenia i modyfikacji pliku oraz o ścieżce dostępu do pliku.		
uniemożliwia zapisanie pojedynczego pliku o rozmiarze powyżej 4 GB.		
umożliwia administratorowi nadawanie pojedynczym użytkownikom lub grupom użytkowników praw dostępu do plików i katalogów.		

Zadanie 48.

Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe, a które jest fałszywe.

W pewnej firmie znajdują się m.in. komputery o następujących adresach IP:

- komputer A: 10.20.30.40 / maska 255.255.0.0;
- komputer B: 10.0.0.10 / maska 255.255.255.0;
- komputer C: 1.2.3.4 / maska 255.255.255.0;
- komputer D: 1.2.3.250 / maska 255.255.255.0.

	P	F
Komputer A może być widoczny w sieci Internet pod innym adresem IP.		
Tylko dwa z wymienionych komputerów mogą mieć dostęp do sieci Internet.		
Komputery A i B znajdują się w jednej podsieci.		
Komputery C i D muszą znajdować się w jednym budynku.		

Zadanie 49.

Chmura obliczeniowa jest usługą polegającą na zdalnym udostępnieniu mocy obliczeniowej urządzeń IT, oferowaną przez zewnętrznego dostawcę. Oceń prawdziwość poniższych zdań, umieszczając znak X w odpowiedniej kolumnie tabeli.

		P	F
A	Z aplikacji i danych umieszczonych w chmurze można korzystać z dowolnej lokalizacji i dowolnego sprzętu IT umożliwiającego połączenie internetowe.		
B	Użytkownik nie jest zobowiązany do zakupu licencji na oprogramowanie używane w chmurze i udostępniane przez dostawcę, płaci jedynie za jego użycie (każdorazowo lub w formie abonamentu).		
C	Użytkownik może zdalnie instalować w przydzielonych zasobach chmury dowolne aplikacje i korzystać z nich tak jak na lokalnym komputerze.		
D	Pula zasobów użytkownika (w tym: procesory, pamięć RAM, przestrzeń dyskowa) jest elastycznie skalowana w zależności od jego potrzeb i ograniczona tylko możliwościami dostawcy.		

Zadanie 50.

HTTP Cookie jest niewielką porcją informacji wysyłaną przez witrynę internetową do przeglądarki klienta i zapisywaną w jej ustawieniach. Oceń prawdziwość poniższych zdań, umieszczając znak X w odpowiedniej kolumnie tabeli.

		P	F
A	*Cookie* zawiera polecenia, które konfigurują ustawienia przeglądarki klienta.		
B	*Cookie* umożliwia serwisowi sprawdzenie, czy klient już go odwiedzał w przeszłości, oraz zapamiętanie upodobań klienta.		
C	*Cookie* zapisane przez serwis z domeny *cwaniak.org* może być odczytane przez serwis z domeny *spryciarz.org*.		
D	Zablokowanie obsługi *cookie* w przeglądarce może spowodować utrudnienia dla użytkownika dokonującego zakupów w sklepie internetowym.		

Zadanie 51.

Czterech użytkowników założyło konta w pewnym serwisie internetowym. Wybrali następujące dane logowania:

Użytkownik A	nazwa użytkownika: *jankowalski* hasło: *1234*
Użytkownik B	nazwa użytkownika: *merkuriusz_312* hasło: *merkuriusz_312*
Użytkownik C	nazwa użytkownika: *abc* hasło: *S4o9s2n5a7*
Użytkownik D	nazwa użytkownika: *master* hasło: *password*

Przestępca usiłuje przejąć konta użytkowników, stosując w tym celu następujące dwie techniki:

- atak *brute force* (wszystkie możliwe kombinacje znaków),
- metoda psychologiczna i słownikowa (zgadnięcie hasła, sprawdzenie popularnych haseł i słów).

Wskaż, które hasła są podatne na złamanie tymi metodami, stawiając w odpowiednim polu tabeli znak X, jeśli taka podatność występuje:

	atak *brute force*	metoda słownikowa/psychologiczna
Użytkownik A		
Użytkownik B		
Użytkownik C		
Użytkownik D		

Zadanie 52.

Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe (P), a które jest fałszywe (F).

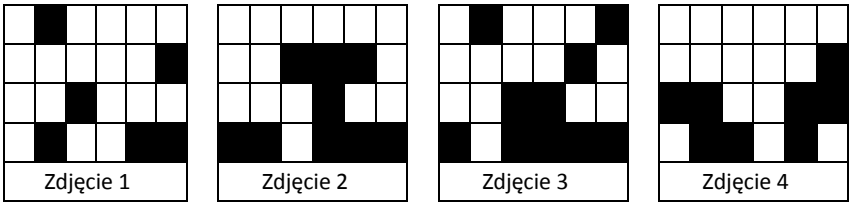
W grafice rastrowej 24-bitowe kodowanie koloru oznacza, że

	P	F
liczba kolorów w palecie barw wynosi ponad 16 milionów.		
informacje o kolorze jednego piksela zajmują 3 bajty.		
liczba kolorów jest niewystarczająca do zapisu zdjęć.		
obraz o rozmiarach 300x300 pikseli zapisany bez kompresji ma wielkość około 2,16 MB.		

Zadanie 53.

Wiązka zadań *Zegar binarny*

Pan Kowalski postanowił wyjechać na wycieczkę na wyspę Binarną. Mieszkańcy tej wyspy do zapisywania cyfr dziesiętnych używają metody obrazkowej, opartej na systemie binarnym. Podczas pobytu na wyspie Pan Kowalski wybrał się na jednodniową wycieczkę objazdową. W jej trakcie zrobił 4 poniższe zdjęcia zegara używanego na wyspie. Stan zegara podawany jest w formacie GG:MM:SS, gdzie GG to godzina, MM to minuty, a SS to sekundy (zapisane zawsze przy pomocy dwóch cyfr). Każda cyfra zapisana jest w osobnej kolumnie.



Pierwsze zdjęcie zostało zrobione w momencie rozpoczęcia wycieczki, ostatnie zdjęcie — w chwili jej zakończenia, zdjęcia 2 i 3 zostały natomiast zrobione w trakcie wycieczki.

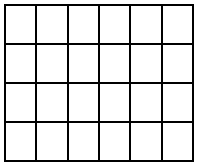
53.1.

Podaj w systemie dziesiętnym stany zegara wyświetlane na zdjęciach 1 oraz 4.

- Stan zegara na zdjęciu nr 1: ……
- Stan zegara na zdjęciu nr 4: ……

53.2.

Pan Kowalski położył się spać o godzinie 23:45:29. Zamaluj odpowiednie pola poniższej planszy tak, aby reprezentowała ona tę godzinę na zegarze używanym przez mieszkańców wyspy.



Zadanie 54.

Zaznacz znakiem X w odpowiedniej kolumnie, które zdanie jest prawdziwe (P), a które jest fałszywe (F).

Stos jest strukturą danych, która umożliwia

	P	F
bezpośredni dostęp do ostatnio zapisanego elementu.		
bezpośredni dostęp do każdego elementu stosu.		
bezpośredni dostęp do najmniejszego i największego elementu stosu.		
dodanie nowego elementu oraz usunięcie najpóźniej dodanego elementu.		

Zadanie 55.

Wiązka zadań *Zadania Zamknięte Funkcja*

Dana jest następujący algorytm F(n) dla $n \in N, n > 0$:

F(n)
jeżeli n = 1, **zwróć** 1 **i zakończ**
w przeciwnym razie zwróć F(n **div** 2) + 1

55.1.

Złożoność tego algorytmu jest

	P	F
wykładnicza.		
logarytmiczna.		
liniowa.		
kwadratowa.		

55.2.

Dla tego algorytmu zachodzi

	P	F
F(8) = 3.		
F(12) = 4.		
F(1) = 0 **lub** F(9) = 4.		
F(1) = 1 **oraz** F(9) = 3.		

Zadanie 56.

Piractwo komputerowe jest przestępstwem polegającym na:

	P	F
wykorzystywaniu oprogramowania w celu osiągnięcia korzyści majątkowej, bez licencji na jego użytkowanie.		
instalowaniu wersji demo oprogramowania, którego licencję planujemy kupić.		
rozpowszechnianiu w Internecie programów komputerowych, których licencję zakupiliśmy.		
oferowaniu za darmo w Internecie oprogramowania rozpowszechnianego na licencji freeware.		

Zadanie 57.

Zgodnie z prawem w Internecie można opublikować zdjęcie osoby:

	P	F
po uzyskaniu od niej zezwolenia.		
gdy jest to osoba powszechnie znana i zdjęcie zostało wykonane podczas pełnienia przez nią funkcji publicznych, w szczególności politycznych, społecznych, zawodowych.		
gdy osoba ta jest naszym bliskim znajomym.		
gdy stanowi ona jedynie szczegół całości takiej jak: zgromadzenie, krajobraz, publiczna impreza.		

2. Zadania praktyczne rozwiązywane z użyciem komputera

Wczytywanie danych

Dane do zadań rozwiązywanych na komputerze zapisane są w plikach tekstowych. Prosty plik tekstowy (`txt`) zawiera dane w postaci alfanumerycznej bez jakiegokolwiek formatowania. Każdy system operacyjny posiada program do otwierania plików tekstowych. W systemie Windows jest to Notatnik.

Importowanie danych do arkusza kalkulacyjnego MS Excel

W założeniu każdy wiersz pliku tekstowego ma być zaimportowany do osobnego wiersza w pliku arkusza kalkulacyjnego. Skąd jednak program ma wiedzieć, jak rozłożyć dane pomiędzy kolumnami? Spójrzmy na poniższy przykład, który przedstawia jeden wiersz z przykładowego pliku tekstowego:

```
Jan, Malinowski; 4,000; 1980-01-12
```

Dla nas jest to dość oczywiste: powyżej mamy imię i nazwisko osoby, jakąś liczbę (np. jej zarobki) i datę (np. datę urodzenia). Poszczególne pola rozdziela średnik. Powyższe dane powinny być więc zaimportowane w taki sposób:

	A	B	C	D
1	Jan	Malinowski	4,000	1980-01-12
2				

Rysunek 1- przykładowe dane po zaimportowaniu

Jednak arkusz kalkulacyjny nie potrafi analizować znaczenia danych, może je więc inaczej zaimportować. Dlatego tak ważny jest separator, który oddziela dane w wierszu.

	A	B	C	D	E
1	Jan, Malinowski; 4,000; 1980-01-12				separatorem jest tabulator
2					
3	Jan, Malinowski	4	1980-01-12		separatorem jest średnik
4					
5	Jan	Malinowski; 4	000; 1980-01-12		separatorem jest przecinek

Rysunek 2- Przykłady błędnie zaimportowanych danych

Do zaimportowania danych do arkusza służy *kreator*. Podpowiada on, jak ma być traktowany plik tekstowy, na którym pracujemy, i jak prawidłowo rozmieścić dane w komórkach. Przejdźmy więc wreszcie do arkusza kalkulacyjnego i zobaczmy, jakie pytania zadaje nam program oraz jak możemy na nie odpowiedzieć.

Rozpoczynamy import

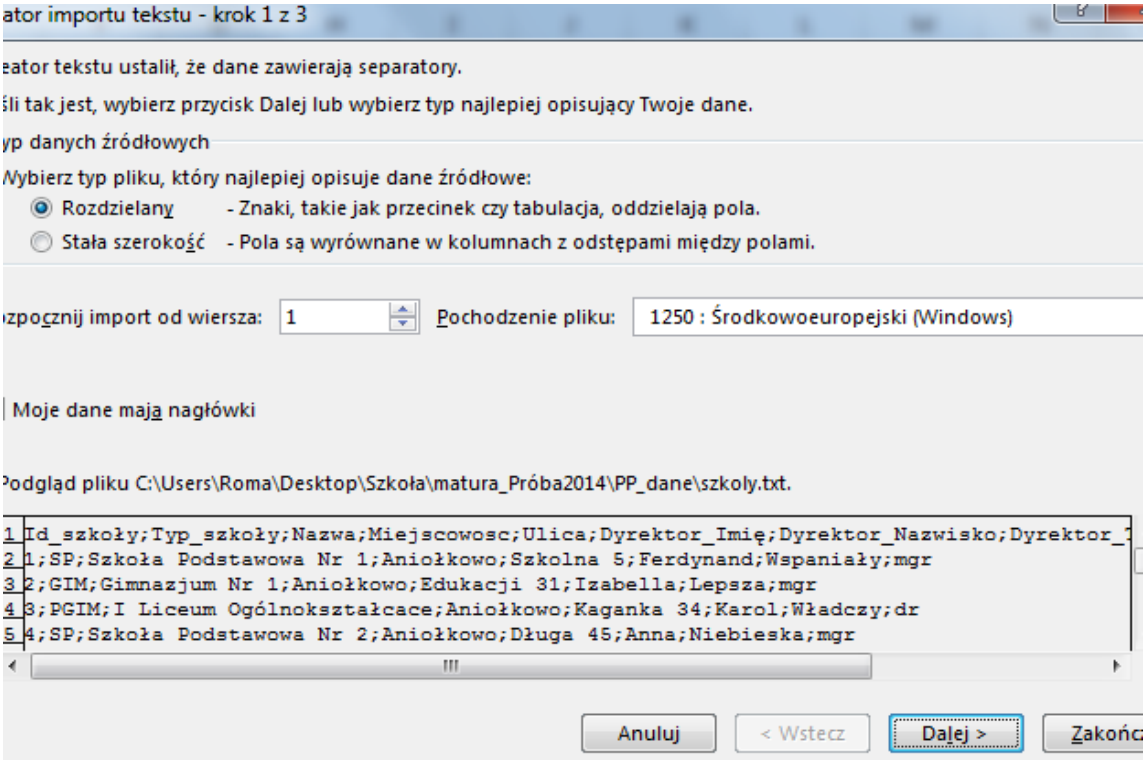
Kreator uruchamiamy przyciskiem *Z tekstu*, znajdującym się na karcie *Dane*, w grupie *Dane zewnętrzne*.

Rysunek 3 — Przycisk uruchamiający kreatora

W nowym oknie wskazujemy plik źródłowy (w naszym przypadku będzie to plik `szkoly.txt`). Załóżmy, że jest to plik zawierający informacje o różnych szkołach. Poszczególne kolumny oddzielone są znakiem **średnika**.

Przechodzimy do pierwszego kroku kreatora.

Krok 1 z 3



Rysunek 4 — Krok pierwszy importu pliku

Pierwsza decyzja dotyczy rodzaju pliku:

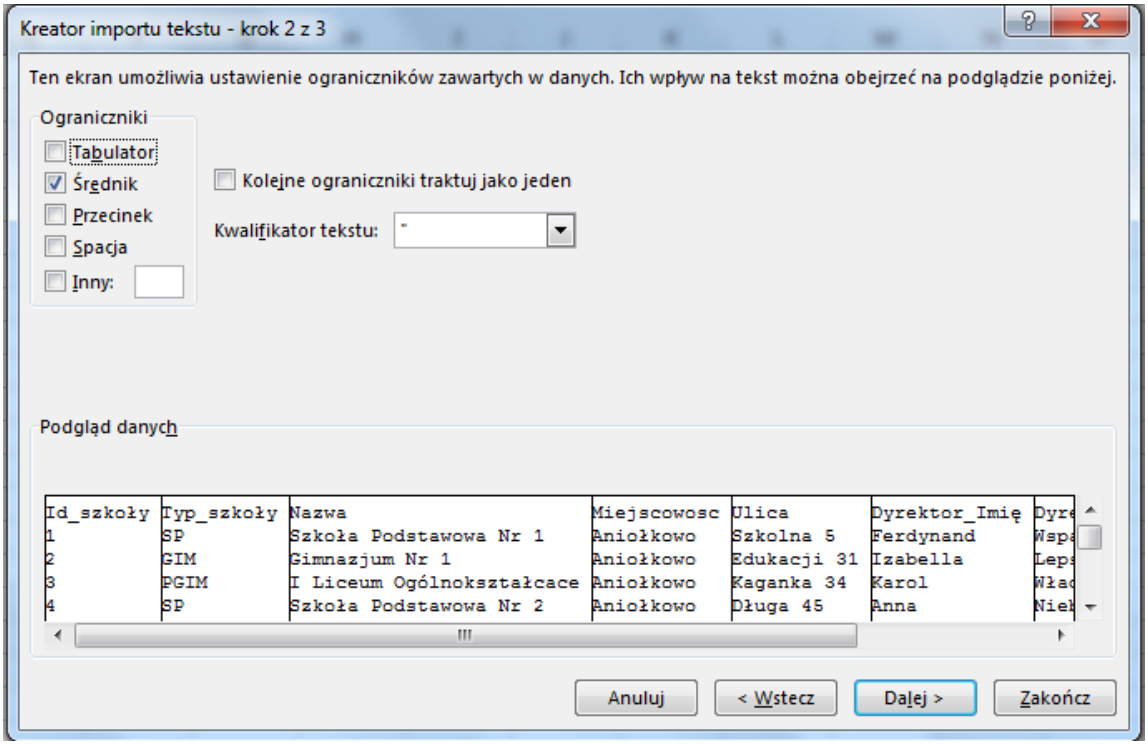
- *Czy jest to plik rozdzielany?* W pliku rozdzielanym kolejne kolumny wyróżnione są jakimś znakiem (spacja, tabulator, średnik). Arkusz, napotykając taki znak w wierszu pliku, wie, że kolejne dane ma wpisywać w następnej kolumnie.
- *Czy jest to plik z polami o stałej szerokości?* W takim pliku kolejne kolumny zajmują określoną liczbę znaków. Na przykład pierwsza kolumna zajmuje dziesięć znaków, a druga — osiem. Po wpisaniu do pierwszej kolumny dziesięciu znaków kreator wpisze do kolejnej następne osiem i tak dalej.

Nasz plik jest rozdzielany średnikiem. Zdarza się, że dane są w pliku poprzedzone nagłówkiem. System generujący plik tekstowy może umieścić w nim datę wygenerowania pliku, swoją nazwę lub inny komentarz do danych. Pole *Rozpocznij import od wiersza* pozwala nam ominąć te informacje i zacząć pobieranie danych od odpowiedniego miejsca. W naszym pliku `szkoly.txt` w pierwszym wierszu są nagłówki kolumn, znowu więc zostawiamy domyślne ustawienia.

Pole *Pochodzenie pliku* najczęściej zostawiamy bez zmian. Jeżeli w podglądzie pliku poniżej zamiast liter widać "krzaczki", niezrozumiałe znaki, trzeba zmienić tę opcję.

Krok drugi wygląda różnie, zależnie od tego, jaki rodzaj pliku wybraliśmy przed chwilą.

Krok 2 z 3 — plik rozdzielany

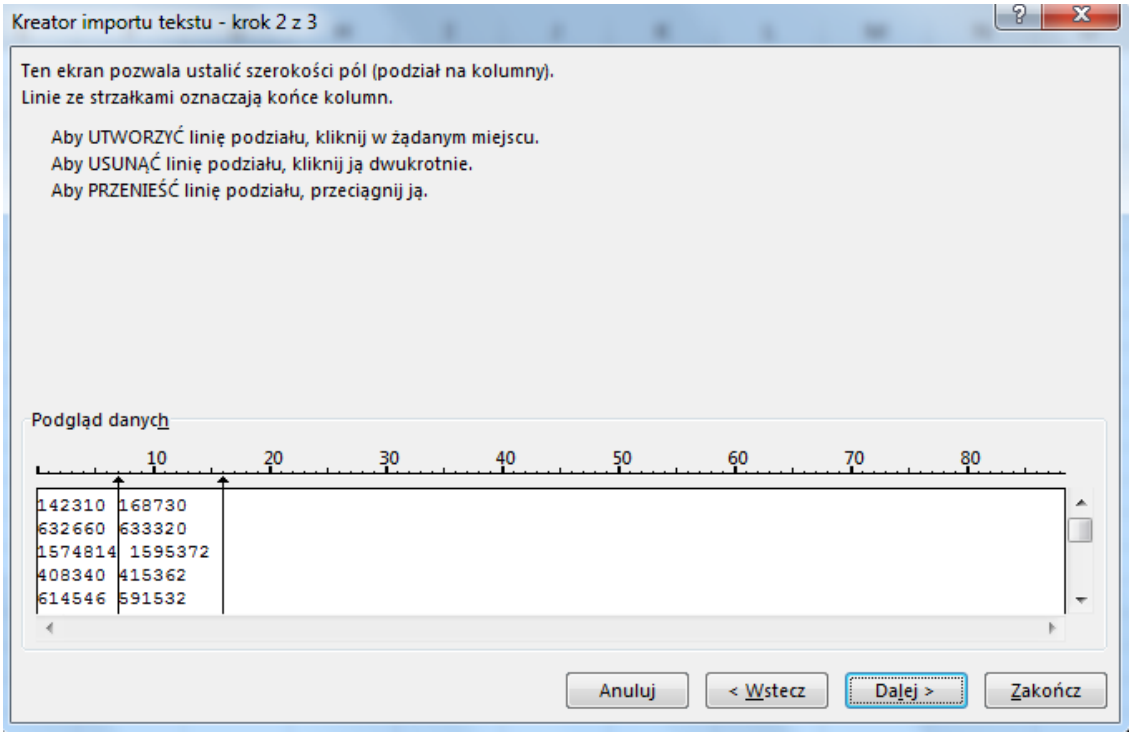


Rysunek 5 — Krok drugi — dla pliku rozdzielanego

W tym kroku określamy, jaki znak rozdziela kolumny — w naszym pliku to średnik. Poniżej widać podgląd importowanego pliku, odpowiednie dane są już w swoich kolumnach.

Krok 2 z 3 — stała szerokość

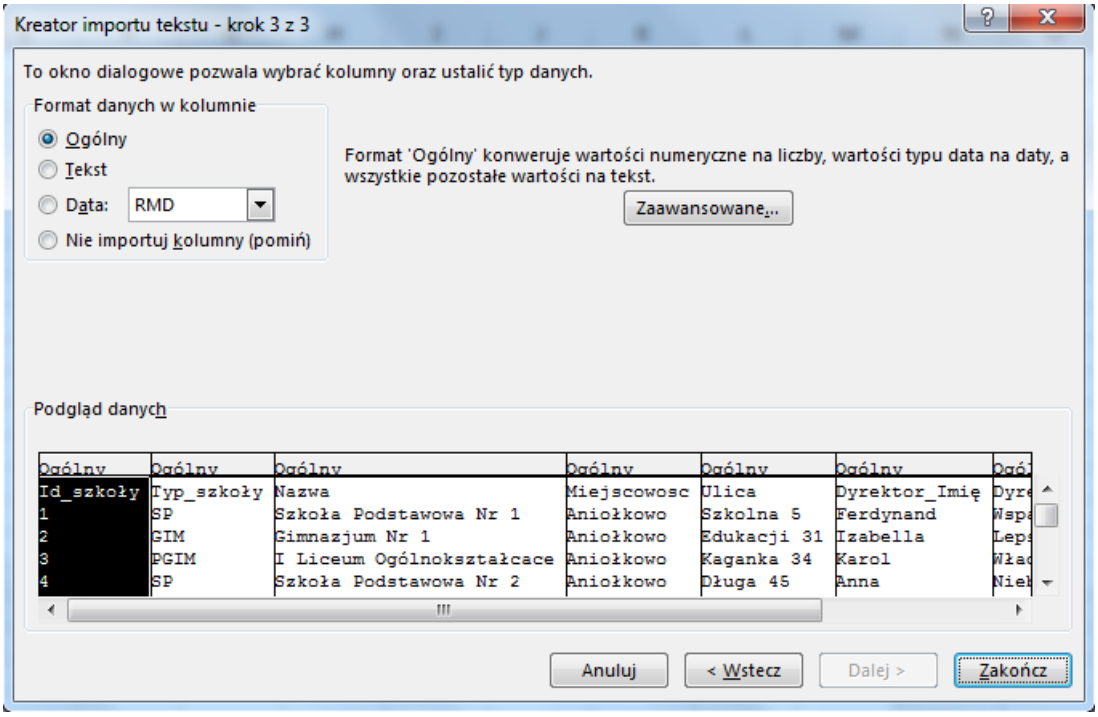
Nie dotyczy to naszego pliku, jednak spójrzmy na chwilę, jak wygląda krok drugi w tym przypadku, oczywiście dla innego pliku.



Rysunek 6 — Krok drugi dla pliku o stałej szerokości pól

W powyższym oknie musimy wskazać, gdzie zaczynają się kolejne kolumny danych, zaznaczając kursorem miejsce podziału.

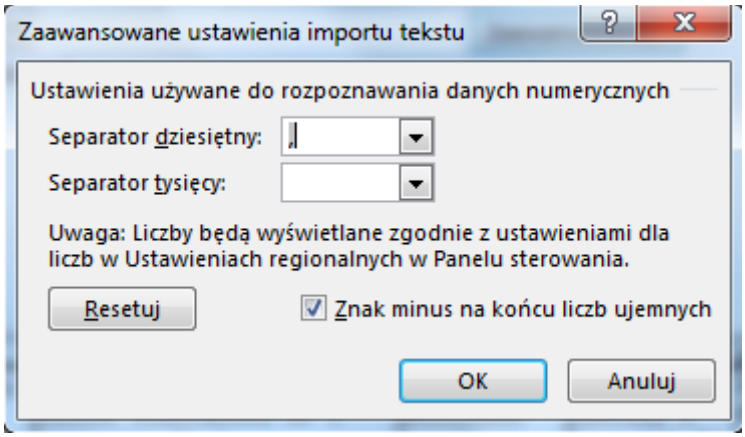
Krok 3 z 3



Rysunek 7 — Krok trzeci importu

W trzecim kroku na ogół akceptujemy domyślne ustawienia kreatora importu. Zwróćmy jednak uwagę na dostępne tu opcje. Wybierając kolumny w podglądzie danych, możemy zmieniać sposób, w jaki są one importowane. Możemy też zdecydować, które kolumny powinny być pominięte w procesie importu. Szczególnie należy zwrócić uwagę na przycisk *Zaawansowane*.

W tym miejscu potrzebna jest mała dygresja. W polskim systemie Windows część całkowita i ułamkowa oddzielane są przecinkiem. Ustawienia stosowane przez system Windows można sprawdzić w opcji *Ustawienia regionalne* Panelu sterowania. Plik tekstowy może pochodzić z innego systemu, w którym na przykład do tego stosowana jest.



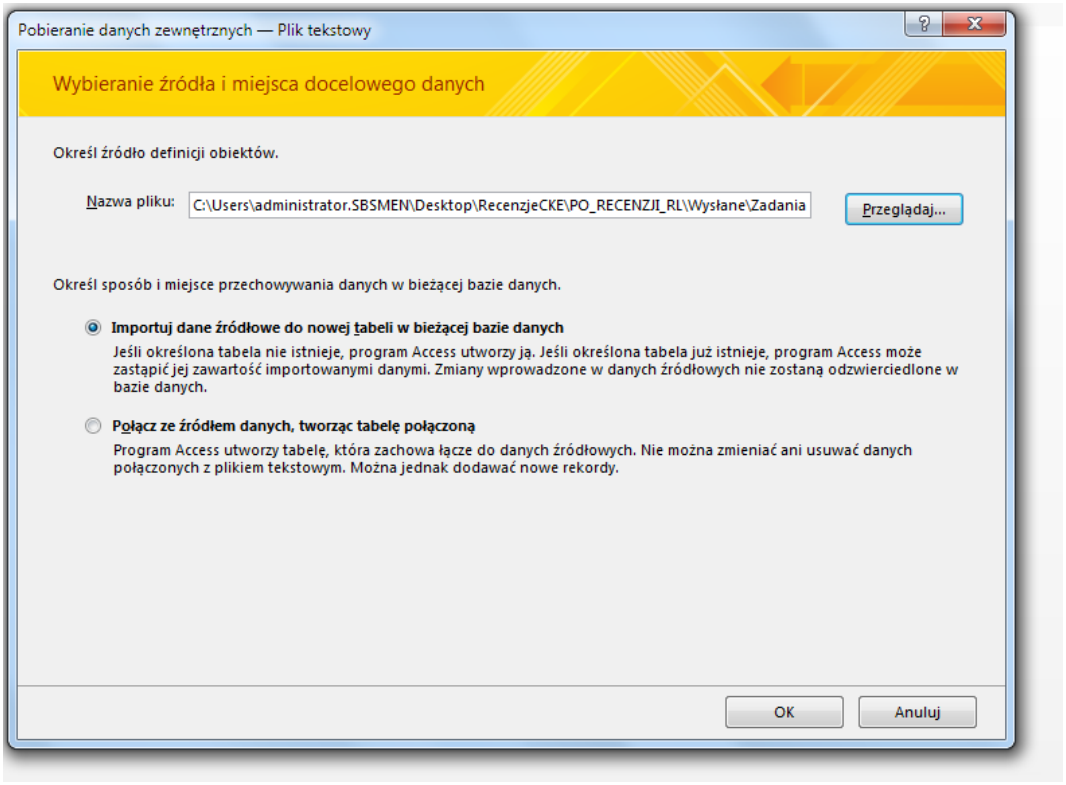
Rysunek 8 — Ustawienia zaawansowane importu liczb

Na przykład liczba 1982,45 zostanie zinterpretowana w polskim systemie prawidłowo, ale ta liczba może być też zapisana jako 1,982.45 (w zapisie amerykańskim). Przy poniższych ustawieniach podpowiadamy, jak zapisać liczby w naszym pliku w taki sposób, aby w jego komórkach ostatecznie pojawiły się właściwe liczby. Dzięki temu będziemy mogli wykonywać różne działania (sumy, średnie itp.). Przy błędnych ustawieniach Excel zaimportuje ciąg znaków „1,982.45", będzie go traktował jako tekst i nie pozwoli nam na działania, jakie byłyby dostępne w przypadku liczb.

W przypadku naszego pliku możemy jednak zostawić ustawienia domyślne. Naciskam więc *Zakończ* i następnie *OK*, aby zaobserwować, jak mój arkusz wypełnia się danymi z pliku.

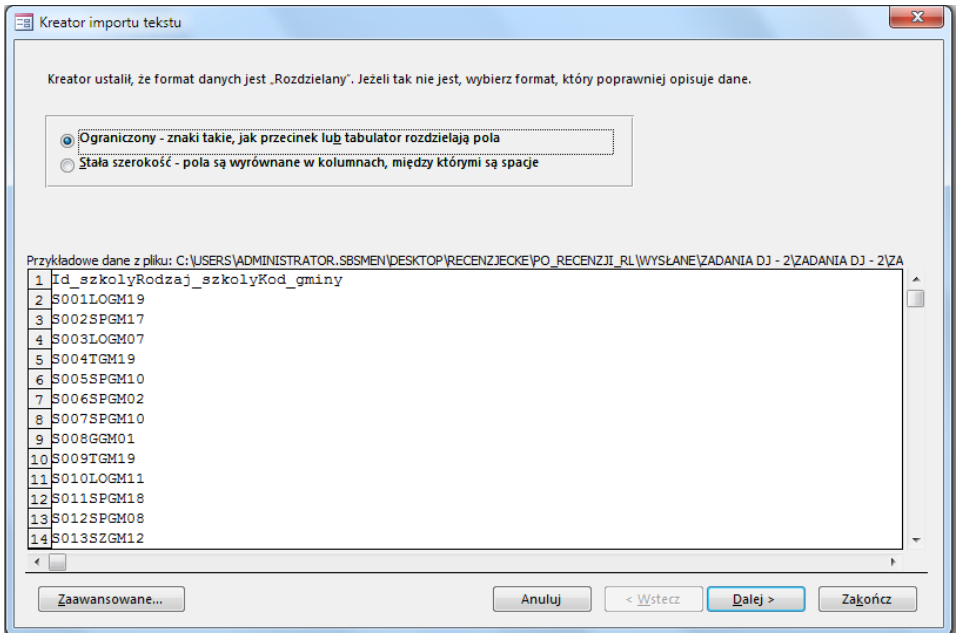
Importowanie danych do MS Access

Dane importujemy do założonej pustej bazy danych, przy czym dla każdego pliku tworzymy osobną tabelę. Zademonstrujmy import na przykładzie pliku `szkoly.txt`, który zawiera zestawienie różnych szkół w poszczególnych gminach. Pierwszy wiersz jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.



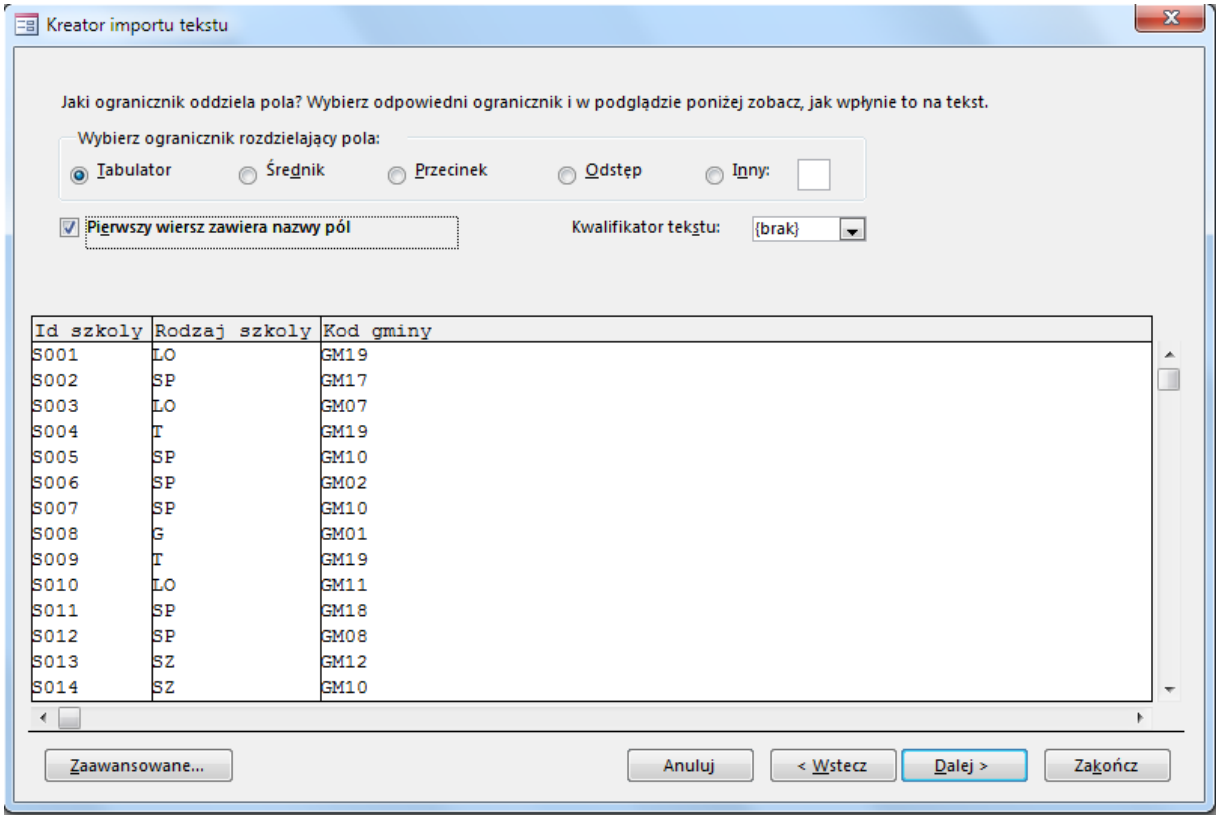
Krok 1 to wywołanie kreatora importu:

Decydujemy, jaki jest format danych.

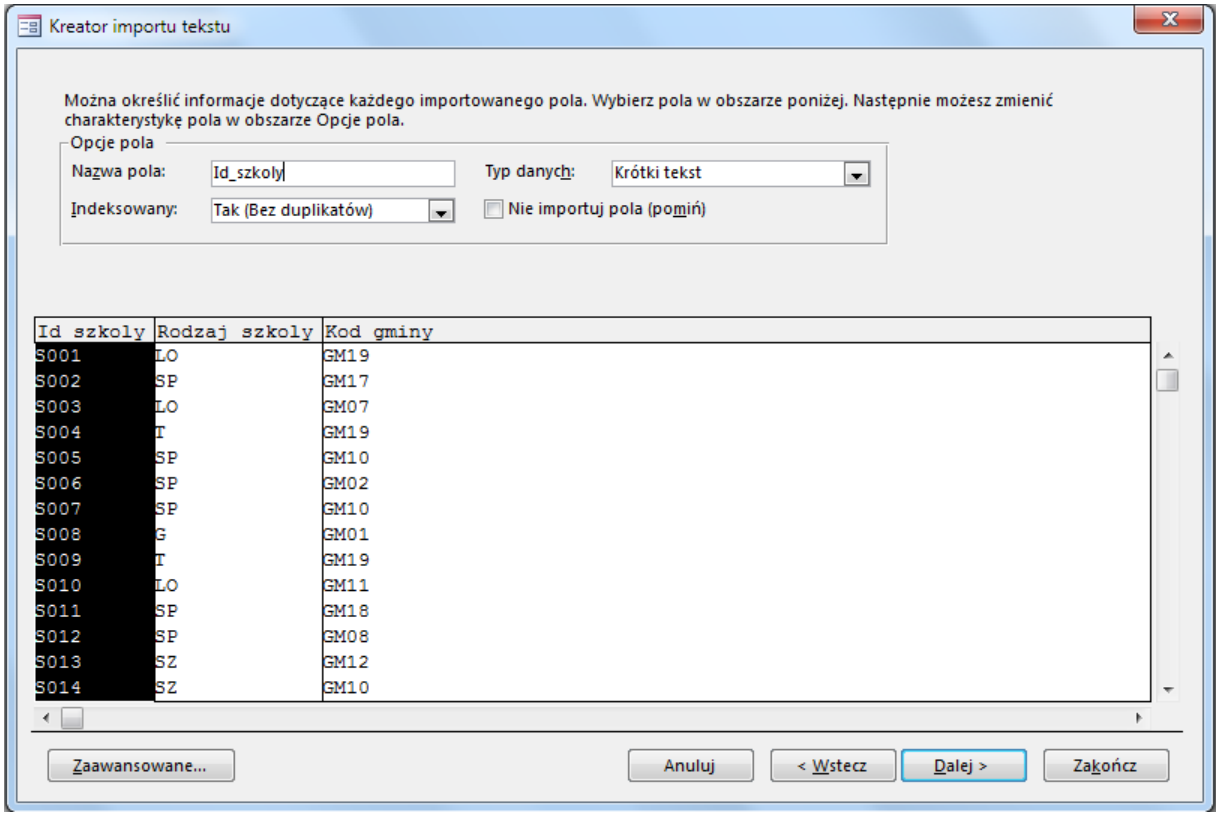


Wybieramy w tym wypadku format ograniczony separatorem (to może być znak tabulacji, przecinek, średnik, odstęp lub inny znak).

W **kroku 2** zaznaczamy, który znak jest separatorem oddzielającym pola, oraz zaznaczamy (lub nie), czy pierwszy wiersz zawiera nazwy pól:



Następnie określamy informacje dotyczące **każdego pola**: jego nazwę, typ przechowywanych danych, to, czy pole będzie indeksowane i czy może zawierać duplikaty.



Nie zawsze wybór typu danych dla pola jest oczywisty na pierwszy rzut oka. Przykład: importując kolumnę zawierającą numery PESEL, wybieramy dla pola typ tekstowy, ponieważ wartości liczbowe numerów PESEL są zbyt duże, aby przechowywać je w polu typu *liczba całkowita* (a nawet *liczba całkowita długa*).

Problem może też stwarzać import danych zawierających zera wiodące. Jeśli zaimportujemy takie dane do pola typu liczbowego, to zera wiodące zostaną usunięte, na przykład dana 0020 po zaimportowaniu będzie mieć wartość 20. Aby utrzymać zera wiodące, taką daną importujemy do pola typu tekstowego.

Kolejna decyzja dotyczy klucza głównego tabeli, a ściślej: czy ma on w ogóle być, a jeżeli tak, to jaki. MS Access domyślnie proponuje dodanie kolumny stanowiącej klucz główny, warto jednak samodzielnie zdecydować, które z już istniejących pól może pełnić rolę klucza. Możemy także zdecydować, że nasza tabela nie będzie miała klucza podstawowego.

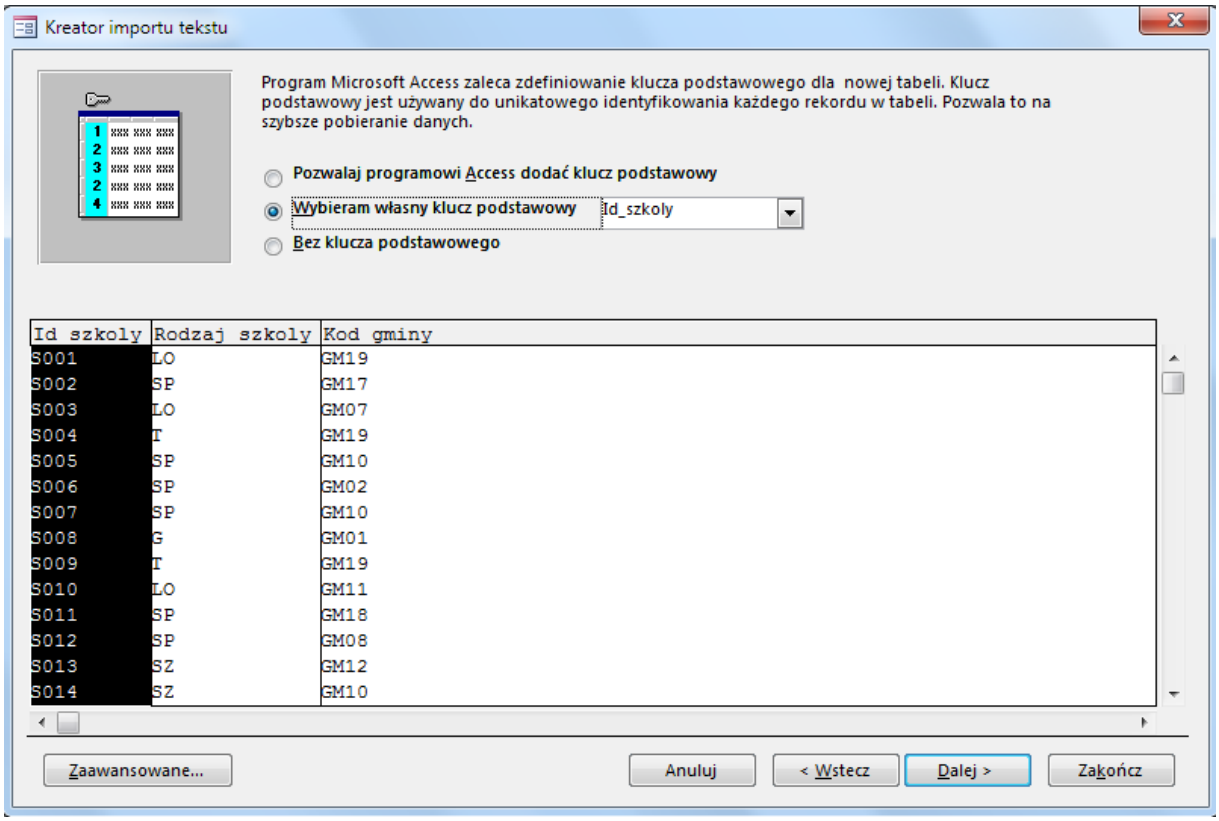
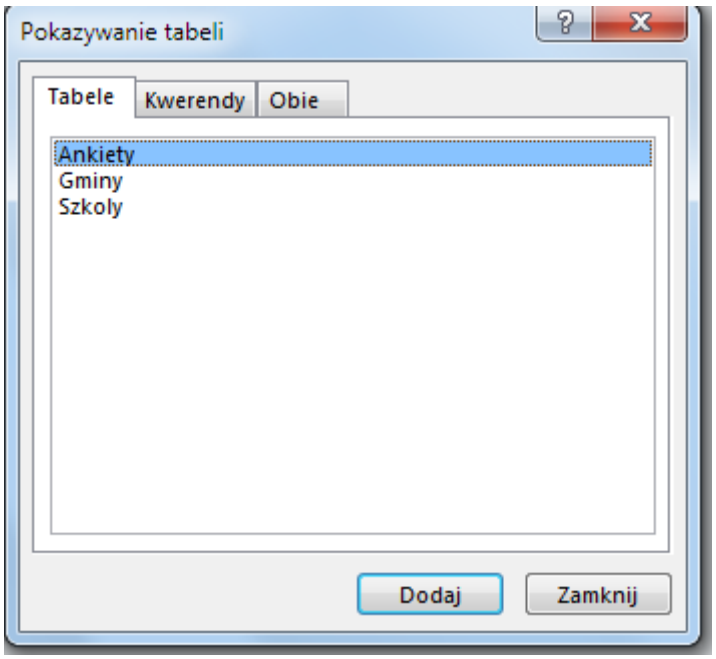
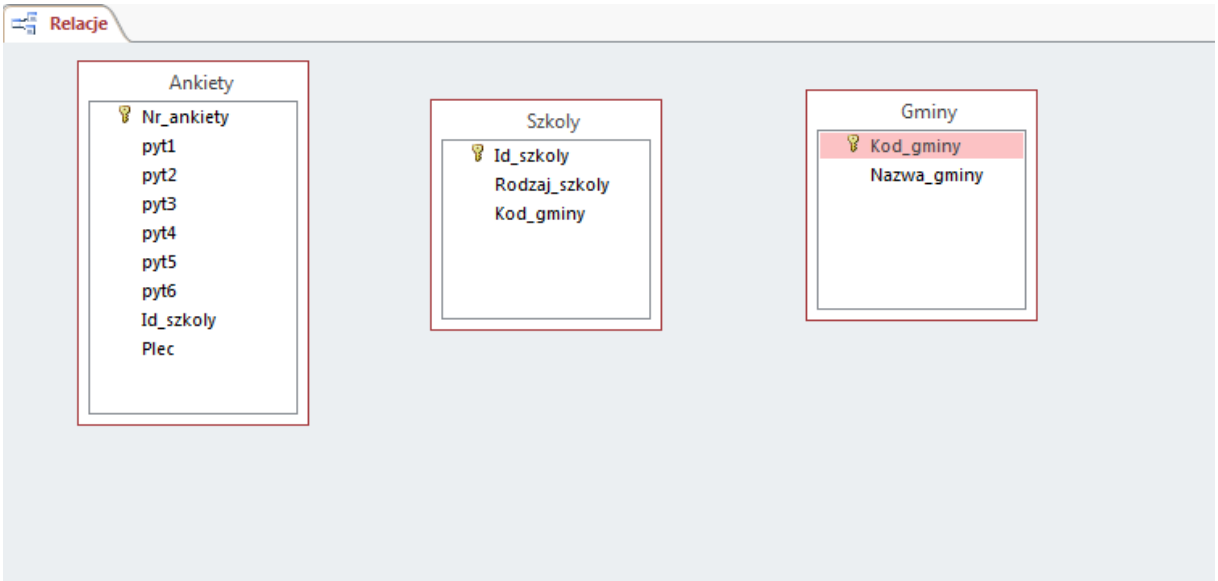


Tabela domyślnie otrzymuje nazwę taką jak nazwa pliku, z którego pochodzą dane, ale możemy tę nazwę zmienić.

W programie Microsoft Access można po zaimportowaniu danych utworzyć wiązania wybranych pól z różnych tabel i zapisać je jako stały element systemu bazodanowego (relacje). Można również tworzyć wiązania tylko na potrzeby konkretnego zapytania (kwerendy) — jest to naturalny sposób programowania w języku SQL.

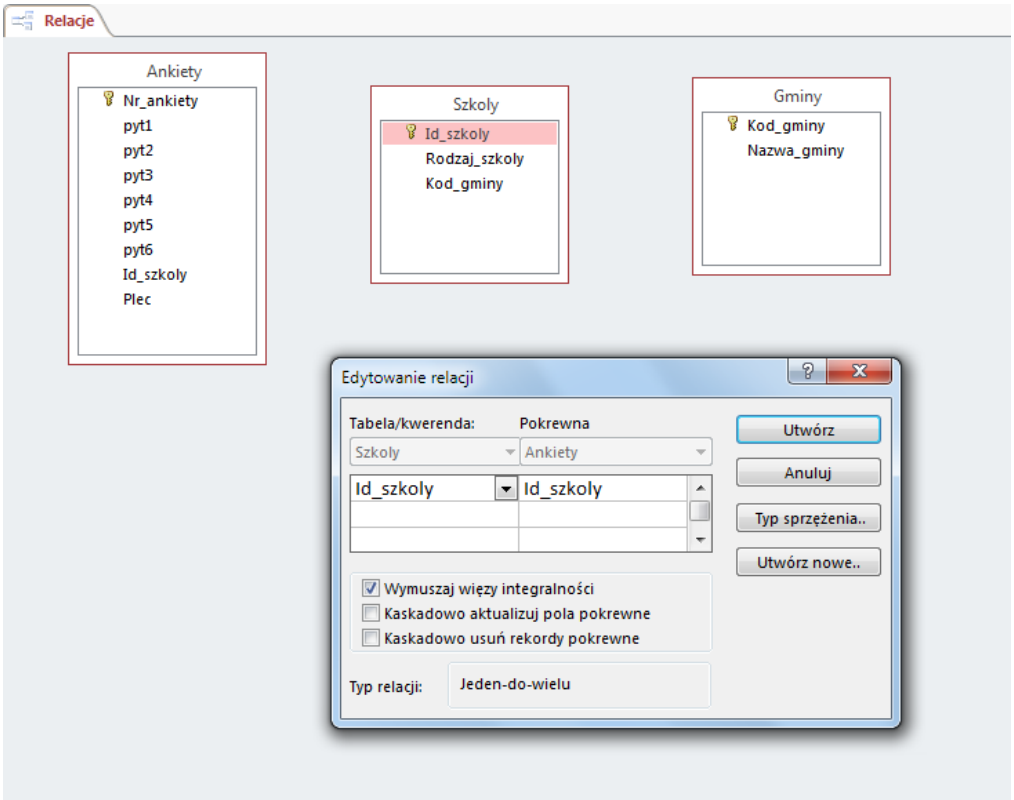
Jeśli zdecydujemy się utworzyć relacje na początku, w programie Microsoft Access wybieramy zakładkę Narzędzia bazy danych/Relacje. Wybieramy te spośród tabel, dla których chcemy stworzyć relacje, czyli na ogół wszystkie tabele w bazie.

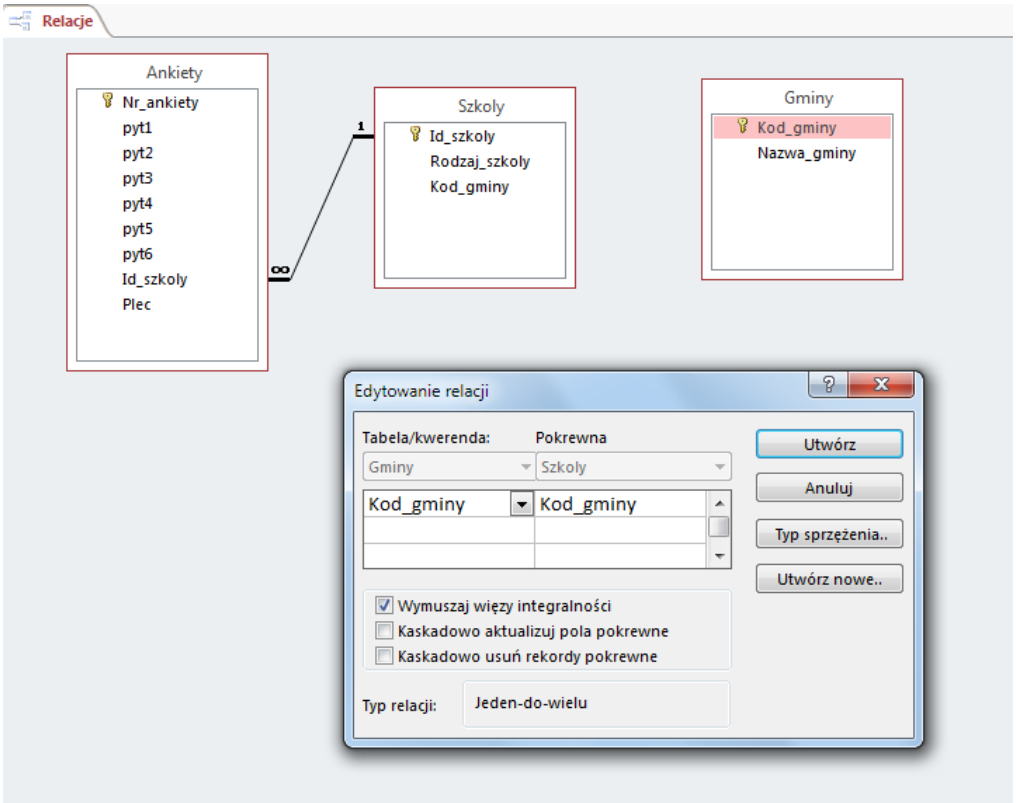


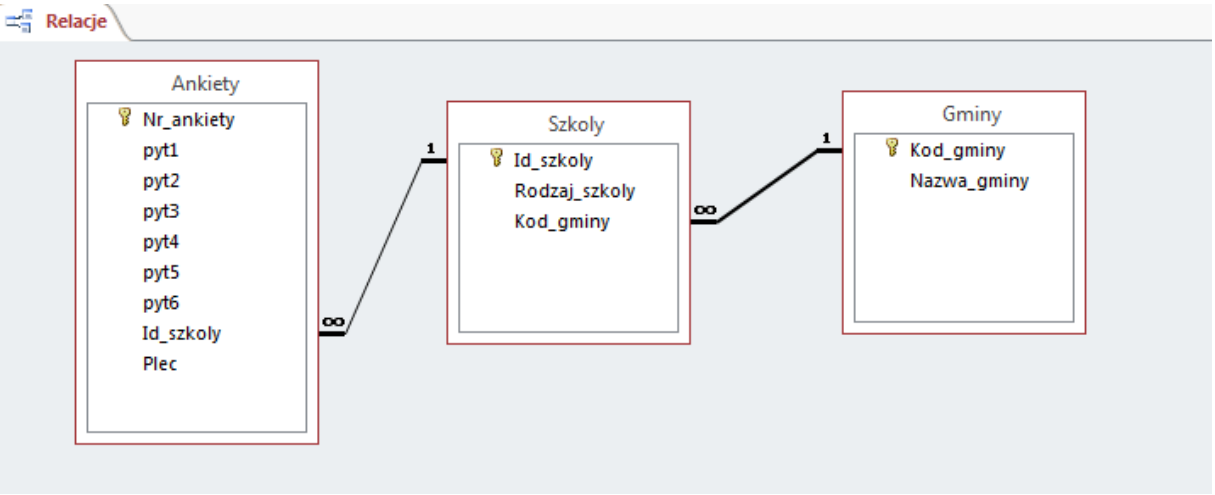


Wiązanie tworzymy, przeciągając myszką wybrane pole z jednej tabeli i „puszczając" je nad wybranym polem drugiej tabeli. Oba pola muszą być tego samego typu, choć mogą mieć różne nazwy. Domyślnie tworzone jest wiązanie typu *wewnętrzne*, które z obu wiązanych tabel udostępnia tylko rekordy pasujące. We właściwościach wiązania można zmienić jego typ na zewnętrzne lewo- lub prawostronne, które będzie udostępniać wszystkie rekordy z jednej tabeli, a z drugiej tabeli tylko rekordy pasujące do połączenia.

W oknie kreatora, obok linii łączącej pola, pojawia się informacja o typie relacji: 1:1 lub 1:∞.







Wczytywanie i wypisywanie danych w zadaniach programistycznych

W języku C++ do odczytu i zapisu danych służą specjalne obiekty zwane *strumieniami*. Taki obiekt powiązany jest ze źródłem danych — w naszym wypadku będą to zwykle pliki na dysku, ale można też obsługiwać strumienie odpowiadające np. urządzeniom zewnętrznym.

Aby użyć w swoim programie strumieni powiązanych z plikami, należy zadeklarować użycie biblioteki *fstream* (z angielskiego *„file-stream”*, czyli „strumienie plikowe”):

```
#include <fstream>
```

W samym programie deklarujemy obiekt, który będzie czytał z konkretnego pliku. Obiekt taki powinien mieć jeden z następujących trzech typów:

- *ifstream* — strumień wejścowy, czyli służący do czytania danych z pliku (*input file stream*)

- *ofstream* — strumień wyjściowy, do zapisywania danych do pliku (*output file stream*)
- *fstream* — strumień, z którego można zarówno czytać dane, jak i zapisywać dane do niego.

Deklaracja obiektu wygląda tak, jak każdej innej zmiennej w C++. Wybierzmy dla niego nazwę „wejscie”:

```
ifstream wejscie;
```

Teraz kojarzymy program z konkretnym plikiem na dysku — niech nazywa się, na przykład, *dane.txt:*

```
wejscie.open(”dane.txt”);
```

Od tej chwili zmienna strumieniowa *wejscie* pobiera kolejne dane z pliku dane.txt. Nadmieńmy jeszcze, że te dwie instrukcje można połączyć w jedną, mającą ten sam skutek:

```
ifstream wejscie(”dane.txt”);
```

Ten obiekt jest typu *ifstream*, czyli służy do czytania danych. Deklaracja obiektu, który odpowiada za zapis danych do pliku wygląda bardzo podobnie:

```
ofstream wyjscie;
wyjscie.open(”wynik.txt”);
```

albo:

```
ofstream wyjscie(”wynik.txt”);
```

Wreszcie, zmienna typu *fstream* może służyć zarówno do czytania, jak i do pisania, trzeba jednak przy otwieraniu pliku podać, do czego w tym momencie zostanie użyta, za pomocą następującej konstrukcji:

```
fstream plik;
plik.open(”plik.txt”, ios::in);
```

jeśli zamierzamy czytać z pliku, lub:

```
plik.open(”plik.txt”, ios::out);
```

jeśli będziemy do niego pisać.

Ważne: deklaracja zmiennej typu *ofstream*, albo *fstream* z parametrem *ios::out* spowodują, że **plik zostanie wyczyszczony**, a nowe dane zastąpią to, co było w nim wcześniej. Aby dopisać nowe dane na koniec istniejącego pliku, trzeba użyć instrukcji:

```
ofstream plik;
plik.open(”wynik.txt”,ios::app);
```

Oczywiście można użyć wariantu z pojedynczą instrukcją:

```
ofstream plik(”wynik.txt”,ios::app);
```

albo użyć *fstream* w miejsce *ofstream*.

Mamy zatem pliki otwarte do odczytu i zapisu — w zadaniach maturalnych najczęściej będziemy potrzebowali jednego pliku do odczytu i jednego do zapisu.

Dane czytamy ze strumieni i piszemy do strumieni za pomocą operatorów << i >>. Jeśli, na przykład, mamy zadeklarowaną zmienną typu *int*:

```
int a;
```

i chcemy odczytać jedną liczbę z pliku, któremu odpowiada zmienna strumieniowa *wejscie*, piszemy:

```
wejscie >> a;
```

Jeśli chcemy odczytać dwie liczby i zapisać je do zmiennych *a* i *b*, możemy napisać:

```
wejscie >> a >> b;
```

co jest skrótem dla dwóch osobnych instrukcji (`wejscie >> a; wejscie >> b;`)

Jeśli, na przykład, w pliku jest 1000 liczb, zapisanych każda w osobnym wierszu, możemy zadeklarować tablicę odpowiedniej długości:

```
int A[1000];
```

a następnie przeczytać wszystkie liczby za pomocą pętli for:

```
for(int i=0; i<1000; i++)
     wejscie >> A[i];
```

Dla przykładu, analogicznie do pliku wyjściowego zapisujemy np. pojedynczą liczbę za pomocą instrukcji:

```
wyjscie << a;
```

Rozwiążmy dla przykładu bardzo proste zadanie: w pliku `dane.txt` znajduje się 1000 wierszy, z których każdy zawiera po dwie liczby, oddzielone pojedynczym odstępem. Należy dla każdej pary liczb policzyć ich sumę i zapisać ją do pliku `wynik.txt`, każdą sumę w osobnym wierszu. Odpowiedni program może wyglądać tak:

```
#include <fstream>

using namespace std;
     // zaznaczamy której przestrzeni nazw będziemy używać

int main()
{
    // deklarujemy tablice A i B długości 4
    int A[4], B[4];

    // zmienna strumieniowa na plik wejsciowy
    ifstream wejscie("dane.txt");

    // zmienna strumieniowa na plik wejsciowy
    ofstream wyjscie("wynik.txt");


    for(int i=0; i<4; i++)
        // wczytaj z pliku dwie liczby i zapisz je do A[i] oraz B[i]
        wejscie >> A[i] >> B[i];


    for(int i=0; i<4; i++)
        // wypisz sumę do pliku wyjściowego
        wyjscie << A[i]+B[i] << endl;


    wejscie.close();
```

```
    wyjscie.close();
}
```

W tym programie użyliśmy dwóch dodatkowych instrukcji. Instrukcja

```
wyjscie << endl;
```

wypisuje do pliku znak końca wiersza (czyli przechodzi do następnego wiersza). Instrukcje

```
wejscie.close();
wyjscie.close();
```

zamykają pliki, czyli informują system operacyjny, że nie będziemy już ich potrzebować. Brak instrukcji zamknięcia pliku może spowodować (choć na ogół nie powoduje) różnego rodzaju problemy (np. niekompletne zapisanie się danych).

Warto zauważyć, że w tym programie nie musieliśmy najpierw czytać wszystkich liczb z pliku wejściowego, a potem wypisywać wszystkich sum. Jak najbardziej możliwe jest wczytanie dwóch liczb, wypisanie ich sumy, a potem zajęcie się kolejnymi dwiema liczbami. W takich sytuacjach nie potrzebujemy nawet tablic do przechowywania danych, co skraca program:

```
#include <fstream>

using namespace std;

int main()
{
    int a, b;
    ifstream wejscie("dane.txt");
    ofstream wyjscie("wynik.txt");
    for(int i=0; i<4; i++)
    {
        // wczytaj z pliku dwie liczby
        wejscie >> a >> b;
        // wypisz sumę do pliku wyjściowego
        wyjscie << a+b << endl;
    }
    wejscie.close();
    wyjscie.close();
}
```

Czasem zamiast liczb chcemy czytać napisy (łańcuchy znaków), które w C++ przechowuje się w zmiennych typu *string*. Wczytywanie i wypisywanie takich danych wygląda prawie identycznie:

```
string S;       // deklaracja zmiennej typu łańcuchowego
wejscie >> S;
```

Warto wiedzieć, że taka instrukcja wczytuje znaki tylko **tak długo, aż napotka pierwszy biały znak** (spację, koniec wiersza, tabulatora). Jeśli w pliku w pierwszym wierszu znajduje się napis:

```
Dzien dobry!
```

to instrukcja

```
wejscie >> S;
```

zapisze do zmiennej S tylko słowo „Dzien”. Gdybyśmy mieli dwie zmienne, S i T, obie typu *string*, to instrukcja:

```
wejscie >> S >> T;
```

zapisałaby do S napis „Dzien”, a do T napis „dobry!”, ignorując spacje.

Jednak przy wyprowadzaniu tej danej do pliku wyjściowego, zapisany zostanie zawsze cały łańcuch. Instrukcja:

```
S = ”Dzien dobry!”
wyjscie << S;
```

umieści w pliku wyjściowym pełny napis „Dzien dobry!”.

Aby więc na przykład przeczytać z pliku wejściowego 200 słów, z których każde zapisane jest w osobnym wierszu, należy wykonać następujący fragment programu:

```
string Tablica[200];
for(int i=0; i<200; i++)
  wejscie >> Tablica[i];
```

Tego samego programu można użyć, jeśli słowa nie są zapisane w osobnych wierszach, ale oddzielone odstępami.

Wczytywanie/wypisywanie liczb i napisów można bez problemu mieszać ze sobą, na przykład instrukcja:

```
string a = ”Basnie z ”;
int b = 1000;
string c = ” i jednej nocy”;

wyjscie << a << b << c;
```

wypisze `Basnie z 1000 i jednej nocy`. Zawartość np. takiego pliku:

```
Zakroczym 2
Wyszogrod 34
Plock 71
Wloclawek 115
```

zostanie prawidłowo odczytana, jeśli wykonamy następujące instrukcje:

```
string A[4];
int B[4];

for(int i=0; i<4; i++)
        wejscie >> A[i] >> B[i];
```

Napisy (nazwy miast) znajdą się w tablicy A, zaś liczby (odległości) — w tablicy B.

Czasem zadanie wymaga pracy na liczbach, które nie są całkowite — w C++ do przechowywania takich liczb służą zmienne typu *float* i *double*. Jeśli na przykład zmiennej *x* przypiszemy wartość 1/7:

```
double x = 1.0 / 7;
```

i użyjemy polecenia

```
wyjscie << x;
```

do pliku wyjściowego zostanie wypisane `0.142857`. W zadaniach maturalnych potrzebujemy jednak najczęściej pewnej ustalonej precyzji zapisu dziesiętnego — na przykład polecenie może brzmieć: „Wypisz liczbę z dokładnością do dwóch miejsc po przecinku”. Aby ustalić precyzję, posługujemy się instrukcją *precision*, używaną w następujący sposób:

```
wyjscie.precision(3);
wyjscie << x;
```

Teraz otrzymujemy wynik 0.143, jednak wypisanie tą samą metodą 1/70 może prowadzić do niespodzianek — fragment kodu:

```
double x = 1.0 / 70;
wyjscie.precision(3);
wyjscie << x;
```

spowoduje wypisanie `0.0143`, a nie `0.014`, ponieważ kompilator, wypisuje trzy cyfry **znaczące**, czyli nie liczy zer bezpośrednio po przecinku. Aby wypisywał on zawsze dokładnie trzy cyfry po przecinku, konieczna jest jeszcze instrukcja *fixed*:

```
wyjscie.precision(3);
wyjscie << fixed;
wyjscie << x;
```

Aby używać instrukcji *precision* i *fixed*, konieczne jest dołączenie nagłówka *iomanip* na początku programu:

```
#include <iomanip>
```

Warto pamiętać, że jeśli nie zmieniamy precyzji, domyślnie użytym zapisem w C++ jest 6 cyfr znaczących.

2.1. Algorytmy w praktyce

Zadanie 58.

Wiązka zadań *Systemy liczbowe*

Centralny ośrodek meteorologiczny planety Cyfrak codziennie w południe rejestruje wskazania zegarów oraz temperaturę w trzech stacjach pogodowych: S1, S2, S3. Zegary w stacjach pogodowych odliczają liczbę godzin, które upłynęły od uruchomienia stacji. W stacji S1 wszystkie wartości (wskazania zegara i temperatury) zapisywane są w systemie binarnym, w stacji S2 — w systemie czwórkowym (czyli systemie pozycyjnym o podstawie 4), a w stacji S3 — w systemie ósemkowym (czyli systemie pozycyjnym o podstawie 8). Temperatury ujemne poprzedzone są znakiem „–”, np. –1101 w systemie dwójkowym oznacza liczbę o zapisie dziesiętnym –13.

Pliki `dane_systemy1.txt`, `dane_systemy2.txt`, `dane_systemy3.txt` zawierają wyniki 1095 kolejnych pomiarów przeprowadzonych w stacjach S1, S2, S3 od czasu ich uruchomienia. Każdy wiersz pliku zawiera wyniki jednego pomiaru: stan zegara i temperaturę. Wartości w wierszach rozdzielone są spacjami.

Przykład

Wiersz opisujący pomiar, w którym zegar wskazuje liczbę 36, a termometr temperaturę –7, wyglądałby następująco:

Plik `dane_systemy1.txt`
```
100100 -111
```
Plik `dane_systemy2.txt`
```
210 -13
```
Plik `dane_systemy3.txt`
```
44 -7
```

Napisz program(-y), który pozwoli rozwiązać poniższe zadania. Odpowiedzi zapisz w pliku `wyniki_systemy.txt`. Odpowiedź do każdego zadania podaj w osobnym wierszu, poprzedzając ją numerem zadania.

58.1.

Dla każdej stacji pogodowej podaj najniższą zarejestrowaną temperaturę, a wszystkie wyniki zapisz w systemie binarnym (dwójkowym).

58.2.

Zgodnie z harmonogramem pomiary wykonywane są co 24 godziny, począwszy od pierwszego pomiaru. Oznacza to, że wyrażone dziesiętnie stany zegarów w kolejnych pomiarach powinny wynosić 12, 12+24=36, 12+2·24=60 itd.

Podaj liczbę pomiarów, w których zarejestrowany stan zegara był niepoprawny jednocześnie we wszystkich stacjach pogodowych.

Przykład

Rozważmy dane, w których pierwsze 3 wiersze pliku ***dane_systemy1.txt*** są następujące:

```
1100 -11
```

```
100100 -111
111101 1
```

Ponieważ zapisane binarnie stany zegara: `1100`, `100100 i 111101` to odpowiednio wartości: 12, 36 i 61, to tylko stany podane w dwóch pierwszych wierszach są poprawne, zaś w trzecim wierszu stan jest nieprawidłowy.

58.3.

Rekordem temperatury dla danej stacji pogodowej nazywać będziemy pomiar temperatury, który jest większy od wszystkich wcześniejszych pomiarów dokonanych w tej stacji.

Przykład

Dla następujących wyników kolejnych pomiarów temperatur dokonanych od pierwszego pomiaru w danej stacji (podanych w zapisie dziesiętnym):

1, -1, 0, 2, 1, 1, 3, 4, 4, 3, 1, 7, 2, 1

rekordami temperatury są wszystkie podkreślone wyniki.

Dniem rekordowym jest dzień, w którym **w co najmniej jednej** stacji pogodowej zarejestrowano rekord temperatury. Podaj liczbę dni rekordowych.

Przykład: przyjmijmy, że — podane w zapisie dziesiętnym — wyniki pomiarów dokonywanych w kolejnych dniach były w trzech stacjach następujące:

Dzień	**S_1**	**S_2**	**S_3**
1	**1**	**0**	**-1**
2	**2**	**1**	-1
3	1	-1	-1
4	0	-2	0
5	1	**2**	**1**

Dla powyższych danych liczba dni rekordowych wynosi: **3**.

58.4.

Oznaczmy kolejne zarejestrowane temperatury w stacji pogodowej S1 przez t_1, t_2, t_3,… Niech r_{ij} oznacza kwadrat różnicy między temperaturami w i-tym i j-tym pomiarze pierwszej stacji pogodowej, $r_{ij}=(t_i - t_j)^2$. *Skokiem temperatury* między i-tym a j-tym pomiarem nazywać będziemy zaokrąglenie w górę do liczby całkowitej ułamka $r_{ij} / |i - j|$.

Przykład

Dla następujących kolejnych pomiarów temperatur (zapisanych dziesiętnie):

3, 5, 4, 7,

skoki temperatur opisuje poniższa tabela

i, j	***t_i, t_j***	***r_{ij}***	***$\|i - j\|$***	**Skok temperatury między *i*-tym a *j*-tym pomiarem**
1, 2	3, 5	$2^2=4$	1	4
1, 3	3, 4	$1^2=1$	2	1
1, 4	3, 7	$4^2=16$	3	6
2, 3	5, 4	$1^2=1$	1	1

2, 4	5, 7	$2^2=4$	2	2
3, 4	4, 7	$3^2=9$	1	9

Podaj największy skok temperatury w stacji pogodowej S1. Wynik podaj w systemie dziesiętnym.

Komentarz

W naszym rozwiązaniu będziemy zamieniać napisy reprezentujące liczby w systemach niedziesiętnych na odpowiednie liczby całkowite (pamiętane w zmiennych typu `int`). Dzięki temu niezbędne obliczenia i porównania wykonywane będą z wykorzystaniem zmiennych liczbowych (a nie napisów). Na koniec, jeśli wymaga tego polecenie w zadaniu, zamienimy liczby uzyskane w wyniku na napisy będące ich reprezentacjami w niedziesiętnym systemie pozycyjnym. Zwróćmy jednak uwagę, że zadania 1, 2 i 3 można rozwiązać bez konieczności zamiany napisów na liczby — o ile wykorzystamy (dość proste) zależności między binarną, czwórkową i ósemkową reprezentacją liczb (na końcu omówimy krótko możliwość takiego rozwiązania).

Kluczowe dla naszego rozwiązania będzie obliczanie wartości liczby zapisanej w systemie niedziesiętnym (i zapamiętywanie jej w zmiennej typu `int`) oraz odwrotnie: przekształcanie liczby przechowywanej w zmiennej typu `int` na napis reprezentujący tę liczbę w systemie o ustalonej podstawie.

Zacznijmy od konwersji liczby zapisanej jako string w systemie o podstawie b na jej wartość zapisaną w zmiennej typu `int`:

```
int zsystemu(string s, int b)
{   int wynik=0, i=0, znak=1;
    if (s[0]=='-'){
        znak=-1;
        i=1;
    }
    int d=s.length();
    for(;i<d;i++){
        wynik=wynik*b+(s[i]-'0');
    }
    return znak*wynik;
}
```

Powyższe rozwiązanie opiera się na następującej obserwacji: jeśli wartość odpowiadająca pierwszym i znakom napisu `s` jest równa `w`, to dopisując na do niej (`i`+1)-szy znak `s[i]`, uzyskamy liczbę `w·b+s[i]`. Na przykład

$$121_{(4)}=25_{(10)}$$

oraz

$$1213_{(4)}=25_{(10)}\cdot 4_{(10)}+3_{(10)}=103_{(10)}.$$

Ponadto wykorzystujemy fakt, że wartość liczbowa cyfry zapamiętanej w `s[i]` jest równa różnicy s[i]-'0'. Musimy też uwzględnić to, że liczba może być poprzedzona znakiem „–", oznaczającym ujemną wartość. Czytelnikowi pozostawiamy prześledzenie, jak ten aspekt został uwzględniony w powyższej funkcji.

Zajmijmy się teraz konwersją reprezentowanej w zmiennej `l` typu `int` liczby na napis reprezentujący jej zapis w systemie o podstawie b. W naszym rozwiązaniu bierzemy pod uwagę, że

- ostatni znak w zapisie `l` w systemie o podstawie `b` jest równy `l%b`;
- pozostałe znaki (znajdujące się na lewo od `l%b`) uzyskamy, znajdując reprezentację `b` liczby $\lfloor$ `l/b` $\rfloor$ w systemie o podstawie ($\lfloor$ `x` $\rfloor$ oznacza tutaj zaokrąglenie liczby `x` w dół do najbliższej liczby całkowitej).

Na przykład dla `l`=57, `b`=4 ostatni znak reprezentacji `l` w systemie o podstawie b jest równy 57%4=1, natomiast pozostałe znaki uzyskujemy jako reprezentację liczby $\lfloor$ 57/4 $\rfloor$=14.

Poniżej podajemy funkcję wykorzystującą powyższe własności:

```
string nasystem(int l, int b){
    string s="";
    char z;
    int znak=1;
    if (l<0) {
        znak=-1; l=-l;
    }
    if (!l) return "0";
    while (l>0){
        z='0'+l%b;
        s=z+s;
        l=l/b;
    }
    if (znak<0) s='-'+s;
    return s;
}
```

Funkcja `nasystem` zwraca napis będący reprezentacją liczby `l` w systemie o podstawie `b`. Zwróćmy uwagę, że operator „+” zastosowany do zmiennej typu string oznacza złączanie (konkatenację napisów), zatem przypisanie `s=z+s` w funkcji `nasystem` dopisuje znak `z` na początku napisu `s`. Uzasadnieniem takiego podstawienia jest to, że kolejne znaki reprezentacji `l` w systemie o podstawie `b` poznajemy „od końca”: najpierw skrajnie prawy, potem drugi od końca itd.

Korzystając z funkcji `zsystemu` (z odpowiednimi parametrami) oraz standardowych metod odczytywania plików tekstowych, wartości odczytów zegara ze stacji pogodowych S1, S2, S3 umieścimy w tablicach `g1`, `g2`, `g3`, a odczyty temperatury — w tablicach `t1`, `t2`, `t3`. Na przykład pomiar temperatury w stacji S2 w dniu 350 umieszczony będzie w t2[349] (tablice w C indeksujemy od zera).

58.1.

Jeżeli dysponujemy wartościami pomiarów przekonwertowanymi do zmiennych typu int, zadanie 1 sprowadza się do wyznaczenia najmniejszej liczby wśród `t1[0],…,t1[1094]`, analogicznie dla tablic `t2` i `t3`. Na koniec wyznaczone minima konwertujemy na reprezentacje binarne za pomocą funkcji `nasystem`: napis będący reprezentacją liczby `x` uzyskujemy, wywołując `nasystem(x,2)`.

58.2.

Zgodnie z treścią zadania musimy wyznaczyć liczbę takich $i \in [0,1094]$, dla których $g1[i] \neq stan$, $g2[i] \neq stan$ oraz $g3[i] \neq stan$, gdzie $stan=12+i \cdot 24$. W poniższym kodzie wartość wynikowa znajdzie się po wykonaniu pętli for w zmiennej `licz`:

```
int licz=0, stan=12;
int ile=ROZM;
for(int i=0;i<1095;i++){
    if (g1[i]!=stan && g2[i]!=stan && g3[i]!=stan)
        licz++;
    stan+=24;
}
```

58.3.

Rozważmy standardowy algorytm wyznaczania największej liczby w ciągu: przeglądamy kolejne liczby z ciągu, pamiętając w zmiennej `max` największą liczbę spośród dotychczas przejrzanych. Liczba rekordów odpowiada wówczas liczbie zmian wartości zmiennej `max`. Należy przy tym pamiętać, że — zgodnie z treścią zadania — rekordem jest też pierwsza wartość w całym ciągu.

Aby rozwiązać zadanie, stosujemy powyższą ideę do ciągów zapamiętanych w tablicach `t1`, `t2` i `t3`, w jednej wspólnej pętli. Aby wyznaczyć liczbę dni, w których w co najmniej jednej stacji wystąpił rekord, tworzymy dodatkowy licznik (zmienna licz), który zwiększamy dla każdego `i` takiego, że `t1[i]` jest rekordem w ciągu `t1` lub `t2[i]` jest rekordem w ciągu `t2`, lub `t3[i]` jest rekordem w ciągu `t3`. Poniżej prezentujemy fragment kodu realizujący opisany powyżej algorytm:

```
int max1=t1[0], max2=t2[0], max3=t3[0];
int licz=1;
int ile=ROZM;
bool czyRekord;
for(int i=1; i<1095; i++){
    czyRekord=false;
    if (t1[i]>max1) { max1=t1[i]; czyRekord=true; }
    if (t2[i]>max2) { max2=t2[i]; czyRekord=true; }
    if (t3[i]>max3) { max3=t3[i]; czyRekord=true; }
    if (czyRekord) licz++;
}
```

58.4.

Jeśli pominiemy kwestię zaokrągleń wartości do liczby całkowitej, zadanie sprowadza się do wyznaczenia największej wartości $(t1[i]-t1[j])^2/(j-i)$ dla $i=0,1,\ldots,1093$ oraz j takiego, że $i<j \leq 1094$. Zadanie takie nietrudno zrealizować przy pomocy dwóch zagnieżdżonych pętli: zewnętrznej dla $i=0,1,\ldots,1093$ i wewnętrznej dla $j=i+1,i+2,\ldots,1094$.

Skoncentrujmy się teraz na zaokrągleniach wyniku. Gdyby wynik miał być zaokrąglany w dół do liczby całkowitej, byłby on w języku C reprezentowany przez wyrażenie

```
(t1[i]–t1[j])*(t1[i]–t1[j]) / (j–i)
```

o ile tablica `t1` zawiera elementy typu `int` (wynika to z faktu, że operator dzielenia / dla argumentów całkowitych zwraca wynik dzielenia całkowitego, czyli zaokrąglenia w dół do liczby całkowitej). Aby wyznaczyć zaokrąglenie wyniku w górę do liczby całkowitej, skorzystamy z następującej własności:

- Niech `r` oznacza zaokrąglenie w dół do liczby całkowitej wyniku dzielenia `x/y`.
- Jeśli `r·y=x`, to `r` jest dokładnym wynikiem dzielenia `x/y`, a zatem `r` to również zaokrąglenie `x/y` w górę do liczby całkowitej.
- Jeśli `r·y<x`, to `r` jest mniejsze od `x/y`, a zatem zaokrąglenie `x/y` w górę do liczby całkowitej jest równe `r+1`.

Poniżej prezentujemy kod znajdujący największy skok temperatury oparty na powyższych obserwacjach:

```
int skok;
int maxS=0;
int ileR, ile=1095, kwadrat;
for(int i=0; i<ile; i++)
    for(int j=i+1; j<ile; j++){
        kwadrat=(t1[i]-t1[j])*(t1[i]-t1[j]);
        skok=kwadrat/(j-i);
        if (skok*(j-i)<kwadrat) skok++;
        if (skok>maxS)
            maxS=skok;
    }
```

Największy dotychczas znaleziony skok temperatury w powyższym fragmencie programu przechowujemy w zmiennej `maxS`; natomiast w zmiennej `skok` zapisujemy skok temperatury między dniami `i` oraz `j`.

Na koniec, zgodnie z zapowiedzią, omówimy krótko zależność między reprezentacjami liczb w systemach o podstawie 2, 4 i 8, dzięki którym zadania 1, 2 i 3 można by rozwiązać bez konwersji napisów na liczby i bez zamiany liczb na napisy oznaczające ich reprezentacje pozycyjne o odpowiedniej podstawie. Ponieważ $4=2^2$, zachodzą zależności:

- reprezentację liczby w systemie czwórkowym można uzyskać z jej reprezentacji w systemie binarnym (czyli o podstawie 2), zamieniając (od końca) pary cyfr na ich czwórkowe reprezentacje;
- reprezentację liczby w systemie binarnym można uzyskać z jej reprezentacji w systemie czwórkowym, zamieniając każdą cyfrę na jej dwucyfrową reprezentację binarną.

Na przykład liczba $1011011_{(2)}$ jest równa $1123_{(4)}$. Wynik uzyskujemy, dzieląc 1011011 na bloki 1, 01, 10, 11, a następnie zapisując czwórkowe reprezentacje tych bloków: 1, 1, 2, 3. Z kolei $1321_{(4)}=1111001_{(2)}$, co uzyskujemy, zamieniając 1, 3, 2, 1 odpowiednio na 01, 11, 10 i 01.

Ponieważ $8=2^3$, analogiczna własność zachodzi dla konwersji między systemem binarnym a systemem ósemkowym, z tą różnicą, że zamiast bloków 2 cyfr rozważamy bloki o długości 3.

Gdyby w zadaniach wystarczyło posługiwać się reprezentacjami liczb w systemie dziesiętnym i systemie o podstawie 8, osoby rozwiązujące zadanie w języku C mogłyby uprościć sobie rozwiązanie, korzystając z formatowania `%x` w instrukcjach `scanf` i `printf`.

Rozwiązanie

58.1.

Min temp w stacji S1: –1011

Min temp w stacji S2: –1001100

Min temp w stacji S3: –1001011

Wartości minimów podane w systemie dziesiętnym:

Min temp w stacji S1: –11

Min temp w stacji S2: –76

Min temp w stacji S3: –75

Wartości minimów podane w systemach, w których zapisane zostały w plikach wejściowych:

Min temp w stacji S1: –1011

Min temp w stacji S2: –1030

Min temp w stacji S3: –113

58.2.

Liczba dni, w których stan zegarów był niepoprawny: 182

Rozwiązanie, w którym zliczane są dni, w których co najmniej jeden zegar podaje błędną wartość (1018)

58.3.

Liczba dni rekordów: 42

Odpowiedź 2, wynikająca ze zliczania dni, w których jednocześnie wystąpił rekord w każdej stacji.

58.4.

Największy skok temperatury: 25

Odpowiedź 24, w której wynik został zaokrąglony w dół a nie w górę.

Zadanie 59.

Wiązka zadań *Ciekawe liczby*

W pliku `liczby.txt` w oddzielnych wierszach znajduje się **1000 różnych liczb**, każda o długości od 2 do 9 cyfr. **Napisz program(-y)**, który da odpowiedzi do poniższych zadań. Odpowiedzi zapisz do pliku `wyniki_liczby.txt`, a każdą odpowiedź poprzedź numerem zadania.

59.1.

Czynnikiem pierwszym danej liczby naturalnej złożonej jest dowolna liczba pierwsza, która dzieli tę liczbę całkowicie. Podaj, ile jest w pliku `liczby.txt` liczb, w których rozkładzie

na czynniki pierwsze występują **dokładnie trzy różne czynniki** (mogą się one powtarzać, z których każdy jest **nieparzysty**.

Przykład

Liczba	**Czynniki pierwsze**	**Czy w rozkładzie występują dokładnie trzy różne nieparzyste czynniki pierwsze?**
32	2, 2, 2, 2, 2	NIE
210	2, 3, 5, 7	NIE
1331	11, 11, 11	NIE
1157625	3, 3, 3, 5, 5, 5, 7, 7, 7	TAK
105	3, 5, 7	TAK
429	3, 11, 13	TAK
1287	3, 3, 11, 13	TAK
3465	3, 3, 5, 7, 11	NIE
255255	3, 5, 7, 11, 13, 17	NIE

59.2.

Podaj, ile jest w pliku `liczby.txt` liczb, dla których suma danej liczby i liczby odwróconej jest liczbą palindromiczną, tzn. jej zapis dziesiętny jest palindromem.

Przykład

Liczba	Liczba odwrócona	Suma	Czy suma jest palindromem?
45	54	99	TAK
471046105	501640174	972686279	TAK
11264	46211	57475	TAK
19	91	110	NIE
8542	2458	11000	NIE

59.3.

Niech $w(n)$ oznacza iloczyn cyfr liczby n. Dla danej liczby n tworzymy ciąg, w którym kolejny element jest iloczynem cyfr występujących w poprzednim elemencie:

$$n_1 = w(n)$$
$$n_2 = w(n_1)$$
$$n_3 = w(n_2)$$
$$\ldots$$

Ciąg kończy się, gdy liczba n_k jest liczbą jednocyfrową. Wówczas **mocą liczby *n*** jest liczba k.

Podaj, ile jest w pliku `liczby.txt` liczb o mocy 1, 2, 3, ..., 8. Dodatkowo podaj minimalną i maksymalną liczbę o mocy równej 1.

Przykład

Liczba 678 ma moc **4**, ponieważ:

$$6 * 7 * 8 = 336$$

$$3 * 3 * 6 = 54$$

$$5 * 4 = 20$$

$$2 * 0 = 0$$

Liczba 1991 ma moc **2**, ponieważ

$$1 * 9 * 9 * 1 = 81$$

$$8 * 1 = 8$$

Komentarz

Przed przystąpieniem do rozwiązania zadania zadeklarujemy odpowiednie zmienne i obiekty, m.in. takie, które będą odpowiadały za zawartość pliku z danymi. Można to zrobić w następujący sposób:

```
#include<fstream>

ifstream fin;
fin.open("liczby.txt");
```

Ponieważ wiemy, ile liczb znajduje się w pliku, do przetwarzania danych (czytania z pliku kolejnych elementów) wystarczy użyć pętli for, na przykład:

```
for(i=0; i<1000; i++)
   {
      fin>>liczba[i];
      ....
   }
```

Warto również utworzyć plik, który będzie przechowywał rozwiązania zadania, np.:

```
ofstream fout;
fout.open("wyniki.txt");
```

Po zakończeniu zadania należy pamiętać o instrukcjach:

```
fin.close();
fout.close();
```

59.1.

W rozwiązaniu zadania 1. wykorzystamy algorytm rozkładu liczby na czynniki pierwsze i zmodyfikujemy go na potrzeby zadania — szukamy tych liczb, które mają tylko nieparzyste czynniki pierwsze oraz mają dokładnie trzy różne czynniki pierwsze.

Jeżeli liczba jest parzysta, możemy ją wykluczyć z naszych rozważań. W pozostałych przypadkach stosujemy znany algorytm rozkładu na czynniki pierwsze z drobną modyfikacją. Zaczynamy od czynnika równego 3 i liczymy, ile różnych czynników pierwszych wystąpiło w liczbie. Za to odpowiedzialna jest zmienna `ile`. Jeżeli wartość tej zmiennej przekroczy 3, wiemy, że liczba ma więcej niż 3 różne czynniki pierwsze nieparzyste i nie wymaga dalszego

sprawdzania. Funkcja zwraca wartość `true` jedynie wtedy, gdy liczba różnych nieparzystych czynników pierwszych jest równa dokładnie 3.

Poniżej przedstawiono przykładową funkcję, która realizuje to zadanie.

```
bool czynniki(int liczba)
{
  int ile=0;
  int czynnik=3;
  if(liczba %2 == 0) return false;
  while (liczba>1)
  {
    if (liczba % czynnik == 0) ile++;
    while (liczba%czynnik==0){
      liczba = liczba/czynnik;
    }
    czynnik=czynnik+2;
    if (ile>3) return false;
  }
  if (ile == 3) return true;
  if (ile <3) return false;
}
```

59.2.

Zadanie można podzielić na dwa etapy: znajdowanie odwrotności danej liczby oraz sprawdzanie, czy liczba jest palindromem. Do rozwiązywania tego zadania można podejść na dwa sposoby. Jeśli wczytywaną liczbę potraktujemy jako napis, to jego odwrócenie będzie bardzo proste, podobnie jak sprawdzenie, czy napis jest palindromem. Problemem dla uczniów może okazać się sumowanie dwóch liczb, które są pamiętane jako napisy.

Drugi sposób polega na przetwarzaniu liczby przechowywanej w zmiennej typu `int`. Aby otrzymać liczbę odwróconą, należy uzyskać z niej kolejne cyfry, które będą tworzyć nową, odwróconą liczbę. Dodatkowo chcemy zapewnić dostęp do kolejnych cyfr liczby poprzez „odcięcie" cyfr liczby, począwszy od cyfry najmniej znaczącej. Można otrzymać taki rezultat dzięki dwóm operacjom: obliczaniu reszty z dzielenia przez 10 (operacja modulo 10) oraz obliczaniu dzielenia całkowitego liczby przez 10. Prosty algorytm odwracania może wyglądać następująco: dopóki nie odcięliśmy wszystkich cyfr liczby (liczba nie jest zerem), pobierz ostatnią cyfrę z liczby, zmodyfikuj liczbę odwróconą, wykorzystując pobraną cyfrę, odetnij ostatnią cyfrę z liczby. Realizuje to poniższa funkcja:

```
int odwroc (int liczba)
{
   int nowa=0;
   while(liczba>0)
   {
      nowa=10*nowa+liczba%10;
      liczba=liczba/10;
   }
   return nowa;
}
```

Teraz wystarczy sprawdzić, czy suma liczby i jej odwróconej postaci tworzą palindrom. Można napisać funkcję, która będzie sprawdzała, czy uzyskana suma jest palindromem, ale możemy też wykorzystać funkcję `odwroc` – palindrom to takie słowo (w naszym przypadku liczba), która czytana od lewej do prawej da ten sam wynik co czytana od prawej do lewej. Wystarczy zatem sprawdzić, czy obliczona suma jest równa sumie odwróconej. Powyższe rozważania obrazuje następujący fragment kodu:

```
odwrocona = odwroc(liczba);
suma=odwrocona+liczba;
if (odwroc(suma)==suma)
   ile++;
```

59.3.

Zadanie wprowadza pojęcie mocy liczby. Dla danych liczb z pliku należy policzyć, ile jest liczb mocy od 1 do 8, oraz podać minimalną i maksymalną liczbę o mocy 1. To zadanie również można podzielić na etapy.

Aby obliczyć moc liczby, trzeba obliczyć iloczyn kolejnych cyfr danej liczby. Należy zauważyć, że obliczony iloczyn staje się nową liczbą, dla której, o ile nie jest to liczba jednocyfrowa, obliczamy ponownie iloczyn cyfr. Wbrew definicji, która mogłaby sugerować wykorzystanie tablicy do pamiętania kolejnych iloczynów, można obliczyć moc liczby „na bieżąco". Na pewno przyda się następująca funkcja `iloczyn_cyfr`:

```
int iloczyn_cyfr(int x)
{
int wynik=1;
    while (x>0)
    {
       wynik=wynik*(x%10);
       x=x/10;
    }
    return wynik;
    }
```

Funkcji `iloczyn_cyfr` będziemy używać do obliczania mocy liczby. Obliczamy iloczyn początkowy, który staje się nową liczbą. Tak długo, jak liczba nie jest jednocyfrowa (jest większa niż 9), wyliczamy iloczyn cyfr liczby, który to iloczyn za każdym razem ponownie staje się liczbą, której iloczyn dalej obliczamy. Każde obliczenie iloczynu cyfr powoduje zwiększenie licznika obliczającego moc liczby o 1. Poniżej przedstawiono funkcję realizującą obliczanie mocy liczby.

```
int moc(int liczba)
{

    int ile=1;

    liczba=iloczyn_cyfr(liczba); //iloczyn poczatkowy
    while (liczba>9)
    {
       liczba=iloczyn_cyfr(liczba);
       ile++;
     }
    return ile;
```

```
}
```

Teraz wystarczy dla danych liczb zliczać, ile jest liczb o mocy od 1 do 8. Do zliczania można użyć tablicy w taki sposób, by indeksy jej komórek odpowiadały liczbom od 1 do 8, zaś wartości tablicy odpowiadały za liczbę wystąpień liczb o danej mocy.

Dodatkowo w przypadku liczby o mocy 1 należy wyznaczyć minimalną i maksymalną liczbę. Ponieważ wiemy, jaki jest zakres danych, możemy sobie uprościć wyszukiwanie, przyjmując jako początkowe minimum maksymalną liczbę, jaka może się pojawić w pliku, zaś jako maksimum — minimalną liczbę, jaka może wystąpić w pliku. Dalej wystarczy zastosować klasyczny algorytm znajdowania minimum (lub maksimum) w nieuporządkowanym ciągu liczb — w sytuacji, gdy liczba będzie miała moc równą 1 sprawdzamy, czy jest ona większa od aktualnego maksimum/ mniejsza od aktualnego minimum, i w razie potrzeby ustalamy nowego kandydata na minimum/maksimum. Oto fragment kodu:

```
int i, liczba, tmp;
int min=999999999, max=10;

int ile=0;
int t[9]={0};

for(i=1; i<=1000; i++)
   {
      fin>>liczba;
      tmp=moc(liczba);
      t[tmp]++;
      if (tmp==1)
      {
         if (liczba<min) min=liczba;
         if (liczba>max) max=liczba;
      }

   }
```

Zadanie 60.

Wiązka zadań *Dzielniki*

W pliku `liczby.txt` danych jest 200 różnych liczb całkowitych z przedziału [2, 1 000 000], każda w osobnym wierszu pliku. Napisz program (lub kilka programów), który poda odpowiedzi do poniższych zadań. Odpowiedzi zapisz do pliku `wyniki.txt`.

60.1.

Policz, ile jest w pliku wejściowym liczb mniejszych niż 1000, oraz podaj dwie takie liczby, które pojawiają się w pliku jako ostatnie (możesz założyć, że będą co najmniej dwie).

60.2.

Wśród liczb występujących w pliku wejściowym znajdź te, które mają dokładnie 18 dzielników naturalnych (wliczając w nie 1 i samą liczbę). Dla każdej znalezionej liczby wypisz, oprócz jej wartości, listę wszystkich jej dzielników, posortowaną rosnąco.

60.3.

Znajdź największą liczbę w pliku, która jest względnie pierwsza ze wszystkimi pozostałymi, czyli taką, która z żadną z pozostałych liczb nie ma wspólnego dzielnika innego niż 1.

Zadanie 61.

Wiązka zadań *Ciągi arytmetyczne*

Ciąg liczb całkowitych nazywamy *ciągiem arytmetycznym*, jeśli różnica między każdymi dwoma kolejnymi jego wyrazami jest identyczna. Ciągiem arytmetycznym jest na przykład ciąg (1, 3, 5, 7, 9). Jest to ciąg o różnicy 2, ponieważ każdy wyraz tego ciągu, poza pierwszym, różni się od poprzedniego wyrazu o 2. Ciąg (17, 22, 27, 32, 37) jest ciągiem arytmetycznym o różnicy 5. W tym zadaniu rozpatrujemy tylko takie ciągi arytmetyczne, które mają dodatnią różnicę oraz co najmniej pięć wyrazów.

W pliku *ciagi.txt* danych jest 100 ciągów składających się z liczb całkowitych dodatnich, nieprzekraczających 1 000 000. Każdy ciąg opisany jest w dwóch wierszach: pierwszy zawiera liczbę wyrazów ciągu (co najmniej 5 i co najwyżej 1000), zaś drugi — kolejne wyrazy ciągu, oddzielone pojedynczymi odstępami. Dla przykładu pierwsze cztery wiersze pliku mają następującą postać:

```
5
1 3 6 7 9
5
17 22 27 32 37
```

Napisz program (lub kilka programów), który wykona podane poniżej polecenia.

61.1.

Podaj, ile spośród podanych w pliku `ciagi.txt` ciągów jest ciągami arytmetycznymi. Znajdź wśród nich ciąg o największej różnicy i oblicz jego różnicę. Liczbę ciągów arytmetycznych oraz największą różnicę zapisz w pliku `wynik1.txt`.

61.2.

Dla każdego podanego ciągu znajdź — jeśli istnieje — największą występującą w nim liczbę, która jest pełnym sześcianem jakiejś liczby naturalnej (w pierwszym z przykładowych ciągów jest to $1 = 1^3$, w drugim — $27 = 3^3$).

Znalezione liczby (po jednej dla każdego ciągu, w którym taka liczba występuje) zapisz w pliku `wynik2.txt`, w kolejności zgodnej z kolejnością ciągów, z których pochodzą.

61.3.

Plik `bledne.txt` ma identyczną strukturę jak `ciagi.txt`, ale zawiera tylko 20 ciągów. Wiadomo jednak, że wszystkie występujące w nim ciągi są ciągami arytmetycznymi z jednym błędem: jeden z wyrazów w każdym ciągu został zastąpiony przez liczbę naturalną nienależącą do ciągu.

Dla każdego ciągu znajdź i wypisz błędny wyraz. Odpowiedzi zapisz w pliku `wynik3.txt`, podając dla każdego ciągu błędną liczbę w osobnym wierszu, w kolejności zgodnej z kolejnością ciągów w pliku wejściowym.

Zadanie 62.

Wiązka zadań *Liczby ósemkowe*

W pliku `liczby1.txt` znajduje się 1000 liczb całkowitych dodatnich, zapisanych ósemkowo, maksymalnie sześciocyfrowych. Każda liczba umieszczona jest w osobnym wierszu.

W pliku `liczby2.txt` znajduje się 1000 liczb całkowitych dodatnich, zapisanych dziesiętnie, maksymalnie sześciocyfrowych. Każda liczba umieszczona jest w osobnym wierszu.

Napisz program(-y), za pomocą którego(-ych) rozwiążesz poniższe zadania. Do oceny oddaj dokument `wyniki.txt` z rozwiązaniami poszczególnych zadań oraz pliki źródłowe programów wykorzystanych do uzyskania rozwiązania.

62.1.

Wyszukaj w pliku `liczby1.txt` dwie liczby, najmniejszą i największą. Podaj wartości tych liczb w zapisie ósemkowym.

62.2.

Znajdź najdłuższy niemalejący ciąg liczb występujących w kolejnych wierszach pliku `liczby2.txt`. Podaj pierwszy element tego ciągu oraz liczbę jego elementów. Możesz założyć, że jest jeden taki ciąg.

Dla przykładowych danych:

23156
1231
1345
1456
1456
897

najdłuższy niemalejący ciąg liczb rozpoczyna się liczbą 1231 i składa się z 4 elementów.

62.3.

Porównaj wartości liczb zapisanych w wierszach o tych samych numerach w plikach `liczby1.txt` i `liczby2.txt`. Podaj liczbę wierszy, w których:

a) liczby mają w obu plikach taką samą wartość;

b) wartość liczby z pliku `liczby1.txt` jest większa od wartości liczby z pliku `liczby2.txt`.

Dla przykładowych danych:

```
liczby1.txt    liczby2.txt
11456          1302
22666          9654
546            499
```

odp. a) 1 wiersz, bo tylko w drugim wierszu liczby mają taką samą wartość: $22666_8 = 9654_{10}$

odp. b) 1 wiersz, bo tylko w pierwszym wierszu wartość liczby w pierwszym pliku jest większa niż odpowiadająca jej wartość w drugim pliku: $11456_8 > 1302_{10}$.

62.4.

Podaj, ile razy w zapisie dziesiętnym wszystkich liczb z pliku `liczby2.txt` występuje cyfra 6 oraz ile razy wystąpiłaby ta cyfra, gdyby te same liczby były zapisane w systemie ósemkowym.

Zadanie 63.

Wiązka zadań *Ciągi zerojedynkowe*

W pliku `ciagi.txt` w oddzielnych wierszach znajduje się **1000 różnych ciągów zerojedynkowych**, każdy o długości od 2 do 18. **Napisz program(-y)**, który pozwoli rozwiązać poniższe zadania. Następnie je rozwiąż, a odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym `wyniki_ciagi.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie.

63.1.

Ciągiem dwucyklicznym będziemy nazywać taki ciąg zerojedynkowy w o długości parzystej, który składa się z dwóch fragmentów w_1 oraz w_2, $w = w_1 w_2$, takich że $w_1 = w_2$. Podaj wszystkie ciągi dwucykliczne zapisane w pliku `ciagi.txt`.

Przykład

Dla zestawu ciągów:

```
100101010100110010101001
11001101001
1001000
11001100
101010011100
110011110011
```

3 podkreślone ciągi są dwucykliczne.

63.2.

Podaj liczbę ciągów z pliku `ciagi.txt`, w których nie występują obok siebie dwie jedynki.

Przykład

Dla zestawu ciągów:

```
101010101001010010010101
11001101001
10001000
101010011100
000011
```

wynikiem jest liczba 2 (w dwóch podkreślonych ciągach dwie jedynki nie występują obok siebie).

63.3.

Liczbą półpierwszą nazywamy taką liczbę, która jest iloczynem dwóch liczb pierwszych. Podaj, ile ciągów z pliku `ciagi.txt` jest reprezentacją binarną liczb półpierwszych. Dodatkowo podaj największą i najmniejszą liczbę półpierwszą w zapisie dziesiętnym.

Przykład

Dla zestawu ciągów:

```
100010
1101001001
1100101
1111111111
10010110000010010010
```

podkreślone ciągi są zapisem binarnym liczb półpierwszych, ponieważ:

$(100010)_2 = 34 = 2 * 17$, więc jest liczbą półpierwszą;

$(1101001001)_2 = 841 = 29 * 29$, więc jest liczbą półpierwszą;

$(1100101)_2 = 101 = 101 * 1$;

$(1111111111)_2 = 1023 = 3 * 11 * 31$;

$(10010110000010010010)_2 = 614546 = 2 * 307273$, więc jest liczbą półpierwszą.

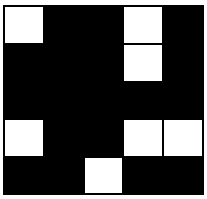
Zadanie 64.

Wiązka zadań *Obrazki*

Bit parzystości ciągu złożonego z zer i jedynek jest równy 0, gdy w ciągu tym występuje parzysta liczba jedynek, w przeciwnym razie bit parzystości jest równy 1.

Czarno-biały obrazek rozmiaru $n \times n$ składa się z n wierszy po n pikseli. Każdy wiersz pikseli reprezentujemy jako ciąg zer i jedynek, każdy biały piksel reprezentujemy przez 0, czarny — przez 1. Na końcu każdego wiersza dodany jest bit parzystości, podobnie pod ostatnim wierszem obrazka dołączony jest wiersz bitów parzystości każdej z n kolumn. **Bitów parzystości nie traktujemy jako części obrazka.**

Przykład: Poniżej podajemy obrazek rozmiaru 5×5 oraz jego reprezentację, wraz z odpowiednimi bitami parzystości (bity parzystości zostały podkreślone):



Obrazek

```
0 1 1 0 1 1
1 1 1 0 1 0
1 1 1 1 1 1
0 1 1 0 0 0
1 1 0 1 1 0
1 1 0 0 0
```

Reprezentacja

Plik `dane_obrazki.txt` składa się z opisu 200 czarno-białych obrazków o rozmiarze 20×20 pikseli. **Sąsiednie obrazki oddzielone są w pliku pustym wierszem.**

Napisz program(-y), który poda odpowiedzi na pytania postawione w poniższych zadaniach. Odpowiedzi zapisz w pliku `wyniki_obrazki.txt`. Odpowiedź do każdego zadania rozpocznij w nowym wierszu, poprzedzając ją numerem zadania.

64.1.

Obrazek nazywamy *rewersem*, jeśli liczba występujących w nim pikseli czarnych jest większa od liczby pikseli białych.

Przykład: W obrazku z powyższego przykładu występuje 18 pikseli czarnych i 7 pikseli białych. Zatem jest on rewersem.

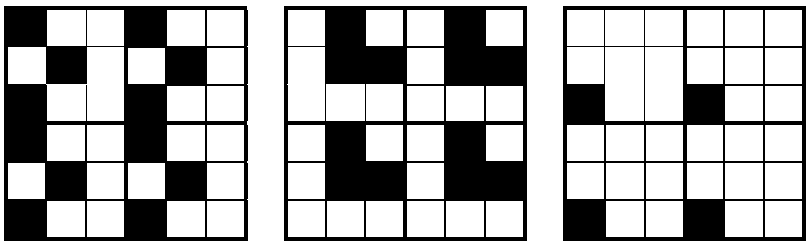
Podaj, ile jest w pliku obrazków, które są rewersami. Podaj też największą liczbę pikseli czarnych występujących w jednym obrazku.

64.2.

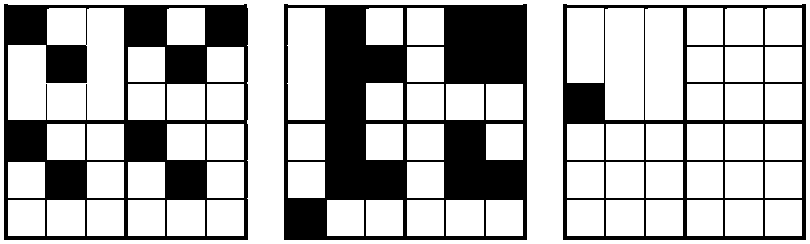
Obrazek rozmiaru $n \times n$ będziemy nazywać *rekurencyjnym*, jeśli n jest parzyste oraz obrazek składa się z 4 kopii tego samego obrazka rozmiaru $\frac{n}{2} \times \frac{n}{2}$.

Przykład

Poniżej podajemy 3 obrazki rozmiaru 6×6, które są rekurencyjne.



Natomiast poniższe obrazki nie są rekurencyjne:



Podaj liczbę obrazków rekurencyjnych w pliku wejściowym. Ponadto podaj opis pierwszego obrazka rekurencyjnego występującego w pliku. W opisie obrazka pomiń bity parzystości (pamiętaj, że obrazek składa się z 20 wierszy po 20 pikseli, które reprezentujemy jako ciąg zer i jedynek).

64.3.

Obrazek nazywamy *poprawnym*, jeśli wszystkie bity parzystości są w nim poprawne (zarówno w wierszach, jak i kolumnach). Obrazek nazywamy *naprawialnym*, jeśli nie jest poprawny, a jednocześnie co najwyżej jeden bit parzystości wiersza i co najwyżej jeden bit parzystości kolumny jest w nim niepoprawny.

Natomiast *nienaprawialnym* nazywamy obrazek, który nie jest poprawny i nie jest naprawialny.

Przykład

Poniżej podajemy przykłady obrazków poprawnych, naprawialnych i nienaprawialnych rozmiaru 5 × 5. Niepoprawne bity parzystości w obrazkach zostały wyróżnione podkreśleniem.

011000	011011	011011	011011	011011
111010	11101**1**	11101**1**	11101**1**	11101**1**
111111	111111	111111	111111	111111
011000	011000	011000	01100**1**	01100**1**
110011	110110	110110	110110	110110
11011	11**1**00	11000	11000	**0**1000
poprawny	naprawialny	naprawialny	nienaprawialny	nienaprawialny

Podaj liczbę obrazków poprawnych, liczbę obrazków naprawialnych oraz liczbę obrazków nienaprawialnych. Ponadto podaj największą liczbę błędnych bitów parzystości występujących w jednym obrazku.

64.4.

W obrazku naprawialnym wystarczy zmienić jedną wartość, aby uzyskać obrazek poprawny. Dokładniej, jeśli niepoprawne są bity parzystości *i*-tego wiersza i *j*-tej kolumny, wystarczy zmienić *j*-ty piksel w *i*-tym wierszu. Jeśli niepoprawny jest dokładnie jeden bit parzystości (wiersza **albo** kolumny), wystarczy zmienić ten bit parzystości.

Przykład

Rozważmy następujące dwa obrazki naprawialne rozmiaru 5 × 5 (niepoprawne bity parzystości w obrazkach zostały podkreślone).

Obraz 1:	Obraz 2:
011011	011011
11**1**01**1**	11101**1**
111111	111111
011000	011000
110110	110110
11**1**00	11000

Zmieniając trzecią jedynkę w drugim wierszu pierwszego obrazka, uzyskamy obrazek poprawny (niepoprawne były bity parzystości trzeciej kolumny i drugiego wiersza). Zmieniając niepoprawny bit parzystości w drugim wierszu drugiego obrazka z 1 na 0, również uzyskamy obrazek poprawny.

Podaj numery obrazków naprawialnych, przyjmując, że numery kolejnych obrazków w pliku to 1, 2, 3 itd. Przy numerze każdego obrazka naprawialnego podaj numer wiersza i kolumny wartości, którą wystarczy zmienić, aby uzyskać obrazek poprawny.

Zadanie 65.

Wiązka zadań *Ułamki*

W pliku `dane_ulamki.txt` znajduje się 1000 par liczb naturalnych dodatnich, mniejszych niż 12 000. Każda para liczb jest zapisana w osobnym wierszu, liczby w wierszu rozdzielone są pojedynczym znakiem odstępu. Parę liczb zapisanych w tym samym wierszu interpretujemy jako ułamek, którego licznikiem jest pierwsza liczba, a mianownikiem — druga liczba.

Przykład

Plik o zawartości

```
3 6
2 3
5 3
2 4
15 5
```

odpowiada ułamkom $\frac{3}{6},\frac{2}{3},\frac{5}{3},\frac{2}{4},\frac{15}{5}$.

Postacią nieskracalną ułamka $\frac{a}{b}$ nazywamy taki ułamek $\frac{x}{y}$, że $\frac{a}{b}=\frac{x}{y}$ oraz x i y są względnie pierwsze (czyli x i y nie mają wspólnego dzielnika większego od 1).

Napisz program(-y), który poda odpowiedzi na pytania postawione w poniższych zadaniach. Odpowiedzi zapisz w pliku `wyniki_ulamki.txt`. Odpowiedź do każdego zadania podaj w osobnym wierszu, poprzedzając ją numerem zadania.

65.1.

Podaj ułamek o minimalnej wartości. Jeśli w pliku występuje więcej niż jeden taki ułamek, to podaj ten spośród nich, który ma najmniejszy mianownik. Twoja odpowiedź powinna zawierać parę liczb oznaczającą licznik i mianownik ułamka.

Przykład

Dla podanego powyżej pliku, opisującego ułamki $\frac{3}{6},\frac{2}{3},\frac{5}{3},\frac{2}{4},\frac{15}{5}$, minimalną wartość mają ułamki $\frac{3}{6},\frac{2}{4}$. Ponieważ $\frac{2}{4}$ ma mniejszy mianownik niż $\frac{3}{6}$, więc odpowiedzią jest para liczb: 2 i 4.

65.2.

Podaj liczbę zapisanych w pliku ułamków, które zostały podane w postaci nieskracalnej.

Przykład

Dla podanego powyżej pliku, opisującego ułamki $\frac{3}{6},\frac{2}{3},\frac{5}{3},\frac{2}{4},\frac{15}{5}$, w postaci nieskracalnej zapisane zostały $\frac{2}{3},\frac{5}{3}$. Natomiast $\frac{3}{6}$ i $\frac{2}{4}$ nie są ułamkami w postaci nieskracalnej (ich liczniki i mianowniki dzielą się odpowiednio przez 3 i 2), podobnie $\frac{15}{5}$ (jego licznik i mianownik dzielą się przez 5). Zatem odpowiedzią jest liczba 2.

65.3.

Zapis danych w postaci nieskracalnej uzyskamy, zamieniając każdy ułamek na jego postać **nieskracalną**. Podaj sumę liczników wszystkich podanych w pliku ułamków, jaką otrzymalibyśmy po sprowadzeniu ułamków do nieskracalnej postaci.

Przykład

Dla podanego powyżej pliku, opisującego ułamki $\frac{3}{6},\frac{2}{3},\frac{5}{3},\frac{2}{4},\frac{15}{5}$, nieskracalne postacie kolejnych ułamków to: $\frac{1}{2},\frac{2}{3},\frac{5}{3},\frac{1}{2},\frac{3}{1}$. Suma liczników tych ułamków to 1+2+5+1+3=12. Zatem odpowiedzią jest 12.

65.4.

Ułamki w pliku zostały tak dobrane, że każdy mianownik jest dzielnikiem liczby $b=2^2\cdot3^2\cdot5^2\cdot7^2\cdot13$, a wartość każdego ułamka jest nie większa niż 3. Oznacza to, że sumę wszystkich ułamków można przedstawić jako ułamek $\frac{a}{b}$, którego mianownikiem jest $b=2^2\cdot3^2\cdot5^2\cdot7^2\cdot13$. Wyznacz sumę ułamków ze wszystkich wierszy i podaj licznik takiego ułamka, że suma ułamków jest równa $\frac{a}{b}$.

Przykład

Dla podanego powyżej pliku, opisującego ułamki 1/2, 2/3, 5/3, 2/4, 15/5, suma ułamków to:

$$\frac{1}{2}+\frac{2}{3}+\frac{5}{3}+\frac{2}{4}+\frac{15}{5}=\frac{\frac{b}{2}+\frac{2b}{3}+\frac{5b}{3}+\frac{2b}{4}+\frac{15b}{5}}{2^2\cdot3^2\cdot5^2\cdot7^2\cdot13}=\frac{3630900}{2^2\cdot3^2\cdot5^2\cdot7^2\cdot13},$$

gdzie $b=2^2\cdot3^2\cdot5^2\cdot7^2\cdot13$. Poprawna odpowiedź wynosi więc 3630900.

Zadanie 66.

Wiązka zadań *Trójki liczb*

W pliku `trojki.txt` w oddzielnych wierszach znajduje się 1000 trójek liczb naturalnych z przedziału od 1 do 550000000. W każdym wierszu są umieszczone trzy liczby rozdzielone pojedynczymi odstępami.

Przykład

3 4 5

12 5 13

12 491 17

11 13 143

15 28 91

Napisz program(-y), który da odpowiedzi do poniższych zadań. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym `wyniki_trojki.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie.

66.1.

Wypisz wszystkie trójki liczb z pliku `trojki.txt`, w których suma cyfr dwóch pierwszych liczb jest równa ostatniej (trzeciej) liczbie.

Przykład

12 491 17

1+2+4+9+1=17

66.2.

Wypisz wszystkie wiersze z pliku `trojki.txt` zawierające takie trzy liczby *a*, *b*, *c*, w których *a* i *b* są liczbami pierwszymi oraz $c = a \cdot b$.

Przykład

11 13 143

11 i 13 są liczbami pierwszymi i $11 \cdot 13 = 143$

66.3.

Wypisz z pliku `trojki.txt` wszystkie pary sąsiadujących ze sobą wierszy, takie że liczby w tych wierszach są długościami boków trójkątów prostokątnych.

Przykład

3 4 5 $3^2+4^2=5^2$

12 5 13 $5^2+12^2=13^2$

66.4.

Podaj, ile jest w pliku `trojki.txt` wierszy, w których znajdują się liczby reprezentujące długości boków trójkąta. Ciąg wierszy nazywamy **trójkątnym**, jeśli liczby w każdym wierszu reprezentują długości boków trójkąta. Podaj długość najdłuższego ciągu trójkątnego w pliku.

Zadanie 67.

Wiązka zadań *Binarny fraktal Fibonacciego*

Ciąg Fibonacciego to ciąg liczb naturalnych określony rekurencyjnie w sposób następujący:

$F_1 = 1, F_2 = 1$, a każdy następny element ciągu jest sumą dwóch poprzednich, czyli:

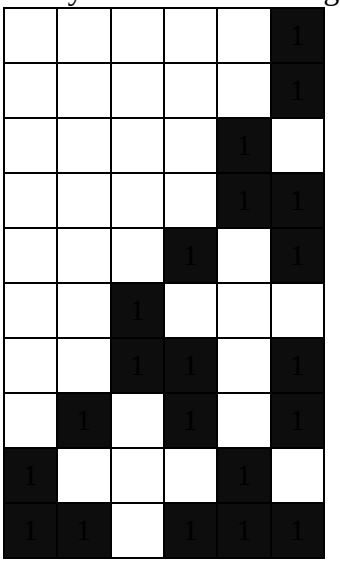
$$F_n = \begin{cases} 1 & dla\ n = 1 \\ 1 & dla\ n = 2 \\ F_{n-1} + F_{n-2} & dla\ n > 2 \end{cases}$$

Binarny fraktal Fibonacciego to dwuwymiarowa tablica zawierająca w kolejnych wierszach binarne zapisy kolejnych liczb Fibonacciego, gdzie każde zero w zapisie zastąpiono białym kwadratem, a każdą jedynkę czarnym kwadratem (p. rysunek). Wszystkie binarne zapisy powinny składać się z jednakowej liczby cyfr, czyli do zapisów krótszych niż najdłuższy należy dodać zera wiodące.

Przykład binarnego fraktala dla pierwszych 10 liczb Fibonacciego:

n	F_n	zapis binarny F_n
1	1	000001
2	1	000001
3	2	000010
4	3	000011
5	5	000101
6	8	001000
7	13	001101
8	21	010101
9	34	100010
10	55	110111

Binarny fraktal Fibonacciego



Napisz program komputerowy, za pomocą którego uzyskasz odpowiedzi do poniższych zadań. Rysunek fraktala (zadanie nr 3) wykonaj, wykorzystując dostępne narzędzia informatyczne. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki.txt`, natomiast rysunek fraktala w pliku `fraktal.xxx`, gdzie `xxx` oznacza rozszerzenie pliku, w którym zapisany jest obraz fraktala.

67.1.

Podaj wartości F_{10}, F_{20}, F_{30}, F_{40}. Zapisz każdą z liczb w osobnym wierszu.

67.2.

Znajdź wszystkie liczby pierwsze wśród liczb F_1, F_2, … , F_{40}. Zapisz każdą z liczb w osobnym wierszu.

67.3.

Dla pierwszych 40 liczb Fibonacciego utwórz binarny fraktal Fibonacciego:

- Wypisz reprezentację binarną wszystkich liczb Fibonacciego od F_1 do F_{40}.
- Wyrównaj długości reprezentacji binarnych wszystkich liczb Fibonacciego od F_1 do F_{40} i na ich podstawie sporządź obraz binarnego fraktala Fibonacciego.

67.4.

Podaj w zapisie binarnym wyrazy ciągu Fibonacciego z zakresu od F_1 do F_{40}, które w tym zapisie mają dokładnie 6 jedynek.

Zadanie 68.

Wiązka zadań *Napisy — anagramy*

Dwa napisy *a* i *b* są swoimi **anagramami**, jeżeli napis *a* (napis *b*) można zapisać za pomocą liter występujących w napisie *b* (napisie *a*), wykorzystując **wszystkie** jego litery.

W pliku `dane_napisy.txt` znajduje się 1000 par napisów, z których każdy jest długości od 2 do 20 znaków, składających się z wielkich liter: A, B, C, D, E, F, G, H, I, J. Każda para napisów jest zapisana w osobnym wierszu, a napisy oddzielone są pojedynczym znakiem odstępu.

Przykład

```
AIHAHGHBEAFJAJDI HGIHFEHHJGBCBGD
FBJHCFFGADD EHADJAJBJBEGD
JHGHADJ AGFEHHEHIAEJFC
EJJHFHIGCEBDAIB DCAFFDICGBEAHAEJ
FBAEEGICHFFACICIGB EEHAHHCABHDHGDFFEGD
```

Napisz program(-y), który poda odpowiedzi dla następujących zadań. Odpowiedzi zapisz w pliku `wyniki_anagramy.txt`. Odpowiedź do każdego zadania podaj w osobnym wierszu, poprzedzając ją identyfikatorem zadania.

68.1.

Napis nazywamy **jednolitym**, jeżeli wszystkie jego litery są takie same. Przykładem takiego napisu jest *AAAA*. Podaj liczbę wierszy zawierających parę napisów jednolitych, które są wzajemnie swoimi anagramami.

Przykład

Dla pliku zawierającego następujące dane:

```
AAAA AAAA
AHHAH AHHAH
AAAA AAAAAAA
BBBBBBB BABBAB
CCCCC CCCCC
```

wynikiem jest liczba 2 (pierwszy i ostatni wiersz). Zwróć uwagę, że napisy w trzecim wierszu są napisami jednolitymi, ale nie są wzajemnie swoimi anagramami.

68.2.

Podaj liczbę wierszy, które zawierają napisy będące wzajemnie swoimi **anagramami**.

Przykład

Dla pliku zawierającego następujące dane:

```
BBBAAB BBBABA
AAAA AAAAA
AHHAH AHHAH
BBABBABB BBBABB
BABABB CACACC
```

wnikiem jest liczba 2 (pierwszy i trzeci wiersz).

68.3.

Podaj największą liczbę *k* taką, że w pliku znajduje się *k* napisów, z których każde dwa są wzajemnie swoimi anagramami.

Przykład

Dla pliku zawierającego następujące dane:

```
BABABB BBBABA
AAAA AAAA
AHHAH AHHAH
BBABBABB BABBAB
BBAABB CCCCC
```

wnikiem jest liczba 4 (*BABABB BBBABA BABBAB BBAABB*).

Zadanie 69.

Wiązka zadań *Geny*

Informację genetyczną (genotyp) każdego osobnika z galaktyki Madgen opisuje słowo (napis), w którym występują litery ze zbioru {A, B, C, D, E}. Obowiązują przy tym następujące zasady:

1. Organizmy żyjące na Madgen tworzą gatunki g_1, g_2, g_3, …, gdzie g_i to zbiór osobników o długości genotypu równej *i*.
2. W skład genotypu mogą wchodzić geny. Pierwszy gen rozpoczyna się pierwszą występującą w genotypie sekwencją AA, a kończy się najbliższą napotkaną po niej sekwencją BB. Każdy kolejny gen rozpoczyna się pierwszą sekwencją AA, występującą za końcem poprzedniego genu, i analogicznie kończy się najbliższą napotkaną sekwencją BB.
3. Geny nazywamy częścią kodującą genotypu, pozostałe fragmenty tworzą część niekodującą.

Przykład 1.

Genotyp AACDBABBBCDAABCBBAAE

zawiera geny AACDBABB oraz AABCBB. Zwróćmy uwagę, że:

- ciąg AA występujący za genem AABCBB nie jest początkiem genu, ponieważ nie występuje za nim ciąg BB kończący gen;
- część kodująca genotypu AACDBABBBCD***AABCBB***AAE jest równa AACDBABB***AABCBB***.

Przykład 2.

Genotyp AADBAADDDDEEEBBEE

zawiera gen **AA**DB**AA**DDDDEEEBB. Zwróćmy uwagę, że:

- pierwsze pojawienie się ciągu AA determinuje początek genu, dlatego w powyższym genotypie występuje gen AADBAADDDDEEEBB, a nie gen AADDDDEEEBB.

Plik `dane_gen.txt` zawiera genotypy 1000 osobników z galaktyki Madgen. Każdy wiersz pliku zawiera genotyp jednego osobnika o długości nie większej niż 500 znaków.

Przykład

```
ABAEACBAADAACAABBABCDA
ABAEACBADEADACACABBABCDA
```

Napisz program(-y), który poda odpowiedzi na pytania postawione w poniższych zadaniach. Odpowiedzi zapisz w pliku `wyniki_gen.txt`. Odpowiedź do każdego zadania rozpocznij w nowym wierszu, poprzedzając ją numerem zadania.

69.1.

Podaj liczbę wszystkich gatunków, których genotypy zapisane są w pliku `dane_gen.txt`. Podaj największą liczbę osobników reprezentujących ten sam gatunek.

69.2.

Występowanie w jakimkolwiek **genie** ciągu **`BCDDC`** oznacza mutację powodującą małą odporność osobnika na zmęczenie. Podaj, ile osobników spośród tych, których genotypy zapisane są w pliku, ma tę mutacje.

Przykład

Osobnik o genie `AAC`**`BCDDC`**`BBACDE` ma mutację `BCDDC` (ciąg `BCDDC` występuje w obrębie podkreślonego genu), natomiast osobnik o genie `C`**`BCDDC`**`BBAACDEBB` nie ma tej mutacji, gdyż występujący ciąg `BCDDC` nie jest ulokowany w żadnym genie.

69.3.

Wyznacz i podaj największą liczbę genów występujących u jednego osobnika. Podaj też największą długość genu zapisanego w całym pliku.

Przykład

Rozważmy plik składający się z genotypów:

```
EAABCDBBDCBAAE
EAABCDBBDCBAAEBCDBBEE
EAABCDBBECAAB
```

Pierwszy osobnik ma jeden gen (`AABCDBB`), drugi ma dwa geny (`AABCDBB` i `AAEBCDBB`), a trzeci osobnik ma jeden gen (`AABCDBB`). Zatem największa liczba genów u jednego osobnika wynosi 2, a największa długość genu to 8 (gen `AAEBCDBB` ma tę długość).

69.4.

Genotyp odczytywany z materiału biologicznego może być odkodowany w kierunku od strony lewej do prawej lub odwrotnie: od strony prawej do lewej. Genotyp nazywać będziemy *odpornym*, jeśli czytany od strony lewej do prawej oraz od strony prawej do lewej ma dokładnie taką samą część kodującą. Natomiast genotyp *silnie odporny* to taki, który czytany od strony lewej do prawej oraz od strony prawej do lewej daje dokładnie ten sam napis. (Inaczej mówiąc, genotyp jest silnie odporny, gdy jest palindromem).

Przykład

Rozważmy genotypy:

```
EAABCDBBDCBAAE
EAABCDBBDCBAAEBCDEE
EAABCDBBECAAB
```

Genotyp `EAABCDBBDCBAAE` jest silnie odporny (jest palindromem). Genotyp `EAABCDBBDCBAAEBCDEE` nie jest silnie odporny (nie jest palindromem), ale jest odporny, gdyż czytany od strony lewej do prawej, jak i od strony prawej do lewej ma taką samą część kodującą: `AABCDBB`. Natomiast genotyp `EAABCDBBECAAB` nie jest silnie odporny (nie jest palindromem), nie jest też odporny, gdyż czytany od strony lewej do prawej daje część kodującą `AABCDBB`, a czytany od strony prawej do lewej ma część kodującą równą `AACEBB`.

Wyznacz liczbę genotypów odpornych oraz liczbę genotypów silnie odpornych.

Zadanie 70.

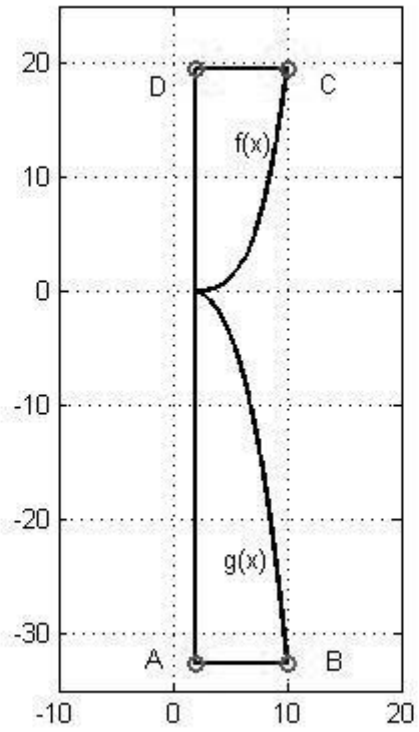
Wiązka zadań *Zasłona*

Pani Binarna dostała zlecenie na uszycie zasłony. Na rysunku poniżej przedstawiono zasłonę, która jest ograniczona:

- od góry prostą $y = 19\frac{61}{125}$,
- od dołu prostą $y = -32\frac{2}{3}$,
- z lewej strony prostą $x = 2$,
- z prawej strony dwoma krzywymi: $f(x) = \frac{x^4}{500} - \frac{x^2}{200} - \frac{3}{250}$ oraz $g(x) = -\frac{x^3}{30} + \frac{x}{20} + \frac{1}{6}$.

Uwaga: Zauważ, że $f(10) = 19\frac{61}{125}$, zaś $g(10) = -32\frac{2}{3}$.

Rysunek pomocniczy:



Korzystając z dostępnych narzędzi informatycznych, wykonaj poniższe zadania. Odpowiedzi do nich umieść w pliku `zadanie_zaslona.txt`. Każda odpowiedź powinna być poprzedzona numerem je oznaczającym.

70.1.

Pani Binarna zakupiła tyle materiału, ile wynosi pole prostokąta ABCD, w którym mieści się zasłona. Oblicz, jaka będzie powierzchnia materiału pozostałego po wykrojeniu zasłony. Wynik podaj z dokładnością do 1/1000.

70.2.

Pani Binarna zamierza obszyć taśmą zasłonę ze wszystkich czterech stron, w tym celu chce wyznaczyć obwód zasłony. Część obwodu ograniczoną wykresem funkcji $f(x)$ szacujemy w następujący sposób: Odcinek [2,10] dzielimy na 1000 równych części, których prawe końce oznaczamy przez $x_1,\ldots,x_{1000}$. Długość krzywej odpowiadającej wykresowi $f(x)$ na przedziale [2,10] przybliżamy długością łamanej łączącej punkty $(2, f(2))$, $(x_1, f(x_1))$, $(x_2, f(x_2))$ itd. aż do $(x_{1000}, f(x_{1000}))$. Analogicznie wyznaczamy część obwodu ograniczoną przez $g(x)$.

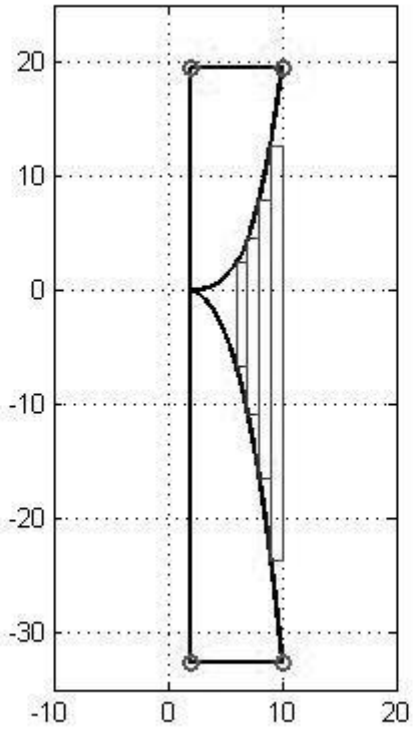
Stosując powyższą metodę wyznaczania obwodu, oblicz długość taśmy, jaką musi zakupić pani Binarna, zakładając, że w sprzedaży jest tylko taśma o długościach będących wielokrotnością jednego metra.

70.3.

Pani Binarna postanowiła wykorzystać pozostały fragment materiału i wyciąć z niego pasy o szerokości 0,25 m i o bokach równoległych do osi układu współrzędnych. Podaj sumę długości pasów, które można wyciąć z pozostałego fragmentu materiału. Załóż, że długość każ-

dego wyciętego pasa jest liczbą całkowitą oraz że pani Binarna zaczyna wycinać pasy od prawej strony materiału.

Rysunek pomocniczy:



Zadanie 71.

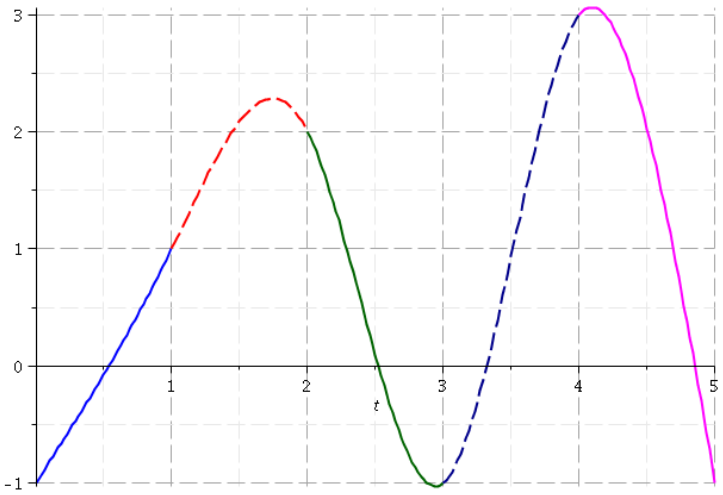
Wiązka zadań *Funkcja*

Wykres funkcji *f* złożony jest z pięciu fragmentów:

$$f(x) = \begin{cases} f_1(x), & x \in [0,1), \\ f_2(x), & x \in [1,2), \\ f_3(x), & x \in [2,3), \\ f_4(x), & x \in [3,4), \\ f_5(x), & x \in [4,5), \end{cases}$$

gdzie każda z funkcji $f_i(x)$ jest wielomianem stopnia trzeciego. W pliku `funkcja.txt` zapisane są współczynniki postaci ogólnej wielomianów $f_i(x)(i = 1,2,\dots,5)$; w *i*-tym wierszu pliku zapisane są cztery liczby rzeczywiste: a_0, a_1, a_2, a_3 (oddzielone pojedynczym odstępem), dla których $f_i(x) = a_0 + a_1x + a_2x^2 + a_3x^3$.

Poniższy rysunek przedstawia wykres funkcji f.



Napisz program, który da odpowiedzi do poniższych zadań. Zapisz je w pliku tekstowym `wyniki_funkcja.txt`. Wyniki do każdego zadania poprzedź numerem je oznaczającym. Wszystkie wyniki należy wypisać, stosując zaokrąglenie do podanej liczby cyfr po przecinku.

71.1.

Podaj wartość $f(1.5)$ z dokładnością do 5 cyfr po przecinku.

71.2.

Znajdź wartość $x \in [0,5)$, dla której wartość $f(x)$ jest największa.

Jako wynik podaj wartość x z dokładnością do trzech, a wartość $f(x)$ z dokładnością do 5 cyfr po przecinku.

71.3.

Znajdź wszystkie miejsca zerowe funkcji f w przedziale $[0,5)$. Odpowiedzi podaj z dokładnością do 5 cyfr po przecinku.

Zadanie 72.
Wiązka zadań *Podobne napisy*

W pliku `napisy.txt` znajduje się 200 wierszy, z których każdy zawiera dwa napisy o długości od 1 do 50 znaków, oddzielone pojedynczym odstępem. Napisy składają się wyłącznie z małych liter alfabetu angielskiego.

Napisz program (lub kilka programów), który pozwoli rozwiązać poniższe zadania. Odpowiedzi zapisz w pliku `wyniki.txt`.

72.1.

Oblicz, w ilu wierszach jeden (którykolwiek) z napisów jest przynajmniej trzy razy dłuższy od drugiego. Jako odpowiedź wypisz liczbę takich wierszy oraz parę napisów z pierwszego z nich.

72.2.

Znajdź (i wypisz) wszystkie takie wiersze pliku, w których drugi napis da się otrzymać z pierwszego przez dopisanie na jego końcu pewnej dodatniej liczby liter (na przykład *kot*

i *kotara*). Dla każdego wiersza podaj oba znajdujące się w nim napisy, a osobno wypisz litery, które należy dopisać.

72.3.

Niektóre z podanych par napisów mają identyczne zakończenia (na przykład *komputer* i *krater*). Znajdź i wypisz największą możliwą długość takiego zakończenia, a także wszystkie pary napisów w wierszach, które osiągają tę maksymalną długość.

Zadanie 73.
Wiązka zadań *Statystyki tekstu*

W pliku `tekst.txt` dany jest tekst złożony ze słów pisanych wielkimi literami alfabetu angielskiego. Słów jest 1876, oddzielone są one pojedynczymi odstępami, a inne znaki poza literami i spacjami w tekście nie występują. Napisz program(-y), który poda odpowiedzi do poniższych zadań. Odpowiedzi zapisz w pliku `wyniki.txt`.

73.1.

Oblicz, ile jest w tekście słów, w których występują dwie kolejne takie same litery.

73.2.

Sporządź statystykę częstotliwości występowania liter w tekście: dla każdej litery podaj liczbę jej wystąpień we wszystkich słowach tekstu oraz jej procentowy udział wśród wystąpień wszystkich liter w tekście(do statystyki nie wliczaj spacji). Odpowiedź zapisz w następującej postaci:

```
A: 632 (7.56%)
B: 196 (2.34%)
...
```

Wartości procentowe podaj w zaokrągleniu do dwóch miejsc po przecinku.

73.3.

Wśród słów w tekście policz długość najdłuższego podsłowa (fragmentu złożonego z kolejnych liter) złożonego z samych spółgłosek. Pamiętaj, że samogłoski to: A, E, I, O, U oraz Y, zaś pozostałe litery są spółgłoskami.

Podaj długość najdłuższego takiego podsłowa, liczbę słów, które zawierają podsłowo o takiej długości, oraz pierwsze z nich, które występuje w pliku `tekst.txt`.

Zadanie 74.
Wiązka zadań *Hasła*

W pliku `hasla.txt` danych jest 200 haseł użytkowników pewnego systemu. Każdy użytkownik posiada jedno hasło (każde zapisane jest w osobnym wierszu), które zawiera od 1 do 20 znaków alfanumerycznych, tzn. cyfr od 0 do 9 lub liter alfabetu łacińskiego (małych lub dużych). Polityka bezpieczeństwa systemu wymaga, aby hasła były odpowiednio skomplikowane i nie powtarzały się.

Poniżej podano pierwsze pięć haseł zapisanych w pliku `hasla.txt`:

```
ZXUhkPLcjKo
ikfLDegQXj
8Y7JGYXXR5
603624722555
50q4252ax5
```

Napisz program, który da odpowiedzi do poniższych zadań. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym `wyniki_hasla.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie.

74.1.

Podaj liczbę haseł złożonych jedynie ze znaków numerycznych, tzn. cyfr od 0 do 9.

74.2.

Wypisz hasła, które zostały użyte przez co najmniej dwóch różnych użytkowników, tzn. występujące w dwóch różnych wierszach. Hasła wypisz (bez powtórzeń) w kolejności leksykograficznej.

74.3.

Podaj liczbę użytkowników posiadających hasła, w których występuje fragment złożony z czterech kolejnych znaków ASCII (w dowolnej kolejności).

Przykłady haseł zawierających taki fragment to:

```
A5mnpoR89cd
A5876RRcg
As45FGHFEk90nba
```

75.4.

Podaj liczbę haseł, które spełniają jednocześnie poniższe warunki:

- hasło zawiera co najmniej jeden znak numeryczny, tzn. cyfrę od 0 do 9,
- hasło zawiera co najmniej jedną małą literę,
- hasło zawiera co najmniej jedną dużą literę.

Zadanie 75.

Wiązka zadań *Szyfr afiniczny*

Dany jest tekst złożony ze słów zbudowanych z małych liter alfabetu angielskiego. Metoda szyfrowania afinicznego — dla której *kluczem szyfrującym* są dwie liczby całkowite A i B — polega na wykonaniu kolejno następujących operacji:

- zastąpienia kolejnych liter alfabetu liczbami od 0 do 25: 'a' przez 0, 'b' przez 1, 'c' przez 2 itd. według przyporządkowania przedstawionego w poniższej tabeli:

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

- pomnożenia liczby odpowiadającej każdej literze przez A i dodania otrzymanego wyniku do B,

- zamiany otrzymanych liczb z powrotem na litery; jeśli liczba jest większa niż 25, bierze się jej resztę z dzielenia przez 26.

Parametry klucza, czyli liczby A i B, powinny być liczbami całkowitymi z przedziału [0, 25].

Dla przykładu, jeśli kluczem szyfrującym jest (3, 7), czyli A = 3, zaś B = 7, to litera 'n' jest najpierw zastępowana liczbą 13. Po pomnożeniu jej przez A i dodaniu B otrzymujemy wynik równy 46. W następnym kroku otrzymujemy literę o numerze 46 – 26 = 20, czyli 'u'.

Okazuje się, że do odszyfrowania szyfru afinicznego można zastosować tę samą metodę, być może z innym kluczem. Na przykład, jeśli napis zaszyfrujemy kluczem (3, 7), to aby go odszyfrować, stosujemy ten sam algorytm z kluczem (9, 15). Dla przykładu, deszyfrując literę 'u' z kluczem (9, 15), otrzymamy liczbę 20 * 9 + 15 = 195, czyli literę 'n', jako że 195 mod 26 = 13. Klucz (9,15) jest wówczas *kluczem deszyfrującym* dla klucza (3,7).

Napisz program(y), który poda odpowiedzi do poniższych zadań. Odpowiedzi zapisz do pliku `wyniki.txt`.

75.1.

W pliku `tekst.txt` dany jest, w pojedynczym wierszu, tekst złożony z dokładnie 805 słów zapisanych małymi literami alfabetu angielskiego, oddzielonych znakami odstępu. Żadne słowo nie jest dłuższe niż 15 znaków.

Znajdź i wypisz te słowa, których zarówno pierwszą, jak i ostatnia literą jest 'd'.

75.2.

Zaszyfruj szyfrem afinicznym o kluczu (5, 2) te słowa z pliku `tekst.txt`, które składają się z co najmniej 10 liter. Wypisz je w postaci zaszyfrowanej, po jednym w wierszu.

75.3.

Plik `probka.txt` składa się z 5 wierszy, każdego zawierającego dwa napisy. Pierwszy z nich to pewne słowo zapisane tekstem jawnym, drugi zaś to to samo słowo zaszyfrowane za pomocą szyfru afinicznego (każde słowo innym kluczem).

Dla każdego z tych słów znajdź i wypisz klucz szyfrujący oraz klucz deszyfrujący.

Zadanie 76.

Wiązka zadań *Szyfr*

Rozważamy szyfrowanie przestawieniowe, w którym kluczem jest *n-elementowa* tablica zawierająca różne liczby całkowite z przedziału [1, *n*]. Na przykład kluczem *5*-elementowym może być tablica [3, 2, 5, 4, 1].

Szyfrowanie napisu *A* (o długości co najmniej *n*) kluczem *n*-elementowym *P*[1..*n*] odbywa się w następujący sposób:

- pierwsza litera słowa *A* zamieniana jest miejscami z literą na pozycji *P*[1],
- następnie druga litera słowa *A* zamieniana jest z literą na pozycji *P*[2]
- itd.

Uzyskane na końcu słowo jest szyfrem napisu *A* z kluczem *P*.

Jeśli napis *A* ma więcej niż *n* liter, to po *n*-tym kroku powyższego algorytmu kolejną literę zamieniamy znów z literą na pozycji *P*[1] itd. Oznacza to, że w *i*-tym kroku zamieniamy litery na pozycjach *i* oraz *P*[1+(*i*-1) mod *n*].

Przykład

Poniższa tabelka ilustruje szyfrowanie słowa „INFORMATYKA” kluczem P równym [3, 2, 5, 4, 1]:

i	1	2	3	4	5	6	7	8	9	10	11
P[1+(i-1) mod n]	3	2	5	4	1	3	2	5	4	1	3
Słowo	I	N	F	O	R	M	A	T	Y	K	A
Krok 1	**F**	N	**I**	O	R	M	A	T	Y	K	A
Krok 2	F	**N**	I	O	R	M	A	T	Y	K	A
Krok 3	F	N	**R**	O	**I**	M	A	T	Y	K	A
Krok 4	F	N	R	**O**	I	M	A	T	Y	K	A
Krok 5	**I**	N	R	O	**F**	M	A	T	Y	K	A
Krok 6	I	N	**M**	O	F	**R**	A	T	Y	K	A
Krok 7	I	**A**	M	O	F	R	**N**	T	Y	K	A
Krok 8	I	A	M	O	**T**	R	N	**F**	Y	K	A
Krok 9	I	A	M	**Y**	T	R	N	F	**O**	K	A
Krok 10	**K**	A	M	Y	T	R	N	F	O	**I**	A
Krok 11	K	A	**A**	Y	T	R	N	F	O	I	**M**

Napis „KAAYTRNFOIM” jest zatem szyfrem napisu „informatyka” z kluczem [3, 2, 5, 4, 1]. **Napisz program**(-y), który da odpowiedzi do poniższych zadań.

76.1.

W pliku `szyfr1.txt` dane są:

- w wierszach o numerach od 1 do 6 — napisy złożone z 50 liter alfabetu łacińskiego;
- w wierszu nr 7 — klucz 50-elementowy; liczby oddzielone są pojedynczym odstępem.

Zaszyfruj wszystkie sześć napisów zgodnie z opisaną metodą. Wynik, czyli zaszyfrowane napisy, zapisz w osobnych wierszach w pliku `wyniki_szyfr1.txt`.

76.2.

W pliku `szyfr2.txt` dane są:

- w pierwszym wierszu — napis złożony z 50 liter alfabetu łacińskiego;
- w drugim wierszu — klucz 15-elementowy; liczby oddzielone są pojedynczym odstępem.

Zaszyfruj dany napis zgodnie z opisaną metodą. Wynik, czyli zaszyfrowany napis, zapisz w pliku `wyniki_szyfr2.txt`.

76.3.

W pliku `szyfr3.txt` dany jest napis złożony z 50 liter alfabetu łacińskiego. Napis ten powstał po zaszyfrowaniu pewnego napisu *A* kluczem [6, 2, 4, 1, 5, 3].

Podaj napis *A*. Wynik zapisz w pliku `wyniki_szyfr3.txt`.

Zadanie 77.

Wiązka zadań *Szyfr Vigenère'a*

W zadaniu rozważamy teksty zbudowane tylko z wielkich liter alfabetu angielskiego, znaków odstępu i znaków przestankowych (przecinek, kropka). Oto litery alfabetu i numery ich pozycji w alfabecie:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Szyfrowanie Vigenère'a polega na zastąpieniu każdej litery tekstu źródłowego literą odległą od niej cyklicznie w alfabecie o *k* pozycji.

Wartość *k* nie jest z góry ustalona dla całego tekstu źródłowego, lecz dla każdej litery w tekście jest określana osobno, w oparciu o słowo przyjęte jako klucz szyfrowania.

Przystępując do szyfrowania, należy przyporządkować kolejnym literom tekstu źródłowego kolejne litery klucza, chodząc po nim cyklicznie, jeśli jest krótszy od szyfrowanego tekstu. Znaki inne niż litery nie są szyfrowane, pomijamy je podczas przypisywania liter klucza. P**ozycja litery klucza** w alfabecie jest tą wartością ***k***, o jaką należy wykonać przesunięcie względem litery tekstu źródłowego w celu znalezienia odpowiadającej jej litery szyfru.

Przykład

tekst źródłowy: `"JEST OK"`, klucz: `"EWA"`

tekst źródłowy	J	E	S	T	spacja	O	K
Klucz	E	W	A	E		W	A
pozycja litery klucza	4	22	0	4		22	0
Szyfr	J→4 = **N**	E→22=**A**	S→0 = **S**	T→4 = **X**	spacja	O→22=**K**	K→0 = **K**

wynik szyfrowania: `"NASX KK"`.

Napisz w wybranym języku programowania program, który wyznaczy rozwiązania zadań podanych niżej. Wszystkie wyniki zapisz w pliku tekstowym `Vigenere_wyniki.txt`, wyraźnie oddzielając odpowiedzi do poszczególnych zadań. Do oceny oddaj ten plik oraz plik (pliki) zawierający reprezentację komputerową rozwiązania.

77.1.

W pliku `dokad.txt` znajduje się jeden wiersz z tekstem. Długość tekstu nie przekracza 1024 znaków. Należy zaszyfrować ten tekst metodą Vigenère'a, używając jako klucza słowa: "LUBIMYCZYTAC".

a) Podaj liczbę powtórzeń klucza niezbędną do zaszyfrowania całego tekstu źródłowego (uwzględniając w nich ostatnie rozpoczęte powtórzenie).

b) Podaj zaszyfrowany tekst i zapisz go w pliku z odpowiedziami.

77.2.

W pliku `szyfr.txt` zapisano dwa wiersze. W pierwszym wierszu znajduje się tekst zaszyfrowany metodą Vigenère'a. W drugim wierszu znajduje się klucz użyty do tego szyfrowania.

Szyfr zawiera wiele słów. Jego łączna długość nie przekracza 1024 znaków. Szyfrowaniu podlegały tylko wielkie litery tekstu, zaś odstępy i znaki przestankowe pozostały bez zmiany.

Odszyfruj tekst i umieść jego postać źródłową w pliku z odpowiedziami.

77.3.

a) Podaj liczby wystąpień poszczególnych liter A, B, ..., Z w treści szyfru zawartego w pierwszym wierszu pliku `szyfr.txt`.

b) Chcąc złamać szyfr Vigenère, nie znając klucza, w pierwszym kroku należy oszacować długość klucza (rozumianą jako liczba znaków). Istnieje przybliżony wzór na szacunkową długość klucza *d* danego szyfru Vigenère'a dla tekstu nad alfabetem 26-literowym. Oszacowanie jest tym lepsze, im dłuższy jest szyfr.

$$d = \frac{0{,}0285}{\kappa_o - 0{,}0385}$$

gdzie κ_o to indeks koincydencji znaków obliczany następująco:

$$\kappa_o = \frac{l_A * (l_A - 1) + l_B * (l_B - 1) + \cdots + l_Z * (l_Z - 1)}{n * (n - 1)}$$

n — łączna liczba wystąpień **wszystkich liter** w tekście szyfru (nie liczymy odstępów i znaków przestankowych),

$l_A, l_B, \ldots, l_Z$ — liczby wystąpień **poszczególnych liter** `A`, `B`, …, `Z` w tekście szyfru.

Wykorzystując powyższe wzory, wyznacz szacunkową długość klucza dla szyfru danego w pierwszym wierszu pliku `szyfr.txt` i porównaj z dokładną długością klucza umieszczonego w drugim wierszu tego pliku. Wypisz obie wartości, wartość szacunkową zaokrąglij do 2 cyfr po przecinku.

Zadanie 78.

Wiązka zadań *Podpis elektroniczny*

Bajtek otrzymał od przyjaciela 11 jawnych wiadomości. Do każdej wiadomości przyjaciel dołączył podpis elektroniczny.

Podpis elektroniczny jest **zaszyfrowanym skrótem** wiadomości. Przyjaciel Bajtka utworzył skrót za pomocą funkcji *skrot*(), która przekształca dowolną wiadomość w 8-znakowy napis, a następnie zaszyfrował ten skrót algorytmem *A* o sobie tylko znanym kluczu prywatnym (*e*,*n*). Opis obu algorytmów podano poniżej.

Bajtek chciałby być pewien, że zachowano:

- integralność danych (treść nie została zmieniona w trakcie przesyłania),
- uwierzytelnienie nadawcy (nikt się pod nadawcę nie podszył).

W tym celu powinien sprawdzić każdą wiadomość następująco:

- zaprogramować funkcję *skrot*(*wiadomosc*) i za jej pomocą utworzyć skrót wiadomości,
- odszyfrować skrót z podpisu elektronicznego algorytmem *A* przy pomocy ogólnie znanego klucza publicznego (***d***,***n***) o wartościach (**3**,**200**),

- porównać oba skróty: jeśli są identyczne, znaczy to, że wiadomość jest wiarygodna.

Pomóż Bajtkowi sprawdzić, czy otrzymane wiadomości są wiarygodne.

W pliku `wiadomosci.txt` znajduje się 11 wiadomości, każda w osobnym wierszu. Liczba znaków każdej wiadomości nie przekracza 255. Wiadomości zawierają znaki pojedynczego odstępu, które są integralną częścią informacji.

W pliku `podpisy.txt` znajduje się 11 wierszy, każdy z nich zawiera 8 liczb całkowitych, stanowiących elementy podpisu elektronicznego jednej wiadomości. Liczby w wierszu oddzielone są pojedynczymi znakami odstępu. Kolejność wierszy podpisów jest zgodna z kolejnością wierszy wiadomości (pierwszy wiersz podpisów odpowiada pierwszej wiadomości, drugi — drugiej itd.)

Funkcja skrótu *skrot*(*wiadomość*)

Skrót wiadomości jest 8-znakowym napisem, złożonym z wielkich liter alfabetu angielskiego.

Aby go wyznaczyć, wykonaj następujące kroki:

a) Wpisz do 8-elementowej tablicy *S* kody ASCII znaków słowa ”ALGORYTM”.
b) Treść wiadomości uzupełnij na końcu znakami kropki ’.’ do wielokrotności 8 znaków.
c) Rozpatrz po kolei 8-znakowe porcje treści wiadomości. W zależności od kodów ich znaków aktualizuj wartości elementów w tablicy S. Dla każdej porcji treści wiadomości powtarzaj:
   dla *j* =1,2 .. 8 wykonuj
   S[*j*] ← **(***S*[*j*]) + kod znaku na *j*-tej pozycji w bieżącej porcji wiadomości) *mod* 128

d) Zbuduj wynik, wyznaczając jego kolejne znaki na podstawie elementów tablicy S:
   wynik = ””
   dla *j* =1,2 .. 8 wykonuj
   wynik ← *wynik* + char(65 + *S*[*j*] *mod* 26)

   gdzie: *mod* jest operatorem dzielenia modulo,
   funkcja *char*(*kod*) zwraca reprezentację graficzną znaku o podanym kodzie

Otrzymany w ten sposób *wynik* jest skrótem wiadomości.

Algorytm *A* szyfrowania z kluczem prywatnym (*e,n*) i deszyfrowania kluczem publicznym (*d,n*)

Deszyfrowanie polega na wykonaniu operacji *x=(y*d mod n)*, gdzie za *y* należy przyjąć kolejne liczby tworzące podpis elektroniczny. Tekst wynikowy można otrzymać, składając w jeden napis reprezentacje graficzne kolejnych liczb *x* zgodnie ze standardem ASCII.

Uwaga dla dociekliwych

Zaszyfrowanie algorytmem *A* polegało na wykonywaniu operacji ***y=(x*e mod n)***, gdzie za *x* należało podstawić kody ASCII kolejnych znaków tekstu źródłowego. Uzyskany w ten sposób ciąg liczb jest podpisem elektronicznym wiadomości. Gdyby ktoś chciał złamać szyfr *A*, czyli wyznaczyć nieznany element *e* klucza prywatnego, musiałby znaleźć taką wartość *e,* względnie pierwszą z *d*, że *e*d mod n = 1*. Uzasadnienia szukaj w prawach arytmetyki modularnej.

Napisz program rozwiązujący poniższe zadania. Do oceny oddaj plik tekstowy `epodpis_wynik.txt`, zawierający odpowiedzi, oraz plik (pliki) zawierający reprezentację komputerową Twojego rozwiązania.

78.1.

Wyznacz skrót **pierwszej** wiadomości z pliku `wiadomosci.txt` i udokumentuj wyniki kolejnych etapów obliczania tego skrótu. Zapisz w kolejnych wierszach pliku wynikowego:

a) liczbę znaków wiadomości po jej uzupełnieniu do najmniejszej długości o wielokrotności 8 znaków,

b) wartości liczbowe 8 kolejnych bajtów skrótu (elementy tablicy *S*) po przetworzeniu całej wiadomości — wszystkie wartości w jednym wierszu, oddzielone pojedynczymi znakami odstępu,

c) skrót wiadomości w postaci napisu o długości 8, złożonego z wielkich liter alfabetu angielskiego.

78.2.

Odszyfruj skróty wiadomości ze wszystkich podpisów elektronicznych umieszczonych w pliku `podpisy.txt`, stosując algorytm *A* z kluczem publicznym (*d*,*n*) = (3,200). Zapisz uzyskane skróty w kolejnych, osobnych wierszach pliku z odpowiedziami.

78.3.

Zweryfikuj wiarygodność wszystkich wiadomości i podaj numery wiadomości wiarygodnych. Zapisz w jednym wierszu pliku z odpowiedziami, jako liczby z zakresu 1..11, zgodnie z kolejnością umieszczenia ich w pliku danych, oddzielone pojedynczym znakiem odstępu.

Zadanie 79.

Wiązka zadań *Okręgi*

Okrąg na płaszczyźnie reprezentujemy za pomocą trzech liczb x, y, r, gdzie (x, y) oznaczają współrzędne środka okręgu, a r — jego promień.

W pliku `okregi.txt` danych jest 2000 okręgów; każdy zapisany jest w osobnym wierszu, zawierającym trzy liczby rzeczywiste x, y, r $(-10^9 \leq x, y \leq 10^9,\ 0 < r < 10^9)$.

Pierwsze **dziewięć wierszy** tego pliku zawiera liczby:

```
  8.000          5.000         40.815
 22.623         21.558         50.000
 53.551         38.508         25.663
 63.429         42.655         18.844
 77.352         48.799         20.286
 80.274         52.627         16.127
178.318        128.106         23.993
191.501        140.711          8.379
197.073        143.275         13.129
```

Można zauważyć, że ostatnie siedem okręgów w całości zawiera się w pierwszej ćwiartce układu współrzędnych. Natomiast pierwsze dwa okręgi nie leżą w całości w żadnej ćwiartce.

Napisz program, który umożliwi wykonanie poniższych poleceń. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym `wyniki_okregi.txt`. Wyniki do każdego zadania poprzedź numerem je oznaczającym.

79.1.

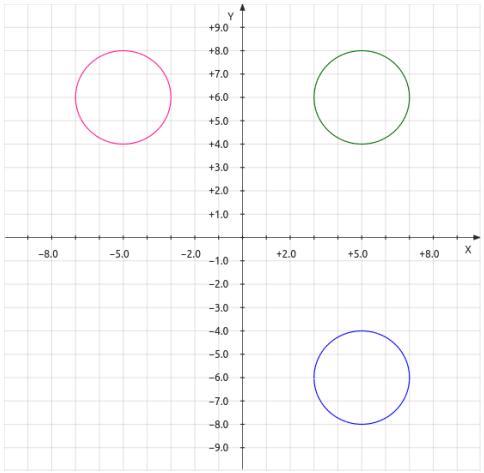
Podaj liczbę okręgów, które całkowicie zawierają się w I, II, III i IV ćwiartce układu współrzędnych. Podaj również liczbę okręgów, które nie zawierają się w całości w żadnej ćwiartce, tzn. mają co najmniej jeden punkt wspólny z jedną z osi Ox lub Oy. Jako odpowiedź wypisz pięć liczb: liczba okręgów I ćwiartki, liczba okręgów II ćwiartki, liczba okręgów III ćwiartki, liczba okręgów IV ćwiartki oraz liczba okręgów, które nie zawierają się w całości w żadnej ćwiartce.

79.2.

Powiemy, że dwa okręgi tworzą *lustrzaną parę*, jeśli jeden z nich powstaje przez odbicie drugiego względem jednej z osi Ox lub Oy. Podaj liczbę lustrzanych par spośród wszystkich okręgów zapisanych w pliku `okregi.txt`.

Uwagi

1. Układ trzech okręgów powstałych przez odbicia względem osi Ox lub Oy zawiera **dwie** lustrzane pary; np. wśród okręgów o środkach w punktach (-5,6), (5,6), (5,-6) (i o tych samych promieniach) są dokładnie dwie lustrzane pary; zob. poniższy rysunek.

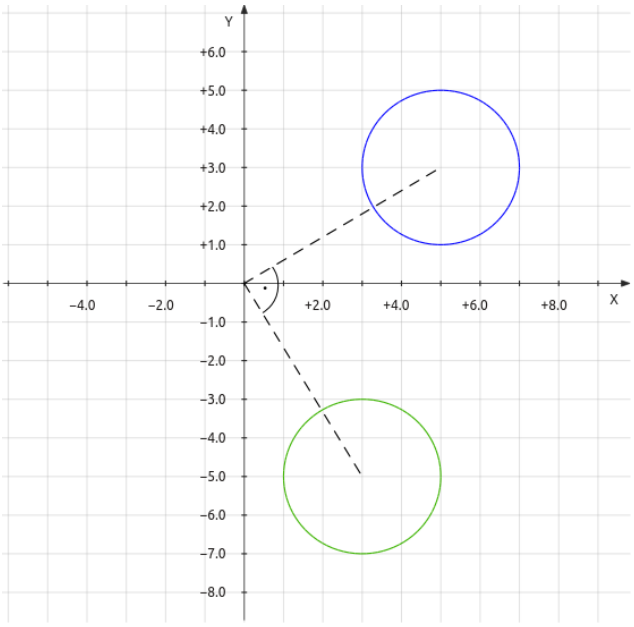


2. Analogicznie do poprzedniego punktu układ czterech okręgów zawiera **cztery** lustrzane pary.

79.3.

Powiemy, że dwa okręgi tworzą *prostopadłą parę*, jeśli jeden z nich powstaje przez obrót drugiego o 90 stopni względem środka układu współrzędnych.

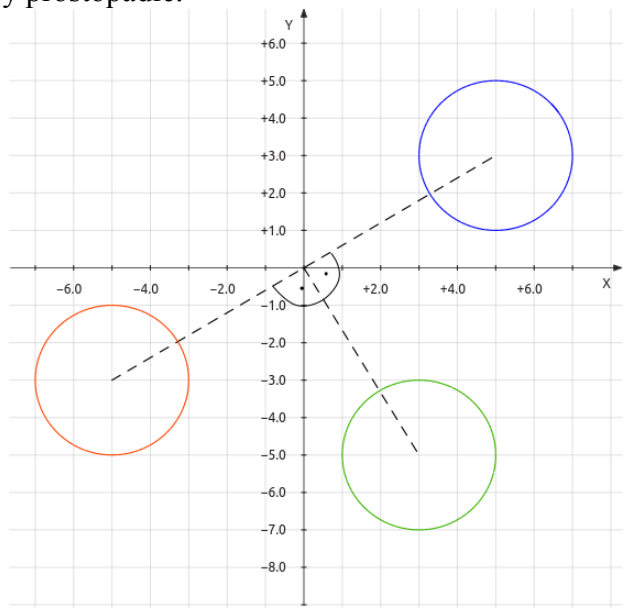
Przykład: okręgi o środkach w punktach (3,-5), (5,3) (i o tych samych promieniach) tworzą parę prostopadłą; zob. rysunek.



Podaj liczbę prostopadłych par okręgów spośród wszystkich okręgów zapisanych w pliku `okregi.txt`.

Uwagi

1. układ trzech okręgów powstałych przez obrót o 90 stopni względem środka układu współrzędnych zawiera **dwie** pary prostopadłe;
   np. wśród okręgów o środkach (5,3), (3,-5), (-5,-3) (i o tych samych promieniach) są dokładnie dwie pary prostopadłe.



2. Analogicznie do poprzedniego punktu układ czterech okręgów zawiera **cztery** pary prostopadłe.

79.4.

Powiemy, że ciąg okręgów tworzy *łańcuch*, jeśli kolejne okręgi tego ciągu mają ze sobą co najmniej jeden punkt wspólny; przyjmujemy, że ciąg zawierający tylko jeden okrąg również tworzy łańcuch. Można zauważyć, że wśród podanych okręgów mamy dwa łańcuchy: pierwszy o długości 6, a drugi o długości 3.

Znajdź długości wszystkich łańcuchów tworzonych przez okręgi zapisane w wierszach o numerach **od 1 do 1000**. Podaj długość najdłuższego łańcucha.

Zadanie 80.

Wiązka zadań *Trójkąty*

W pliku `dane_trojkaty.txt` znajduje się 500 liczb całkowitych dodatnich. Każda liczba jest zapisana w osobnym wierszu, żadna liczba **nie występuje w pliku więcej niż jeden raz**. Liczby podane w pliku `dane_trojkaty.txt` to długości odcinków, z których będziemy próbować budować trójkąty.

Napisz program(-y), który pozwoli rozwiązać poniższe zadania. Odpowiedzi zapisz w pliku `wyniki_trojkaty.txt`. Odpowiedź do każdego zadania podaj w osobnym wierszu, poprzedzając ją numerem zadania.

80.1.

Wypisz wszystkie trójki **kolejnych** liczb z pliku `dane_trojkaty.txt`, które są długościami boków trójkąta prostokątnego.

Przykład

Rozważmy plik składający się z dziesięciu liczb: 8, 7, 4, 3, 5, 9, 12, 13, 85, 84. Wynikiem są dwie trójki liczb: 4, 3, 5 oraz 13, 85, 84. Trójkąt prostokątny tworzą też odcinki o bokach 5, 12 i 13, ale liczby te **nie są kolejnymi** liczbami w podanym pliku.

80.2.

Podaj największy obwód trójkąta, którego boki mają długości równe liczbom występującym w różnych wierszach pliku `dane_trojkaty.txt`.

Przykład

Dla pliku składającego się z dziesięciu liczb: 10, 18, 70, 100, 15, 13, 21, 12, 1, 2 wynikiem jest 54, ponieważ trójkąt o największym obwodzie ma boki 18, 15 i 21.

80.3.

Podaj, ile **nieprzystających** trójkątów można utworzyć z odcinków o długościach podanych w pliku `dane_trojkaty.txt`.

Uwaga: Dwa trójkąty są przystające wtedy i tylko wtedy, gdy trzy boki jednego trójkąta są odpowiednio równe trzem bokom drugiego trójkąta, np. trójkąt o bokach (10, 18, 15) jest przystający z trójkątem o bokach (18, 15, 10).

Przykład

Dla pliku składającego się z dziesięciu liczb: 10, 18, 70, 100, 15, 13, 21, 12, 1, 2 wynikiem jest 21, gdyż z podanych długości odcinków można utworzyć 21 trójkątów o następujących bokach:

(10, 18, 15); (10, 18, 13); (10, 18, 21); (10, 18, 12); (10, 15, 13); (10, 15, 21); (10, 15, 12); (10, 13, 21); (10, 13, 12); (10, 21, 12); (18, 15, 13); (18, 15, 21); (18, 15, 12); (18, 13, 21); (18, 13, 12); (15, 13, 21); (15, 13, 12); (15, 21, 12); (13, 21, 12); (13, 12, 2); (18, 21, 12).

Zadanie 81.

Wiązka zadań *Czworokąty*

Plik `wierzcholki.txt` zawiera 100 wierszy. W każdym wierszu zapisano 6 liczb całkowitych z przedziału <–100; 100>, będących współrzędnymi trzech różnych punktów: A, B i C w kartezjańskim układzie współrzędnych (odpowiednio x_a, y_a, x_b, y_b, x_c, y_c). Liczby w wierszu są oddzielone pojedynczymi znakami tabulacji.

Podobny plik `wierzcholkiTR.txt` zawiera również 100 wierszy. W każdym wierszu zapisano 6 liczb całkowitych należących do przedziału <–100; 100>, będących współrzędnymi trzech wierzchołków trójkąta ABC (odpowiednio x_a, y_a, x_b, y_b, x_c, y_c). Liczby w wierszu są oddzielone pojedynczymi znakami tabulacji.

Wykorzystując dane zawarte w plikach oraz dostępne narzędzia informatyczne, rozwiąż poniższe zadania.

Napisz program(-y), za pomocą którego(-ych) uzyskasz odpowiedzi do poniższych zadań. Do oceny oddaj dokument wyniki.txt z zapisanymi odpowiedziami na poszczególne zadania oraz pliki źródłowe programów wykorzystanych do uzyskania rozwiązania.

81.1.

Podaj liczbę wierszy z pliku `wierzcholki.txt`, w których wszystkie zapisane punkty leżą w I ćwiartce układu współrzędnych i nie należą do osi OX i OY.

81.2.

Podaj liczbę wierszy z pliku `wierzcholki.txt`, w których zapisane są współrzędne punktów leżących na jednej prostej.

81.3.

Podaj (z pliku `wierzcholkiTR.txt`) współrzędne wierzchołków trójkąta o największym obwodzie oraz obwód tego trójkąta. Obwód zaokrąglij do dwóch miejsc po przecinku. Uwaga: możesz założyć, że jest tylko jeden taki trójkąt.

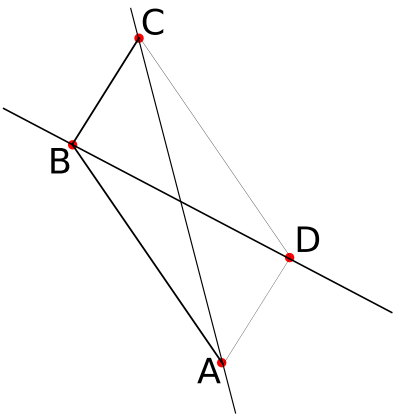
79.4.

Dla każdego wiersza z pliku `wierzcholkiTR.txt` sprawdź, czy punkty zapisane w tym wierszu są wierzchołkami pewnego trójkąta prostokątnego. Podaj liczbę trójkątów prostokątnych zapisanych w tym pliku

Uwaga: Takich trójkątów jest więcej niż cztery.

81.5.

Dla każdego wiersza z pliku `wierzcholkiTR.txt`, w których zapisane są kolejno współrzędne punktów A, B i C, wyznacz współrzędne punktu D, tak aby czworokąt ABCD był równoległobokiem. Podaj współrzędne wszystkich wierzchołków czworokątów ABCD, których punkt D leży na prostej $y=x$.

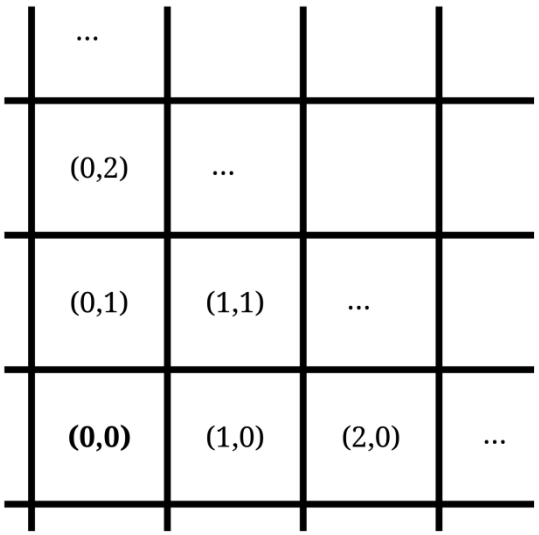


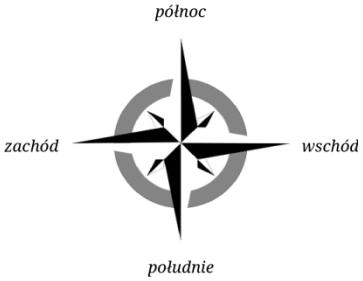
2.2. Symulacja

Zadanie 82.

Wiązka zadań *Piraci*

Na karaibskiej wyspie Santo Domingo piraci poszukują skarbu, który niegdyś ukrył tam zbuntowany hiszpański wicekról. Przeszukiwany teren podzielili na jednostkowe kwadratowe obszary o boku jednej mili. Każdemu obszarowi przypisali współrzędne: odciętą i rzędną, przy czym odcięta rośnie w kierunku wschodnim, a rzędna w kierunku północnym, tak jak przedstawiono na poniższym rysunku:





W niedzielę 1 października 1902 roku piraci lądują na wyspie, na obszarze (0,0), i postępują według następującej, znalezionej w tajemniczych okolicznościach, instrukcji:

> *„Każdego dnia przejdź 8 mil prosto na północ, a potem skręć w prawo i przejdź 11 mil na wschód. Policz, ile w sumie mil na północ przeszedłeś od zejścia ze statku — kopiąc w tym miejscu, znajdziesz tyle złotych dublonów, ile cyfr ma ta łączna odległość. Ponadto każdego trzeciego dnia miesiąca znajdziesz dodatkowo dwa dublony.*
>
> *Po zebraniu złota zawróć i przejdź na zachód tyle mil, ile dublonów właśnie zebrałeś. Tam rozbij obóz na noc, a następnego dnia możesz kontynuować swoją wędrówkę.*
>
> *Nie bądź zbyt chciwy, aby nie spadła na ciebie klątwa!”*

Wiedząc, że piraci spędzili na wyspie 150 dni, każdego dnia (łącznie z dniem lądowania) wypełniając dokładnie te polecenia. Jeżeli piraci znajdują się w kwadracie o współrzędnych (i,j), to po przejściu 1 mili na północ znajdą się w obszarze o współrzędnych $(i,j+1)$. Z kolei w obszarze o współrzędnych $(i+1,j)$ znajdą się, gdy pójdą na wschód, $(i,j-1)$ — gdy pójdą na południe, $(i-1,j)$ — gdy pójdą na zachód.

Używając dostępnych narzędzi informatycznych, znajdź odpowiedzi na poniższe pytania. Odpowiedzi zapisz w pliku o nazwie *wyniki.txt,* każdą z nich umieszczając w osobnym wierszu i poprzedzając numerem odpowiedniego zadania.

82.1.

Codziennie wieczorem piraci obozują w miejscu, gdzie znaleźli się po wykopaniu dublonów i przejściu odpowiedniej liczby mil na zachód. Podaj współrzędne obozu piratów, w którym spędzają noc wigilijną 24/25 grudnia 1902.

82.2.

Oblicz, ile mil łącznie przejdą piraci przez cały okres poszukiwań. Uwzględnij mile przebyte na północ, wschód, a także na zachód, w czasie cofania się.

82.3.

W każdą sobotę część piratów wymyka się z obozu, aby popłynąć łódką na sąsiednią wyspę Tortuga, gdzie na różne rozrywki tracą 10% (zaokrąglone w dół do liczby całkowitej) majątku posiadanego przez całą bandę.

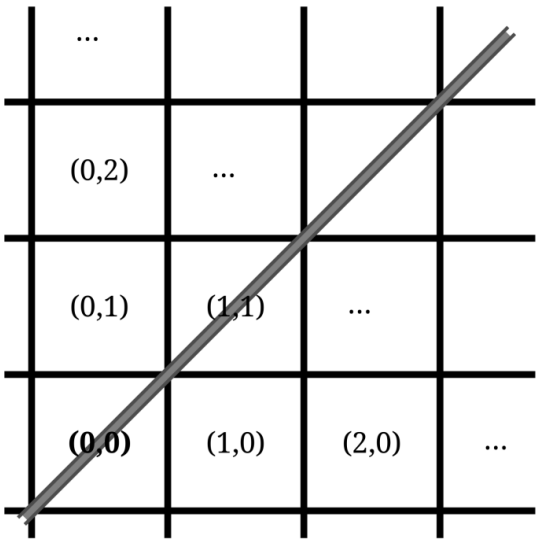
Oblicz, ile łącznie dublonów piraci zostawią na Tortudze przez cały okres swojej wyprawy. Załóż, że na wyspę przybyli, nie mając ani jednego dublona.

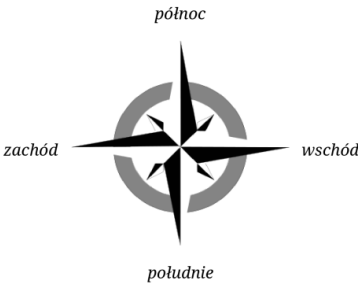
82.4.

Klątwa uaktywnia się (o czym piraci nie wiedzą) każdego dnia, w którym piraci znajdują 4 lub więcej dublonów. Klątwa ta skutkuje nieprzyjemną niespodzianką: po powrocie na statek piraci będą musieli zmierzyć się z oddziałem wojska miejscowego gubernatora, składającym się z tylu żołnierzy, ile piraci zebrali łącznie dublonów w feralne dni. Oblicz, ilu przeciwników spotkają piraci.

82.5.

Przez Santo Domingo płynie rzeka, pod kątem 45 stopni do brzegów wyspy, wpadająca do morza w pobliżu punktu lądowania piratów. Odległość obozu piratów (będącego w danym dniu na polu (x,y) od rzeki można opisać wzorem *odległość* = $|x - y|$, czyli jako wartość bezwzględną z różnicy współrzędnych obozu[1].





[1] W rzeczywistości odległość ta wynosi $\frac{|x-y|}{\sqrt{2}/2}$ (licząc od środka obszaru do rzeki), użyjemy jednak uproszczonej formuły.

Znajdź średnią odległość wieczornego obozu piratów od rzeki przez cały okres poszukiwań (150 dni) w zaokrągleniu do dwóch miejsc po przecinku, a następnie sporządź wykres kolumnowy przedstawiający tę odległość w kolejnych dniach.

Komentarz do zadania

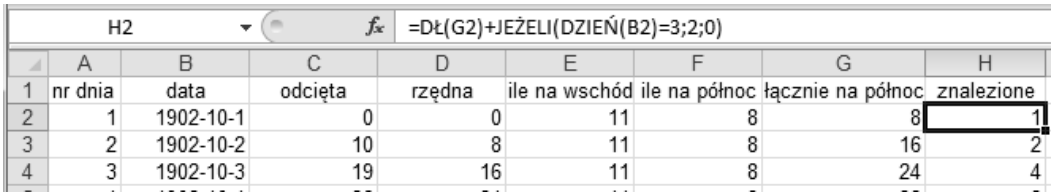
Utwórzmy arkusz, którego każdy wiersz będzie przechowywał następujące informacje:

1. *nr dnia* i *data* — kolejny dzień symulacji (1,2,3,…) oraz aktualna data (począwszy od 1 października 1902),
2. *odcięta* i *rzędna* — współrzędne pola, z którego piraci rano rozpoczynają wędrówkę (na początku (0,0)),
3. *ile na wschód*, *ile na północ* — ile mil przeszli tego dnia na wschód oraz na północ,
4. *łącznie na północ* — całkowita liczba mil, jakie piraci pokonali w kierunku północnym
5. *znalezione* — ile dublonów znaleźli kopiąc (zarazem: ile mil cofnęli się potem na zachód).

	A	B	C	D	E	F	G	H
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione
2	1	1902-10-1	0	0	11	8	8	1
3	2	1902-10-2	10	8	11	8	16	2
4	3	1902-10-3	19	16	11	8	24	4

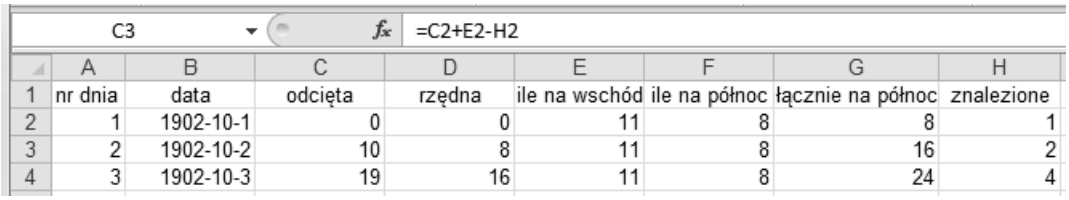
Wartości w wierszu *k* obliczamy na podstawie wartości z poprzedniego wiersza w następujący sposób:

- kolejny *nr dnia* i *data* obliczą się automatycznie po rozszerzeniu pierwszych komórek na całą kolumnę,
- *ile na wschód* oraz *ile na północ* to zawsze odpowiednio 11 i 8,
- *łącznie na północ* obliczamy z analogicznej wartości poprzedniego wiersza, dodając do niej *ile na północ,*
- *znalezione* obliczamy zgodnie z piracką instrukcją, stosując do wartości *łącznie na północ* funkcję DŁ w MS Excel, która podaje długość napisu w komórce. Musimy dodać jeszcze 2 dublony, jeśli jest to trzeci dzień miesiąca — odzyskujemy go z *daty* funkcją DZIEŃ.



	A	B	C	D	E	F	G	H
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione
2	1	1902-10-1	0	0	11	8	8	1
3	2	1902-10-2	10	8	11	8	16	2
4	3	1902-10-3	19	16	11	8	24	4

- *odciętą* oraz *rzędną* obliczamy z ich wartości z poprzedniego wiersza wartości, dodając do rzędnej mile przebyte na północ (*ile na północ*), a do odciętej — mile przebyte na wschód (*ile na wschód*), pomniejszone o mile przebyte na zachód (*znalezione*).



	A	B	C	D	E	F	G	H
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione
2	1	1902-10-1	0	0	11	8	8	1
3	2	1902-10-2	10	8	11	8	16	2
4	3	1902-10-3	19	16	11	8	24	4

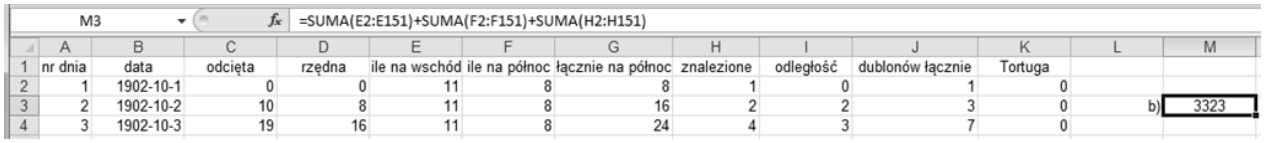
Mając już kluczową część symulacji, możemy zrealizować polecenia podane w kolejnych zadaniach.

82.1.

Odpowiedzią są wartości *odciętej* i *rzędnej* w wierszu odpowiadającym 25 grudnia 1902.

82.2.

Sumujemy wartości *ile na północ*, *ile na wschód* oraz *znalezione* w całym zakresie.



M3 =SUMA(E2:E151)+SUMA(F2:F151)+SUMA(H2:H151)

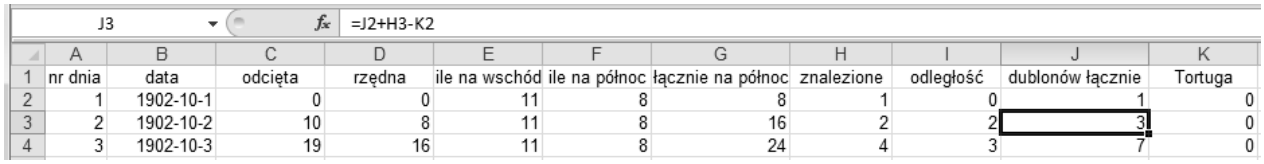
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione	odległość	dublonów łącznie	Tortuga		
2	1	1902-10-1	0	0	11	8	8	1	0	1	0		
3	2	1902-10-2	10	8	11	8	16	2	2	3	0	b)	3323
4	3	1902-10-3	19	16	11	8	24	4	3	7	0		

82.3.

Konieczne jest dopisanie dwóch kolumn: łącznego majątku piratów oraz dublonów zostawionych na Tortudze. Łączny majątek w dniu *k* to majątek w dniu *k-1* powiększony o dublony znalezione w dniu *k*, a pomniejszony o złoto stracone na rozrywki. Dublony stracone w dniu *k* obliczamy za pomocą instrukcji warunkowej:

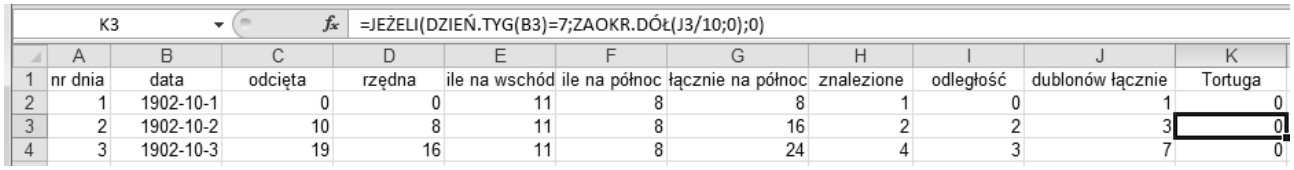
- jeśli dniem tygodnia (funkcja DZIEN.TYG) jest sobota, liczbą straconych dublonów jest aktualny majątek piratów podzielony przez 10 i zaokrąglony w dół.
- jeśli dzień tygodnia jest inny niż sobota, straconych dublonów jest 0.

Pozostało jeszcze zsumować stracone dublony za pomocą instrukcji SUMA.



J3 =J2+H3-K2

	A	B	C	D	E	F	G	H	I	J	K
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione	odległość	dublonów łącznie	Tortuga
2	1	1902-10-1	0	0	11	8	8	1	0	1	0
3	2	1902-10-2	10	8	11	8	16	2	2	3	0
4	3	1902-10-3	19	16	11	8	24	4	3	7	0



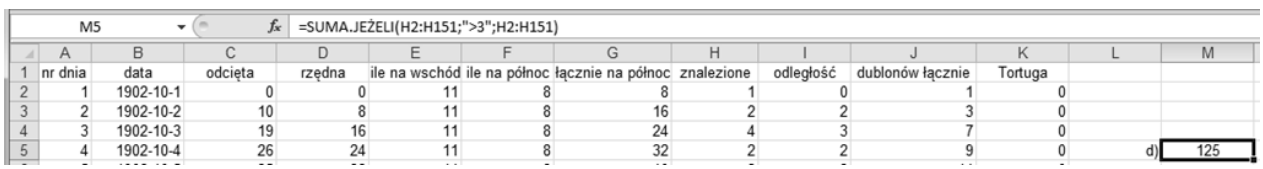
K3 =JEŻELI(DZIEŃ.TYG(B3)=7;ZAOKR.DÓŁ(J3/10;0);0)

	A	B	C	D	E	F	G	H	I	J	K
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione	odległość	dublonów łącznie	Tortuga
2	1	1902-10-1	0	0	11	8	8	1	0	1	0
3	2	1902-10-2	10	8	11	8	16	2	2	3	0
4	3	1902-10-3	19	16	11	8	24	4	3	7	0

82.4.

Zadanie można rozwiązać bardzo szybko instrukcją warunkowej sumy, czyli SUMA.JEŻELI. Chcemy zsumować w pewnym zakresie tylko te komórki, które mają odpowiednio duże wartości (większe niż 3). Używamy więc następującej formuły:

SUMA.JEŻELI(*zakres*;">3"; *zakres*)



M5 =SUMA.JEŻELI(H2:H151;">3";H2:H151)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	nr dnia	data	odcięta	rzędna	ile na wschód	ile na północ	łącznie na północ	znalezione	odległość	dublonów łącznie	Tortuga		
2	1	1902-10-1	0	0	11	8	8	1	0	1	0		
3	2	1902-10-2	10	8	11	8	16	2	2	3	0		
4	3	1902-10-3	19	16	11	8	24	4	3	7	0		
5	4	1902-10-4	26	24	11	8	32	2	2	9	0	d)	125

82.5.

Odległość liczymy, używając funkcji ŚREDNIA. Ważne jest, aby do średniej nie wliczyć punktu położenia obozu rankiem pierwszego dnia (czyli punktu lądowania (0,0)), za to uwzględnić położenie obozu wieczorem ostatniego dnia.

Rozwiązanie

Realizacja komputerowa rozwiązania (całej wiązki) znajduje się w pliku *piraci_rozw.xlsx.*

Prawidłowe odpowiedzi:

82.1.

(687, 680)

82.2.

3323 kroki

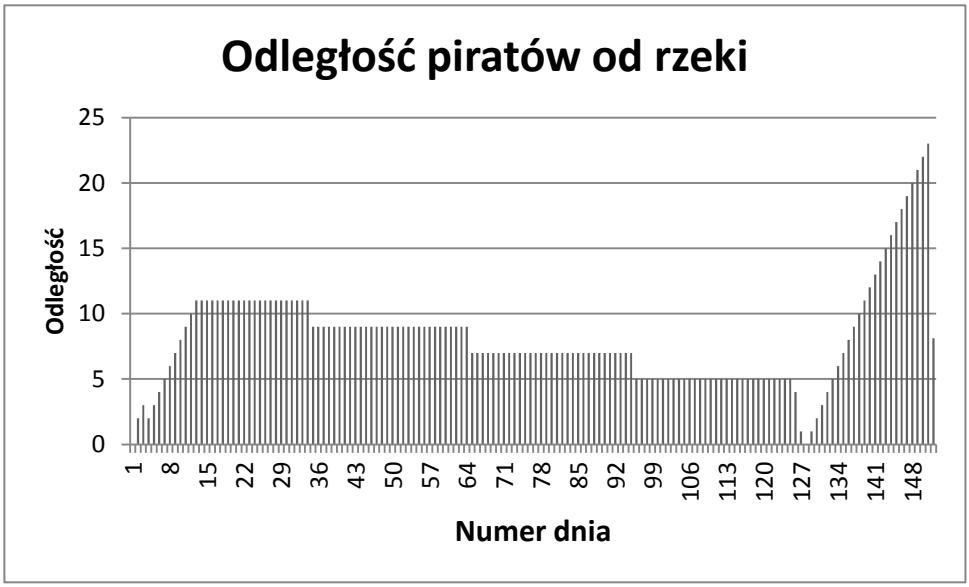
82.3.

260 dublonów

82.4.

125 żołnierzy

82.5.



Średnia odległość: 8,13.

Zadanie 83.

Wiązka zadań Wilki i zające

Klasyczny model Lotki-Volterry opisuje interakcje między populacjami dwóch gatunków. Niech W_n i Z_n oznaczają liczebność populacji odpowiednio wilków i zajęcy, gdzie n jest numerem kolejnego miesiąca. Model definiują następujące dwa wzory:

$$Z_{n+1} = Z_n + a\,Z_n - b\,Z_n W_n,$$
$$W_{n+1} = W_n + b\,Z_n W_n - c\,W_n,$$

gdzie a, b, c są parametrami modelu i mają następujące znaczenie:

- a — współczynnik przyrostu liczby zajęcy,
- b — współczynnik umierania zajęcy na skutek polowań wilków,
- c — współczynnik umierania wilków.

Przyjmujemy następujące początkowe wartości populacji:

$$Z_0 = 100, \qquad W_0 = 30.$$

Rozważamy model Lotki-Volterry z parametrami:

$$a = 0{,}02, \qquad b = 0{,}0005, \qquad c = 0{,}05.$$

Dla potrzeb matematycznych modeli symulacyjnych przyjmujemy, że wartości Z_n i W_n mogą być liczbami niecałkowitymi. Poniżej podano wartości populacji wilków i zajęcy w pierwszych 5 miesiącach symulacji (wartości zaokrąglono do 2 miejsc po przecinku):

n	**Zające** (Z_n)	**Wilki** (W_n)
0	100,00	30,00
1	100,50	30,00
2	101,00	30,01
3	101,51	30,02
4	102,01	30,05
5	102,52	30,08

Wykorzystując dostępne narzędzia informatyczne, wykonaj poniższe zadania. Odpowiedzi (z wyjątkiem wykresu do zadania 3) zapisz w pliku o nazwie `wyniki_wilki.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 3 umieść w pliku `wykres_wilki.xxx`, gdzie *xxx* oznacza rozszerzenie odpowiednie dla formatu pliku.

83.1.

Podaj liczebność populacji wilków i zajęcy po upływie 5 lat (60 miesięcy). Wynik podaj z dokładnością do 2 miejsc po przecinku.

83.2.

Dla podanych powyżej wartości *a*, *b*, *c* i wartości początkowych Z_0, W_0 przebieg symulacji jest następujący:

- na początku następuje przyrost obu populacji;
- gdy wilków przybędzie odpowiednio dużo, to populacja zajęcy zaczyna maleć;
- gdy liczebność populacji zajęcy zaczyna spadać, to zaczyna też spadać liczebność populacji wilków.

Podaj, kiedy zacznie maleć populacja zajęcy, tzn. podaj najmniejszą wartość n, dla której $Z_n < Z_{n-1}$. Podaj też, kiedy nastąpi spadek populacji wilków, tzn. podaj najmniejszą wartość m, dla której $W_m < W_{m-1}$.

83.3.

Utwórz wykres liniowy przedstawiający liczebność populacji wilków i zajęcy w kolejnych miesiącach w ciągu pierwszych 20 lat symulacji. Pamiętaj o czytelnym opisie wykresu.

83.4.

Rozważ symulację liczebności populacji wilków i zajęcy w okresie 40 lat. Podaj najmniejszą i największą liczebność wilków i zajęcy, jaka może być wskazana w rozważanym modelu. Odpowiedzi podaj z dokładnością do 2 miejsc po przecinku.

	Zające	Wilki
Najmniejsza liczebność populacji		
Największa liczebność populacji		

Komentarz do zadania

W zadaniu rozważany jest pewien model symulacji liczebności populacji wilków i zajęcy w pewnym środowisku. Do realizacji tej symulacji można wykorzystać arkusz kalkulacyjny lub napisać odpowiedni program. Na potrzeby rozwiązania zadań 1, 2 i 4 napiszemy programy w języku C++. Zadanie 3 zostanie rozwiązane z wykorzystaniem zarówno programu, jak i arkusza kalkulacyjnego. Ten ostatni wykorzystamy jedynie do sporządzenia wykresu. Podkreślmy, że w języku C++ wszelkie obliczenia numeryczne warto wykonywać na zmiennych typu double. Wówczas wypisywanie wyniku, np. z dokładnością do dwóch miejsc po przecinku, można zrealizować za pomocą funkcji printf w następujący sposób:

```
printf("%.2f", x);  // wypisywanie liczby x
                    // z dokł. do 2 miejsc
```

We wszystkich poniższych programach wykorzystujemy stałe globalne:

```
const double a = 0.02, b = 0.0005, c = 0.05;
```

83.1.

Rozważaną w zadaniu symulację można dość łatwo zaprogramować w języku C++. Wystarczy bowiem napisać pętlę, która realizuje obliczanie liczności populacji wilków i zajęcy wprost z podanych w treści wzorów:

```
double Z = 100.0, W = 30.0;
for (int n=1; n<=5*12; n++)
{
    double nZ = Z + a*Z - b*Z*W;
    double nW = W + b*Z*W - c*W;
    Z = nZ;
    W = nW;
}
```

Istotne jest, aby w każdej iteracji pętli nie zastąpić liczebności populacji wilków i zajęcy przy obliczaniu nowych wartości, gdyż w obu wzorach potrzebujemy starych wartości `Z` i `W`. Dlatego w powyższym programie dodano dwie nowe zmienne `nZ` i `nW`. Ponieważ liczba `n` zmienia się od 1 do 5*12, więc po wykonaniu powyższej pętli w zmiennych `Z` i `W` znajdą się wyniki do zadania.

83.2.

Aby rozwiązać zadanie ponownie utworzymy pętlę, w której obliczać będziemy liczby wilków i zajęcy w kolejnych iteracjach symulacji.

W każdej iteracji będziemy po obliczeniu nowej liczby wilków i zajęcy sprawdzać, czy jest ona mniejsza od liczby z poprzedniej iteracji. Musimy zapamiętać pierwszy taki moment zarówno dla populacji wilków, jak i zajęcy. Pętlę przerwiemy dopiero wtedy, gdy zostanie stwierdzony co najmniej raz spadek populacji wilków i co najmniej raz spadek populacji zajęcy. Wykrycie numerów iteracji, w których występują te zdarzenia, oraz końcowa organizacja całej symulacji będą wówczas wyglądać następująco:

```
double Z = 100.0, W = 30.0;
int k=1, odpZ=0, odpW=0;
while (odpZ==0 || odpW==0)
{
    double nZ = Z + a*Z - b*Z*W;
    double nW = W + b*Z*W - c*W;
    if ( nZ < Z && odpZ == 0 ) odpZ = k;
    if ( nW < W && odpW == 0 ) odpW = k;
    Z = nZ;
    W = nW;
    k++;
}
```

W powyższym fragmencie programu zapamiętujemy numer iteracji, dla której nastąpił pierwszy spadek liczby zajęcy, w zmiennej `odpZ`, a w wypadku wilków — w zmiennej `odpW`. Zmienne te inicjujemy wartością 0, która wskazuje, że moment spadku liczebności nie został jeszcze zaobserwowany. Jeśli w zmiennych `Z` i `nZ` pamiętamy liczbę zajęcy w bieżącej i w poprzedniej iteracji, pierwszy spadek liczby zajęcy wystąpi wówczas, gdy spełniony zostanie warunek `( nZ < Z && odpZ == 0 )`. Gdy warunek ten jest spełniony, zapamiętujemy numer bieżącej iteracji w zmiennej `odpZ`. Zagwarantujemy w ten sposób, że w dalszym przebiegu symulacji zajdzie warunek `odpZ>0`, co wskaże, że wystąpił już spadek liczby zajęcy, i przy kolejnych spadkach (jeśli wystąpią) wartość `odpZ` nie będzie zmieniana. Analogicznie postępujemy w przypadku populacji wilków, aktualizując odpowiednio wartość zmiennej `odpW`.

Pętlę przebiegającą kolejne kroki symulacji przerywamy po zaobserwowaniu zarówno spadku liczby zajęcy, jak i spadku liczby wilków. Sytuacja taka ma miejsce, gdy `odpZ!=0` i `odpW!=0`. A zatem pętlę będziemy kontynuować, dopóki choć jedna ze zmiennych `odpZ`, `odpW` będzie równa zeru. Do rozwiązania zadania pozostaje (po zakończeniu pętli) tylko wypisanie wartości zmiennych `odpZ` i `odpW`.

83.3.

Aby rozwiązać zadanie za pomocą arkusza kalkulacyjnego, należy najpierw przygotować odpowiednie dane do wykresu. Można to zrobić wykorzystując poprzednie rozwiązania, tj. program do symulacji liczby wilków i zajęcy, lub zrealizować całą symulację w arkuszu kalkulacyjnym. Skupimy się na pierwszym sposobie. Przygotowanie danych realizujemy zatem za pomocą następującego programu:

```
double Z = 100.0, W = 30.0;
printf("0 %f %f\n", Z, W);
for (int n=1; n<=20*12; n++)
```

```
    {
        double nZ = Z + a*Z - b*Z*W;
        double nW = W + b*Z*W - c*W;
        Z = nZ;
        W = nW;
        printf("%d %f %f\n", n, Z, W);
    }
```

którego wynik możemy zapisać w pliku tekstowym, powiedzmy `dane.txt`. Jego początkowe wiersze będą wówczas wyglądały następująco:

```
0 100.000000 30.000000
1 100.500000 30.000000
2 101.002500 30.007500
3 101.507134 30.022541
4 102.013525 30.045165
5 102.521289 30.075414
```

W każdym wierszu znajdują się trzy liczby: numer iteracji, liczba zajęcy, liczba wilków. Warto podkreślić, że dwie ostatnie liczby podane są w formacie zmiennoprzecinkowym, z wykorzystaniem symbolu kropki do oddzielenia części całkowitej od ułamkowej. Jest to istotne przy importowaniu danych do arkusza kalkulacyjnego. Można to bowiem zaznaczyć w zaawansowanych opcjach podczas dokonywania importu. Podobnie należy pamiętać, aby przy imporcie zaznaczyć, że dane liczby są oddzielone pojedynczym odstępem (w tym celu należy wybrać separator „spacja"). Dalej zakładamy, że dysponujemy arkuszem, w którym:

- w komórkach o adresach A1:A241 znajdują się kolejne numery iteracji, a więc liczby całkowite od 0 do 240,
- w komórkach o adresach B1:B241 znajdują się liczby rzeczywiste określające populację wilków w kolejnych iteracjach,
- w komórkach o adresach C1:C241 znajdują się liczby rzeczywiste określające populację zajęcy w kolejnych iteracjach.

Sporządzenie wykresu można zrealizować za pomocą arkusza kalkulacyjnego Excel w następujący sposób:

1. Wybieramy Wstaw → Wykres liniowy.
2. Dodajemy dwie serie danych: B1:B241 oraz C1:C241, nadając im odpowiednie nazwy.
3. Dla czytelności wykresu zmieniamy również etykiety na osi Ox, wybierając dane A1:A241.
4. Nadajemy wykresowi odpowiedni tytuł.

83.4.

Aby rozwiązać zadanie, wystarczy zmodyfikować program napisany do poprzednich zadań. Mianowicie długość symulacji należy ustawić na 40 lat. Aby wyznaczyć minimalną i maksymalną liczbę wilków i zajęcy, wprowadzamy pomocniczo cztery zmienne: `minW`, `maxW`, `minZ` i `maxZ`. W każdej iteracji symulacji próbujemy poprawić te wartości w następujący sposób (skorzystano w nim z funkcji *min* i *max*):

```
    double Z = 100.0, W = 30.0;
```

```
double minZ, maxZ, minW, maxW;
minZ = maxZ = Z; minW = maxW = W;
for (int n=1; n<=40*12; n++)
{
    double nZ = Z + a*Z - b*Z*W;
    double nW = W + b*Z*W - c*W;
    Z = nZ;
    W = nW;
    minZ = min( minZ, Z ); maxZ = max( maxZ, Z );
    minW = min( minW, W ); maxW = max( maxW, W );
}
```

Po zakończeniu powyższej pętli wypisujemy wyniki z dokładnością do dwóch miejsc po przecinku za pomocą instrukcji:

```
printf("MIN : %.2f | %.2f\n", minZ, minW);
printf("MAX : %.2f | %.2f\n", maxZ, maxW);
```

Zadanie 84.

Wiązka zadań *LPG*

Pan Binarny założył instalację gazową w swoim samochodzie, dzięki czemu mógł wykorzystywać do jazdy 2 rodzaje paliwa. Parametry pojazdu są przedstawione w poniższej tabeli:

paliwo	pojemność zbiornika	zużycie paliwa (spalanie)
Pb95	45 litrów	6 litrów na 100 km
LPG	30 litrów	9 litrów na 100 km

Pan Binarny postanowił prowadzić rejestr długości pokonywanych tras w kolejnych dniach 2014 roku. Dane te zostały zebrane w pliku `lpg.txt`. Są to data wyjazdu (`data`) oraz liczba kilometrów przejechanych tego dnia (`km`). Pierwszy wiersz jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.

Przykład

```
data          km
2014-01-01    159
2014-01-02    82
2014-01-03    108
```

1 stycznia 2014 r. zbiorniki z paliwem (Pb95 i LPG) były zatankowane do pełna. Codziennie rano pan Binarny decydował, w jaki sposób będzie zużywał paliwo. Jeżeli w zbiorniku LPG było więcej niż 15 litrów, to do jazdy wykorzystywał tylko paliwo LPG, w przeciwnym razie połowę trasy pokonywał, korzystając z paliwa Pb95, a połowę — z paliwa LPG.

Liczbę zużytych litrów paliwa na trasie obliczał ze wzoru

$$litry = \frac{spalanie * km}{100},$$

gdzie *spalanie* oznacza zużycie paliwa w litrach na 100 km (zgodnie z parametrami zużycia paliwa w tabeli powyżej), a *km* oznacza liczbę przejechanych kilometrów, przy czym wyniki **zaokrąglał do dwóch miejsc po przecinku**.

Bez względu na rodzaj paliwa tankowanie wykonywane było **po przejechaniu trasy, wieczorem. W każdy czwartek**, o ile w zbiorniku z Pb95 znajdowało się poniżej 40 litrów, pan

Binarny tankował do pełna Pb95. Paliwo LPG było tankowane do pełna wtedy, gdy w zbiorniku LPG znajdowało się mniej niż 5 litrów tego paliwa i odbywało się **niezależnie od dnia tygodnia**.

Wykorzystując dane zawarte w pliku `lpg.txt` i dostępne narzędzia informatyczne, wykonaj poniższe zadania. Odpowiedzi (za wyjątkiem wykresu do zadania 3) zapisz w pliku o nazwie `wyniki_lpg.txt`. Wyniki do każdego zadania poprzedź numerem je oznaczającym. Wykres do zadania 3 umieść w pliku `wykres_lpg.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku.

Uwaga: Dane podane w pliku `lpg.txt` wykluczają sytuację, że panu Binarnemu kiedykolwiek zabrakło paliwa na trasie i w związku z tym musiał on zmieniać decyzję co do rodzaju używanego paliwa.

84.1.

Podaj, ile razy pan Binarny tankował paliwo Pb95, a ile razy LPG.

Podaj liczbę dni, w których pan Binarny korzystał podczas jazdy wyłącznie z paliwa LPG.

84.2.

Podaj dzień, w którym w zbiorniku LPG po raz pierwszy było rano mniej niż 5,25 litra paliwa.

84.3.

Utwórz wykres, w którym porównasz stan zbiornika LPG przed podróżą ze stanem zbiornika LPG po podróży dla pierwszych 31 dni. Pamiętaj o czytelnym opisie wykresu.

84.4.

Przyjmijmy, że w całym rozważanym okresie koszt 1 litra LPG jest równy 2,29 zł, a koszt 1 litra Pb95 wynosi 4,99 zł. Koszt instalacji gazowej wynosi 1600 zł.

Oblicz dla rozważanego okresu koszt eksploatacji samochodu z wykorzystaniem instalacji gazowej (uwzględnij koszt instalacji oraz zużyte LPG) oraz koszt eksploatacji samochodu wykorzystującego tylko paliwo Pb95 (w oparciu o zużytą ilość Pb95).

W swoich obliczeniach dotyczących eksploatacji z wykorzystaniem instalacji gazowej przyjmij wszystkie opisane założenia dotyczące tankowania i sposobu przejazdów pana Binarnego. Zauważ, że w obliczeniach związanych z wykorzystaniem wyłącznie paliwa Pb95 nie ma znaczenia dzień tankowania, istotny jest jedynie roczny koszt eksploatacji.

Podaj oba koszty eksploatacji. Podczas obliczeń wszystkie koszty oraz wyniki końcowe zaokrąglaj do dwóch miejsc po przecinku.

Zadanie 85.
Wiązka zadań *Oscypki*

Oscypek to ser sezonowy. Robi się go z mleka owiec wypasanych na halach. Sezon wypasu trwa od 23 kwietnia (św.Wojciecha) do 29 września (św. Michała).

Baca ma stado owiec składające się z **600** sztuk. Każda owca przez pierwsze 7 dni sezonu daje **0,5 l** mleka dziennie. Począwszy od ósmego dnia (czyli 30.04) ta ilość mleka wzrasta skokowo o **4%** co **7** dni. Tak dzieje się do 24 czerwca (św.Jana) włącznie. Od 25 czerwca

co 7 dni ilość mleka od jednej owcy zmniejsza się skokowo o **10%** w stosunku do ilości mleka z poprzedniego 7-dniowego okresu. Proces ten trwa do końca wypasu owiec. Liczbę odnoszącą się do rzeczywistej ilości mleka oddawanego przez jedną owcę w każdym dniu, wyliczoną zgodnie z powyższymi regułami, należy zaokrąglić do dwóch miejsc po przecinku.

Ser jest wyrabiany z mleka zebranego od owiec poprzedniego dnia. Jego wyrób trwa jedną dobę, a następnie jest on wędzony przez 4 doby i kolejnego dnia jest gotowy do sprzedaży. Na wytworzenie jednego sera potrzeba **6** litrów mleka. Z mleka, jakie jest do dyspozycji, baca produkuje tylko całe sery i maksymalną możliwą ich liczbę. Reszta mleka, niezużyta w danym dniu do produkcji serów, jest wykorzystywana do innych celów.

Każdego dnia w okresie od 29.04 do 29.09 włącznie do bacówki przychodzą po sery turyści. W dni powszednie kupują 36 serów dziennie (o ile baca ma tyle na sprzedaż), a w dni weekendu (sobotę, niedzielę) — 100 serów dziennie. Jeśli danego dnia baca ma na sprzedaż mniej serów niż chcą kupić turyści, to wykupują je wszystkie, jeśli więcej, to baca przechowuje sery i ma je do sprzedania w kolejnych dniach. Rozważmy, jak wyglądało wyrabianie i sprzedaż serów w bacówce w 2014 roku.

Przykład

23.04 — pierwsze dojenie owiec,

24.04 — wyrób serów z mleka z dnia 23.04,

25.04 – 28.04 — wędzenie serów zrobionych 24.04,

29.04 — pierwsze sery gotowe do sprzedaży.

We wtorek 29.04.2014 do bacówki od rana przychodzili turyści i kupowali gotowe sery. Baca miał 50 gotowych serów, z których turyści kupili 36 sztuk.

Wykorzystując dostępne narzędzia informatyczne, wykonaj poniższe polecenia. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki.txt` (z wyjątkiem wykresu do zadania 4). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 4 umieść w pliku `wykres.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku z Twoimi rozwiązaniami.

85.1.

Podaj liczbę litrów mleka, jaką uzyskał baca ze swojego stada owiec w okresie od 23.04.2014 do 29.09.2014 włącznie.

85.2.

Podaj łączną liczbę serów, które turyści byli gotowi zakupić u bacy począwszy od dnia, kiedy sery były gotowe do sprzedaży, czyli od 29.04.2014, do ostatniego dnia wypasu, czyli do 29.09.2014.

85.3.

Podaj dzień, w którym po raz pierwszy w sezonie (od 29.04.2014 do 29.09.2014) baca miał mniej gotowych serów, niż chcieli kupić turyści. Ile było takich dni w całym sezonie?

85.4.

Wykonaj zestawienie, w którym dla każdego miesiąca, od kwietnia do września włącznie, podasz liczbę sprzedanych w tym miesiącu oscypków. Do wykonanego zestawienia utwórz wykres kolumnowy. Pamiętaj o czytelnym opisie wykresu.

85.5.

Podaj najmniejszą liczbę owiec, jaką musiałby posiadać baca, aby każdego dnia od 29 kwietnia do 29 września zaspokoić popyt turystów na oscypki.

85.6.

W przepisie na produkcję oscypka dopuszcza się dodawanie mleka krowiego do mleka owczego (maksymalniy udział mleka krowiego może wynieść 40%). Podaj liczbę serów (gotowych do sprzedaży), które można byłoby wyprodukować w całym okresie wypasu przy opisanych wyżej warunkach uzyskiwania mleka owczego, jeżeli ser tworzony byłby:

- tylko z mleka owczego,
- z mieszanki składającej się z 80% mleka owczego i 20% mleka krowiego,
- z mieszanki składającej się z 60% mleka owczego i 40% mleka krowiego.

Zadanie 86.

Wiązka zadań *Wybory*

Jedną z metod obsadzania mandatów w systemach wyborczych opartych na proporcjonalnej reprezentacji partii politycznych jest metoda Sainte-Laguë. W metodzie tej kolejne mandaty w danym okręgu wyborczym przydzielane są po kolei. Przed przydzieleniem każdego mandatu wyznacza się współczynnik w_K dla każdego komitetu wyborczego K równy

$$w_k = \frac{g_k}{2m_K+1},$$

gdzie:

g_k — liczba głosów zdobytych przez komitet K w rozważanym okręgu,

m_K — liczba mandatów przydzielonych komitetowi K do tej pory.

Kolejny mandat otrzymuje ten komitet wyborczy K, który ma największą wartość współczynnika w_k.

Uwaga. Jeśli kilka komitetów ma taką samą wartość współczynnika w_k. Sposób rozstrzygnięcia takich remisów nie ma wpływu na rozwiązania poniższych zadań, dlatego takiego przypadku nie opisujemy.

Przykład

Przyjmijmy, że do wyborów zgłosiły się dwa komitety wyborcze $K1$ i $K2$, które otrzymały odpowiednio $g_{K1} = 810$ oraz $g_{K2} = 300$ głosów w okręgu, w którym do obsadzenia są 4 mandaty.

Mandat pierwszy otrzymuje $K1$, gdyż $m_{K1} = 0$, $m_{K2} = 0$ oraz:

$$w_{K1} = \frac{810}{1} = 810,\ w_{K2} = \frac{300}{1} = 300.$$

Mandat drugi otrzymuje $K2$, gdyż $m_{K1} = 1$, $m_{K2} = 0$ oraz:

$$w_{K1} = \frac{810}{3} = 270,\ w_{K2} = \frac{300}{1} = 300.$$

Mandat trzeci otrzymuje $K1$, gdyż $m_{K1} = 1$, $m_{K2} = 1$ oraz:

$$w_{K1} = \frac{810}{3} = 270,\ w_{K2} = \frac{300}{3} = 100.$$

Mandat czwarty również otrzymuje $K1$, gdyż $m_{K1} = 2$, $m_{K2} = 1$ oraz:

$$w_{K1} = \frac{810}{5} = 162,\ w_{K2} = \frac{300}{3} = 100.$$

W rezultacie komitet $K1$ otrzyma 3 mandaty, a komitet $K2$ otrzyma 1 mandat.

Wybory odbyły się w 20 okręgach wyborczych, zgłosiło się do nich pięć komitetów wyborczych o nazwach $K1$, $K2$, $K3$, $K4$ i $K5$. Do obsadzenia jest 400 mandatów poselskich. Plik `dane_wybory.txt` zawiera wyniki przeprowadzonych wyborów. Każdy wiersz pliku zawiera dane dotyczące jednego okręgu wyborczego: nazwę okręgu i 5 liczb oznaczających liczbę głosów oddanych na kolejne komitety wyborcze ($K1$, $K2$, $K3$, $K4$ i $K5$) w tym okręgu (zakładamy, że każdy głos wskazuje na jeden komitet, nie ma głosów nieważnych). Wartości w wierszu oddzielone są spacjami.

Przykład

```
A1 16573 13009 19177 21253 9656
B2 24574 10394 9756 13299 3464
```

Wykorzystując dane zawarte w tym pliku i dostępne narzędzia informatyczne, wykonaj poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_wybory.txt` (z wyjątkiem wykresu do zadania 1). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 1 umieść w pliku `wyniki_wybory.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku.

86.1.

Podaj, ile ogółem głosów oddano (na wszystkie komitety razem) w każdym z okręgów wyborczych. Na tej podstawie sporządź wykres kolumnowy porównujący liczby głosów oddanych w poszczególnych okręgach. Pamiętaj o prawidłowym i czytelnym opisie wykresu.

86.2.

Poparcie komitetu wyborczego w danym okręgu to procent głosów zdobytych w tym okręgu, w zaokrągleniu do dwóch miejsc po przecinku. *Matecznikiem* komitetu wyborczego nazywamy okręg, w którym komitet ten zdobył największe procentowe poparcie w porównaniu z jego poparciem w innych okręgach. Dla każdego komitetu wyborczego podaj nazwę okręgu będącego jego *matecznikiem*.

Przykład

Przyjmijmy, że wybory odbyły się tylko w dwóch okręgach wyborczych $A1$ i $B2$, z następującymi wynikami:

	$K1$	$K2$	$K3$	$K4$	$K5$
$A1$	100	300	400	50	150
$B2$	150	500	750	400	200

Wówczas matecznikiem komitetu $K1$ jest okręg $A1$ (100 głosów, poparcie 10%), mimo że uzyskał on większą liczbę głosów w okręgu $B2$ (150 głosów, poparcie 7,5%).

86.3.

Przyjmijmy, że w każdym okręgu wyborczym jest do obsadzenia 20 mandatów, przy czym przyznawane są one w oparciu o rozkład głosów w tym okręgu zgodnie z metodą Sainte-Laguë. Dla każdego z komitetów wyborczych (*K*1, *K*2, *K*3, *K*4, *K*5) wskaż maksymalną liczbę mandatów zdobytych przez ten komitet w jednym okręgu.

86.4.

Pierwsza litera nazwy każdego okręgu oznacza region, w którym ten okręg się znajduje: *A*, *B*, *C* lub *D*. Komisja wyborcza rozważa różne warianty podziału mandatów:

a) standardowy: w każdym okręgu mandaty obsadzane są osobno, przyznaje się po 20 mandatów na okręg metodą Sainte-Laguë.

b) regionalny: wszystkie okręgi z tego samego regionu łączone są w duży okręg i w tak utworzonych większych okręgach (*A*, *B*, *C* i D) obsadza się po 100 mandatów metodą Sainte-Laguë.

Podaj liczbę mandatów zdobytych w obu wariantach przez każdy komitet w całym kraju.

86.5.

Rozważmy sytuację, w której o mandaty rywalizują dwa komitety wyborcze, *Q* i *R*, w jednym okręgu wyborczym, w którym każdy spośród 100 000 wyborców oddaje głos na komitet *Q* albo komitet *R*. Do obsadzenia jest 2*m* mandatów. Ustal najmniejszą liczbę głosów, jakie musi zdobyć komitet *Q*, aby zdobyć dokładnie połowę, czyli *m* mandatów, dla *m*=10, *m*=20 i *m*=50.

Zadanie 87.

Wiązka zadań *U-977*

Okręt podwodny U-977 wypłynął w swój pierwszy patrol bojowy z Kristiansand w czwartek 3 maja 1945 o godzinie 00:00 z zadaniem dotarcia do Southampton, a celem wyprawy było zatopienie jak największej liczby jednostek alianckich. W czasie rejsu na okręt dotarła wiadomość o kapitulacji Niemiec. W tej sytuacji dowódca wraz z większością załogi zdecydowali się płynąć do Argentyny i rozpoczęli długi rejs na południową półkulę.

Ze względu na trudności w podróży 14 lipca 1945, tj. 73. dnia podróży, załoga postanowiła się zatrzymać na 24-godzinny odpoczynek w pobliżu archipelagu Wysp Zielonego Przylądka (14 lipca 1945, od godziny 0:00 do godziny 24:00). Mimo że w tym czasie okręt stał w miejscu, to wszystkie jego podzespoły pracowały, co powodowało ich naturalny proces zużywania się.

18 sierpnia 1945 o godzinie 00:00, po 107 dniach podróży, U-977 zawinął do Mar del Plata, gdzie załoga poddała się argentyńskiej marynarce.

Okręt był wyposażony w silniki elektryczne i akumulatory umożliwiające rejs w zanurzeniu, początkowo z prędkością średnią 4 węzłów (4 mile morskie na godzinę) oraz silniki spalinowe, umożliwiające rejs na powierzchni początkowo z prędkością średnią 10 węzłów.

Pojemność akumulatorów wystarczała na maksymalnie 20 godzin pracy silników elektrycznych. Podczas rejsu na silnikach spalinowych jednocześnie ładowane były akumulatory silników elektrycznych. Pełne naładowanie akumulatorów następowało po czterech godzinach używania silników spalinowych, niezależnie od prędkości.

Aby zmniejszyć prawdopodobieństwo wykrycia okrętu przez siły alianckie o ile było to możliwe okręt płynął w zanurzeniu korzystając z silników elektrycznych.

W związku z różnego rodzaju awariami i ogólnym zużyciem mechanizmów prędkość średnia na każdym z napędów zmniejszała się codziennie o 1%, niezależnie od czasu ich wykorzystywania. Począwszy od 15 lipca, w związku z uszkodzeniem akumulatorów, okręt płynął codziennie po 11 godzin w wynurzeniu. Pozostałą część doby przebywał w zanurzeniu, nie zmieniając pozycji nad dnem. Ostatniej doby rejsu, w związku z trudnościami nawigacyjnymi okręt płynął tylko 9 godzin.

Wszystkie określenia godzin w zadaniu wyrażone niezależnie od miejsca zdarzenia podane są według czasu uniwersalnego (GMT).

Wykorzystując dostępne narzędzia informatyczne, rozwiąż podane poniżej zadania. Obliczenia wykonuj na liczbach rzeczywistych i nie zaokrąglaj wyników częściowych. Do oceny oddaj dokument `wyniki.txt` z zapisanymi odpowiedziami do poszczególnych zadań oraz plik zawierający obliczenia.

87.1.

Którego dnia podróży średnia prędkość rejsu w zanurzeniu po raz pierwszy wyniosła poniżej 3 węzłów?

87.2.

Jaką drogę (w milach morskich) przebył okręt na trasie z Kristiansand do Mar del Plata? Odpowiedź podaj w zaokrągleniu do jednego miejsca po przecinku.

87.3.

Oblicz drogę (w milach morskich) przebytą przez okręt każdego dnia i sporządź wykres ilustrujący odległość przebytą w kolejnych dniach rejsu. Zadbaj o czytelny opis wykresu.

87.4.

Którego dnia zakończyłby się rejs okrętu, gdyby po 14 lipca okręt płynął już ciągle dokładnie tym samym kursem i będąc w zanurzeniu utrzymywał prędkość według wcześniejszego harmonogramu (20 godzin w zanurzeniu na silnikach elektrycznych i 4 godziny na powierzchni na silnikach spalinowych)?

87.5.

Dokładnie w tym samym czasie co U-977 (tj. 3 maja 1945 o godzinie 00:00) z Kristiansand wypłynął też okręt U-997 o tych samych parametrach technicznych. Jego dowódca, podobnie jak dowódca U-977, postanowił płynąć do Ameryki Południowej. Mając świadomość przeszkód/problemów, jakie mogą nastąpić w podróży dowódca postanowił płynąć z prędkością

równą 90% maksymalnej oraz w każdą niedzielę, poza pierwszą, będąc w zanurzeniu, wyłączyć na dwie godziny wszystkie silniki i dokonać dokładnego ich przeglądu oraz konserwacji. Dzięki temu ubytek uzyskiwanej prędkości wynosił tylko 0,3% na dobę na każdym z napędów i nie trzeba było robić 24-godzinnej przerwy w podróży. Można było, bowiem przez cały czas poza dniami konserwacji płynąć po 20 godzin na dobę w zanurzeniu, a przez 4 godziny płynąć na powierzchni i ładować akumulatory. Okręt płynął inną drogą, ale również zakończył swój rejs 17 sierpnia 1945 o północy. Podaj, w zaokrągleniu do jednego miejsca po przecinku, jaką odległość przebył okręt do 17 sierpnia 1945, do godziny 24:00.

Zadanie 88.

Wiązka zadań *Sprzedaż choinek*

Firma produkcyjno-handlowa „Świerk" sprzedaje choinki przed Bożym Narodzeniem. Co roku rozpoczyna ich sprzedaż pierwszego i kończy dwudziestego czwartego grudnia. Firma dysponuje jednym samochodem przeznaczonym do transportu choinek. Samochód wieczorem wyjeżdża z plantacji i dojeżdża na plac sprzedaży rano przed jej rozpoczęciem. W ciągu dnia wraca na plantację.

Zaobserwowano, że ze względu na zainteresowanie klientów firma może sprzedać danego dnia liczbę choinek wyznaczoną wg wzoru:

$$L = \lfloor(-d^2 + 40d + 50)/10\rfloor,$$

gdzie d oznacza numer dnia sprzedaży (1 grudnia jest 1. dniem sprzedaży, a 24 grudnia jest 24. dniem sprzedaży). Jeśli na placu jest co najmniej L choinek, to L choinek zostaje sprzedanych. Gdy na placu znajduje się mniej drzewek, niż klienci gotowi byliby kupić (mniej niż L), to część klientów wybierała inne punkty sprzedaży, gdzie wybór drzewek był większy. W takim dniu firmie udaje się maksymalnie sprzedać zaokrąglone w dół do liczby całkowitej 90% choinek, które znalazły się na placu rano po przyjęciu dostawy. Firma zaplanowała dostawy po 50 drzewek w następujących dniach: pierwszego grudnia, czwartego grudnia oraz następnie w każdy dzień o numerze parzystym aż do 24 grudnia włącznie.

Wykorzystując dostępne narzędzia informatyczne, rozwiąż następujące zadania. Do oceny oddaj dokument `wyniki.txt` z odpowiedziami do poszczególnych zadań oraz plik z przeprowadzonymi obliczeniami.

88.1.

Ile choinek zostanie w sumie sprzedanych w całym okresie sprzedaży i ile choinek pozostanie na placu wieczorem 24 grudnia?

88.2.

Wykonaj wykres kolumnowy przedstawiający dla każdego dnia sprzedaży liczbę choinek które znajdowały się tego dnia rano na placu (już po dostawie), z podziałem na sprzedane później tego dnia i pozostałe na placu wieczorem (do sprzedaży w kolejnych dniach).

88.3.

Którego dnia firma po raz pierwszy sprzeda mniej choinek, niż wynikałoby to z zainteresowania klientów (mniej niż L)?

88.4.

Zaproponuj przykładowy harmonogram dostaw po 50 drzewek jednego dnia, zgodnie z którym firma sprzeda możliwie najwięcej choinek i jednocześnie ostatniego dnia na placu pozostanie mniej niż 50 choinek. Podaj liczbę dostaw, dni tych dostaw oraz liczbę choinek sprzedanych w całym okresie sprzedaży.

88.5.

Podaj maksymalną liczbę choinek, jaka powinna być przy opisanym popycie codziennie (czyli 24 razy) dostarczana, aby 24 grudnia wieczorem na placu zostało mniej choinek niż było ich przywiezionych. Podaj liczbę choinek, które pozostaną na placu 24 grudnia wieczorem.

2.3. Przetwarzanie i tworzenie informacji

Zadanie 89.

Wiązka zadań *Punkty rekrutacyjne*

W postępowaniu rekrutacyjnym przy przyjmowaniu do pierwszej klasy szkoły ponadgimnazjalnej bierze się pod uwagę wyniki egzaminu gimnazjalnego, oceny ze świadectwa ukończenia gimnazjum oraz dodatkowe osiągnięcia ucznia. Maksymalnie można uzyskać 100 punktów, w tym:

1) 50 punktów za wyniki egzaminu gimnazjalnego, które oblicza się zgodnie z poniższymi zasadami:
   a) liczba punktów rekrutacyjnych za każdy wynik procentowy z zakresu: języka polskiego, historii i wiedzy o społeczeństwie, matematyki, przedmiotów przyrodniczych, języka obcego jest równa liczbie określającej wynik procentowy, uzyskany z danego zakresu, podzielonej przez dziesięć.
   b) liczba punktów rekrutacyjnych za wyniki egzaminu gimnazjalnego jest równa sumie punktów rekrutacyjnych uzyskanych z wymienionych wyżej zakresów egzaminu.

      Przykład:

      Uczeń uzyskał następujące wyniki z egzaminu gimnazjalnego z poszczególnych zakresów:

      — język polski — 74%

      — historia i wiedza o społeczeństwie — 88%

      — matematyka — 60%

      — przedmioty przyrodnicze — 71%

      — język obcy — 79%

      Liczba punktów rekrutacyjnych wynosi: 74 : 10 + 88 : 10 + 60 : 10 + 71 : 10 + 79 : 10 = 7,4 + 8,8 + 6 +7,1 + 7,9 = 37,2
2) 40 punktów za oceny z języka polskiego, matematyki, biologii i geografii otrzymane na świadectwie ukończenia gimnazjum, przyznawanych zgodnie z przeliczeniem przedstawionym w tabeli:

liczba punktów	ocena
0	dopuszczająca
4	dostateczna
6	dobra
8	bardzo dobra
10	celująca

3) 10 punktów za dodatkowe osiągnięcia, w tym:
   a) 2 punkty za ukończenie gimnazjum ze wzorową (6) oceną z zachowania,
   b) maksymalnie 8 punktów za szczególne osiągnięcia ucznia wymienione na świadectwie ukończenia gimnazjum.

W pliku `punkty_rekrutacyjne.txt` znajduje się 514 wierszy z danymi uczniów, potrzebnymi do przeprowadzenia rekrutacji. Pierwszy wiersz pliku jest wierszem nagłówkowym. Kolejne wiersze składają się z następujących informacji: nazwisko ucznia (`Nazwisko`), imię ucznia (`Imie`), liczba punktów za osiągnięcia (`Osiagniecia`), ocena z zachowania (`Zachowanie`), oceny na świadectwie ucznia z czterech przedmiotów branych pod uwagę przy rekrutacji: języka polskiego (`JP`), matematyki (`Mat`), biologii (`Biol`) i geografii (`Geog`), oraz wyniki egzaminów gimnazjalnych z zakresów: języka polskiego (`GHP`), historii i wiedzy o społeczeństwie (`GHH`), matematyki (`GMM`), przedmiotów przyrodniczych (`GMP`), języka obcego (`GJP`). Wszystkie dane liczbowe są prezentowane w postaci liczb całkowitych nieujemnych, przy czym oceny z zachowania i z przedmiotów mieszczą się w przedziale <2, 6>, wyniki za osiągnięcia w przedziale <0, 10>, zaś wyniki procentowe egzaminów gimnazjalnych w przedziale <0, 100>. Dane w wierszach pliku rozdzielone są średnikami.

Przykład fragmentu danych

```
Nazwisko;Imie;Osiagniecia;Zachowanie;JP;Mat;Biol;Geog;GHP;GHH;GMM;GMP;GJP
Swistek;Damian;0;4;4;5;6;6;62;13;26;67;62
Kowalik;Mateusz;7;4;4;2;5;6;90;8;21;52;33
```

Przykład obliczenia liczby punktów rekrutacyjnych

Swistek Damian – liczba punktów rekrutacyjnych = 0+0+(6+8+10+10)+(62+13+26+67+62)/10=57
Kowalik Mateusz – liczba punktów rekrutacyjnych = 7+0+(6+0+8+10)+ (90+8+21+52+33)/10=51,4

Korzystając z dostępnych narzędzi informatycznych, rozwiąż poniższe zadania. Odpowiedzi zapisz do pliku `wyniki_punkty.txt` (z wyjątkiem wykresu do zadania 4), a każdą odpowiedź poprzedź cyfrą oznaczającą to zadanie.

89.1.

Utwórz zestawienie uczniów, którzy spełniają jednocześnie następujące warunki: mają 0 punktów za osiągnięcia, co najmniej bardzo dobrą ocenę (5) z zachowania oraz średnią z przedmiotów punktowanych w rekrutacji większą od 4. Zestawienie uporządkuj alfabetycznie według nazwisk. Podaj imiona i nazwiska pierwszych 5 osób.

89.2.

Oblicz zgodnie z zasadami opisanymi w treści zadania, ile punktów rekrutacyjnych uzyskał każdy z uczniów. Podaj liczbę punktów rekrutacyjnych występującą najczęściej oraz listę nazwisk i imion uczniów, którzy uzyskali tę liczbę punktów.

89.3.

Utwórz zestawienie zawierające imiona i nazwiska uczniów, którzy uzyskali 100% punktów z co najmniej 3 zakresów egzaminu gimnazjalnego.

89.4.

Utwórz zestawienie zawierające rozkład ocen dla każdego punktowanego w rekrutacji przedmiotu (przykład schematu zestawienia poniżej). Dla otrzymanego zestawienia wykonaj wykres procentowy skumulowany. Zadbaj o czytelny opis wykresu.

Liczba ocen	**Język polski**	**Matematyka**	**Biologia**	**Geografia**
dopuszczających				
dostatecznych				
dobrych				
bardzo dobrych				
celujących				

89.5.

Podaj liczbę uczniów, którzy uzyskali łącznie więcej punktów rekrutacyjnych za oceny z przedmiotów punktowanych i dodatkowe osiągnięcia (w tym ocenę z zachowania) niż za wyniki egzaminu gimnazjalnego.

Komentarz do zadania

Po wczytaniu danych z pliku do arkusza kalkulacyjnego są one umieszczone w następujących kolumnach:

punkty_rekrutacyjne.xlsx

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nazwisko	Imie	Osiągniecia	Zachowanie	JP	Mat	Biol	Geog	GHP	GHH	GMM	GMP	GJP
2	Swistek	Damian	0	4	4	5	6	6	62	13	26	67	62
3	Kowalik	Mateusz	7	4	4	2	5	6	90	8	21	52	33
4	Hintzke	Nikola	7	4	4	6	6	5	96	99	16	85	65

89.1.

W zadaniu należy wyszukać uczniów, którzy spełniają wszystkie trzy warunki:

- 0 punktów za osiągnięcia,
- co najmniej bardzo dobra ocena (5) z zachowania,
- średnia z przedmiotów punktowanych w rekrutacji większa od 4.

W tym celu konstruujemy formułę warunkową, której wartością jest 1, gdy jest spełniona koniunkcja podanych warunków:

```
=JEŻELI(ORAZ(C2=0;ORAZ(D2>=5;ŚREDNIA(E2:H2)>4));1;0)
```

Następnie stosujemy filtr, aby wybrać wiersze, w których powyższa formuła przyjmuje wartość 1, i otrzymaną listę sortujemy alfabetycznie według nazwisk.

	A	B	C
1	Nazwisko	Imie	zad. 1
11	Bialaszewski	Wiktor	1
82	Cicherski	Szymon	1
181	Florek	Sandra	1
202	Kaminski	Mikolaj	1
222	Krol	Malgorzata	1
250	Kurowska	Karolina	1
289	Pettka	Jan	1
376	Romanowska	Julia	1
441	Smal	Franciszek	1
456	Szczepanska	Emilia	1
463	Szubarczyk	Dawid	1
476	Toczek	Antonina	1

Jako odpowiedź kopiujemy imiona i nazwiska pięciu pierwszych uczniów.

89.2.

Aby obliczyć sumę punktów rekrutacyjnych, obliczamy kolejno punkty za:

- wyniki egzaminów gimnazjalnych,
- przedmioty punktowane,
- dodatkowe osiągnięcia.

Do obliczenia punktów za wyniki egzaminów gimnazjalnych posłuży formuła:

```
=SUMA(I2:M2)/10
```

Punkty za przedmioty punktowane obliczamy, stosując pomocniczą tabelę z liczbą punktów za poszczególne oceny, umieszczoną w arkuszu o nazwie `'punkty za oceny'`:

	A	B
1	ocena	liczba punktów
2	2	0
3	3	4
4	4	6
5	5	8
6	6	10

Dla każdego przedmiotu utworzymy formułę, która wyszuka liczbę punktów odpowiadającą uzyskanej ocenie. Dla języka polskiego jest to formuła:

```
=WYSZUKAJ.PIONOWO(E2; 'punkty za oceny'! $A$2:$B$6;2;FAŁSZ)
```

Następnie sumujemy wartości tych formuł dla wszystkich czterech punktowanych przedmiotów.

Punkty za dodatkowe osiągnięcia są obliczane za pomocą formuły, w której do liczby punktów za osiągnięcia, podanych w pliku z danymi, dodajemy 2 punkty, gdy ocena z zachowania jest wzorowa (równa 6):

```
=C2+JEŻELI(D2=6;2;0)
```

Po obliczeniu sumy punktów rekrutacyjnych znajdujemy najczęściej występującą wartość za pomocą funkcji `WYST.NAJCZĘŚCIEJ.WART`. Następnie stosujemy filtr do wybrania uczniów, którzy taką liczbę punktów rekrutacyjnych (wyznaczoną w sposób opisany w poprzednim zdaniu) uzyskali.

89.3.

Dla każdego ucznia zliczamy, z ilu egzaminów uzyskał on wynik 100. Służy do tego formuła:

`=LICZ.JEŻELI(I2:M2;"=100")`

Następnie stosujemy filtr do wybrania uczniów, którzy z co najmniej trzech egzaminów uzyskali wynik 100.

89.4.

Rozwiązaniem zadania jest zestawienie o poniższej zawartości:

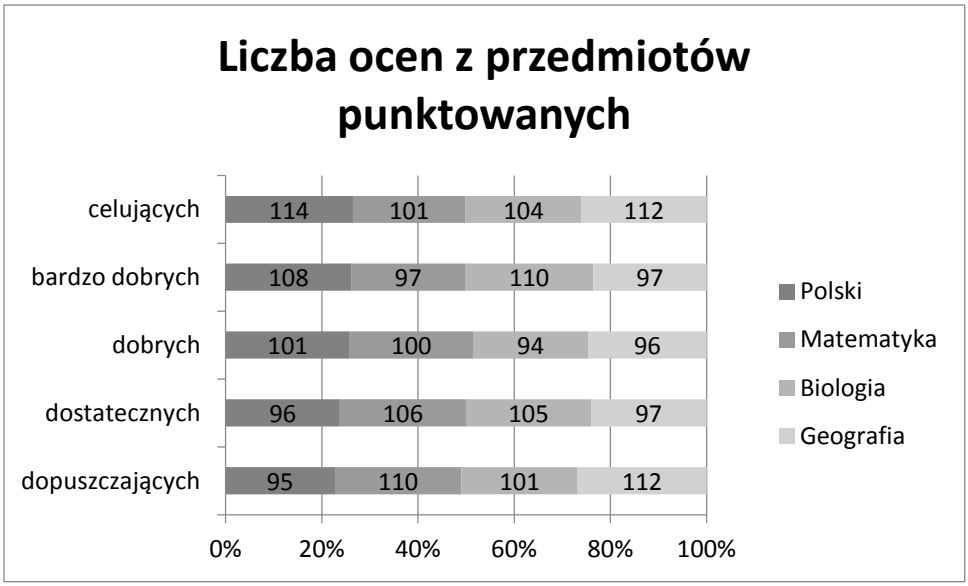
Liczba ocen	Język polski	Matematyka	Biologia	Geografia
dopuszczających	95	110	101	112
dostatecznych	96	106	105	97
dobrych	101	100	94	96
bardzo dobrych	108	97	110	97
celujących	114	101	104	112

Aby uzyskać powyższe zestawienie, stosujemy funkcję `LICZ.JEŻELI`. Formułę `=LICZ.JEŻELI(E$2:E$515;"=2")` wpisaną dla języka polskiego (JP) i oceny dopuszczającej wystarczy skopiować do kolejnych kolumn dotyczących matematyki, biologii i geografii. Należy zwrócić uwagę na stosowanie adresowania mieszanego. W podanej formule:

- nie blokujemy adresu kolumny, aby przy kopiowaniu zmienił się adres kolumny, w której wyszukujemy oceny (zmiana przedmiotu);
- blokujemy tylko adres wiersza, aby po skopiowaniu formuły do następnych wierszy zestawienia nie uległ zmianie zakres wierszy, do których się odwołujemy przy wyszukiwaniu ocen.

W kolejnych wierszach zestawienia zmieniamy w formule tylko wyszukiwaną wartość oceny.

Dla tak przygotowanego zestawienia tworzymy procentowy wykres skumulowany.



Wykres ten dla każdego typu oceny zawiera słupek tej samej wielkości, reprezentujący liczbę takich ocen z każdego przedmiotu. Z wykresu tego możemy na przykład odczytać, że ocen bardzo dobrych z języka polskiego jest 108 i stanowi to ponad 20% ocen bardzo dobrych z wszystkich przedmiotów punktowanych.

89.5.

W zadaniu należy wyszukać uczniów, którzy uzyskali więcej punktów rekrutacyjnych za oceny z przedmiotów punktowanych i dodatkowe osiągnięcia (w tym ocenę z zachowania) niż za wyniki egzaminów gimnazjalnych. Jeśli w zadaniu 2 rozbiliśmy obliczanie liczby punktów rekrutacyjnych, sumując trzy składniki:

- wyniki egzaminów gimnazjalnych,
- przedmioty punktowane,
- dodatkowe osiągnięcia,

to teraz wystarczy porównać sumę dwóch ostatnich składników z pierwszym. Posłuży do tego formuła warunkowa `=JEŻELI(zad.2!C2+zad.2!D2>zad.2!E2;1;0)`, gdzie w arkuszu o nazwie `zad.2`, w kolumnach C, D, E, są wyliczone wartości opisanych wyżej składników. Wartość tej formuły jest równa 1, gdy warunek podany w zadaniu jest spełniony. Dlatego wystarczy zsumować wartości tej formuły dla wszystkich uczniów, aby uzyskać liczbę uczniów, którzy spełniają warunek podany w zadaniu.

Rozwiązanie

89.1.

Bialaszewski	Wiktor
Cicherski	Szymon
Florek	Sandra
Kaminski	Mikolaj
Krol	Malgorzata

89.2.

Najczęściej występująca liczba punktów 55,6

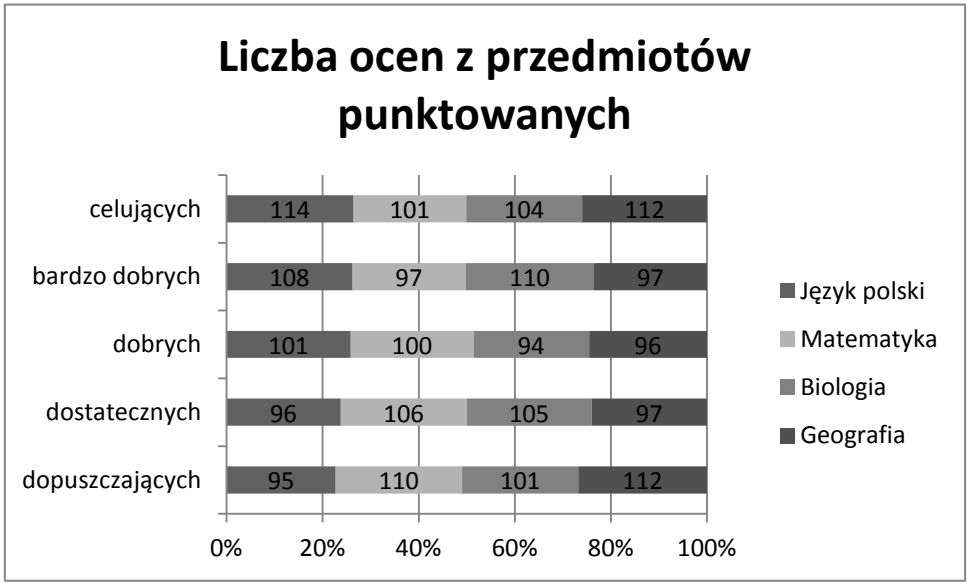
Majtas	Lucja
Broszkow	Zofia
Sokolnicka	Inga
Sochacka	Inka
Murczynska	Laura
Odya	Kacper

89.3.

Grzelecki	Oliwier
Wojcicki	Aleks
Rembiewski	Jakub
Przytula	Jakub

89.4.

Liczba ocen	Język Polski	Matematyka	Biologia	Geografia
dopuszczających	95	110	101	112
dostatecznych	96	106	105	97
dobrych	101	100	94	96
bardzo dobrych	108	97	110	97
celujących	114	101	104	112



89.5.

300 uczniów.

Zadanie 90.

Wiązka zadań *Akademiki*

W pliku `studenci.txt` zamieszczono listę 1616 studentów z 10 różnych uczelni, którzy starają się o miejsce w akademikach miasteczka studenckiego Mordor. W osobnych wier-

szach zostały zapisane informacje o każdym studencie: identyfikator studenta (`Id_studenta`), imię (`Imie`), nazwisko (`Nazwisko`), nazwa uczelni, na której on studiuje (`Uczelnia`), miejsce zamieszkania (`Miejsce_zam`), rok studiów (`Rok_studiow`) oraz dochód przypadający na jedną osobę w rodzinie wyrażony w denarach (`Dochod_na_osobe`). Pierwszy wiersz pliku jest wierszem nagłówkowym, a dane w wierszu są rozdzielone znakiem średnika.

Przykład

```
Id_studenta;Imie;Nazwisko;Uczelnia;Miejsce_zam;Rok_studiow;Dochod_na_osobe
1;Jadwiga;Baranowska;Uniwersytet Krolewski;Krosno;I;2382
2;Zofia;Chorzowska;Akademia Sztuk Pieknych;Pulawy;II;1549
3;Katarzyna;Bilska;Uniwersytet Krolewski;Lubliniec;I;3037
```

Korzystając z danych umieszczonych w pliku `studenci.txt`, wykonaj podane niżej zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku `wyniki.txt` (poza wykresem do zadania 88.4), a każdą z nich poprzedź numerem zadania.

90.1.

Podaj nazwę miejscowości (miejsce zamieszkania), z której pochodzi najwięcej studentów starających się o akademiki, oraz liczbę studentów z tej miejscowości. Ponadto podaj nazwę uczelni, na której studiuje najwięcej studentów z tej miejscowości.

90.2.

W akademikach jest w sumie 1000 miejsc. Początkowo Rada Mordoru ustaliła, że miejsce w akademiku otrzymają studenci, u których w rodzinach dochód na jedną osobę nie przekracza 2000 denarów. Podaj, ile miejsc pozostałoby wolnych, gdyby obowiązywało takie kryterium przydziału miejsc w akademikach.

Rada Mordoru postanowiła obsadzić wszystkie miejsca w akademikach. Podaj, jak należy zmienić próg kryterium dochodowego, aby dokładnie 1000 miejsc zostało wykorzystanych przez studentów. Próg musi być liczbą całkowitą. Taki próg istnieje dla podanych danych.

90.3.

Wykonaj zestawienie zawierające nazwę uczelni oraz średni dochód na osobę w rodzinie studentów tej uczelni, którzy starają się o pokój w Mordorze. Średni dochód podaj z dokładnością do dwóch miejsc po przecinku.

Do wykonanego zestawienia utwórz wykres kolumnowy. Aby uwypuklić różnicę w średnim dochodzie na osobę, przeskaluj oś OY tak, aby w początku układu współrzędnych znajdowała się wartość 1800 denarów.

90.4.

Dla każdej uczelni podaj liczbę, starających się o miejsce w Mordorze, studentów studiujących na I, II, III, IV i V roku studiów.

90.5.

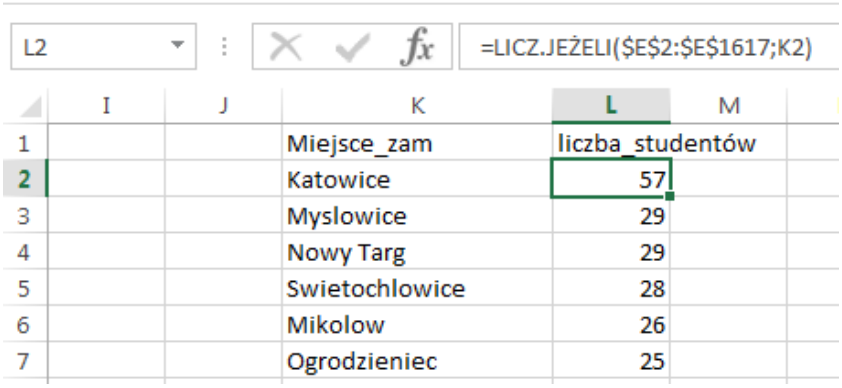
Wśród studentów są rodzeństwa. Rozpoznać je można po wspólnym nazwisku (z dokładnością do ostatniej litery, np. Nowicka i Nowicki mogą być rodzeństwem), takim samym miejscu zamieszkania i takim samym dochodzie na osobę. Wypisz wszystkie rodzeństwa wśród

podanych studentów. W zestawieniu podaj: imię, nazwisko, miejsce zamieszkania i dochód na osobę każdego rodzeństwa.

Komentarz do zadania

90.1.

Aby wyszukać nazwę miejscowości, z której pochodzi najwięcej studentów starających się o akademiki, można skorzystać z sum częściowych lub dla każdej miejscowości policzyć liczbę jej wystąpień funkcją `LICZ.JEŻELI`. Poniżej przedstawiamy rozwiązanie drugim proponowanym sposobem: wszystkie miejscowości kopiujemy do kolumny K, a następnie usuwamy z nich duplikaty, dalej funkcją `LICZ.JEŻELI` liczymy studentów z danych miejscowości i zestawienie porządkujemy malejąco według liczby studentów.



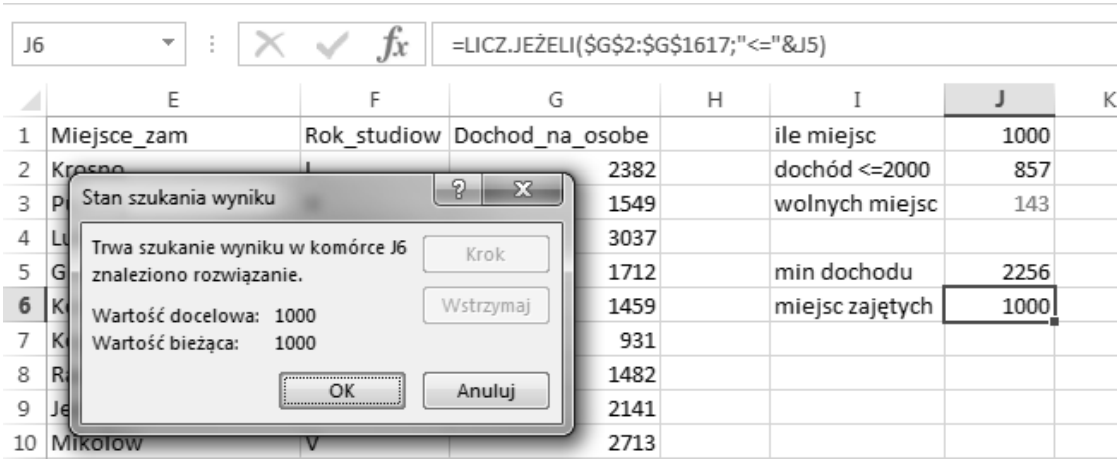
	I	J	K	L	M
1			Miejsce_zam	liczba_studentów	
2			Katowice	57	
3			Myslowice	29	
4			Nowy Targ	29	
5			Swietochlowice	28	
6			Mikolow	26	
7			Ogrodzieniec	25	

Aby podać nazwę uczelni najpopularniejszej wśród studentów z Katowic, odfiltrowujemy rekordy ze studentami z Katowic i dla każdej uczelni obliczamy liczbę studentów, a następnie sprawdzamy, dla jakiej uczelni ta liczba jest największa.

90.2.

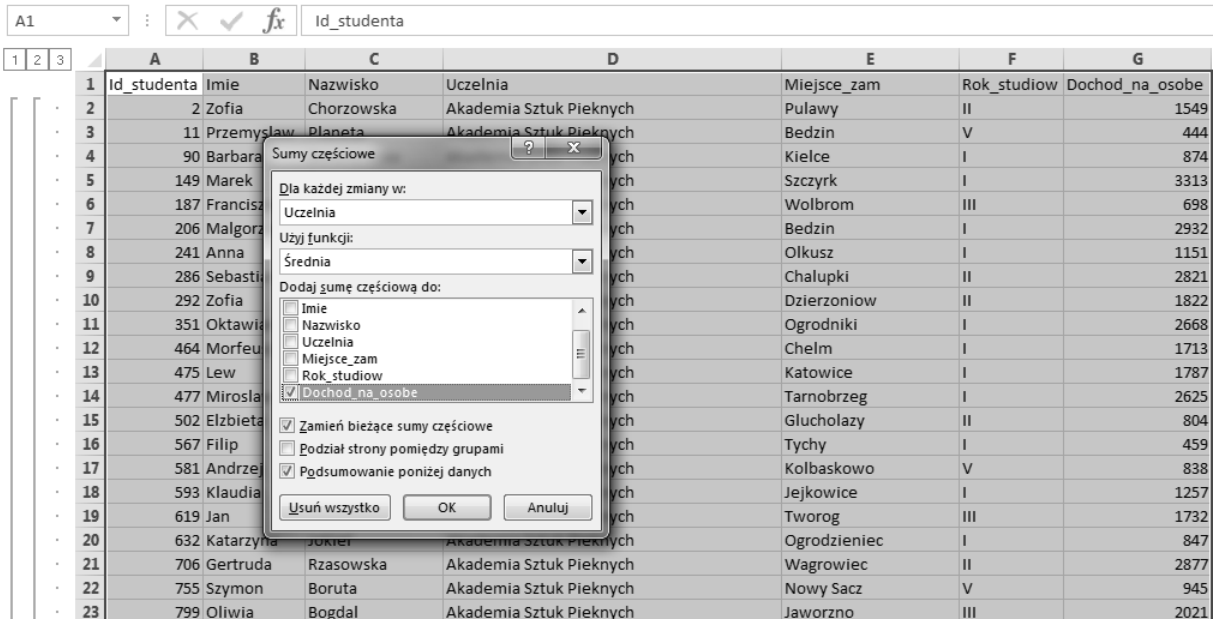
W pierwszej części zadania sprawdzamy, ile osób ma dochód niższy niż 2000 denarów, korzystając z funkcji `LICZ.JEŻELI`, i otrzymaną liczbę odejmujemy od liczby miejsc w akademikach.

Aby zasiedlić wszystkie miejsca w akademikach, należy zmienić kryterium dochodowe (zwiększyć minimalny dochód). Ustalamy zależność pomiędzy komórkami, w jednej z nich (w przykładowym rozwiązaniu to J5) zapisujemy minimalny dochód, a w drugiej — funkcję `LICZ.JEŻELI`, gdzie jako kryterium zostanie uwzględniona wartość dochodu z J5. Modyfikując komórkę J5, dążymy do uzyskania wartości 1000 w komórce docelowej (w przykładowym rozwiązaniu to J6). Można skorzystać z narzędzia *Analiza warunkowa* | *Szukaj wyniku*.



90.3.

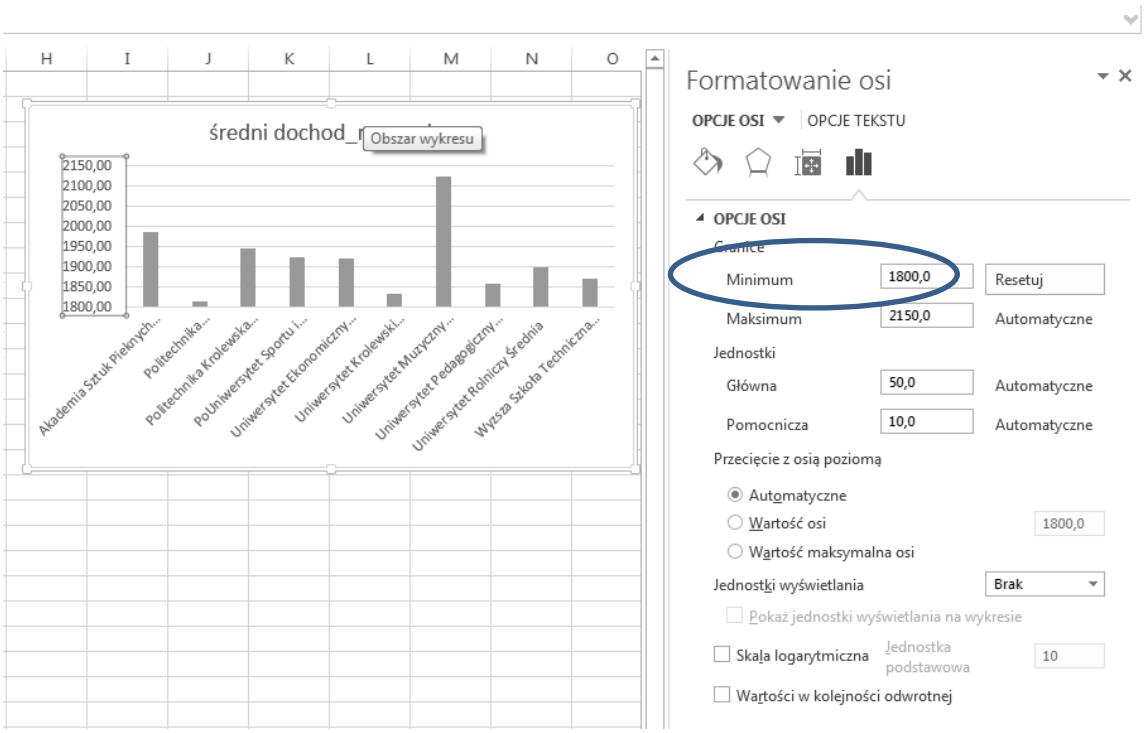
Aby wykonać zestawienie zawierające nazwę uczelni oraz średni dochód na osobę studentów tej uczelni, skorzystamy z narzędzia sumy częściowe. Należy pamiętać o wcześniejszym posortowaniu danych według uczelni.



	Id_studenta	Imie	Nazwisko	Uczelnia	Miejsce_zam	Rok_studiow	Dochod_na_osobe
2	2	Zofia	Chorzowska	Akademia Sztuk Pieknych	Pulawy	II	1549
3	11	Przemyslaw	Planeta	Akademia Sztuk Pieknych	Bedzin	V	444
4	90	Barbara		ych	Kielce	I	874
5	149	Marek		ych	Szczyrk	I	3313
6	187	Francisz		ych	Wolbrom	III	698
7	206	Malgorz		ych	Bedzin	I	2932
8	241	Anna		ych	Olkusz	I	1151
9	286	Sebasti		ych	Chalupki	II	2821
10	292	Zofia		ych	Dzierzoniow	II	1822
11	351	Oktawia		ych	Ogrodniki	I	2668
12	464	Morfeu		ych	Chelm	I	1713
13	475	Lew		ych	Katowice	I	1787
14	477	Mirosla		ych	Tarnobrzeg	I	2625
15	502	Elzbieta		ych	Glucholazy	II	804
16	567	Filip		ych	Tychy	I	459
17	581	Andrzej		ych	Kolbaskowo	V	838
18	593	Klaudia		ych	Jejkowice	I	1257
19	619	Jan		ych	Tworog	III	1732
20	632	Katarzyna	Jokiel	Akademia Sztuk Pieknych	Ogrodzieniec	I	847
21	706	Gertruda	Rzasowska	Akademia Sztuk Pieknych	Wagrowiec	II	2877
22	755	Szymon	Boruta	Akademia Sztuk Pieknych	Nowy Sacz	V	945
23	799	Oliwia	Bogdal	Akademia Sztuk Pieknych	Jaworzno	III	2021

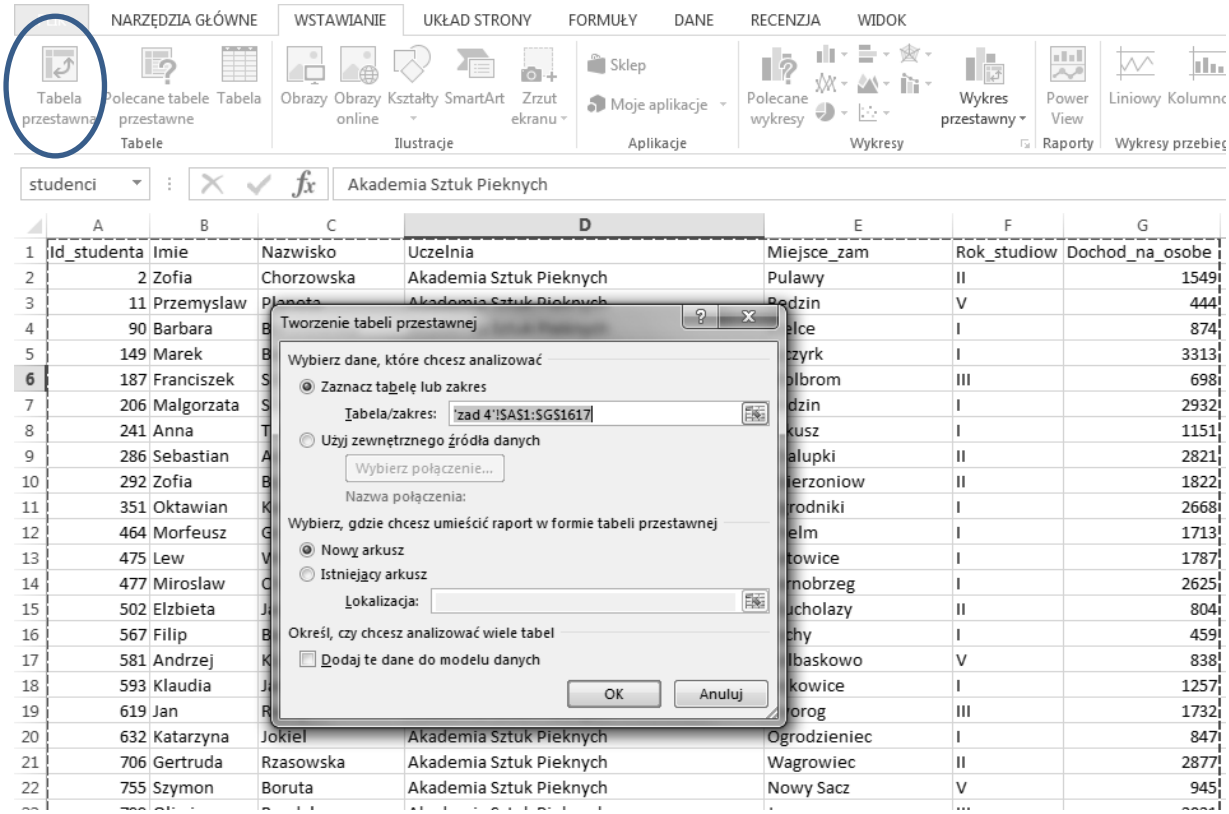
Po otrzymaniu wyników warto jeszcze sprawdzić, czy średni dochód podany został z dokładnością do dwóch miejsc po przecinku.

Po utworzeniu wykresu kolumnowego, aby uwypuklić różnicę w średnim dochodzie na osobę, przeskalowujemy oś OY tak, aby w początku układu współrzędnych znajdowała się wartość 1800 denarów.

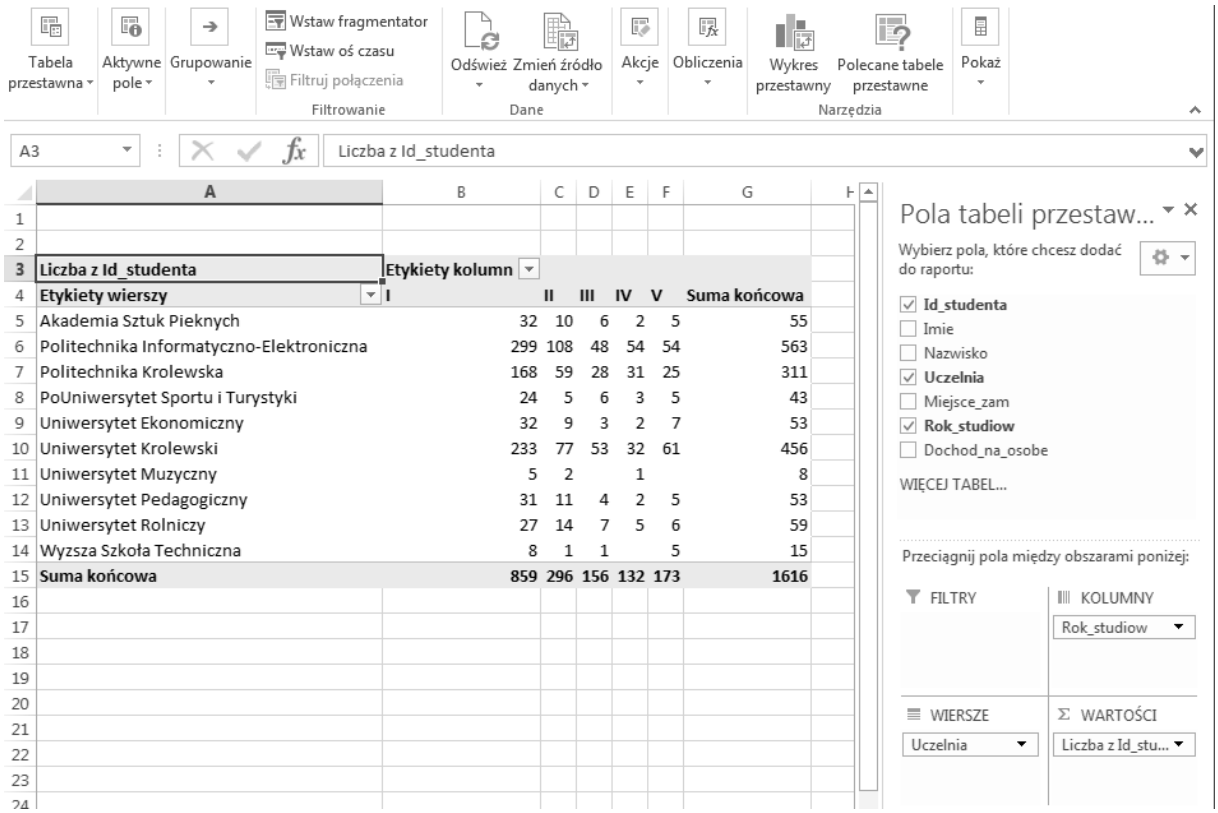


90.4.

W zadaniu 4 należy dla każdej uczelni podać liczbę studentów studiujących na I, II, III, IV i V roku studiów. Utworzymy tabelę dwuwymiarową, korzystając z narzędzia tabela przestawna.



Id_studenta	Imie	Nazwisko	Uczelnia	Miejsce_zam	Rok_studiow	Dochod_na_osobe
2	Zofia	Chorzowska	Akademia Sztuk Pieknych	Pulawy	II	1549
11	Przemyslaw	Planeta	Akademia Sztuk Pieknych	Rodzin	V	444
90	Barbara	B		elce	I	874
149	Marek	B		czyrk	I	3313
187	Franciszek	S		olbrom	III	698
206	Malgorzata	S		dzin	I	2932
241	Anna	T		kusz	I	1151
286	Sebastian	A		alupki	II	2821
292	Zofia	B		erzoniow	II	1822
351	Oktawian	K		rodniki	I	2668
464	Morfeusz	G		elm	I	1713
475	Lew	V		towice	I	1787
477	Miroslaw	C		rnobrzeg	I	2625
502	Elzbieta	J		ucholazy	II	804
567	Filip	B		chy	I	459
581	Andrzej	K		lbaskowo	V	838
593	Klaudia	J		kowice	I	1257
619	Jan	R		orog	III	1732
632	Katarzyna	Jokiel	Akademia Sztuk Pieknych	Ogrodzieniec	I	847
706	Gertruda	Rzasowska	Akademia Sztuk Pieknych	Wagrowiec	II	2877
755	Szymon	Boruta	Akademia Sztuk Pieknych	Nowy Sacz	V	945



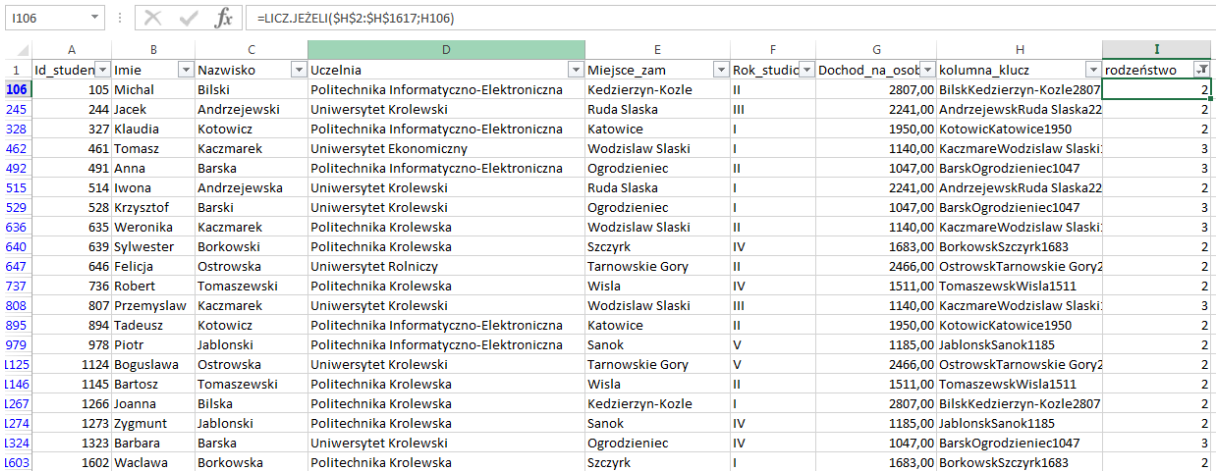
Liczba z Id_studenta	Etykiety kolumn					
Etykiety wierszy	I	II	III	IV	V	Suma końcowa
Akademia Sztuk Pieknych	32	10	6	2	5	55
Politechnika Informatyczno-Elektroniczna	299	108	48	54	54	563
Politechnika Krolewska	168	59	28	31	25	311
PoUniwersytet Sportu i Turystyki	24	5	6	3	5	43
Uniwersytet Ekonomiczny	32	9	3	2	7	53
Uniwersytet Krolewski	233	77	53	32	61	456
Uniwersytet Muzyczny	5	2		1		8
Uniwersytet Pedagogiczny	31	11	4	2	5	53
Uniwersytet Rolniczy	27	14	7	5	6	59
Wyzsza Szkoła Techniczna	8	1	1		5	15
Suma końcowa	**859**	**296**	**156**	**132**	**173**	**1616**

90.5.

Aby wyszukać rodzeństwa, które rozpoznajemy po takim samym nazwisku, takim samym miejscu zamieszkania i takim samym dochodzie na osobę, zbudujemy kolumnę klucz, zawierającą złączenie kolumn: nazwiska bez ostatniej litery, miejsca zamieszkania i dochód:

```
=ZŁĄCZ.TEKSTY(FRAGMENT.TEKSTU(C106;1;DŁ(C106)-1);E106;G106)
```

Następnie wyznaczymy rodzeństwa przez warunek `LICZ.JEŻELI(…)>1` na tej kolumnie.



	A	B	C	D	E	F	G	H	I
1	Id_studen	Imie	Nazwisko	Uczelnia	Miejsce_zam	Rok_studic	Dochod_na_osob	kolumna_klucz	rodzeństwo
106	105	Michal	Bilski	Politechnika Informatyczno-Elektroniczna	Kedzierzyn-Kozle	II	2807,00	BilskKedzierzyn-Kozle2807	2
245	244	Jacek	Andrzejewski	Uniwersytet Krolewski	Ruda Slaska	III	2241,00	AndrzejewskRuda Slaska22	2
328	327	Klaudia	Kotowicz	Politechnika Informatyczno-Elektroniczna	Katowice	I	1950,00	KotowicKatowice1950	2
462	461	Tomasz	Kaczmarek	Uniwersytet Ekonomiczny	Wodzislaw Slaski	I	1140,00	KaczmareWodzislaw Slaski	3
492	491	Anna	Barska	Politechnika Informatyczno-Elektroniczna	Ogrodzieniec	II	1047,00	BarskOgrodzieniec1047	3
515	514	Iwona	Andrzejewska	Uniwersytet Krolewski	Ruda Slaska	I	2241,00	AndrzejewskRuda Slaska22	2
529	528	Krzysztof	Barski	Uniwersytet Krolewski	Ogrodzieniec	I	1047,00	BarskOgrodzieniec1047	3
636	635	Weronika	Kaczmarek	Politechnika Krolewska	Wodzislaw Slaski	II	1140,00	KaczmareWodzislaw Slaski	3
640	639	Sylwester	Borkowski	Politechnika Krolewska	Szczyrk	IV	1683,00	BorkowskSzczyrk1683	2
647	646	Felicja	Ostrowska	Uniwersytet Rolniczy	Tarnowskie Gory	II	2466,00	OstrowskTarnowskie Gory2	2
737	736	Robert	Tomaszewski	Politechnika Krolewska	Wisla	IV	1511,00	TomaszewskWisla1511	2
808	807	Przemyslaw	Kaczmarek	Uniwersytet Krolewski	Wodzislaw Slaski	III	1140,00	KaczmareWodzislaw Slaski	3
895	894	Tadeusz	Kotowicz	Politechnika Informatyczno-Elektroniczna	Katowice	II	1950,00	KotowicKatowice1950	2
979	978	Piotr	Jablonski	Politechnika Informatyczno-Elektroniczna	Sanok	V	1185,00	JablonskSanok1185	2
1125	1124	Boguslawa	Ostrowska	Uniwersytet Krolewski	Tarnowskie Gory	V	2466,00	OstrowskTarnowskie Gory2	2
1146	1145	Bartosz	Tomaszewski	Politechnika Krolewska	Wisla	II	1511,00	TomaszewskWisla1511	2
1267	1266	Joanna	Bilska	Politechnika Krolewska	Kedzierzyn-Kozle	I	2807,00	BilskKedzierzyn-Kozle2807	2
1274	1273	Zygmunt	Jablonski	Politechnika Krolewska	Sanok	IV	1185,00	JablonskSanok1185	2
1324	1323	Barbara	Barska	Uniwersytet Krolewski	Ogrodzieniec	IV	1047,00	BarskOgrodzieniec1047	3
1603	1602	Waclawa	Borkowska	Politechnika Krolewska	Szczyrk	I	1683,00	BorkowskSzczyrk1683	2

Z uzyskanego zestawienia usuwamy niepotrzebne informacje, a dla zaspokojenia własnego poczucia estetyki możemy zestawienie posortować alfabetycznie według nazwisk.

Zadanie 91.

Wiązka zadań *Numery PESEL*

Numer PESEL to 11-cyfrowy kod jednoznacznie identyfikujący określoną osobę fizyczną. Dla wszystkich urodzonych w latach 1900-1999 skonstruowany został w następujący sposób:

- cyfry na pozycjach od 1 do 6 to data urodzenia (w kolejności: ostatnie dwie cyfry roku, numer miesiąca w postaci dwóch cyfr oraz numer dnia w postaci dwóch cyfr),
- cyfry na pozycjach od 7 do 9 to liczba porządkowa,
- cyfra na pozycji 10 oznacza płeć (cyfra parzysta dla kobiet, nieparzysta dla mężczyzn),
- cyfra z pozycji 11 to cyfra kontrolna.

Dla osób urodzonych w roku 2000 oraz w późniejszych latach do numeru miesiąca dodana jest liczba 20. W ten sposób w numerze PESEL odróżniane są od siebie dwa stulecia.

W pliku `pesele.txt` znajdują się 494 wiersze z danymi osób zarejestrowanymi w systemie ewidencji ludności. Pierwszy wiersz pliku jest wierszem nagłówkowym. Kolejne wiersze składają się z następujących informacji: numer PESEL (`PESEL`), nazwisko (`Nazwisko`) oraz imię (`Imie`).[2] Dane w wierszach pliku rozdzielone są średnikami.

Przykład

```
PESEL;Nazwisko;Imie
08242501475;Micun;Krzysztof
```

Korzystając z dostępnych narzędzi informatycznych, podaj odpowiedzi do poniższych zadań. Odpowiedzi zapisz do pliku `pesele_wyniki.txt` (z wyjątkiem wykresu do zadania 4), a każdą odpowiedź poprzedź cyfrą oznaczającą to zadanie.

91.1.

Wypisz imiona kobiet występujące w pliku, które **nie** kończą się na literę „a”. Aby ustalić, które wiersze w pliku zawierają dane kobiet, skorzystaj z cyfry numeru PESEL oznaczającej płeć osoby.

91.2.

Wśród osób, których dane zostały zapisane w pliku, są takie, które mają to samo imię i nazwisko. Wskaż te osoby, podając ich numery PESEL, imiona i nazwiska.

91.3.

Wypisz imiona i nazwiska osób, których liczba porządkowa w numerze PESEL (zapisana na pozycjach 7-9) jest odpowiednio największa i najmniejsza.

91.4.

Utwórz zestawienie zawierające nazwy wszystkich miesięcy roku oraz liczbę osób urodzonych w poszczególnych miesiącach. Dla otrzymanego zestawienia wykonaj wykres kolumnowy. Zadbaj o czytelny opis wykresu.

---

[2] Z uwagi na prawną ochronę danych osobowych, imiona i nazwiska w podanym pliku są fikcyjne.

91.5.

Dla każdej osoby utwórz jej identyfikator złożony z pierwszej litery imienia, trzech pierwszych liter nazwiska oraz ostatniej cyfry numeru PESEL. Podaj w porządku alfabetycznym identyfikatory, które wystąpią więcej niż jeden raz.

Zadanie 92.

Wiązka zadań *Olimpiady*

W pliku o nazwie `dane_medale.txt` znajdują się informacje o liczbie medali zdobytych przez poszczególne państwa uczestniczące w letnich i zimowych igrzyskach olimpijskich w latach 1896 – 2014. W każdym wierszu znajdują się następujące informacje, oddzielone pojedynczymi znakami tabulacji: nazwa państwa (`Panstwo`), kontynent (`Kontynent`), liczba olimpiad letnich, w których dane państwo brało udział (`OL_letnie`), liczba poszczególnych medali zdobytych w olimpiadach letnich (`Zloty; Srebrny; Brazowy`), liczba olimpiad zimowych, w których brało udział państwo (`OL_zimowe`), liczba poszczególnych medali zdobytych w olimpiadach zimowych (`Zloty; Srebrny; Brazowy`). Pierwszy wiersz jest wierszem nagłówkowym.

Przykład

```
Panstwo    Kontynent OL_letnie  Zloty  Srebrny Brazowy OL_zimowe Zloty  Srebrny Brazowy
Afganistan Azja      13         0      0       2       0         0      0       0
Algieria   Afryka    12         5      2       8       3         0      0       0
```

Dane opracowane na podstawie:
http://pl.wikipedia.org/wiki/Klasyfikacja_medalowa_wszech_czas%C3%B3w_igrzysk_olimpijskich#cite_note-1

Wykorzystując dane zawarte w tym pliku i dostępne narzędzia informatyczne, wykonaj poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_olimpiady.txt` (z wyjątkiem wykresu do zadania 2). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 2 umieść w pliku `wykres_olimpiady.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku.

92.1.

Podaj liczbę państw, z których każde spełnia poniższe warunki:

- brało udział w co najmniej jednej olimpiadzie letniej,
- brało udział w co najmniej jednej olimpiadzie zimowej,
- zdobyło co najmniej 1 medal na olimpiadadch letnich, a nie zdobyło **żadnego medalu na olimpiadach zimowych**.

Podaj łączną liczbę medali zdobytych na olimpiadach letnich przez państwa spełniające podane warunki.

92.2.

Przyjmijmy, że każdemu kontynentowi przydzielamy za daną olimpiadę tyle punktów, ile państw z tego kontynentu w niej wystąpiło. Utwórz zestawienie zawierające dla każdego kontynentu łączną liczbę punktów zz olimpiady letnie oraz łączną liczbę punktów za olimpiady zimowe. Dla otrzymanego zestawienia sporządź wykres procentowy, skumulowany słupkowy. Pamiętaj o prawidłowym i czytelnym opisie wykresu.

92.3.

Podaj nazwy państw, który zdobyły na wszystkich olimpiadach (letnich i zimowych) więcej medali złotych niż (łącznie) medali srebrnych i brązowych.

92.4.

Dla każdego kontynentu podaj nazwę państwa z tego kontynentu, które zdobyło łącznie największą liczbę medali na wszystkich olimpiadach, oraz liczbę tych medali.

92.5.

Kraj nazywamy *letnim*, jeżeli **dla każdego typu medali** (złoty, srebrny, brązowy) kraj ten zdobył **więcej** medali tego typu w olimpiadach letnich niż w olimpiadach zimowych. Podobnie kraj nazywamy *zimowym*, jeżeli dla każdego typu medali zdobył on więcej medali tego typu w olimpiadach zimowych niż w olimpiadach letnich. Podaj, ile jest w Europie krajów letnich i ile jest krajów zimowych.

Zadanie 93.

Wiązka zadań *Podróże*

Warszawska firma komputerowa *Sofcik* ma swoich klientów w 13 miastach i wysyła do nich swoich 101 pracowników. W pliku ***podróże.txt*** zapisane zostały wszystkie wyjazdy pracowników firmy do klientów w 2014 roku.

Pierwszy wiersz pliku jest wierszem nagłówkowym. W każdym wierszu zapisano: imię (`Imie`) i nazwisko pracownika (`Nazwisko`), miasto, do którego pracownik był delegowany (`Miasto`), datę wyjazdu (`D_wyj`) i datę powrotu z delegacji (`D_powr`) oraz koszt wyjazdu (`Koszt_wyj`). Daty w pliku są podawane w formacie: `RRRR-MM-DD`.

Dane w wierszach pliku rozdzielone są pojedynczymi znakami tabulacji.

Przykład

```
Imie     Nazwisko  Miasto    D_wyj        D_powr      Koszt_wyj
Karolina Arska     Malbork   2014-01-02   2014-01-03  891,00
Justyna  Kolska    Siedlce   2014-01-02   2014-01-03  295,40
Dorota   Morska    Radom     2014-01-02   2014-01-03  302,50
```

Wykorzystując dane zawarte w tym pliku i dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_podroze.txt` (z wyjątkiem wykresu do zadania 4). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 4 umieść w pliku `wykres_podroze.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku.

93.1.

Podaj imię i nazwisko osoby, która w ciągu całego roku była tylko raz w delegacji.

93.2.

Utwórz zestawienie zawierające nazwiska osób, które w sumie były w delegacji więcej niż 40 dni. Dla każdej osoby podaj liczbę dni, jaką spędziła ona w delegacji.

93.3.

Na koszt delegacji składa się koszt wyjazdu oraz koszt diety za każdy dzień podróży.

Dieta obliczana jest zgodnie z następującą zasadą: jeżeli pracownikowi nie zapewniono posiłku, to dieta wynosi 30 zł za każdy dzień, jeżeli zaś pracownik jadł w hotelu śniadanie (co zdarza się zawsze po noclegu w hotelu), to dieta wynosi 24 zł.

Przykład

Pracownik był w delegacji 4 dni, koszt jego diety wynosi 30 zł (za pierwszy dzień)+ 3 · 24 zł (za 3 dni ze śniadaniem w hotelu), razem 102 zł.

Utwórz zestawienie zawierające informacje o sumie kosztów (koszt wyjazdów), jakie firma poniosła na delegacje pracowników do każdego z 13 miast. Zestawienie posortuj rosnąco ze względu na kwotę kosztów.

93.4.

Utwórz zestawienie zawierające łączną liczbę wyjazdów pracowników w delegacje w poszczególnych miesiącach. Sporządź wykres kolumnowy przedstawiający otrzymane zestawienie. Pamiętaj o prawidłowym i czytelnym opisie wykresu.

Uwaga: dla każdego miesiąca weź pod uwagę tylko te wyjazdy w delegacje, których data wyjazdu (`D_wyj`) znajduje się w tym miesiącu.

93.5.

Podaj średnią liczbę noclegów pracowników we **wszystkich** delegacjach oraz średnią liczbę noclegów pracowników przebywających w delegacjach, które były **co najmniej 2-dniowe**. Wyniki podaj w zaokrągleniu do dwóch miejsc po przecinku.

Zadanie 94.

Wiązka zadań *Centyle*

W celu uaktualnienia siatek centylowych wzrostu dzieci wykonywano regularne pomiary wzrostu 2 262 wylosowanych dzieci. W pliku `wzrost.txt` zapisano w kolejnych kolumnach, rozdzielonych średnikiem, identyfikator dziecka, jego płeć, długość ciała dziecka w dniu narodzin, a następnie wzrost po ukończeniu każdego kolejnego roku życia. Długość i wzrost mierzymy w centymetrach. Pierwszy wiersz jest wierszem nagłówkowym.

Przykład

```
ID_dz;plec;dl_ur;1rok;2lata;3lata;4lata;5lat;6lat;7lat;8lat;9lat;10lat;11lat;12lat;
13lat;14lat;15lat;16lat;17lat;18lat;19lat
1;d;54;72;88;97;105;112;118;124;130;136;142;149;155;161;164;166;167;167;168;168
2;d;46;64;82;91;99;106;111;117;123;128;134;141;146;151;154;156;156;156;157;157
3;ch;54;75;88;97;104;111;117;123;129;134;139;145;151;158;165;171;175;177;178;179
```

Wykorzystując dane zawarte w pliku `wzrost.txt` i dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w jednym pliku o nazwie `wyniki` (z wyjątkiem wykresu do zadania 6). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 6 umieść w pliku `wykres.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku z Twoim rozwiązaniem.

94.1.

Znajdź dziecko, które od urodzenia do ukończenia 19 roku życia osiągnęło największy przyrost wysokości ciała, wyrażony procentowo w stosunku do długości ciała w dniu urodzin. Podaj ten przyrost oraz identyfikator dziecka, które uzyskało ten przyrost.

94.2.

Podaj , w którym roku życia (od 0 do 19 lat) średnia arytmetyczna wzrostu chłopców jest mniejsza o co najmniej 1 cm od średniej arytmetycznej wzrostu dziewcząt. Dzieci w dniu urodzin są w wieku 0.

94.3.

Podaj liczbę dzieci, które przestały rosnąć po ukończeniu 15 lat.

94.4.

Przyrost w *n*-tym roku życia to różnica między wzrostem po ukończeniu *n* lat a wzrostem po ukończeniu $n - 1$ lat.

Przyrost wysokości ciała z każdym kolejnym rokiem życia dziecka maleje, jednak tuż przed okresem dojrzewania zmienia trend i zaczyna się zwiększać. Oblicz średni przyrost wysokości ciała w kolejnych latach, dla każdej płci osobno. Na tej podstawie podaj, od którego roku życia dziewcząt i od którego roku życia chłopców przyrost wysokości ciała zaczyna się zwiększać.

94.5.

Centylem rzędu *p* (*p*-tym centylem) wzrostu dzieci nazywamy taki najmniejszy wzrost, że co najmniej *p*% dzieci ma wzrost mniejszy lub równy tej liczbie oraz co najmniej 100% – *p*% dzieci ma wzrost większy lub równy tej liczbie, przy czym liczba *p* jest liczbą całkowitą i $p \in (0; 100\%)$.

Na przykład 95. centyl wzrostu chłopców to taki wzrost, że co najmniej 95% chłopców ma wzrost mniejszy lub równy niż podana wartość, a co najmniej 5% ma wzrost większy lub równy podanej wartości.

Jeżeli liczba dzieci jest parzysta i granica tych dwóch grup wypada pomiędzy dwiema osobami o różnym wzroście należy podać średnią tych dwóch wartości.

Dla chłopców w wieku 1 roku, 10 lat i 19 lat oblicz 5. centyl oraz 95. centyl.

94.6.

Dla każdego badanego wieku chłopców (od 0 do 19) podaj medianę wzrostu.

Dla wykonanego zestawienia wykonaj wykres, którego pionowa oś będzie oznaczać wzrost, a pozioma — wiek. Pamiętaj o czytelnym opisie wykresu.

Zadanie 95.

Wiązka zadań *Giełda Papierów Wartościowych*

W pliku `gpw.txt` znajdują się dane dotyczące notowań akcji na Giełdzie Papierów Wartościowych w Warszawie w dniach: 21, 22 i 23 stycznia 2015.

W pierwszym wierszu pliku umieszczono nagłówki kolumn. Każdy następny wiersz pliku zawiera 7 danych dotyczących jednej spółki notowanych w jednej sesji: datę notowań (`data`), nazwę spółki (`nazwa`), jej identyfikator (`ISIN`), kurs jednej akcji na koniec dnia wyrażony w zł (`kurs_zamkniecia`), liczbę akcji spółki, które zmieniły właściciela w danym

dniu (`wolumen`), łączną wartość transakcji wszystkich akcji w danym dniu wyrażoną w zł (`obrot`) oraz liczbę akcji przyjętą do obliczenia indeksu WIG (`pakiet_wig`). Jeśli spółka nie wchodzi w skład portfela WIG, jej pole `pakiet_wig` ma wartość 0. Dane w wierszach oddzielone są znakami tabulacji.

Przykład

data	*nazwa*	*ISIN*	*kurs_zamkniecia*	*wolumen*	*obrot*	*pakiet_wig*
2015-01-21	ALTA	PLTRNSU00013	2,1	4664	9710	7353000
2015-01-22	ELZAB	PLELZAB00010	15,3	16599	249530	2716000
2015-01-23	PLAZA	NL0000686772	0,19	101576	19300	0

Dane pochodzą ze strony http://www.gpw.pl

Rozwiąż poniższe zadania, wykorzystując dostępne narzędzia informatyczne. Wyniki umieść w pliku tekstowym o nazwie `gpw_wyniki.txt`. Do oceny oddaj plik tekstowy zawierający wyniki oraz plik zawierający realizację komputerową rozwiązania.

95.1

Średni kurs akcji spółki w danym dniu obliczany jest jako iloraz *obrot*/*wolumen*. W przypadku gdy obrót i wolumen są równe 0 (czyli żadna akcja tej spółki nie zmieniła właściciela w ciągu dnia), średni kurs akcji spółki jest równy kursowi zamknięcia sesji.

Podaj **nazwy** trzech spółek, które w dniu **2015-01-21** osiągnęły najwyższe średnie kursy akcji, podaj również wartości tych kursów. Zapisz wyniki z dokładnością do 2 cyfr po przecinku.

95.2.

Dzienna zmiana procentowa kursu spółki jest obliczana na podstawie kursów zamknięcia z dwóch kolejnych dni:

$$d = \left(\frac{kurs\ zamknięcia\ z\ dnia\ bieżącego}{kurs\ zamknięcia\ z\ dnia\ poprzedniego} - 1\right) * 100\%.$$

Podaj nazwę spółki, która w dniu **2015-01-23** uzyskała największą dzienną zmianę kursu. Podaj wielkość tej zmiany w procentach, w zaokrągleniu do dwóch cyfr po przecinku.

95.3.

ISIN (*International Securities Identification Number*) jest międzynarodowym identyfikatorem spółki na rynku finansowym. Pierwsze dwa znaki stanowią kod kraju. Dla Polski przyjęto ”PL”. Wykorzystując tę informację:

a) podaj **liczbę spółek** krajowych i liczbę spółek zagranicznych notowanych na giełdzie w podanym okresie;

b) podaj **łączną wartość obrotów** spółek krajowych i łączną wartość obrotów spółek zagranicznych w ciągu trzech danych dni oraz **procentowy udział spółek krajowych** w łącznych obrotach wszystkich spółek. Wyniki podaj w zaokrągleniu do dwóch cyfr po przecinku.

95.4.

Warszawski Indeks Giełdowy WIG jest miarą koniunktury na giełdzie i wyraża łączną wartość spółek obecnych na Giełdzie Papierów Wartościowych (GPW) w stosunku do ich wartości w pierwszym dniu notowania (16 kwietnia 1991).

WIG obejmuje tylko wybrane spółki, spełniające kryteria co do procentu i wartości akcji w wolnym obrocie. Portfel spółek uczestniczących w WIG jest aktualizowany co kwartał. Obecnie (styczeń 2015) w portfelu jest 377 spółek. Dla każdej z nich określono *pakiet*, czyli liczbę akcji branych pod uwagę przy obliczaniu WIG. Wartość WIG oblicza się według wzoru:

$$WIG = \frac{M}{M_b * K} * W_b$$

gdzie: M — aktualna łączna wartość rynkowa pakietów wszystkich spółek w portfelu WIG, równa sumie iloczynów *pakiet*kurs* wszystkich spółek

M_b = 57 140 000 zł — wartość bazowa spółek na początku notowania;

W_b = 1000 — wartość bazowa indeksu WIG na początku notowania;

K = 96,482 137 39 — korektor wynikający ze zmian składu spółek zaliczanych do WIG.

WIG zmienia się wraz ze zmianą kursu akcji. Opierając się na wartościach kursów zamknięcia sesji, podaj dla każdego z trzech danych dni:

- łączną wartość rynkową pakietów wszystkich spółek przy kursie zamknięcia,
- wartość indeksu WIG zaokrągloną do dwóch cyfr po przecinku.

95.5.

Inwestor giełdowy śledzi notowania kursu zamknięcia sesji. Na podstawie wyników z trzech kolejnych dni następująco ocenia, co warto zrobić:

- jeżeli kurs spółki rośnie, i to rośnie coraz szybciej (drugi wzrost jest większy od pierwszego), warto kupić akcje,
- jeżeli kurs spółki spada, i to spada coraz szybciej (drugi spadek jest większy od pierwszego), warto sprzedać akcje,
- w pozostałych sytuacjach warto poczekać i obserwować rozwój sytuacji.

Wykorzystując notowania z trzech danych dni, podaj **liczbę spółek**, których akcje warto zdaniem inwestora **kupić**, liczbę spółek, których akcje jego zdaniem warto **sprzedać,** i liczbę spółek, które według niego warto dalej obserwować.

Zadanie 96.

Wiązka zadań *Prognoza liczby ludności w Polsce do roku 2050*

W pliku `ludnosc.txt` znajdują się dane: prognostyczne dotyczące liczby ludności Polski w latach 2013–2050 z podziałem na wiek (od 0 do 100 lat), płeć oraz miejsce zamieszkania: miasto lub wieś.

Pierwszy wiersz pliku jest wierszem nagłówkowym, a każdy następny wiersz pliku zawiera 6 liczb: rok kalendarzowy (`rok`), wiek osób w latach (`wiek`), liczbę mężczyzn (`m_miasto`) i liczbę kobiet w tym wieku (`k_miasto`) zamieszkałych w miastach oraz liczbę mężczyzn

(`m_wies`) i liczbę kobiet w tym wieku (`k_wies`) zamieszkałych na wsi. Osoby w wieku ponad 100 lat zaliczono dla uproszczenia do grupy 100-latków. Dane w wierszu oddzielone są znakiem tabulacji.

Przykład

```
rok     wiek    m_miasto  k_miasto   m_wies   k_wies
2013    0       107301    101414     77659    73821
2013    1       116232    109575     83756    79140
2013    2       118851    111765     85217    80842
```

Dane pochodzą ze strony http://stat.gov.pl

Rozwiąż poniższe zadania, wykorzystując dostępne narzędzia informatyczne. Wyniki umieść w pliku tekstowym o nazwie `ludnosc_wyniki.txt`. Do oceny oddaj plik tekstowy zawierający wyniki oraz plik zawierający realizację komputerową rozwiązania.

96.1.

Porównaj na wspólnym wykresie strukturę wiekową ludności Polski w roku 2013 i 2050. Sporządź wykres typu *Punktowy*, zawierający dwie serie danych: dla roku 2013 i roku 2050, który będzie przedstawiał dla każdego wieku od 0 do 100 łączną liczbę ludności w tym wieku.

96.2.

Podaj **stosunek** liczby ludności miast do liczby ludności wsi obliczony dla roku 2013 i dla roku 2050. Wyniki zapisz w postaci zaokrąglonej do dwóch cyfr po przecinku.

96.3.

Podaj średni wiek **mężczyzny** zamieszkałego **w mieście** w roku 2013 oraz średni wiek **mężczyzny** zamieszkałego **w mieście** w roku 2050.

Wyniki zapisz w postaci zaokrąglonej do liczby całkowitej.

Uwaga: Dla danego roku należy obliczyć średnią ważoną po wszystkich grupach wiekowych od 0 do 100:

$$\textit{średni wiek} = \frac{0 * m_0 + 1 * m_1 + 2 * m_2 + \cdots + 99 * m_{99} + 100 * m_{100}}{m_0 + m_1 + \cdots + m_{99} + m_{100}}$$

gdzie m_i — liczba mężczyzn w mieście w grupie wiekowej *i* lat.

96.4.

Strukturę ludności charakteryzuje taka prawidłowość, że w każdym roku kalendarzowym w młodszych rocznikach mężczyźni stanowią większość, jednak wraz z wiekiem ta przewaga się zmniejsza. W pewnym wieku zaczynają przeważać liczebnie kobiety.

Sporządź zestawienie, w którym dla każdego roku kalendarzowego z zakresu 2013-2050 podasz najniższy wiek, w jakim kobiety przeważają liczebnie mężczyzn.

96.5.

Rozważ trzy grupy wiekowe:

- młodzież do 18 roku życia włącznie;
- osoby w wieku produkcyjnym od 19 do 67 roku życia włącznie;
- emeryci powyżej 67 roku życia.

Dla każdego roku kalendarzowego oblicz liczbę ludności w każdej z tych grup.

Sporządź wykres kolumnowy skumulowany procentowy, ilustrujący liczebność tych trzech grup w kolejnych latach okresu 2013–2050.

Sporządź zestawienie, w którym dla każdego roku podasz, jaki procent ludności stanowią osoby w wieku produkcyjnym. Wynik zapisz w postaci zaokrąglonej do całkowitej liczby procentów.

Zadanie 97.

Wiązka zadań *Stopa bezrobocia*

W pliku o nazwie `stopa_bezrobocia.txt` znajdują się miesięczne stopy bezrobocia w państwie Klingonów w latach 1945–2014. W każdym wierszu znajduje się 13 liczb, oddzielonych pojedynczymi znakami tabulacji: rok oraz stopy bezrobocia w kolejnych dwunastu miesiącach. Pierwszy wiersz jest wierszem nagłówkowym.

Przykład

```
ROK    I      II     III    IV     V      VI     VII    VIII   IX     X      XI     XII
1945   3,2    4,2    4,1    3,9    3,9    4,0    4,0    4,2    4,4    4,1    4,0    3,8
1946   4,5    3,9    3,6    3,6    3,7    3,5    3,5    3,6    3,7    3,9    4,1    4,5
1947   5,0    5,9    6,2    6,7    6,9    6,9    6,6    6,8    7,0    7,1    6,7    6,3
```

Wykorzystując dane zawarte w tym pliku i dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_bezrobocie.txt` (z wyjątkiem wykresu do zadania 3). Odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie. Wykres do zadania 3 umieść w pliku `wyniki_bezrobocie.xxx`, gdzie `xxx` oznacza rozszerzenie odpowiednie dla formatu pliku.

97.1.

Miesiącem wysokiego zagrożenia dla gospodarki w państwie Klingonów nazywamy taki miesiąc, w którym stopa bezrobocia jest większa niż 10. Podaj liczbę miesięcy wysokiego zagrożenia, jakie wystąpiły latach 1945–2014.

97.2.

Podaj najniższą i najwyższą średnią roczną stopę bezrobocia (wyniki podaj w zaokrągleniu do dwóch miejsc po przecinku) oraz rok, w którym one wystąpiły.

97.3.

Utwórz zestawienie zawierające dla każdego roku w latach 1945–2014 minimalną i maksymalną miesięczną stopę bezrobocia odnotowaną w tych latach.

Dla otrzymanego zestawienia sporządź wykres liniowy. Pamiętaj o prawidłowym i czytelnym opisie wykresu.

97.4.

Znajdź najdłuższy nierosnący ciąg miesięcznych stóp bezrobocia w kolejnych miesiącach w latach 1945–2014. Podaj miesiąc i rok początkowy oraz miesiąc i rok końcowy znalezionego ciągu oraz jego długość.

Przykład

Dla pliku zawierającego następujące dane:

	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
2002	12,90	12,90	12,50	13,00	12,50	12,20	11,00	11,70	11,50	11,30	11,40	11,60
2003	14,20	14,40	14,30	**14,40**	**13,60**	**13,20**	**13,10**	**13,00**	**13,00**	**13,00**	**12,90**	**12,80**
2004	**12,50**	12,60	13,30	12,90	12,60	12,30	12,30	12,40	12,40	12,50	12,90	13,40

długość najdłuższego nierosnącego ciągu stóp bezrobocia wynosi 10 (ciąg liczb: 14,40; 13,60; 13,20; 13,10; 13,00; 13,00; 13,00; 12,90; 12,80; 12,50). Miesiąc i rok początku ciągu to IV 2003. Miesiąc i rok końca ciągu to I 2004.

97.5.

Podaj liczbę lat, w których stopa bezrobocia w każdym miesiącu była większa od stopy bezrobocia w tym samym miesiącu roku poprzedniego.

2.4. Bazy danych

Zadanie 98.

Wiązka zadań *Dziennik ocen*

Dane są trzy pliki tekstowe: `uczniowie.txt`, `przedmioty.txt` i `oceny.txt`, w których zapisano oceny wystawiane uczniom w pewnym technikum informatycznym w okresie od 1.09.2014 r. do 18.12.2014 r.

Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a kolumny w wierszach rozdzielone są znakami tabulacji.

Plik o nazwie `uczniowie.txt` zawiera informacje dotyczące uczniów szkoły. W każdym wierszu znajduje się: identyfikator ucznia (`ID_ucznia`), jego imię (`Imie`), nazwisko (`Nazwisko`) oraz oznaczenie klasy za pomocą rzymskiej liczby i litery (`Klasa`).

Przykład

```
Id_ucznia Imie      Nazwisko       Klasa
123/2011  Wojciech  Banasik        IV E
124/2011  Monika    Baranowska     IV E
125/2011  Janusz    Czerwinski     IV E
```

Plik `przedmioty.txt` zawiera identyfikator przedmiotu (`Id_przedmiotu`) oraz nazwę przedmiotu (`Nazwa_przedmiotu`).

Przykład

```
Id_przedmiotu  Nazwa_przedmiotu
1              j.polski
2              j.angielski
3              j.niemiecki
```

W pliku `oceny.txt` zapisane są w każdym wierszu: identyfikator oceny (`Id_oceny`), data wystawienia oceny (`Data`), identyfikator ucznia (`Id_ucznia`), identyfikator przedmiotu (`Id_przedmiotu`) oraz ocena (`Ocena`).

Przykład

```
Id_oceny  Data        Id_ucznia Id_przedmiotu  Ocena
1         2014-09-08  704/2014  1              2
2         2014-09-08  312/2012  1              4
3         2014-09-08  649/2013  3              5
```

Korzystając z danych zawartych w tych plikach oraz z dostępnych narzędzi informatycznych, wykonaj poniższe polecenia. Każdą odpowiedź umieść w pliku `wyniki.txt`, poprzedzając ją numerem odpowiedniego zadania.

98.1.

Imiona dziewcząt w zestawieniu kończą się na literę „a”. Podaj klasy, w których ponad 50% wszystkich uczniów to dziewczęta.

98.2.

Podaj daty, kiedy w szkole wystawiono więcej niż 10 jedynek jednego dnia.

98.3.

Podaj, z dokładnością do dwóch miejsc po przecinku, średnie ocen z języka polskiego dla każdej klasy czwartej.

98.4.

Podaj zestawienie zawierające dla każdego przedmiotu liczbę piątek wystawionych w kolejnych miesiącach od września do grudnia łącznie we wszystkich klasach.

98.5.

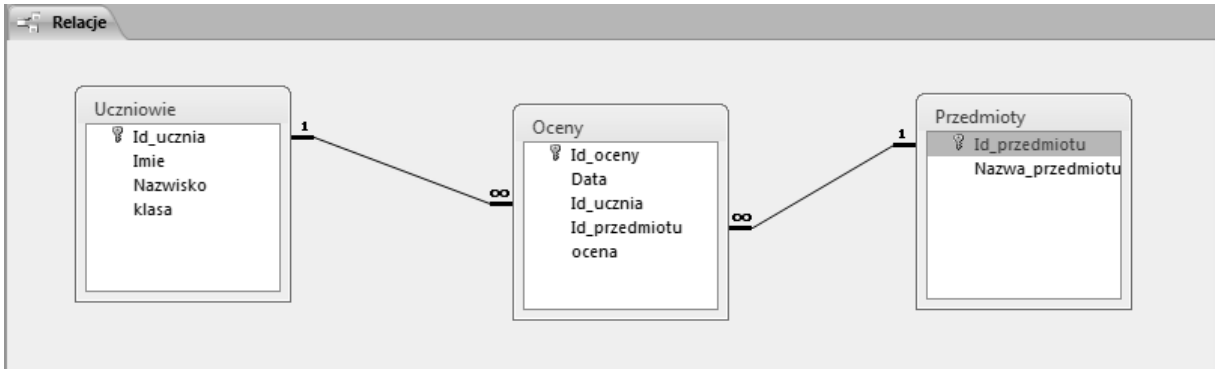
Podaj zestawienie imion i nazwisk uczniów klasy II A, którzy nie otrzymali żadnej oceny z przedmiotu sieci komputerowe.

Komentarz do zadania

Przedstawiamy rozwiązanie zadania za pomocą programu Access z pakietu MS Officce 2010 oraz w języku SQL.

Kluczami głównymi (PRIMARY KEY) w tabelach `Uczniowie`, `Oceny`, `Przedmioty` będą odpowiednio kolumny `Id_ucznia`, `Id_przedmiotu` oraz `Id_oceny`, to one jed-

noznacznie identyfikują rekordy w tabelach. Tabele połączone są relacjami typu jeden do wielu:



98.1.

Aby wyszukać klasy, w których ponad 50% wszystkich uczniów to dziewczęta, skorzystamy z informacji zawartych w tabeli Uczniowie. Utworzymy kwerendy pomocnicze, za pomocą których wyznaczymy liczbę dziewcząt (kryterium w kolumnie `Imie`: Like "*a") w klasach (liczymy liczbę identyfikatorów uczniów) oraz liczbę wszystkich uczniów w klasach.

Pole:	klasa	PoliczOfId_ucznia1: Id_ucznia	Imie
Tabela:	Uczniowie	Uczniowie	Uczniowie
Podsumowanie:	Grupuj według	Policz	Gdzie
Sortuj:			
Pokaż:	☑	☑	☐
Kryteria:			Like "*a"
lub:			

.

Pole:	Klasa	liczba_uczniow: Id_ucznia
Tabela:	Uczniowie	Uczniowie
Podsumowanie:	Grupuj według	Policz
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

W kolejnej kwerendzie dla każdej klasy obliczymy wyrażenie $\frac{liczba\ dziewcząt}{liczba\ wszystkich\ uczniów}$, przy czym wyrażenie to powinno być większe od 0,5.

Pole:	Klasa	udzial_dziewczat: [pom1b]![Liczba_dziewczat]/[pom1a]![liczba_uczniow]
Tabela:	pom1a	
Sortuj:		
Pokaż:	☑	☑
Kryteria:		>0,5
lub:		

Zapytanie w języku SQL:

SELECT Klasa
FROM Uczniowie
GROUP BY Klasa
HAVING SUM(CASE WHEN Imie LIKE "%a" **THEN** 1 **ELSE** 0 **END)** >**COUNT**(*) / 2

98.2.

Chcąc wyszukać daty, kiedy w szkole wystawiono więcej niż 10 jedynek jednego dnia, odfiltrujemy spośród wszystkich ocen jedynki, a następnie posłużymy się funkcjami agregującymi (policzymy dla każdej daty liczby jedynek i odfiltrujemy te dni, w których liczba jedynek przekroczyła 10).

Warunek na ocenę został założony za pomocą klauzuli WHERE. Klauzula GROUP BY umożliwia podział wierszy na kategorie na podstawie wartości w kolumnie `Data` i skorzystanie z funkcji grupującej (`POLICZ`) dla różnych ocen. Dalej klauzulą HAVING ograniczamy zestawienie do takiego, w którym liczba jedynek jest większa od 10.

Pole:	Data	ocena	ocena
Tabela:	Oceny	Oceny	Oceny
Podsumowanie:	Grupuj według	Policz	Gdzie
Sortuj:		Malejąco	
Pokaż:	☑	☑	☐
Kryteria:		>10	1
lub:			

Zapytanie w języku SQL:

SELECT Oceny.Data, Count(*)
FROM Oceny
WHERE Oceny.ocena=1
GROUP BY Oceny.Data
HAVING Count(*)>10

98.3.

Aby ustalić z dokładnością do dwóch miejsc po przecinku średnie ocen z języka polskiego każdej klasy czwartej, odfiltrujemy oceny z języka polskiego uczniów klas IV, a następnie skorzystamy z funkcji agregującej Średnia (`AVG`).

Pole:	klasa	ocena	Nazwa_przedmiotu
Tabela:	Uczniowie	Oceny	Przedmioty
Podsumowanie:	Grupuj według	Średnia	Grupuj według
Sortuj:			
Pokaż:	☑	☑	☐
Kryteria:	Like 'IV*'		Like 'j.polski'
lub:			

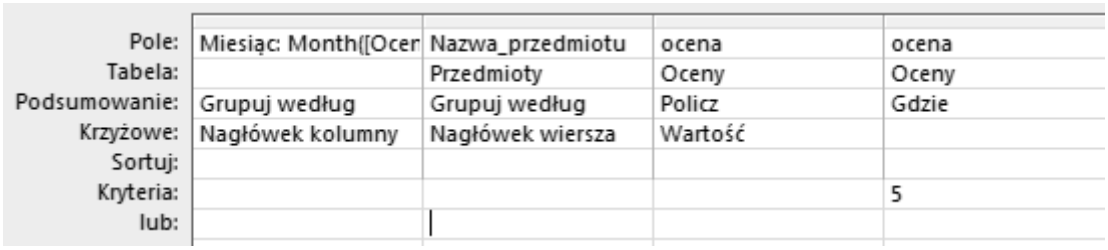
Zapytanie w języku SQL:

SELECT Uczniowie.klasa, **AVG**(oceny.ocena)
FROM Oceny
JOIN Przedmioty **ON** Przedmioty.id_przedmiotu = Oceny.id_przedmiotu
JOIN Uczniowie **ON** Uczniowie.id_ucznia = Oceny.id_ucznia
WHERE Przedmioty.nazwa_przedmiotu = ‘j.polski’ **AND** Uczniowie.klasa LIKE ‘IV%’
GROUP BY Uczniowie.klasa

98.4.

W rozwiązaniu utworzymy kwerendę krzyżową, w której nagłówkiem kolumny będzie wyodrębniamy z daty numer miesiąca (funkcja `MONTH`), zaś nagłówkiem wiersza — nazwa przedmiotu. Jako wartość wstawiamy ocenę lub identyfikator oceny i stosujemy funkcję `Policz`. Dodatkowo należy pamiętać o klauzuli WHERE, za pomocą której odfiltrujemy same piątki.

Widok projektu:



Pole:	Miesiąc: Month([Ocer	Nazwa_przedmiotu	ocena	ocena
Tabela:		Przedmioty	Oceny	Oceny
Podsumowanie:	Grupuj według	Grupuj według	Policz	Gdzie
Krzyżowe:	Nagłówek kolumny	Nagłówek wiersza	Wartość	
Sortuj:				
Kryteria:				5
lub:				

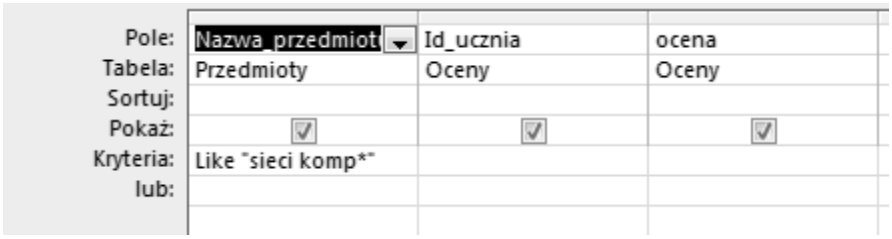
Zapytanie w języku SQL:

```
SELECT Przedmioty.Nazwa_przedmiotu, Month(Oceny.Data), Count(*)
FROM Oceny
JOIN Przedmioty ON Przedmioty.id_przedmiotu = Oceny.id_przedmiotu
WHERE Oceny.ocena=5
GROUP BY Przedmioty.Nazwa_przedmiotu, Month(Oceny.Data)
```

98.5.

Rozwiązywanie poprzedzimy utworzeniem kwerendy pomocniczej, w wyniku której otrzymamy oceny uczniów uczęszczających na zajęcia z przedmiotu sieci komputerowe.

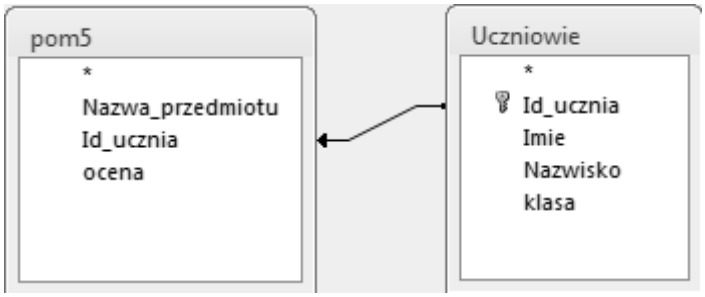
Widok projektu:



Pole:	Nazwa_przedmiot	Id_ucznia	ocena
Tabela:	Przedmioty	Oceny	Oceny
Sortuj:			
Pokaż:	☑	☑	☑
Kryteria:	Like "sieci komp*"		
lub:			

Dalej, korzystając z wyników powyższego zapytania pomocniczego oraz tabeli Uczniowie (złącze pomiędzy tabelami typu RIGHT JOIN), uzyskamy zestawienie uwzględniające wszystkie rekordy z tabeli Uczniowie, niezależnie od tego czy uczniowie mają ocenę, czy nie (pole wypełnione wartościami NULL). Kolejnym krokiem jest wybranie rekordów z wartością NULL. Dodatkowo w kryterium dla pola Klasa podajemy ”II A”.

Widok projektu:



Pole:	Imie	Nazwisko	klasa	ocena
Tabela:	Uczniowie	Uczniowie	Uczniowie	pom5
Sortuj:				
Pokaż:	☑	☑	☐	☐
Kryteria:			"II A"	Is Null
lub:				

Zapytanie w języku SQL:

SELECT Uczen.Imie, Uczen.Nazwisko
FROM Uczen
WHERE Uczen.Klasa = ‘II A’ **AND** (**SELECT COUNT**(*) **FROM** Oceny **JOIN**
Przedmioty **ON** Przedmioty.id_przedmiotu = Oceny.id_przedmiotu **WHERE**
Oceny.id_ucznia = Uczen.id_ucznia **AND** Przedmiotu.nazwa LIKE ‘sieci komp%’) = 0

Rozwiązanie

98.1.

I A, I C

98.2.

2014-11-11

2014-10-14

98.3.

Klasa	Średnia
IV A	3,53
IV B	3,45
IV C	3,40
IV D	3,90
IV E	3,51

98.4.

Przykładowa poprawna odpowiedź:

Nazwa_przedmiotu	9	10	11	12
administracja bazami danych	3	13	7	1
administracja sieciowymi systemami operacyjnymi				2
biologia	8	20	9	5
chemia	18	31	31	13
diagnostyka i naprawa urzadzen techniki komputerowej	8	13	9	13
edukacja dla bezpieczenstwa	9	15	14	8
fizyka	14	21	31	9
geografia		1	1	
historia	2	2		
historia i spoleczenstwo - przedmiot uzupelniajacy	8	4	6	3
informatyka	33	31	34	25
j.angielski	50	85	72	50
j.niemiecki	66	106	84	53
j.polski	37	81	62	41
jezyk angielski zawodowy w branzy informatycznej	1	1	1	
matematyka	32	39	48	35
podstawy przedsiebiorczosci	6	16	16	12
projektowanie i montaz lokalnych sieci komputerowych	10	11	11	6
sieci komputerowe		2		
systemy baz danych	6	5	8	5
systemy operacyjne	4	19	12	9
urzadzenia techniki komputerowej	8	18	19	15
wiedza o kulturze	7	16	13	10
wiedza o spoleczenstwie	7	15	9	7
witryny i aplikacje internetowe	5	6	9	3
wychowanie fizyczne	23	39	34	15

98.5.

Imie	Nazwisko
Aneta	Duda
Miroslaw	Gorski
Lukasz	Kostoczko
Donald	Krychowski
Adrian	Lubaczewski
Piotr	Nawrocki
Michal	Nowakowski
Piotr	Prusinski
Grzegorz	Tomkow
Radoslaw	Wojciechowski
Mariusz	Wojtyra

Zadanie 99.

Wiązka zadań *Bezpieczeństwo w szkole*

Wydział Edukacji powiatu Bajtolandia przeprowadził wśród uczniów szkół badanie na temat bezpieczeństwa w szkole. W ankiecie udział wzięli uczniowie ze 130 wybranych szkół różnego typu.

W plikach `ankiety.txt`, `szkoly.txt` i `gminy.txt` znajdują się dane dotyczące ankietowanych szkół w powiecie Bajtolandia oraz wyniki ankiet wypełnionych przez uczniów tych szkół. Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.

W kolejnych wierszach pliku `ankiety.txt` znajdują się wyniki 5600 ankiet: numer ankiety (`Nr_ankiety`), oceny podane w odpowiedzi na poszczególne pytania (`pyt1`, `pyt2`, ..., `pyt6`), `Plec` (*k* — dziewczyna, *m* — chłopak), identyfikator szkoły (`Id_szkoły`).

Przykład

Nr_ankiety	pyt1	pyt2	pyt3	pyt4	pyt5	pyt6	Id_szkoly	Plec
a0001	1	2	2	3	3	4	S001	k
a0002	5	3	3	2	2	3	S001	k
a0003	5	3	5	5	1	4	S001	m
a0004	3	2	2	1	2	3	S001	k

Treść pytań ankiety:

pyt1: Czy w swojej szkole czujesz się bezpiecznie?

pyt2: Czy byłeś na terenie szkoły ofiarą niebezpiecznych zachowań, agresji?

pyt3: Czy byłeś świadkiem niebezpiecznych sytuacji, agresji w szkole?

pyt4: Czy zdarzyło Ci się zrobić komukolwiek jakąś przykrość?

pyt5: Czy nauczyciele rozmawiają z uczniami o przemocy?

pyt6: Czy w szkole odbywają się zajęcia (spotkania/lekcje/wystawy/inne formy) na temat bezpieczeństwa w szkole?

Odpowiedzi na pytania to oceny w skali od 1 do 5, oznaczające odpowiednio:

1. nigdy
2. bardzo rzadko
3. rzadko
4. często
5. bardzo często

W pliku `szkoly.txt` znajduje się 130 wierszy z informacjami o szkołach: identyfikator szkoły (`Id_szkoly`), `Rodzaj_szkoly` (SP, G, LO, T, ZS), `Kod_gminy`.

Przykład

Id_szkoly	Rodzaj_szkoly	Kod_gminy
S001	LO	GM19
S002	SP	GM17
S003	LO	GM07
S004	T	GM19

W pliku `gminy.txt` każdy wiersz zawiera informacje o jednej z 20 gmin w Bajtolandii: `Kod_gminy`, `Nazwa_gminy`.

Przykład

```
Kod_gminy     Nazwa_gminy
GM01          Piatki Gorne
GM02          Piatki Dolne
GM03          Sobotka
```

Wykorzystując dane zawarte w tych plikach i dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie ***wyniki_ankiety.txt***. Odpowiedź do każdego zadania rozpocznij od nowego wiersza i poprzedź numerem oznaczającym to zadanie.

99.1.

Podaj liczbę wszystkich ankietowanych dziewcząt i wszystkich ankietowanych chłopców.

99.2.

Dla każdego rodzaju szkoły podaj średnią ocenę odpowiedzi na każde pytanie. Wyniki podaj w zaokrągleniu do dwóch miejsc po przecinku.

99.3.

Dla każdej gminy wyznacz średnią ocenę uczniów, z jej terenu podaną w odpowiedzi na ostatnie (szóste) pytanie. Wyniki umieść w zestawieniu zawierającym dwie kolumny: kod gminy, średnią ocenę uczniów. Zestawienie uporządkuj malejąco ze względu na średnią ocenę. Średnie podaj w zaokrągleniu do dwóch miejsc po przecinku.

99.4.

Utwórz zestawienie zawierające dla każdego rodzaju szkoły informacje o liczbie uczniów, którzy podali ocenę 5 na pytanie trzecie. Zestawienie posortuj alfabetycznie według rodzaju szkoły.

99.5.

Podaj nazwę gminy z największą liczbą uczniów biorących udział w badaniu oraz liczbę tych uczniów.

99.6.

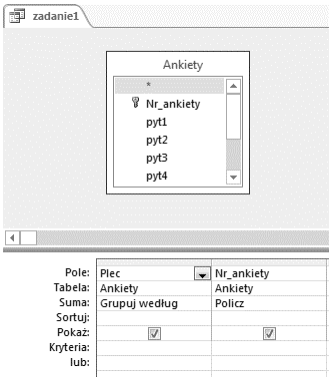
Utwórz zestawienie zawierające informacje o liczbie dziewcząt i chłopców (osobno) z poszczególnych rodzajów szkół, którzy podali najwyższą ocenę 5 na pytanie 1.

Komentarz do zadania

Po wczytaniu danych do tabel oraz ustaleniu związków między tabelami (*Relacje*) przystępujemy do rozwiązania zadania.

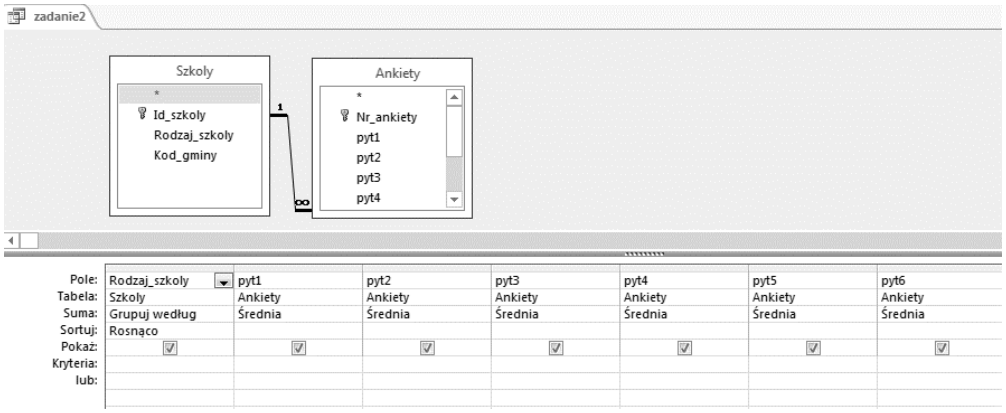
99.1.

W tym zadaniu konstruujemy kwerendę podsumowującą na podstawie tabeli `Ankiety`. Rozpoczynamy od projektu prostej kwerendy wybierającej i umieszczamy w siatce projektowej pola `Plec` oraz `Nr_ankiety`. Następnie ustalamy typ kwerendy na podsumowującą i dla wybranych pól stosujemy odpowiednio `Grupuj według` (dla pola `Plec`) oraz `Policz` (dla pola `Nr_ankiety`). W efekcie kwerenda wyznaczy, ilu chłopców i ile dziewcząt wypełniło ankietę.

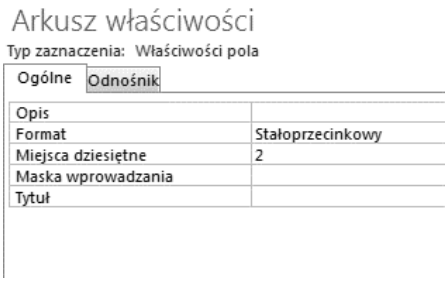


99.2.

W celu rozwiązania zadania utworzymy zapytanie w oparciu o tabele: `Szkoly` oraz `Ankiety`. Podobnie jak przy rozwiązaniu poprzedniego zadania utworzymy kwerendę podsumowującą. Ponieważ mamy podać średnią ocenę odpowiedzi na każde pytanie dla każdego rodzaju szkoły, w kwerendzie potrzebne będą pola: `Rodzaj_szkoly` z tabeli `Szkoly` (wybierzemy w kwerendzie grupowanie według tego pola) oraz pola `pyt1`, `pyt2`, `pyt3`, `pyt4`, `pyt5` i `pyt6` z tabeli `Ankiety`. Dla pól z odpowiedziami na poszczególne pytania w kwerendzie wybieramy funkcję `Średnia`.



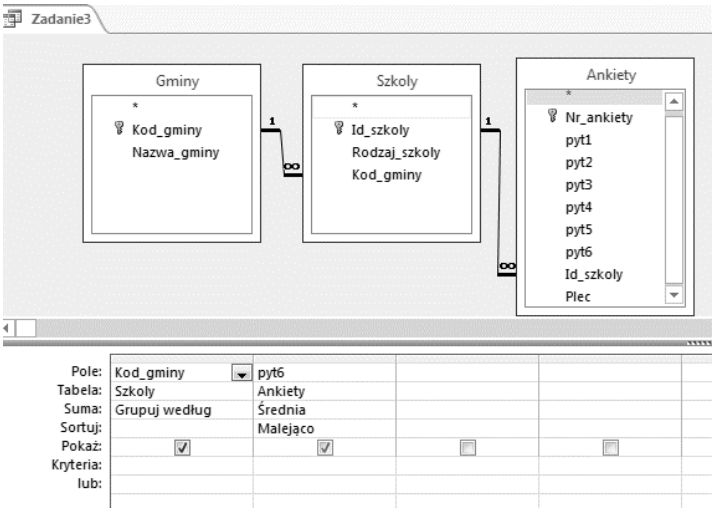
Zauważmy też, że średnie wartości mają być podane z dokładnością do dwóch miejsc po przecinku. Można to zrobić, np. ustawiając właściwości pól `pyt1`, `pyt2`, `pyt3`, `pyt4`, `pyt5` i `pyt6` w arkuszu właściwości: wybieramy format stałoprzecinkowy z 2 miejscami po przecinku.



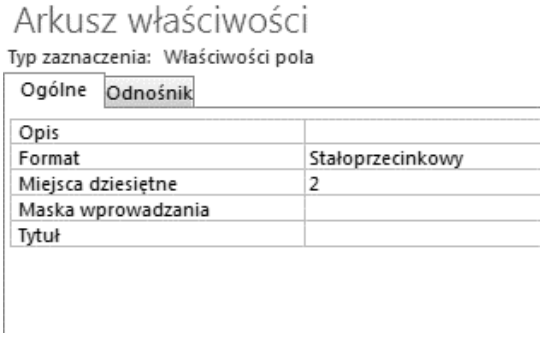
99.3.

W tym zadaniu także warto stworzyć kwerendę podsumowującą. W kwerendzie stosujemy grupowanie według pola `Kod_gminy` z tabeli `Gminy` oraz wyznaczamy średnią ocenę uczniów z odpowiedzi na szóste pytanie (dla pola `pyt6` z tabeli `Ankiety` stosujemy funkcję `Średnia`).

Zwróćmy też uwagę na dalszą część polecenia: „*Zestawienie uporządkuj malejąco ze względu na średnią ocenę*”. Musimy zatem wybrać w siatce projektowej sortowanie malejące dla pola `pyt6`.



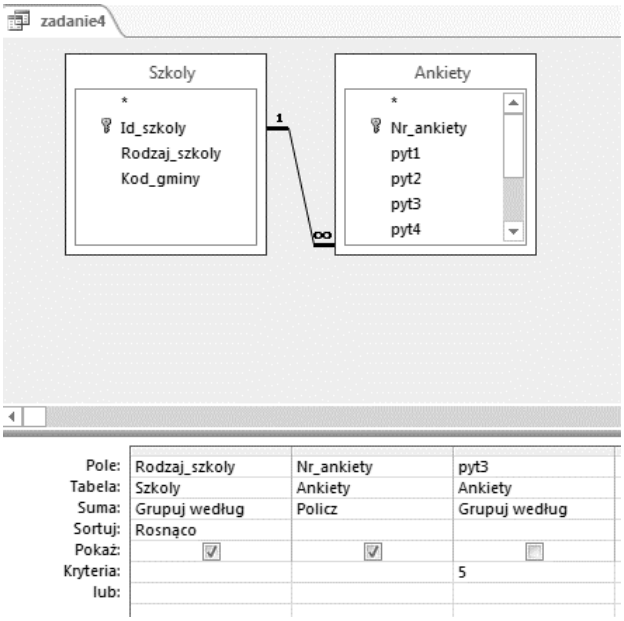
Także i tym razem średnie wartości odpowiedzi na pytanie szóste mają być podane z dokładnością do dwóch miejsc po przecinku, więc i tu w arkuszu właściwości pola `pyt6` wybieramy format stałoprzecinkowy z 2 miejscami po przecinku.



99.4.

W tym zadaniu należy utworzyć zestawienie zawierające dla każdego rodzaju szkoły informacje o liczbie uczniów, którzy podali najwyższą ocenę (5) dla pytania 3. W tym celu przygotujemy kwerendę, w której posłużymy się polami:

- `Rodzaj_szkoly` z tabeli `Szkoly` (grupujemy według tego pola — `Grupuj według`);
- `Nr_ankiety` z tabeli `Ankiety` (policzymy wszystkie ankiety spełniające zadane kryterium, wybieramy więc funkcję `Policz`);
- `pyt3` z tabeli `Ankiety` (pole to jest potrzebne do określenia warunku, zatem w wierszu Kryteria tego pola wpiszemy najwyższą ocenę (`5`), jaką mogli podać uczniowie, pole to nie jest wyświetlane w wyniku działania kwerendy, lecz jest niezbędne do spełnienia zadanego warunku).



Należy również pamiętać o wyborze sortowania rosnąco według pola `Rodzaj_szkoly`.

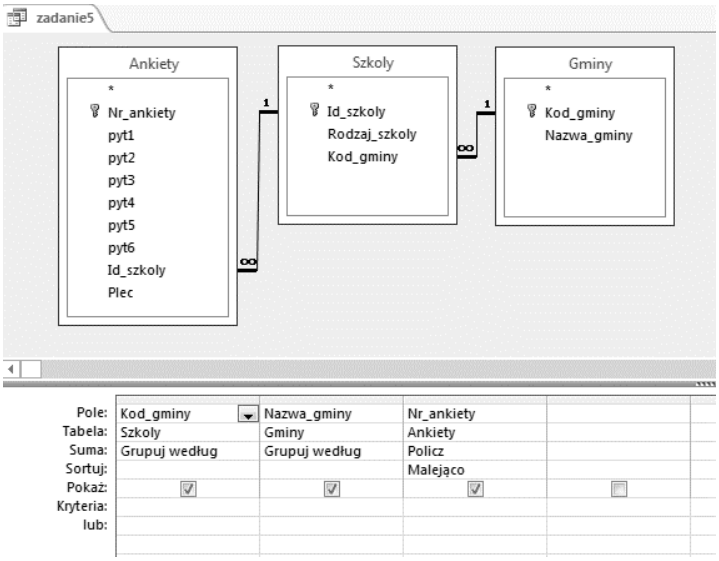
99.5.

Zadanie to można podzielić na dwa mniejsze zadania:

a) Dla każdej gminy wyznacz liczbę uczniów z jej terenu biorących udział w badaniu.

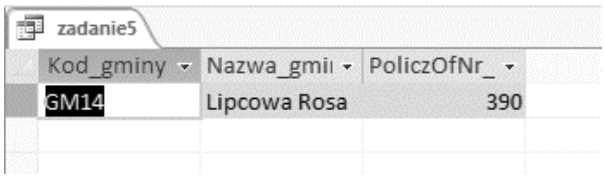
b) Podaj nazwę gminy z największą liczbą uczniów biorących udział w badaniu i liczbę tych uczniów.

Aby uzyskać odpowiedź na pierwsze z nich, przygotujemy kwerendę podsumowującą, wybierając pola: `Kod_gminy` z tabeli `Szkoly`, `Nazwa_gminy` z tabeli `Gminy` (grupujemy według tych pól) oraz pole `Nr_ankiety` z tabeli `Ankiety` (dla niego stosujemy funkcję `Policz`, aby zliczyć wszystkie wypełnione ankiety). Jeśli dla pola `Nr_ankiety` wybierzemy sortowanie malejące, to na górze tabeli wyświetlającej wynik działania tej kwerendy znajdziemy odpowiedź do drugiej części zadania.



Tak przygotowana kwerenda pozwala ustalić rozwiązanie — odczytamy je z pierwszego wiersza wyświetlanych wartości. Możemy jednak tak zmodyfikować kwerendę, aby zwracała tylko jeden (najwyższy) wynik.

Efekt działania byłby następujący:

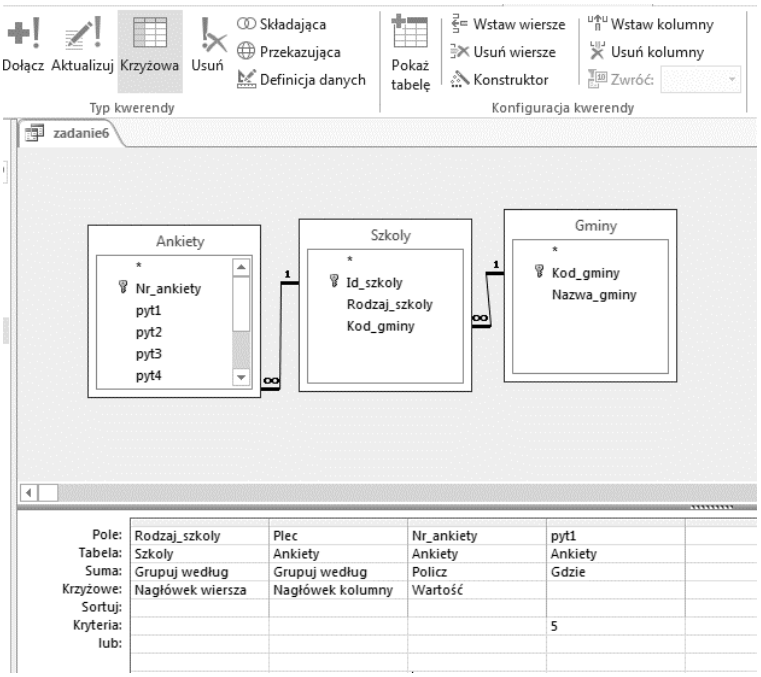


99.6.

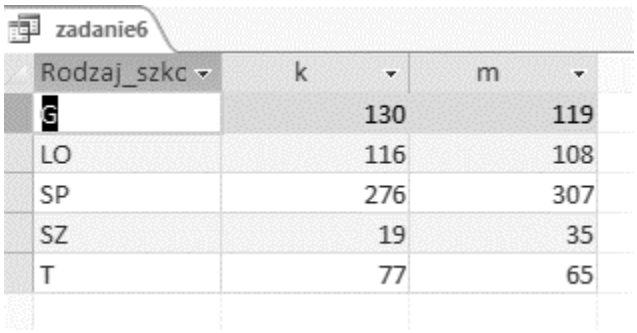
W zadaniu tym utworzymy kwerendę krzyżową, mamy bowiem utworzyć zestawienie zawierające osobno informacje o liczbie dziewcząt i osobno o liczbie chłopców w poszczególnych rodzajach szkół, którzy podali najwyższą ocenę 5 jako odpowiedź na pytanie 1.

Potrzebne zatem będą nam pole `Rodzaj_szkoly` z tabeli `Szkoly` jako **`nagłówek wiersza`** oraz pole `Plec` z tabeli `Ankiety` jako **`nagłówek kolumny`**. Na przecięciu wiersza i kolumny powinna pojawić się **`Wartość`**, uzyskana z policzenia pól `Nr_ankiety`

z tabeli `Ankiety`. Zestawienie dopełni ustalenie dodatkowego kryterium zadanego w poleceniu, czyli niezbędne jest grupowanie według pola `pyt1` z tabeli `Ankiety` oraz zadanie warunku dla tego pola (najwyższa ocena — 5 w odpowiedzi na pytanie 1). Musimy więc dodatkowo dopisać warunek dla pola (w siatce projektowej powtórnie umieszczamy pole `pyt1` z tabeli `Ankiety` i zaznaczamy funkcję `Gdzie` z ustalonym `Kryterium (5)`.



Efekt działania będzie następujący:



Rodzaj_szkc	k	m
G	130	119
LO	116	108
SP	276	307
SZ	19	35
T	77	65

Zadanie 100.

Wiązka zadań *E-learning*

Szkoła postanowiła wprowadzić dla uczniów dodatkowe zajęcia z informatyki z wykorzystaniem systemu e-learning. W plikach: `osoby.txt`, `listy.txt`, `punktacja.txt` znajdują się informacje na temat: uczniów szkoły, którzy uczyli się w systemie e-learning w okresie od 1.09.2014 do 15.02.2015, list zadań oraz uzyskanych przez uczniów wyników. Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.

Plik o nazwie `osoby.txt` zawiera 60 wierszy z informacjami na temat osób, które wysyłały rozwiązania zadań poprzez system. Są to: identyfikator osoby (`id_osoby`), jej nazwisko (`nazwisko`), imię (`imie`) oraz nazwa grupy, do której osoba została przydzielona (`grupa`).

Przykład

```
id_osoby  imie nazwisko  grupa
35   Joanna    Matura    G3
36   Anna      Piasecka  G3
37   Katarzyna Zienowicz G3
```

W pliku `listy.txt` znajduje się 11 wierszy z informacjami na temat list zadań zamieszczonych w systemie: numerem listy (`id_listy`), nazwą listy zadań (`nazwa`) oraz terminem oddania (`termin_oddania`).

Przykład

```
id_listy  nazwa     termin_oddania
7         C7        2015-01-26
8         P1        2014-11-10
```

Plik o nazwie `punktacja.txt` zawiera 653 wiersze z informacjami o wynikach uczniów. Są to: liczba porządkowa (`lp`), identyfikator danej osoby (`id_osoby`), identyfikator listy zadań (`id_listy`), liczba punktów zdobytych przez daną osobę (`punkty`) oraz data przesłania rozwiązania listy zadań (`data`).

Przykład

```
lp   id_osoby  id_listy  punkty    data
1    1         1         12        2014-10-20
2    1         2         12        2014-11-03
3    1         8         22        2014-11-10
```

Wykorzystując dane zawarte w tych plikach i dostępne narzędzia informatyczne, wykonaj poniższe zadania, a wyniki zapisz w pliku o nazwie `wyniki_elearning.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie.

100.1.

Utwórz zestawienie, w którym dla każdej listy zadań podasz średnią liczbę punktów otrzymanych za te zadania przez uczniów. W zestawieniu podaj nazwę listy oraz średnią liczbę punktów zaokrągloną do dwóch miejsc po przecinku.

100.2.

Utwórz zestawienie, w którym podasz imiona i nazwiska osób, które spóźniły się o 14 lub więcej dni z oddaniem dowolnej listy o nazwie zaczynającej się od litery „P”.

100.3.

Na podstawie liczby wszystkich zdobytych przez uczniów punktów wystawione zostały oceny według zasad przedstawionych w tabeli:

przedział punktów	**ocena**
(0, 72)	1
[72, 90)	2
[90, 126)	3
[126, 153)	4
[153, 180)	5

Podaj, ile osób otrzymało ocenę 1, 2, 3, 4, 5.

100.4.

Utwórz czytelne zestawienie tabelaryczne, w którym dla każdej grupy podasz, ile osób otrzymało liczbę punktów równą 10, 11, 12. W swoich obliczeniach weź pod uwagę wszystkie listy zadań.

100.5.

Podaj imiona i nazwiska osób, które nie wysłały przynajmniej jednej listy zadań. Zestawienie posortuj rosnąco ze względu na nazwiska osób.

Zadanie 101.

Wiązka zadań *Karta MaturaSport*

Władze miasta wprowadziły program *MaturaSport*, w ramach którego opłacają zajęcia sportowe dla uczniów ostatnich klas szkół ponadgimnazjalnych. Uczniowie korzystają z kart *MaturaSport*, które umożliwiają wstęp do różnych obiektów sportowych i uczestnictwo w prowadzonych tam zajęciach. W następujących plikach zgromadzono dane dotyczące wykorzystania kart programu *MaturaSport* w kwietniu 2014 roku. Dane w plikach są oddzielone średnikami, a pierwszy wiersz zawiera nagłówki kolumn.

Każdy wiersz w pliku `osoby.txt` zawiera informacje o jednym użytkowniku karty: jego identyfikator (`Id_uzytkownika`), nazwisko (`Nazwisko`), imię (`Imie`) i płeć (`Plec`). Płeć jest oznaczona literą „K” lub „M”.

Przykład

```
Id_uzytkownika;Nazwisko;Imie;Plec
1;Olszowka;Klara;K
2;Wieruszewski;Antoni;M
```

W pliku `zajecia.txt` zawarte są informacje o zajęciach prowadzonych w obiektach sportowych, biorących udział w programie. Każdy wiersz zawiera: identyfikator zajęć (`Id_zajec`), nazwę obiektu (`Obiekt`), rodzaj zajęć (`Zajecia`) i koszt (`Koszt`) jednego wejścia na zajęcia opłacany z programu *MaturaSport.*

Przykład

```
Id_zajec;Obiekt;Zajecia;Koszt
1;Redeco;Basen;9
```

Każdy wiersz pliku `wejscia.txt` zawiera informacje o pojedynczym wejściu użytkownika karty na wybrane zajęcia: numer porządkowy (`Lp`), identyfikator użytkownika (`Id_uzytkownika`), datę (`Data`) i identyfikator zajęć (`Id_zajec`).

Przykład

```
Lp ;Id_uzytkownika;Data;Id_zajec
1;1;2014-04-05;16
```

Wykorzystując dane zawarte w plikach `osoby.txt`, `zajecia.txt`, `wejscia.txt`, rozwiąż poniższe zadania. Odpowiedzi zapisz do pliku `wyniki_sport.txt`, a każdą z nich poprzedź cyfrą oznaczającą właściwe zadanie.

101.1.

Podaj, ile kobiet i ilu mężczyzn uczęszczało na zajęcia `Fitness TBC`. Zwróć uwagę, że niektóre osoby mogły być na tych zajęciach kilkakrotnie, a w zestawieniu powinny być uwzględnione tylko raz.

101.2.

Utwórz zestawienie, w którym dla każdego obiektu podasz, jaką łączną kwotę zapłacono za prowadzone w nim zajęcia.

101.3.

Podaj nazwiska i imiona osób, które w dniu 16 kwietnia 2014 r. uczestniczyły w więcej niż jednych zajęciach.

101.4.

Podaj rodzaj zajęć, w których uczestniczyło najwięcej osób. Podaj podaj liczbę tych osób i nazwę obiektu, w którym te zajęcia były prowadzone.

101.5.

Utwórz zestawienie, w którym dla każdego obiektu podasz, ile odnotowano w nim wejść na zajęcia. Zestawienie uporządkuj alfabetycznie według nazw obiektów.

Zadanie 102.

Wiązka zadań *Portal społecznościowy*

Użytkownicy pewnego niszowego portalu społecznościowego mogą zawierać ze sobą znajomości, a także dzielić się zdjęciami. W plikach tekstowych podane są dane pewnej grupy użytkowników portalu, lista par użytkowników będących znajomymi, a także lista zdjęć, które opublikowali użytkownicy. Dane w wierszach rozdzielone są znakiem tabulacji, a pierwszy wiersz każdego pliku jest wierszem nagłówkowym.

Plik `uzytkownicy.txt` opisuje użytkowników portalu: dla każdego podany jest kolejno: jego identyfikator w bazie (`ID_uzytkownika`), imię (`Imie`), nazwisko (`Nazwisko`), kraj, z którego on pochodzi (`Kraj`) oraz płeć (`Plec`, M — mężczyzna, K — kobieta):

```
ID_uzytkownika Imie           Nazwisko  Kraj
     Plec
1              Joshua         King      Stany Zjednoczone   M
2              Leonardo       De Luca   Wlochy              M
3              Aleksandra     Krawczyk  Polska              K
```

Plik `znajomosci.txt` zawiera listę par znajomych. Każdy wiersz zawiera dwa różne identyfikatory użytkowników, którzy są znajomymi (`Znajomy_1` oraz `Znajomy_2`), oraz datę zawarcia znajomości (`Data`).

```
Znajomy_1 Znajomy_2 Data
64        12        2014-11-23
81        46        2013-01-27
82        13        2013-11-01
```

W pliku `zdjecia.txt` opisane są zdjęcia opublikowane przez użytkowników. Każdy wiersz zawiera: identyfikator zdjęcia (`ID_zdjecia`), datę publikacji zdjęcia (`Data_dodania`), identyfikator użytkownika, który je dodał (`ID_uzytkownika`), oraz podaną w pikselach szerokość i wysokość zdjęcia (`Szerokosc` i `Wysokosc`).

```
ID_zdjecia     Data_dodania   ID_uzytkownika Szerokosc Wysokosc
1              2013-06-29     11             720       480
2              2014-09-08     56             720       480
3              2013-08-30     74             1440      576
```

Wykorzystując dane zawarte w podanych plikach oraz dostępne narzędzia informatyczne, wykonaj poniższe polecenia. Każdą odpowiedź umieść w pliku `wyniki.txt`, poprzedzając ją numerem odpowiedniego zadania.

102.1.

Podaj, ile zdjęć zostało opublikowanych w 2014 roku.

102.2.

Podaj wszystkie pary użytkowników, którzy są znajomymi i mają takie samo imię. Dla każdej takiej pary podaj imię, nazwisko i kraj pochodzenia każdego użytkownika.

102.3.

Podaj zestawienie 10 krajów, z których pochodzi najwięcej zdjęć. Dla każdego kraju podaj jego nazwę i liczbę zdjęć z niego przesłanych. Posortuj listę malejąco według liczby opublikowanych zdjęć.

102.4.

Podaj wysokość i szerokość zdjęcia, które ma najwięcej pikseli, oraz imię i nazwisko użytkownika, które je opublikował. Możesz założyć, że jest tylko jedno takie zdjęcie.

102.5.

Znajdź wśród użytkowników mężczyzn, którzy opublikowali w portalu zdjęcie, nie mając w momencie jego publikacji żadnych znajomych. Dla każdego takiego użytkownika podaj jego imię, nazwisko i kraj pochodzenia. Listę posortuj rosnąco według alfabetycznej kolejności nazwisk użytkowników.

Zadanie 103.

Wiązka zadań *Rentgenodiagnostyka*

W okresie sezonu narciarskiego w pracowni rentgenodiagnostyki szpitalnego oddziału ratunkowego (SOR) wykonano 2361 zdjęć RTG.

W plikach `pacjenci.txt`, `badania.txt` i `rtg.txt` znajdują się informacje na temat pacjentów, wykonanych im zdjęć RTG oraz rodzajach badań RTG wykonywanych przez pracownię. Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są średnikami.

W pliku `pacjenci.txt` znajdują się wiersze z informacjami o 816 pacjentach urodzonych przed rokiem 2000: `PESEL`, `Nazwisko`, `Imie`, `Plec` (*k* — kobieta, *m* — mężczyzna), `NFZ` (oddział Narodowego Funduszu Zdrowotnego, do którego należy pacjent).

Przykład

```
PESEL;Nazwisko;Imie;Plec;NFZ
37112515913;Latacki;Stanislaw;m;Opolski
38012109293;Stachniuk;Przemyslaw;m;Dolnoslaski
47101603441;Dubiel;Zdzislawa;k;Slaski
```

W pliku `badania.txt` znajdują się wiersze z informacjami o zdjęciach RTG wykonanych pacjentom: `PESEL`, `Id_badania`.

Przykład

```
PESEL;Id_badanie;
37112515913;Id_13;
38012109293;Id_16;
45032403378;Id_23;
```

Każdy wiersz pliku `rtg.txt` zawiera następujące informacje o wykonywanych w pracowni zdjęciach RTG: `Id_badania`, `Nazwa_badania`, `Cena_badania`.

Przykład

```
Id_badanie;Nazwa_badania;Cena_Badania
Id_1;RTG kosci pietowej;30
Id_10;RTG zeber;60
Id_11;RTG obojczyka;50
```

Wykorzystując dane zawarte w tych plikach i dostępne narzędzia informatyczne, wykonaj poniższe polecenia. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_RTG.txt`. Odpowiedź do każdego zadania poprzedź numerem je oznaczającym.

103.1.

Podaj nazwisko, imię oraz rok urodzenia pacjenta, któremu wykonano najwięcej zdjęć RTG, oraz liczbę tych zdjęć. Skorzystaj z informacji, że dwie pierwsze cyfry numeru PESEL to rok urodzenia.

103.2.

Pacjentami SOR byli mieszkańcy 16 oddziałów NFZ. Oddziały te pokrywają koszty wykonanych zdjęć RTG osób, które do nich należą. Utwórz zestawienie zawierające informacje o sumie kosztów zdjęć RTG poniesionych przez poszczególne oddziały NFZ. Zestawienie posortuj malejąco ze względu na sumę kosztów.

103.3.

Podaj rodzaj zdjęć RTG, które wykonano:

a) najwięcej razy,

b) u największej liczby pacjentów.

W odpowiedzi do punktu a) podaj też, ile razy wykonano dane badanie. W odpowiedzi do punktu b) podaj też liczbę pacjentów, u których wykonano dane badanie.

103.4.

Podaj liczbę kobiet oraz liczbę mężczyzn, którym wykonano zdjęcia RTG.

103.5.

Dla okresu od roku 1900 do 1999 utwórz zestawienie zawierające numery kolejnych dziesięcioleci oraz liczbę pacjentów urodzonych w tych dziesięcioleciach. Przyjmujemy, że pierwsze dziesięciolecie to okres 1900-1909, drugie dziesięciolecie to okres 1910–1919 itd.

Zadanie 104.

Wiązka zadań Leki refundowane

Pracownik NFZ ma za zadanie skontrolować recepty na leki refundowane wypisane w jednej z przychodni w ciągu pierwszego kwartału 2015 roku. Ma do dyspozycji trzy pliki tekstowe: `recepty.txt`, `leki_refundowane.txt` i `grupy_lekow.txt`[3].

Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są średnikami.

Plik o nazwie `recepty.txt` zawiera informacje dotyczące wypisywanych recept; w każdym wierszu znajduje się: identyfikator recepty (`ID_recepty`), data wypisania recepty (`Data`) oraz 13-cyfrowy kod wypisanego leku (`Kod_leku`). Jedna recepta może zawierać maksymalnie pięć leków.

Przykład

```
ID_recepty;Data;Kod_leku
11/2015;2015-01-02;5909990752720
12/2015;2015-01-02;5909990969753
12/2015;2015-01-02;5909990967247
```

W pliku `leki_refundowane.txt` zapisane są w każdym wierszu: 13-cyfrowy kod leku (`Kod_leku`), maksymalnie 90-znakowa nazwa substancji czynnej (`Subst_czynna`), maksymalnie 160-znakowa nazwa leku, zawierająca postać i dawkę leku, (`Nazwa`), identyfikator

[3] Źródło: http://www.mz.gov.pl/leki/refundacja/lista-lekow-refundowanych-obwieszczenia-ministra-zdrowia

grupy, do której dany lek należy (`Id_grupy`), jego cena (`Cena_detaliczna`) oraz cena refundowana (`Cena_refundowana`).

Przykład

```
Kod_leku;Subst_czynna;Nazwa;Id_grupy;Cena_detaliczna;Cena_refundowana
4013054024331;Metforminum;Siofor 500, tabl. powl., 500 mg;15.0;12,21;3,2
4013054024348;Metforminum;Siofor 850, tabl. powl., 850 mg;15.0;19,87;4,08
5901720140012;Doxazosinum;Dozox, tabl., 4 mg;76.0;70,74;9,6
```

Plik o nazwie `grupy_lekow.txt` zawiera identyfikator grupy (`Id_grupy`) oraz maksymalnie 200-znakową nazwę grupy (`Nazwa_grupy`).

Przykład

```
Id_grupy;Nazwa_grupy
1.0;Leki blokujace receptory histaminowe H2 - stosowane doustnie
10.0;Leki przeciwbiegunkowe - loperamid
100.1;Sulfametoksazol w polaczeniu z trimetoprymem do stosowania doustnego
- postacie stale
```

Korzystając z danych zawartych w tych plikach oraz z dostępnych narzędzi informatycznych, wykonaj poniższe polecenia. Każdą odpowiedź zamieść w pliku `wyniki.txt`, poprzedzając ją numerem odpowiedniego zadania.

104.1.

Podaj, w którym dniu kontrolowanego okresu wypisano w przychodni najwięcej recept, podaj odpowiednią datę i liczbę recept z tego dnia. Pamiętaj, że na jednej recepcie może być wypisanych kilka leków.

104.2.

Podaj nazwę grupy leków, w której znajduje się najwięcej leków refundowanych w 100%, czyli takich, których cena refundowana wynosi 0 zł.

104.3.

Sporządź zestawienie, w którym dla każdego miesiąca podana będzie liczba wszystkich wypisanych recept oraz sumaryczna wartość wszystkich leków z recept wypisanych w tym miesiącu (w obliczeniach weź pod uwagę cenę detaliczną).

104.4.

Leki mają różne ceny detaliczne. Podaj cenę detaliczną najdroższego leku, na który wypisano receptę w badanej przychodni, oraz nazwę grupy, do której ten lek należy.

104.5.

Dopłata funduszu do leku to różnica pomiędzy ceną detaliczną a ceną refundowaną.

Utwórz zestawienie zawierające daty wystawienia i identyfikatory recept, dla których suma dopłat do wszystkich leków z danej recepty jest większa niż 2000 zł. Zestawienie posortuj rosnąco według dat.

Zadanie 105.

Wiązka zadań *Rośliny ogrodowe*

W poniżej opisanych plikach zgromadzono dane dotyczące zamówień złożonych w internetowym sklepie ogrodniczym[4]. Zakupów mogą dokonywać klienci, którzy dokonali wcześniejszej rejestracji w sklepie. Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są średnikami.

W pliku `osoby.txt` każdy wiersz zawiera informacje o zamawiającym: identyfikator klienta (`Id_klienta`), jego nazwisko (`Nazwisko`), imię (`Imie`) i miasto, z którego on pochodzi (`Miasto`).

Przykład

```
Id_klienta;Nazwisko;Imie;Miasto
200;Koprowski;Maurycy;Wroclaw
```

W pliku `rosliny.txt` zawarte są informacje o roślinach ogrodowych dostępnych w sklepie internetowym. Każdy wiersz zawiera następujące informacje: identyfikator rośliny (`Id_rosliny`), nazwę rośliny (`Nazwa`), cenę jej sadzonki (`Cena`), kolor kwiatów (`Kolor_kwiatow`), okres kwitnienia (`Okres_kwitnienia`), rozmiary doniczki (`Rozmiary_doniczki`).

Przykład

```
Id_rosliny;Nazwa;Cena;Kolor_kwiatow;Okres_kwitnienia;Rozmiary_doniczki
54;Aconitum napellus;5;ciemno-niebieskie;IX-X;13x13x13
```

W pliku `zamowienia.txt` każdy wiersz zawiera informacje o pojedynczej pozycji zamówienia: numer porządkowy (`Lp`), identyfikator klienta (`Id_klienta`), datę (`Data`), liczbę sadzonek (`Liczba_sadzonek`) i identyfikator rośliny (`Id_rosliny`).

Przykład

```
Lp;Id_klienta;Data;Liczba_sadzonek;Id_rosliny
1;546;2014-03-01;6;554
```

Wykorzystując dane zawarte w plikach `osoby.txt`, `rosliny.txt`, `zamowienia.txt`, wykonaj poniższe polecenia. Odpowiedzi zapisz do pliku `wyniki_rosliny.txt`, a każdą z nich poprzedź cyfrą oznaczającą odpowiednie zadanie.

105.1.

Na zakończenie dnia sklep przygotowywał dla każdego klienta specyfikację zamówionego w ciągu całego dnia towaru i na tej podstawie sporządzał fakturę, która była dołączana do wysyłki (kumulowanie zakupów od jednego klienta zmniejsza liczbę przesyłek kurierskich). To oznacza, że na jednej fakturze były umieszczane pozycje z zamówień złożonych tego samego dnia. Wartość faktury to suma wartości zakupionych towarów.

Przykład

Klient zakupił w danym dniu 6 sadzonek w cenie 9 zł oraz 15 sadzonek w cenie 10 zł. Wartość jego faktury wynosi $6 \cdot 9 + 15 \cdot 10 = 204$ zł.

---

[4] Dane o roślinach zostały przygotowane na podstawie http://ogrodkomercyjny.pl/

Oblicz, ile faktur wystawił sklep ogrodniczy, oraz podaj najwyższą wartość wystawionej faktury. Podaj też, ilu klientom wystawiono więcej niż jedną fakturę.

105.2.

Utwórz zestawienie, w którym podasz liczbę pozycji złożonych przez osoby z poszczególnych miast zamówień na rośliny kwitnące dokładnie w okresie VII–VIII.

105.3.

Wyszukaj pozycje zamówień, które dotyczą więcej niż 10 sadzonek roślin o kwiatach w kolorze `bialo-liliowe`. Dla znalezionych zamówień podaj nazwiska i imiona osób, które je złożyły, liczbę sadzonek oraz nazwy roślin.

105.4.

Podaj nazwy roślin, które nie zostały uwzględnione w żadnym zamówieniu.

105.5.

Utwórz zestawienie zawierające liczbę pozycji zamówień roślin w poszczególnych rozmiarach doniczek.

Zadanie 106.

Wiązka zadań *Obserwacje ptaków*

Wolontariusze zarejestrowani w serwisie internetowym *www.awibaza.pl* wprowadzają do bazy informacje o swoich obserwacjach ptaków. W trzech plikach tekstowych przedstawiono dane zaczerpnięte z tej bazy. Dane w wierszach oddzielone są pojedynczymi znakami tabulacji. W każdym pliku pierwszy wiersz jest wierszem nagłówkowym

Plik `gatunki.txt` zawiera informacje o gatunkach ptaków: identyfikatory `ID_gatunku`, nazwy zwyczajowe (`nazwa_zwyczajowa`) i łacińskie (`nazwa_lacinska`). Każdy gatunek opisany jest w osobnym wierszu.

Przykład

```
ID_gatunku  nazwa_zwyczajowa     nazwa_lacinska
5           bazant zlocisty      Chrysolophus pictus
17          bielik               Haliaeetus albicilla
54          gawron               Corvus frugilegus
```

W pliku `lokalizacje.txt` jest lista miejsc, w których dokonywano obserwacji. W każdym miejscu można było dokonywać wielu obserwacji w różnych terminach. W pliku podano dla poszczególnych miejsc: identyfikator miejsca (`ID_lokalizacji`), jego nazwę (`lokalizacja`), nazwę powiatu, na terenie którego się ono znajduje (`powiat`), oraz jego krótki opis (`opis`).

```
ID_lokalizacji  lokalizacja     powiat       opis
1               Cieplewo        gdanski      laka podmokla, jezioro
6               Hel             pucki        plaza, brzeg zatoki
9               Koscierzyna     koscierski   transekt przez miasto
```

W pliku `obserwacje.txt` w każdym wierszu znajduje się zapis danych z jednej obserwacji. Każda obserwacja dotyczyła jednego gatunku ptaków, ale mogła obejmować wiele osobników tego gatunku. Dla każdej obserwacji podano: identyfikator obserwowanego gatunku

(`ID_gatunku`), identyfikator lokalizacji obserwacji (`ID_lokalizacji`), datę i czas jej początku (`poczatek`), datę i czas jej zakończenia (`koniec`), liczebność ptaków (`liczebnosc`) oraz ich zachowanie w czasie obserwacji (`zachowanie`). Obserwacja zaczynała się i kończyła zawsze w tym samym dniu.

ID_gatunku	ID_lokalizacji	poczatek	koniec	liczebnosc	zachowanie
92	24	1984-12-16 12:55	1984-12-16 13:55	107	plywa
124	13	2008-01-14 08:15	2008-01-14 11:45	1	odpoczywa
23	3	2014-10-25 07:00	2014-10-25 07:15	4	zeruje

Rozwiąż poniższe zadania, wykorzystując dostępne narzędzia informatyczne. Wyniki zamieść w pliku tekstowym o nazwie `ptaki_wyniki.txt`. Do oceny oddaj plik tekstowy zawierający wynik zadań oraz plik zawierający komputerową realizację twojego rozwiązania.

106.1.

Podać nazwy zwyczajowe trzech gatunków ptaków, które były obiektem największej liczby obserwacji.

106.2.

W których miesiącach można było zaobserwować remiza? Podaj numery miesięcy (liczby z zakresu 1..12), w których obserwowano gatunek o nazwie zwyczajowej „remiz". Dla każdego z tych miesięcy podaj łączną liczbę wszystkich zaobserwowanych osobników remiza. Weź pod uwagę wszystkie obserwacje, z różnych lat.

106.3.

Które z europejskich gatunków krukowatych — ptaków z rodzaju *Corvus* — można zaobserwować w miastach? Ile ich zaobserwowano?

Podaj nazwy zwyczajowe wszystkich gatunków ptaków z rodzaju *Corvus* (słowo *Corvus* stanowi fragment łacińskiej nazwy gatunku) zaobserwowanych w lokalizacjach położonych na terenach miejskich (zawierających słowo "miasto" w opisie lokalizacji obserwacji).

106.4.

Obserwacje dokonane w tej samej lokalizacji i tym samym czasie (mające jednakowe wartości w polu `poczatek` i w polu `koniec`) stanowią **grupę**.

a) Wyznacz **grupę obserwacji trwającą najdłużej.** Podaj jej lokalizację, datę, czas trwania w minutach i łączną liczbę osobników zaobserwowanych w tej grupie.

Sprawnością grupy obserwacji nazywamy średnią liczbę osobników wszystkich gatunków obserwowanych w ciągu 1 minuty przez grupę.

$$\text{sprawność} = \frac{\text{łączna liczba osobników obserwowanych w grupie}}{\text{czas trwania obserwacji grupy w minutach}}$$

b) Podaj **największą osiągniętą sprawność** grupy obserwacji, w zaokrągleniu do 3 cyfr po przecinku, oraz datę i lokalizację grupy, która ją osiągnęła.

106.5.

Wyszukaj wszystkie obserwacje osobników **żurawia** (o nazwie zwyczajowej: „zuraw").

a) Podaj liczbę wszystkich zaobserwowanych osobników żurawia.

b) Utwórz w postaci tabeli 2-wymiarowej zestawienie, w którym dla poszczególnych powiatów podasz liczby osobników żurawia zaobserwowanych na ich terenach, z podziałem na różne zachowania obserwowanego ptaka (*gniazduje, leci* itp.).

Zadanie 107.

Wiązka zadań *Loty pasażerskie*

W plikach: `loty.txt, pasazerowie.txt, bilety.txt` znajdują się informacje na temat lotów, pasażerów i biletów lotniczych zakupionych przez nich w biurze podróży w drugim kwartale 2014 roku. Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.

W pliku `loty.txt` znajduje się 1027 wierszy z informacjami o lotach pasażerskich: numerem identyfikacyjnym (`id_lotu`), miejscem docelowym (`miejsce_docelowe`), datą wylotu (`data`) oraz godziną wylotu (`godzina`).

Przykład

id_lotu	miejsce_docelowe	data	godzina
37	Warszawa	2014-04-04	12:05
38	Zurych	2014-04-04	13:50
39	Londyn Stansed	2014-04-04	18:10

Plik o nazwie `pasazerowie.txt` zawiera 302 wiersze z informacjami na temat pasażerów, którzy kupili bilety. Są to: identyfikator pasażera (`id_pasazera`), jego nazwisko (`nazwisko`), imię (`imie`), ulica, przy której mieszka, wraz z numerem domu (`adres`), miejscowość (`miejscowosc`) oraz telefon (`tel`).

Przykład

id_pasazera	nazwisko	imie	adres	miejscowosc	tel
202	Antczak	Edyta	Czerwcowa 40/6	Walbrzych	735223964
203	Karpik	Hanna	Drewniana 8/6	Dzierzoniow	312271637

W pliku `bilety.txt` znajduje się 2251 wierszy z informacjami na temat zakupionych przez pasażerów biletów: numerem identyfikacyjnym lotu (`id_lotu`) oraz identyfikatorem pasażera (`id_pasazera`).

Przykład

id_lotu	id_pasazera
142	100
161	420
170	161
171	155

Wykorzystując dane zawarte w tych plikach i dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi do poszczególnych zadań zapisz w pliku tekstowym o nazwie `wyniki_loty_pasazerskie.txt`. Wyniki do każdego zadania poprzedź numerem oznaczającym to zadanie.

107.1.

Podaj 3 miejsca docelowe, do których wyloty odbyły się w największą liczbę dni. W zestawieniu podaj miejsce docelowe oraz liczbę dni.

107.2.

Utwórz zestawienie, w którym podasz nazwiska i imiona pasażerów, którzy zakupili więcej niż 15 biletów, oraz liczbę biletów kupionych przez każdego z tych pasażerów.

107.3.

Utwórz zestawienie, w którym dla każdego numeru miesiąca z badanego okresu podasz liczbę biletów kupionych przez osoby z Wrocławia (`Wroclaw`).

107.4.

Utwórz zestawienie, w którym podasz imiona i nazwiska pasażerów, którzy kupili bilety do dowolnego lotniska w Londynie na samoloty, których wyloty odbyły się między 8:00 a 10:00. Zestawienie posortuj rosnąco według kolejności alfabetycznej nazwisk.

107.5.

Utwórz zestawienie lotów, na które nie kupiono w biurze biletów. Podaj w nim numery lotów i ich miejsca docelowe.

Zadanie 108.

Wiązka zadań *Stacje benzynowe*

Informacje o wybranych drogach w Polsce i położonych przy nich stacjach benzynowych zamieszczone są w plikach: `drogi.txt`, `kategorie.txt`, `sieci.txt` oraz `stacje.txt`. Dane w plikach rozdzielone są znakiem tabulatora, pierwszy wiersz każdego pliku jest wierszem nagłówkowym.

Plik `drogi.txt` zawiera informacje dotyczące dróg. Są to: unikalny numer drogi (id_drogi), nazwa drogi (nazwa), jej długość (dlugosc) oraz identyfikator jej kategorii (id kategorii). Długość drogi jest podana w kilometrach i zaokrąglona w dół do pełnych kilometrów.

Przykład

id_drogi	nazwa	dlugosc	id_kategorii
1	Autostrada Bursztynowa	582	A
2	Autostrada Wolnosci	623	A
3	Zachodnia Droga Ekspresowa	480	S

Plik `kategorie.txt` zawiera informacje dotyczące kategorii dróg: unikalny identyfikator kategorii (id_kategori) oraz nazwę kategorii (kategoria).

Przykład

id_kategorii	kategoria
A	autostrada
S	droga ekspresowa

Plik `sieci.txt` zawiera informacje dotyczące sieci stacji benzynowych: unikalny identyfikator sieci (id_sieci) oraz nazwę sieci, do której należy stacja (nazwa_sieci).

Przykład

id_sieci	nazwa_sieci
1	Dobre Paliwo
2	Standard Oil

Plik `stacje.txt` zawiera informacje dotyczące stacji benzynowych: unikalny identyfikator stacji (id_stacji), numer drogi, przy której jest położona stacja (id_drogi), oraz identyfikator sieci, do której ona należy stacja (id_sieci).

Przykład

id_stacji	id_drogi	id_sieci
1	8	8
2	8	5

Korzystając z danych zawartych w tych plikach oraz dostępnych narzędzi informatycznych, wykonaj poniższe polecenia. Odpowiedzi zapisz w pliku `wyniki.txt`, a odpowiedź do każdego zadania poprzedź numerem oznaczającym to zadanie.

108.1.

Podaj sumaryczną długość dróg wszystkich kategorii.

108.2.

Podaj nazwę i długość najdłuższej drogi, przy której nie leży żadna stacja benzynowa.

108.3.

Średnia odległość pomiędzy stacjami benzynowymi to stosunek całkowitej długości drogi do liczby położonych przy niej stacji. Jeżeli przy drodze nie ma stacji benzynowych, odległość ta jest niezdefiniowana i danej drogi nie bierzemy pod uwagę podczas porównywania średnich odległości pomiędzy stacjami.

Znajdź drogę, dla której średnia odległość między stacjami benzynowymi jest najmniejsza. Podaj numer drogi, jej nazwę oraz średnią odległość pomiędzy stacjami na tej drodze, z dokładnością do 0,1 km.

108.4.

Podaj nazwy dróg mających w nazwie słowo „autostrada”, ale będących drogami innych kategorii.

108.5.

Wykonaj zestawienie liczby stacji benzynowych należących do poszczególnych sieci z podziałem na kategorie dróg, przy których są położone te stacje.

Zadanie 109.

Wiązka zadań *Urządzenia budowlane*

Constall jest liderem wśród firm wynajmujących urządzenia i rusztowania dla budownictwa. W trzech plikach tekstowych `sprzet_budowlany.txt`, `klienci.txt` i `wynajem.txt` zostały zapisane dane dotyczące działalności firmy w 2014 roku.

Pierwszy wiersz każdego z plików jest wierszem nagłówkowym, a dane w wierszach rozdzielone są znakami tabulacji.

Plik o nazwie `sprzet_budowlany.txt` zawiera informacje dotyczące oferowanego sprzętu budowlanego. W każdym wierszu znajduje się: identyfikator sprzętu (`ID_sprzetu`), nazwa sprzętu (`Nazwa_sprzetu`), koszt jego wynajęcia na dobę podany w zł (`Koszt_wynajecia`), kaucja za sprzęt w zł (`Kaucja`).

Przykład

```
ID_sprzetu  Nazwa_sprzetu                    Koszt_wynajecia   Kaucja
1           Agregat hydrauliczny ATLAS       100               1200
2           Agregat jednofazowy-moc: 2,8-3,2kW 65              1000
```

W pliku `klienci.txt` zapisane są w każdym wierszu: numer dowodu osobistego osoby wynajmującej sprzęt (`Nr_dowodu_osoby`), imię tej osoby (`Imie`) oraz jej nazwisko (`Nazwisko`).

Przykład

```
Nr_dowodu_osoby     Imie      Nazwisko
XGF208075           Radoslaw  Warszawski
GUZ058053           Kacper    Szwaja
```

Plik o nazwie `wynajem.txt` zawiera: datę wypożyczenia sprzętu (`Data_wypozycz`), datę jego zwrotu (`Data_zwrotu`), identyfikator wypożyczanego sprzętu (`ID_sprzetu`), odległość od magazynu firmy do miejsca, w którym sprzęt ma być odebrany przez klienta, podaną w km (`Transport_km`), numer dowodu osobistego klienta (`Nr_dowodu_osoby`).

Przykład

```
Data_wypozycz     Data_zwrotu ID_sprzetu  Transport_km      Nr_dowodu_osoby
2014-01-02        2014-01-07  106         0                 MZM066623
2014-01-02        2014-01-07  14          0                 FXN638961
```

Firma Constall pobiera opłaty za usługę transportu urządzenia do klienta według poniższego taryfikatora:

- transport do 10 km włącznie — 50 zł,
- transport powyżej 10 km — 100 zł.

Korzystając z danych zawartych w tych plikach oraz z dostępnych narzędzi informatycznych, wykonaj poniższe polecenia. Każdą odpowiedź umieść w pliku `wyniki.txt`, poprzedzając ją numerem odpowiedniego zadania.

109.1.

Podaj nazwę urządzenia wypożyczanego najczęściej oraz liczbę wypożyczeń tego urządzenia.

109.2.

Jaką łączną kwotę kaucji za wszystkie wypożyczone przez siebie urządzenia wpłacił Andrzej Rydawski identyfikujący się dowodem osobistym o numerze JCK343973?

109.3.

Podaj imiona i nazwiska osób, które wynajmowały od firmy Constall w 2014 roku więcej niż trzy różne urządzenia.

109.4.

Podaj liczbę zarejestrowanych w bazie firmy klientów, którzy w 2014 roku nie wypożyczyli żadnego urządzenia.

109.5.

Dla każdego miesiąca (od stycznia do grudnia) podaj:

a) przychód firmy z tytułu wypożyczeń urządzeń budowlanych w tym miesiącu,

b) przychód z tytułu opłat za transport w tym miesiącu.

Przykład do punktu a)

Jeżeli drabina aluminiowa 3-częściowa została wypożyczona 10.12.2014, a zwrócona 13.12.2014, to koszt wynajęcia jest obliczany następująco:

`liczba dni * cena za dobę` (3*300zł=900 zł).

Nie ma możliwości wypożyczenia i oddania sprzętu w tym samym dniu.

Uwaga: Opłata jest pobierana w dniu wynajęcia urządzenia i należy ją uwzględnić jako przychód w miesiącu, w którym urządzenie zostało wypożyczone.

Zadanie 110.

Wiązka zadań *Miejscowości w Polsce*

Informacje o podziale administracyjnym Polski oraz o wszystkich miejscowościach w jej obrębie zapisane są w plikach: `wojewodztwa.txt`, `powiaty.txt`, `gminy.txt` oraz `miejscowosci.txt`. Dane w plikach rozdzielone są znakiem tabulatora, pierwszy wiersz każdego pliku jest wierszem nagłówkowym.

Plik `wojewodztwa.txt` zawiera informacje dotyczące województw: unikalny numer województwa (id_wojewodztwa) oraz nazwę województwa (nazwa_wojewodztwa).

Przykład

id_wojewodztwa	**nazwa_wojewodztwa**
02	dolnoslaskie
04	kujawsko-pomorskie

Plik `powiaty.txt` zawiera informacje dotyczące powiatów: unikalny numer powiatu (id_powiatu), numer województwa, do którego on należy (id_wojewodztwa), oraz nazwę powiatu (nazwa_powiatu).

Przykład

id_powiatu	**id_wojewodztwa**	**nazwa_powiatu**
0201	02	boleslawiecki
0202	02	dzierzoniowski

Plik `gminy.txt` zawiera informacje dotyczące gmin: unikalny numer gminy (id_gminy), numer powiatu, do którego ona należy (id_powiatu), oraz nazwę gminy (nazwa_gminy).

Przykład

id_gminy	id_powiatu	nazwa_gminy
0201022	0201	Boleslawiec
0206011	0206	Karpacz

Plik `miejscowosci.txt` zawiera informacje dotyczące miejscowości: unikalny numer miejscowości (id_miejscowosci), numer gminy, w której jest ona położona(id_gminy), nazwę miejscowości (nazwa_miejscowosci) oraz słowne określenie typu miejscowości (typ_miejscowosci).

Przykład

id_miejscowosci	id_gminy	nazwa_miejscowosci	typ_miejscowosci
0162398	2206032	Abisynia	osada
0954722	0609052	Abramowice Koscielne	wies

Wykorzystując dane zawarte w tych plikach oraz dostępne narzędzia informatyczne, rozwiąż poniższe zadania. Odpowiedzi zapisz w pliku `wyniki.txt`, a odpowiedź do każdego podpunktu poprzedź numerem oznaczającym dane zadanie.

Jeżeli w treści pytań użyte są pojęcia:

- **gmina miejska** — należy przez to rozumieć gminę, na terenie której jest dokładnie jedna miejscowość i jest to miasto,
- **gmina miejsko-wiejska** — należy przez to rozumieć gminę, na terenie której znajduje się kilka miejscowości, z których przynajmniej jedna jest miastem;
- **gmina wiejska —** należy przez to rozumieć gminę, na terenie której nie ma żadnego miasta;
- **miasto na prawach powiatu** — należy przez to rozumieć miasto, którego nazwa jest taka sama jak nazwa powiatu, na terenie którego się znajduje.

110.1.

Podaj liczbę miast na terenie Polski.

110.2.

Oblicz, ile jest miejscowości każdego typu w powiecie brodnickim (wieś, miasto, osada itd.).

110.3.

Znajdź powtarzające się w kraju nazwy powiatów. Podaj nazwy tych powiatów oraz nazwy województw, w których te powiaty się znajdują. Zestawienie posortuj alfabetycznie według nazwy powiatów. Powiaty o tych samych nazwach uporządkuj alfabetycznie według nazw województw.

110.4.

Ile jest gmin wiejskich w województwie kujawsko-pomorskim?

110.5.

Wykonaj zestawienie gmin miejskich o nazwach zaczynających się na „J”, podając nazwy powiatów i województw, w których są one położone.

Zadanie 111.

Wiązka zadań *Malware*

Malware Domain List to niekomercyjny projekt społecznościowy, w którym tworzona jest lista adresów stron internetowych i dokumentów stwarzających zagrożenie: trojanów, exploitów itp. Listę na bieżąco uzupełniają profesjonalni entuzjaści bezpieczeństwa w sieci.

W trzech plikach tekstowych przedstawiono dane zaczerpnięte z tej listy. Dane w wierszach oddzielone są pojedynczymi znakami tabulacji. W każdym pliku pierwszy wiersz jest wierszem nagłówkowym.

W pliku `malware.txt` znajdują się pozycje z tej listy, wybrane z okresu od stycznia 2014 do stycznia 2015 roku włącznie. Podano: datę rejestracji zagrożenia (`data`), adres IP komputera udostępniającego zagrożenie (`IP`), opis zagrożenia (`opis`), numer ASN sieci, do której ten komputer należy (`ASN`), ścieżkę dostępu do szkodliwej strony lub do zasobu (`URL`).

Przykład

data	IP	opis	ASN	URL
2014-12-17	62.76.74.228	Trojan.Downloader	51408	my-screenshot.net/
2014-12-04	31.41.218.232	CryptoLocker	42655	mysda24.com/f/pacchetto_38.zip
2014-11-25	89.218.31.11	Script.exploit	9198	zakonodatelstvo.kz/russ.html

W pliku `asn.txt` znajdują się informacje o sieciach komputerowych, zawierające m.in. informacje o numerze ASN (*Autonomic System Number*) — identyfikatorze sieci, wykorzystywanym w konfiguracji routerów. W pliku podano dla każdej sieci: numer ASN (`ASN`), internetowy identyfikator kraju (`ID_kraju`), nazwę organizacji regionalnej przydzielającej adres ASN (`region`) oraz nazwę firmy zarządzającej siecią (`siec`).

Przykład

ASN	ID_kraju	region	siec
1267	it	ripencc	ASN-INFOSTRADA WIND Telecomunicazioni S.p.A.
2514	jp	apnic	INFOSPHERE NTT PC Communications, Inc
2914	us	arin	NTT-COMMUNICATIONS-2914 - NTT America, Inc

Regionalne organizacje przydzielające adresy ASN obejmują:

- apnic (*Asia Pacific Network Information Centre*) — rejon Azji i Pacyfiku,
- arin (*American Registry for Internet Numbers*) — rejon Ameryki Północnej,
- lacnic (*Latin-American and Caribbean*) — rejon Ameryki Łacińskiej i Wysp Karaibskich,
- ripencc (*Réseaux IP Européens*) — rejon Europy, Bliskiego Wschodu i centralnej Azji,
- afrinic — rejon Afryki.

W pliku `kraje.txt` podano nazwy krajów (`kraj`) oraz ich 2-literowe identyfikatory internetowe (`ID_kraju`).

Przykład

kraj	ID_kraju
Australia	au
France	fr
Hungary	hu

Rozwiąż poniższe zadania, wykorzystując dostępne narzędzia informatyczne. Wyniki zamieść w pliku tekstowym o nazwie `malware_wynik.txt`. Do oceny oddaj plik tekstowy zawierający wyniki oraz plik zawierający realizację komputerową Twojego rozwiązania.

111.1.

Znajdź te pozycje złośliwego oprogramowania, których celem jest **phishing**, czyli wyłudzanie informacji od użytkownika (w polu: *opis* zawierają łańcuch znaków: *phish* lub *Phish*). Podaj listę zawierającą dla każdej pozycji:

- nazwę kraju, z którego pochodzi komputer udostępniający zagrożenie,
- opis zagrożenia (`opis`),
- pełną ścieżkę dostępu do szkodliwego pliku (`URL`).

111.2.

Znajdź pięć sieci, z których komputery udostępniły najwięcej pozycji złośliwego oprogramowania. Podaj zestawienie zawierające dla każdej takiej sieci: nazwę sieci, nazwę kraju, w którym znajduje się ta sieć, liczbę stron lub dokumentów zawierających złośliwe oprogramowanie oraz liczbę różnych adresów IP, z których to oprogramowanie udostępniono.

111.3.

Dla każdego wpisu na listę *malware* określ domenę, z której udostępniono szkodliwe oprogramowanie. Nazwę domeny stanowią znaki pola URL liczone kolejno od lewej aż do pierwszego wystąpienia znaku „/”, bez tego znaku. W każdym polu `URL` danych znak „/” występuje przynajmniej jeden raz.

a) Podaj liczbę domen, z których pochodzi szkodliwe oprogramowanie.

b) Serwer DNS można skonfigurować tak, aby odpowiadał kilkoma adresami IP dla jednej domeny. Wyszukaj wśród domen te pozycje, którym odpowiada więcej niż jeden adres IP. Podaj nazwy tych domen i odpowiadające im liczby różnych adresów IP.

111.4.

Sporządź w postaci tabeli zestawienie, w którym podasz liczbę zarejestrowanych pozycji złośliwego oprogramowania **w każdym miesiącu roku 2014**, w podziale na poszczególne regiony: apnic, arin, lacnic, ripencc, afrinic. W zestawieniu nie uwzględniaj danych ze stycznia 2015 r.

111.5.

Złośliwe oprogramowanie może być ukryte w plikach graficznych i uaktywniać się podczas wyświetlania obrazu, wykorzystując luki w programach odtwarzających obraz.

Znajdź wszystkie wpisane na listę *malware* pozycje informujące o złośliwym kodzie ukrytym w obrazach zapisanych w formatach: jpg i png (pole *URL* kończy się: „.jpg" lub „.png").

Sporządź w postaci tabeli zestawienie, w którym dla każdego, w którym znajdował się komputer udostępniający szkodliwy kod w plikach graficznych, podasz liczby tych plików, z podziałem na format jpg i png.

Zadanie 112.

Wiązka zadań *Kod EAN*

Kod EAN (*European Article Number*) jest kodem kreskowym powszechnie stosowanym na opakowaniach towarów będących przedmiotem handlu. Czytnik kodu przetwarza go na ciąg cyfr z zakresu 0..9.

Dla uproszczenia zadania (choć rzeczywistość wykracza poza te reguły) przyjęto, że:

- kody wszystkich towarów są 13-cyfrowe,
- pierwsze 3 cyfry kodu są oznaczeniem kraju, w którym zarejestrowany jest producent,
- następne 4 cyfry identyfikują oddział firmy producenta,
- kolejne 5 cyfr jest identyfikatorem towaru,
- ostatnia cyfra jest cyfrą kontrolną.

W pliku `towary.txt` zebrano dane o wybranych słodyczach, napojach, deserach i używkach dostępnych na polskim rynku (nieco zmodyfikowane na potrzeby zadania). Kody EAN towarów są unikatowe. Towary o tej samej nazwie, lecz w opakowaniach o różnych masach, posiadają różne kody EAN. Dla każdego towaru podano: nazwę (`nazwa`), jego rodzaj (`rodzaj`), masę towaru w gramach (`masa`), podatek VAT (`VAT`) oraz kod EAN (`EAN`). Każdy towar opisany jest w osobnym wierszu. Przykład:

```
nazwa          rodzaj       masa      VAT      EAN
TIKI TAKI      czekolada    100       0,23     5909102008735
KOPIEC KRETA   ciasto       410       0,05     5909437089003
NIC NAC'S      przekaska     40       0,08     4018077798818
```

W pliku `kraje.txt` znajdują się nazwy krajów (`kraj`), w których zarejestrowani są producenci towarów, oraz przydzielone tym krajom 3-cyfrowe kody (`kod_kraju`), będące pierwszymi trzema cyframi kodów EAN. Większość krajów posiada jeden kod, ale są kraje, którym przydzielono więcej kodów. Przykład:

```
kod_kraju  kraj
304        Francja
338        Francja
380        Bulgaria
```

W pliku `producenci.txt` znajdują się informacje o oddziałach firm producentów towarów. Dla każdego oddziału podano: jego 4-cyfrowy kod stosowany w kodzie EAN (`kod_oddzialu`), 3-cyfrowy kod kraju, w którym oddział został zarejestrowany (`kod_kraju`), nazwę firmy (`nazwa_firmy`) oraz lokalizację jej oddziału — nazwę miasta, ewentualnie z nazwą dzielnicy (`lokalizacja`). Niektóre firmy mają wiele oddziałów, które mogą być zarejestrowane w różnych krajach. Przykład:

```
kod_kraju   kod_oddziału   lokalizacja          nazwa_firmy
590         9020           Torun                Pacific
590         1480           Jankowice            Kraft-Foods
304         5140           Velizy-Villacoublay  Kraft-Foods
```

W każdym pliku pierwszy wiersz jest wierszem nagłówkowym. Dane w wierszach oddzielone są znakiem tabulacji.

Rozwiąż poniższe zadania, wykorzystując dostępne narzędzia informatyczne. Wyniki zamieść w pliku tekstowym o nazwie `ean_wyniki.txt`. Do oceny oddaj ten plik oraz komputerową realizację rozwiązania.

112.1.

Dla każdego **rodzaju** towarów podaj **liczbę różnych krajów**, z których te towary pochodzą.

112.2.

Dla wszystkich producentów obecnych na naszym rynku, a zarejestrowanych w Czechach, podaj nazwy ich firm oraz lokalizacje ich oddziałów.

112.3.

Utwórz zestawienie, w którym dla każdego rodzaju towaru podasz najmniejszą masę towaru tego rodzaju oraz nazwę towaru tego rodzaju o najmniejszej masie.

112.4.

Wyszukaj towary z rodzaju „baton” w opakowaniach o masie przekraczającej 300g. Dla każdego z nich podaj: nazwę towaru, jego masę oraz nazwę firmy i nazwę kraju, w którym zarejestrowano oddział producenta tego towaru.

112.5.

Każdy towar ma określoną stawkę podatku VAT. Sporządź zestawienie, w którym dla każdego kraju podasz liczbę towarów pochodzących z tego kraju, z podziałem na poszczególne stawki podatku VAT.

3. Komentarze do zadań

Zadanie 3.

Zastanówmy się, co dostaniemy w wyniku wywołania funkcji F(x, n). Niech $n = 3 \bullet a + b$, gdzie a i b są nieujemnymi liczbami całkowitymi oraz $b < 3$. Wówczas $x^n = x^{3 \bullet a+b} = x^a \bullet x^a \bullet x^a \bullet x^b$. Rozważmy teraz funkcję F. Nietrudno spostrzec, że gdy $n < 3$, to wartością F(x,n) jest x^n, natomiast dla $n = 3 \bullet a + b > 2$ mamy

$$F(x,n) = x^b \bullet F(x, 3 \bullet a) = x^b \bullet F(x,a) \bullet F(x,a) \bullet F(x,a) = x^b \bullet x^a \bullet x^a \bullet x^a = x^n.$$

3.1.

Aby obliczyć wynik wywołania F(2, 10), wywołujemy rekurencyjnie F(2, 9), ponieważ $2^{10} = 2^{3 \bullet 3+1}$.

Aby obliczyć wynik wywołania F(2, 9), wywołujemy rekurencyjnie F(2, 3) – $2^9 = 2^{3 \bullet 3} = 2^3 \bullet 2^3 \bullet 2^3$.

Aby obliczyć wynik wywołania F(2, 3), wywołujemy rekurencyjnie F(2, 1) – $2^3 = 2^{3 \bullet 1} = 2^1 \bullet 2^1 \bullet 2^1$.

W wyniku wywołania F(2, 1) dostajemy 2.

Do obliczenia wyniku wywołania F(2, 3) podnosimy do potęgi trzeciej otrzymany wynik wywołania F(2,1) i dostajemy 8.

Do obliczenia wyniku wywołania F(2, 9) podnosimy do potęgi trzeciej otrzymany wynik wywołania F(2,3) i dostajemy 512.

Do obliczenia wyniku wywołania F(2, 10) mnożymy przez 2 otrzymany wynik wywołania rekurencyjnego (2,9), otrzymując na koniec 1024.

3.2.

Wiedząc, że funkcja oblicza wartość x^n, łatwo już wypełnić brakujące pola w tabeli.

3.3.

Wywołanie F(2, 2):

wykonujemy 2*F(2, 1), czyli mamy jedno mnożenie.

Wywołanie F(2, 3):

wywołujemy F(2, 1) (bez mnożeń), a następnie podnosimy uzyskany wynik do potęgi trzeciej, czyli w sumie wykonujemy dwa mnożenia.

Wywołanie F(3, 4):

wykonujemy 3*F(3, 3), czyli jedno mnożenie.

Dla uzyskania F(3, 3) obliczamy najpierw F(3, 1) (bez mnożenia), a następnie uzyskany wynik podnosimy do potęgi trzeciej, co oznacza kolejne dwa mnożenia.

W sumie wykonywane są trzy mnożenia.

Podobnie analizujemy pozostałe przypadki.

3.4.

Po wywołaniu F z argumentem *n* będącym potęgą trójki w kolejnych wywołaniach rekurencyjnych za każdym razem zmniejszana jest trzykrotnie wartość argumentu *n*, inaczej mówiąc, zmniejszany jest wykładnik potęgi trójki o jeden. Do momentu zmniejszenia argumentu do 1 takich zmniejszeń będzie $\log_3 n$. Po każdym z tych wywołań zostaną wykonane dwa mnożenia, zatem łączna liczba mnożeń wyniesie $2 \cdot \log_3 n$.

Zadanie 4.

4.1.

Zamieniamy liczbę z systemu silniowego na system dziesiętny:

$(310)_! = 3 \cdot 3! + 1 \cdot 2! + 0 \cdot 1! = 3 \cdot 6 + 2 + 0 = 20$

$(2011)_! = 2 \cdot 4! + 0 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! = 2 \cdot 24 + 0 + 2 + 1 = 51$

$(54211)_! = 5 \cdot 5! + 4 \cdot 4! + 2 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! = 5 \cdot 120 + 4 \cdot 24 + 2 \cdot 6 + 2 + 1 = 711$

4.2.

Korzystamy ze wskazówki umieszczonej w treści zadania. W systemie silniowym współczynnik x_i, odpowiadający mnożnikowi $i!$, spełnia zależność $0 \leq x_i \leq i$. Oznacza to, że maksymalną cyfrą odpowiadającą mnożnikowi 5! jest cyfra 5, maksymalną cyfrą odpowiadającą mnożnikowi 4! jest 4 itd. Zatem maksymalną liczbą 5-cyfrową zapisaną w silniowym systemie pozycyjnym jest $(54321)_!$.

4.3.

Aby wykonać zamianę liczby 5489 z systemu dziesiętnego na silniowy, posługujemy się schematem opisanym w zadaniu.

Szukamy największej liczby *k*, takiej że $k! \leq 5489$. Ponieważ 7!=5040, to $k = 7$. Obliczamy wynik działania: $x \text{ div } k! = 5489 \text{ div } 5040 = 1$. W ten sposób otrzymujemy pierwszą cyfrę zapisu silniowego. Następnie obliczamy: $x \bmod k! = 5489 \bmod 5040 = 449$. Liczba 449 jest nową wartością *x*. Żeby wyznaczyć kolejną cyfrę zapisu silniowego, zmniejszamy *k* o 1. Analogicznie wypełniamy kolejne wiersze tabeli:

x	*k*	*x* div *k*!	*x* mod *k*!
5489	7	1	449
449	6	0	449
449	5	3	89
89	4	3	17
17	3	2	5
5	2	2	1
1	1	1	0

Liczba dziesiętna 5489 w zapisie silniowym: $(1033221)_!$

4.4.

Analizujemy przedstawiony algorytm. W pierwszym etapie szukamy największej liczby *k*, takiej że $k! \leq x$. Zauważmy, że pętla kończy się, gdy $k! \geq x$. Jeżeli $k! = x$, to znaleźliśmy odpowiednie *k*. Jeżeli $k! > x$, to musimy zmienić wartość zmiennej *silnia* (zmniejszamy wartość

silnia k razy) oraz *k* (zmniejszamy *k* o 1). Stąd wynika, że pierwszą lukę w algorytmie wypełniamy zapisem: **jeżeli** *silnia*> *x*.

W drugiej części algorytmu obliczamy kolejne cyfry zapisu silniowego. Aby uzyskać kolejną cyfrę, musimy obliczyć wynik dzielenia całkowitego liczby *x* przez *k*!, co odpowiada instrukcji *cyfra* ← *x* div *silnia*. Po aktualizacji tworzonego zapisu silniowego (instrukcja *s* ← *s* ◦ tekst (*cyfra*)) aktualizujemy wartość *x*, co odpowiada instrukcji *x* ← *x* mod *silnia* lub

x ← *x* — *cyfra* **silnia*. Ostatnim etapem jest przygotowanie nowej wartości silni odpowiadającej kolejnej cyfrze w zapisie silniowym. Będzie to *silnia* ← *silnia* div *k*.

Zadanie 5.

5.1.

W analizie posłużymy się tabelą, w której prześledzimy wartości istotnych zmiennych w trakcie działania algorytmu. Wiersze tabeli opisujące fragmenty działania algorytmu, w których nie zmienia się wartość zmiennej *j*, zostały pogrupowane przez zastosowanie różnych odcieni tła.

Wartość						Która instrukcja wykonana	
j	*p*	*k*	*i*	*x*	*A*[*i*]	*k* ← *i*	*p* ← *i*
5	5	7	6	5	11	+	
4	4	7	5	10	5		+
4	5	7	6	10	11	+	
3	3	7	5	3	5	+	
3	3	5	4	3	10	+	
2	2	7	4	4	10	+	
2	2	4	3	4	3		+
1	1	7	4	12	10		+
1	4	7	5	12	5		+
1	5	7	6	12	11		+

5.2.

Brakujące fragmenty algorytmu dotyczą zapamiętania, a następnie wstawienia w poprawne miejsce w tablicy aktualnie przetwarzanego elementu ciągu, czyli *A*[*j*]. W pętli zewnętrznej elementy ciągu są przetwarzane od przedostatniego do pierwszego, a w pętli wewnętrznej zmienna sterująca *i* rośnie. Dlatego wyszukiwanie liniowe sprowadza się tutaj do przeglądania już uporządkowanych elementów tablicy *A*[*j*+1], *A*[*j*+2],…, *A*[*n*] w celu znalezienia właściwego miejsca dla elementu *A*[*j*]. Ponieważ w pętli wewnętrznej porównujemy kolejne elementy z *x*, pierwsza brakująca instrukcja służy zapamiętaniu *j*-tego elementu w zmiennej *x*. W pętli wewnętrznej poszukiwanie miejsca do wstawienia *j*-tego elementu rozpoczynamy od pozycji *j*+1, dlatego druga brakująca instrukcja to *i* ← *j* + 1. Ostatnia brakująca instrukcja służy wstawieniu zapamiętanego w zmiennej *x* elementu we właściwe miejsce. Indeks *i* zatrzymuje się w wewnętrznej pętli na elemencie większym od wstawianego elementu *x*, dlatego *x* umieszczamy na pozycji i – 1. (Zauważmy, że elementy *A*[*j*+1],…,*A*[*i* – 1] zostają wskutek wykonania pętli wewnętrznej przesunięte o jedną pozycję w lewo.)

5.3.

W obu analizowanych algorytmach porównania elementów ciągu są wykonywane w pętli wewnętrznej. Liczba powtórzeń tej pętli zależy od wartości danych. Dla ciągu liczb podanych w zadaniu liczba porównań jest różna w obu algorytmach.

Natomiast liczba przesunięć elementów tablicy zależy tylko od wartości danych *i* jest dla ustalonych danych wejściowych taka sama w obu algorytmach.

W porównywanych algorytmach liczba powtórzeń zewnętrznej pętli zawsze jest taka sama, bez względu na wartości danych wejściowych. W obu przypadkach pętla wykonywana jest dla $j = n - 1, n - 2, \ldots , 1$. Instrukcja podstawienia dla zmiennej *x* jest wykonywana tylko w pętli zewnętrznej, więc liczba jej wykonań jest (dla ustalonych danych) zawsze taka sama w obu algorytmach.

Zadanie 6.

6.1.

Niewielki rozmiar analizowanych danych pozwala na prześledzenie działania algorytmu i ustalenie końcowej zawartości tablicy *A* bez pełnego zrozumienia istoty działania algorytmu. Istotne jest tutaj, że w kolejnych obrotach pętli „**dopóki** *s*<*n* **wykonuj**” porównujemy ze sobą (i ewentualnie zamieniamy) elementy tablicy *A* w odległościach 1, 2, 4 itd. Śledząc działanie programu, uzyskamy następujące rozwiązanie:

k	Początkowa zawartość tablicy $A[1...2^k]$	Końcowa zawartość tablicy $A[1...2^k]$
2	[4, 3, 1, 2]	[1, 4, 3, 2]
2	[2, 3, 4, 1]	[1, 3, 2, 4]
3	[1, 2, 3, 4, 5, 6, 7, 8]	[1, 2, 3, 4, 5, 6, 7, 8]
3	[8, 7, 6, 5, 4, 3, 2, 1]	[1, 8, 7, 6, 5, 4, 3, 2]
3	[4, 5, 6, 1, 8, 3, 2, 4]	[1, 5, 4, 6, 2, 8, 3, 4]

Postarajmy się jednak zrobić trochę więcej: ustalić ideę podanego algorytmu; wiedzę tę wykorzystamy również przy rozwiązywaniu zadań 2 i 3.

Najpierw zauważmy, że po zakończeniu pętli „dla *i*=1,2,…,*k* wykonuj” zmienna *n* ma wartość 2^k.

W pierwszym obrocie pętli „**dopóki** *s*<*n* **wykonuj**” porównujemy *A*[1] z *A*[2], *A*[3] z *A*[4] itd. Przy porównaniu *A*[*i*] z *A*[*i*+1] wykonujemy zamianę, gdy *A*[*i*]>*A*[*i*+1]. A zatem po pierwszym obrocie pętli w *A*[1] znajduje się minimum z *A*[1] i *A*[2], w *A*[3] znajduje się minimum z *A*[3] i *A*[4], w *A*[5] znajduje się minimum z *A*[5] i *A*[6] itd.

Po każdym obrocie pętli **„dopóki** *s*<*n* **wykonuj”** wartość *s* zwiększa się dwukrotnie. Tak więc przy drugim obrocie tej pętli wartość *s* będzie równa 2. Wówczas porównamy *A*[1] z *A*[3] i, w przypadku gdy *A*[1] >*A*[3], dokonana zostanie zamiana tych elementów. W efekcie w *A*[1] znajdzie się minimum z czterech pierwszych elementów tablicy *A*. Następnie porównamy *A*[5] z *A*[7], skutkiem czego w *A*[5] znajdzie się minimum z *A*[5], *A*[6], *A*[7], *A*[8], a więc minimum z kolejnych czterech elementów tablicy *A*. Przeprowadzając analizę tak dalej, dochodzimy do wniosku, że takie lokalne minima z grup kolejnych czwórek elementów tablicy *A* znajdą się w pierwszych elementach tych czwórek.

Co się stanie w kolejnych obrotach pętli **„dopóki** *s*<*n* **wykonuj”**? Łatwo sprawdzić, że wielkość grup będzie podwajana. Tak więc będziemy mieć do czynienia z grupami kolejnych 8 elementów, następnie — grupami kolejnych 16 elementów itd…. Minima z tych grup będą

zapamiętywane w pierwszych elementach grup. Zatem po kolejnych obrotach pętli w $A[1]$ będą zapamiętane kolejno minima z pierwszych 2^i elementów tablicy A. W szczególności po k-tym obrocie, a więc na zakończenie algorytmu, w $A[1]$ znajdzie się minimum ze wszystkich elementów tablicy A.

6.2.

Dwa pierwsze zdania są fałszywe, co można zauważyć na przykładach z zadania pierwszego. Stwierdzenie „Po zakończeniu działania algorytmu 1 komórka $A[1]$ zawiera najmniejszą z liczb $A[1],\ldots,A[2^k]$" zachodzi dla przykładów z zadania 1, a jego prawdziwość dla każdych danych wejściowych uzasadniliśmy, omawiając powyżej działanie algorytmu 1.

6.3.

Przypomnijmy, że po wykonaniu pierwszej pętli n jest równe 2^k. Zauważmy też, że w pierwszym obrocie pętli „**dopóki** $s<n$ **wykonuj**" instrukcja w wierszu (*) wykona się $n/2$ razy, w drugim obrocie pętli wykona się $n/4$ razy itd., w ostatnim obrocie wykona się jeden raz (dla $s=2^{k-1}=n/2$). Zatem instrukcja w wierszu (*) wykona się

$$1 + 2 + 4 + \ldots + n/4 + n/2 = n - 1$$

razy (zauważ, że liczby 1, 2, 4, … tworzą ciąg geometryczny). Oznacza to, że pierwsze z podanych czterech stwierdzeń jest prawdziwe, a drugie fałszywe ($2n>n-1>n/2$ dla $n>2$).

Ponieważ wartość s jest w trakcie działania algorytmu większa od zera, a instrukcja zamiany z wiersza (**) wykonuje się tylko, gdy $A[j]>A[j+s]$, instrukcja (**) nie zostanie wykonana ani razu dla ciągów uporządkowanych, czyli spełniających $A[1]<A[2]<\ldots<A[n]$. Prawdziwe jest zatem stwierdzenie: „Możliwe jest dobranie takiej początkowej zawartości $A[1..2^k]$, że instrukcja zamiany z wiersza (**) nie wykona się ani razu".

W celu ustalenia prawdziwości ostatniego z podanych zdań zauważmy, że liczba wykonań instrukcji z wiersza (**) jest nie większa od liczby wykonań instrukcji z wiersza (*). Ustaliliśmy wcześniej, że liczba wykonań (*) jest równa $n-1$, zatem ostatnie z podanych w zadaniu zdań jest fałszywe: liczba wykonań (**) jest nie większa $n-1<2n^2$ dla $n>0$.

6.4.

Zwróćmy uwagę na kluczowe różnice między pętlą wewnętrzną „**dopóki** $j<n$ **wykonuj**" algorytmu 1 i analogiczną pętlą algorytmu 2:

- w wierszu (*) algorytmu 1 porównujemy elementy, których indeksy różnią się o s, natomiast w analogicznym wierszu algorytmu 2 porównujemy zawsze elementy sąsiednie;
- wartość j jest zwiększana o s po każdym obrocie pętli „**dopóki** $s<n$ **wykonuj"** w algorytmie 1, natomiast w algorytmie 2 jest zwiększana o 1.

Kolejna kluczowa zmiana (w porównaniu z algorytmem 1) dotyczy modyfikacji zmiennej s na końcu każdego obrotu zewnętrznej pętli „**dopóki** $s<n$ **wykonuj**": „$s \leftarrow 2 \cdot s$" w algorytmie 1 została zmieniona na „$s \leftarrow s+1$" w algorytmie 2.

Okazuje się, że po opisanych powyżej zmianach algorytmu 1 (których efektem jest algorytm 2) uzyskujemy metodę sortowania bąbelkowego:

- w pierwszym obrocie pętli „**dopóki** $s<n$ **wykonuj**" największy element spośród $A[1],\ldots,A[n]$ zostanie przeniesiony na pozycję $A[n]$ (po „napotkaniu" największego

elementu będziemy go zamieniać z kolejnymi elementami aż do umieszczenia go na pozycji $A[n]$);

- w drugim obrocie pętli „**dopóki** $s<n$ **wykonuj**” największy element spośród $A[1],\ldots,A[n-1]$ zostanie przeniesiony na pozycję $A[n-1]$ (po „napotkaniu” tego elementu będziemy go zamieniać z kolejnymi elementami aż do umieszczenia go na pozycji $A[n-1]$);
- ogólnie po i obrotach pętli „**dopóki** $s<n$ **wykonuj**” dla $i \in [1,n-1]$ spełniony jest warunek: i największych elementów spośród $A[1],\ldots,A[n]$ jest umieszczonych w kolejności niemalejącej w komórkach $A[n-i+1], A[n-i+2],\ldots, A[n]$.

Zauważmy, że pętla zewnętrzna „**dopóki** $s<n$ **wykonuj**” wykona $n-1$ obrotów. Z powyższych własności wynika zatem, że po zakończeniu działania algorytmu $n-1$ największych elementów z $A[1],\ldots,A[n]$ znajdzie się w komórkach $A[2], A[3],\ldots, A[n]$ w kolejności

$$A[2] \leq A[3] \leq \ldots \leq A[n].$$

Z powyższych obserwacji również wynika, że najmniejszy element z $A[1],\ldots,A[n]$ znajdzie się po zakończeniu działania algorytmu w $A[1]$, a zatem efektem działania algorytmu 2 jest uporządkowanie niemalejące $A[1],..,A[n]$. Powyższe obserwacje prowadzą do następującego opisu wyniku działania algorytmu 2:

Po zakończeniu działania algorytmu 2 element $A[i]$ jest *nie większy (mniejszy bądź równy)*

niż element $A[i+1]$ dla każdego i większego od *0*

oraz mniejszego od *n*

Pętla „**dopóki** $s<n$ **wykonuj**” wykonuje $n-1$ obrotów, w każdym z nich pętla wewnętrzna (oraz instrukcja w wierszu (*)) wykona się $n-1$ razy. Zatem liczba wykonań wiersza (*) wynosi $(n-1)^2$, co pozwala następująco uzupełnić luki w ostatnim zdaniu:

Wiersz (*) **algorytmu 2** wykonywany będzie w przebiegu algorytmu

- *więcej* niż n razy
- *mniej* niż n^2 razy

Zadanie 7.

7.1.

W zadaniu należy wybrać te dane spośród wymienionych w treści, które są zgodne z podaną specyfikacją rozważanego algorytmu. W specyfikacji tej zakłada się, że dana tablica T zawiera n liczb rzeczywistych, gdzie $n = 2^m$, a więc n jest liczbą będącą potęgą dwójki. Tego założenia nie spełnia jedynie trzecia tablica podana w poleceniu.

Aby rozwiązać drugą część zadania, należy przeanalizować działanie algorytmu z treści zadania. Można zauważyć, że algorytm ten wykorzystuje technikę *dziel i zwyciężaj* w celu uporządkowania tablicy T od elementu najmniejszego do największego. Mianowicie algorytm dzieli tablicę T na dwie części. Następnie rekurencyjnie wywołuje algorytm *uporządkuj*, aby je posortować (w kolejności od najmniejszego elementu). Otrzymane dwie uporządkowane części są następnie scalane, za pomocą procedury *scal*, w jedną całość, która jest ostatecznym wynikiem algorytmu. Algorytm ten to nic innego niż algorytm *sortowania przez scalanie* (ang. *merge-sort*). W celu rozwiązania zadania wystarczy zatem uporządkować wybrane tablice, które spełniają specyfikację.

7.2.

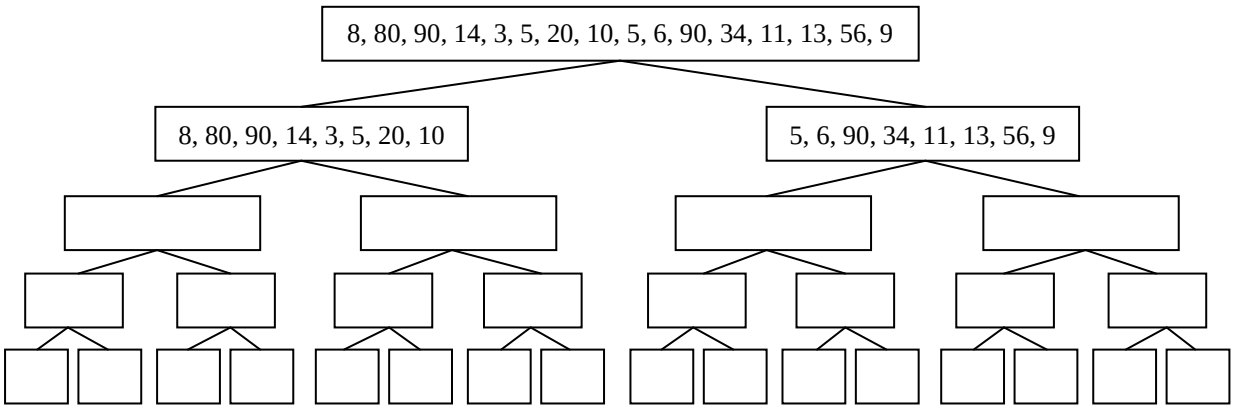
W zadaniu należy uzupełnić zasugerowane drzewo wywołań rekurencyjnych wykonywanych przez algorytm *uporządkuj*. Zgodnie z analizą algorytmu, jaką przedstawiono w opisie poprzedniego zadania, algorytm ten zawsze dzieli daną tablicę $T[1\,..\,n]$ na dwie części:

$$T[1\,..\,k], \qquad T[k+1\,..\,n],$$

gdzie k jest równe *n*/2. Ponieważ w specyfikacji algorytmu powiedziane jest, że $n = 2^m$, więc otrzymane części są równoliczne i zawierają dokładnie po $k = 2^{m-1}$ elementów. To pozwala na rekurencyjne wywołanie algorytmu *uporządkuj* dla obu części, a w konsekwencji uzasadnia poprawność algorytmu. Aby rozwiązać zadanie, wystarczy zatem dzielić daną tablicę na połowy. Z algorytmu można łatwo odczytać, że dla tablic jednoelementowych ($n = 1$) podział na części już nie występuje. Stąd w rozwiązaniu należy drzewo zakończyć na poziomie, w którym we wszystkich wierzchołkach znajdują się tylko pojedyncze liczby.

7.3.

W zadaniu należy podać łączną liczbę kosztownych operacji, jaka zostanie wykonana przez funkcję *uporządkuj* dla tablicy 16-elementowej. Zakładamy, że kosztowne operacje ukryte są w funkcji *scal*, która wykonuje *2k*-1 operacji dla dwóch tablic o długości k. W rozwiązaniu zadania bardzo pomocne jest narysowanie drzewa wywołań rekurencyjnych, które pozwala zauważyć, gdzie są wykonywane kosztowne operacje i ile ich jest. W tym celu skorzystamy z drzewa z poprzedniego zadania.



Pierwsza obserwacja jest taka, że szukana liczba operacji wykonanych przez algorytm *uporządkuj* jest sumą liczby kosztownych operacji wykonywanych przez funkcje *scal*, realizowanych na różnych poziomach tego drzewa.

Aby uzyskać wynik — tablica 16-elementowa na szczycie drzewa(pierwszy poziom), należy scalić dwa wyniki, tj. dwie tablice (na drugim poziomie) o długości 8. Oznacza to, że realizowane jest scalanie, które wykona $2 \cdot 8 - 1 = 15$ kosztownych operacji. Dalej na kolejnym poziomie realizowane są dwie operacje scalania tablic o długości 4, co łącznie daje $2 \cdot (2 \cdot 4 - 1) = 14$ kosztownych operacji. Rozumując dalej w ten sam sposób, dochodzimy do wniosku, że na kolejnym poziomie wykonywanych jest łącznie $4 \cdot (2 \cdot 2 - 1) = 12$ kosztownych operacji, a na końcu $8 \cdot (2 \cdot 1 - 1) = 8$ kosztownych operacji. Ostatecznie uzyskujemy łącznie $15 + 14 + 12 + 8 = 49$ wszystkich kosztownych operacji wykonywanych przez algorytm *uporządkuj(*T*)* dla tablicy 16-elementowej.

Zadanie 8.

8.1.

Dla pierwszego wiersza w tabeli dane wejściowe to A=[3, 5, 12, 17] i B=[8, 10, 13, 14] oraz x=21.

Przeanalizujmy kolejne kroki algorytmu. Na początku zmienne i oraz j przyjmują wartości: i=1, j=4, liczba elementów w każdej tablicy wynosi n=4. Policzmy, ile razy sprawdzony zostanie warunek w wierszu (*).

Lp.	i	j	Czy $i \leq n$ oraz $j > 0$	$A[i]+B[j]$	(*) Czy $A[i] + B[j] = x$	Czy $A[i]+B[j])<x$
1	1	4	tak	17	nie	tak
2	2	4	tak	19	nie	tak
3	3	4	tak	26	nie	nie
4	3	3	tak	25	nie	nie
5	3	2	tak	22	nie	nie
6	3	1	tak	20	nie	tak
7	4	1	tak	25	nie	nie
	4	0	nie	koniec działania algorytmu, wynik: *FAŁSZ*		

Dla danych wejściowych: A=[4, 6, 8, 10] i B=[5, 7, 9, 11] oraz x=13:

Lp.	i	j	Czy $i \leq n$ oraz $j > 0$	$A[i]+B[j]$	(*) Czy $A[i] + B[j] = x$	Czy $A[i]+B[j])<x$
1	1	4	tak	15	nie	nie
2	1	3	tak	13	koniec działania algorytmu, wynik: *PRAWDA*	

8.2.

Analiza działania algorytmu dla danych wejściowych z zadania 8.1 naprowadza na ogólny wynik obliczany przez algorytm. Teraz należy tylko sformułować prawidłowo i precyzyjnie specyfikację. Dla dwóch tablic A i B posortowanych rosnąco oraz liczby x algorytm sprawdza, czy istnieje suma złożona z dowolnej liczby z tablicy A i dowolnej liczby z tablicy B, która jest równa liczbie x. Algorytm pobiera elementy z tablicy A, począwszy od pierwszego, a z tablicy B, zaczynając od elementu ostatniego. Najpierw sprawdza, czy suma obu elementów jest równa x, jeśli tak, to kończy działanie z wynikiem PRAWDA, w przeciwnym razie sprawdza, czy suma jest mniejsza od x. Jeżeli tak, to bierze do sumy kolejny (większy od poprzedniego) element z tablicy A (indeks i zwiększa się o 1). Jeżeli zaś suma jest większa od x, bierze mniejszy element z tablicy B (indeks j zmniejsza się o 1). Dzięki takim przesunięciom i oraz j nie gubi dobrej sumy.

Przyspieszenie takiego sprawdzania jest możliwe dzięki odpowiedniej kolejności przeglądania tablic (zmienne i, j) oraz pominięciu zbędnych sum, o których wiadomo, że są albo za duże, albo za małe w stosunku do x.

8.3.

Skoro algorytm ma zwrócić wartość *PRAWDA*, wybierzmy dwa elementy takie, które należą do dwóch różnych tablic, a ich suma wyniesie 20, np. 10 i 10. Najlepiej jeśli sumy każdych dwóch pozostałych elementów będą różne od 20. Tablice powinny być uporządkowane rosnąco. Chcemy, aby algorytm wykonał dokładnie sześć porównań. Jedna z możliwych strate-

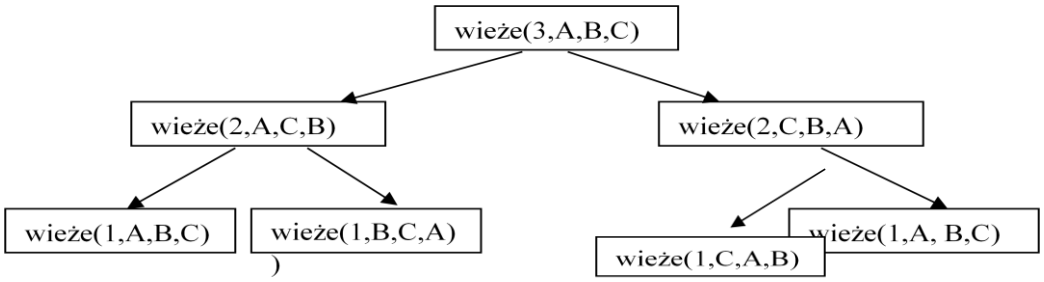
gii polega na zmuszeniu algorytmu do przejścia po wszystkich elementach tablicy *A* (5 porównań), a następnie porównania piątego elementu z tablicy *A* z czwartym elementem z tablicy *B*, po którym algorytm zwróci wynik *PRAWDA* i zakończy działanie. Niech para (*i, j*) oznacza indeks *i*-tego elementu tablicy *A* i *j*-tego elementu tablicy *B*. Można dobrać elementy tablic tak, aby sumy w parach (1,5), (2,5), (3,5), (4,5) były mniejsze od 20, suma pary (5,5) była większa od 20 i dopiero para (5,4) dała sumę równą 20, co spowoduje podanie *PRAWDA* i zakończenie wykonywania algorytmu.

Może to być na przykład *A*=[2, 4, 6, 8, 10] i *B*=[5, 7, 9, 10,11] oraz *x*=20.

Zadanie 9.

9.1.

Działanie *wieże*(3, A, B, C) opiszemy przy pomocy tak zwanego drzewa wywołań rekurencyjnych. W drzewie tym zapisywać będziemy wszystkie wywołania funkcji *wieże* będące wynikiem uruchomienia *wieże*(3, A, B, C). Wywołania *wieże*(*i, x, y, z*) i *wieże*(*i', x', y', z'*) połączymy skierowaną krawędzią, jeśli w treści *wieże*(*i, x, y, z*) następuje wywołanie *wieże*(*i', x', y', z'*). Na przykład prowadzimy krawędź od *wieże*(3, A, B, C) do *wieże*(2, C, B, A), gdyż wykonanie *wieże*(3, A, B, C) powoduje wywołanie *wieże*(2, C, B, A) (jest to wywołanie z ostatniego wiersza: wieże(n – 1, z, y, x)). Poniżej prezentujemy pełne drzewo wywołań rekurencyjnych dla *wieże*(3, A, B, C).



W efekcie uzyskujemy następujące rozwiązanie zadania:

n	x	y	z
3	A	B	C
2	A	C	B
1	A	B	C
1	B	C	A
2	C	B	A
1	C	A	B
1	A	B	C

9.2.

Zadanie możemy rozwiązać, śledząc działanie funkcji „krok po kroku", pamiętając przy tym o całej hierarchii wywołań rekurencyjnych. Przedstawimy nieco inne rozwiązanie, korzystające z drzewa wywołań rekurencyjnych, które utworzyliśmy, rozwiązując zadanie 1. Tworząc drzewo, stosowaliśmy zasadę, że wywołania funkcji *wieże* wykonywane w trakcie działania *wieże*(*i, x, y, z*) są wypisywane pod wywołaniem *wieże*(*i, x, y, z*), w kolejności od lewej do prawej, zgodnie kolejnością ich wykonań w trakcie działania *wieże*(*i, x, y, z*). Na przykład wywołanie *wieże*(2, A, C, B) jest wykonywane przed *wieże*(2, C, B, A) w trakcie *wieże*(3, A, B, C), co odpowiada ich kolejności „od lewej do prawej" w naszym drzewie.

Z powyższej obserwacji i treści funkcji wynika, że dla każdego węzła naszego drzewa najpierw wypisywane będą ruchy z jego „lewego poddrzewa", potem ruch „**wypisz** x ⇒ y", a następnie ruchy z jego „prawego poddrzewa". Na przykład dla *wieże*(2, C, B, A) mamy ciąg ruchów:

C⇒A; C⇒B; A⇒B,

gdzie podkreślony ruch C⇒B jest wynikiem instrukcji „**wypisz** x ⇒ y" wykonanej w *wieże*(2, C, B, A), a pozostałe ruchy są wynikiem wywołań rekurencyjnych.

Stosując powyższą obserwację, widzimy, że ciąg ruchów wygenerowany przez *wieże*(3, A, B, C) to:

A⇒B; A⇒C; B⇒C; A⇒B; C⇒A; C⇒B; A⇒B,

gdzie podkreślony ruch A⇒B jest wynikiem instrukcji „**wypisz** x ⇒ y" wykonanej w *wieże*(3, A, B, C), ruchy na lewo od podkreślonego A⇒B to efekt działania *wieże*(2, A, C, B), a ruchy na prawo od podkreślonego A⇒B to efekt działania *wieże*(2, C, B, A).

9.3.

Obserwację podaną w treści zadania możemy wyrazić zależnością $H(n)=2\cdot H(n-1)+1$ dla $n>1$. Z treści funkcji wynika też, że $H(1)=1$ (przy jednym krążku wykonujemy jeden ruch). Stosując te zależności, uzyskujemy poniższe wartości:

n	H(n)
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

Powyższa tabela pozwala nam odkryć zależność $H(n) = 2^n - 1$. Formalnie zależność tę możemy wykazać na przykład za pomocą zasady indukcji matematycznej, z którą czytelnik być może zetknął się na lekcjach matematyki lub którą będzie miał sposobność poznać na studiach wyższych.

9.4.

Algorytm przedstawiony w treści zadania wygeneruje dokładnie ten sam ciąg ruchów, jaki uzyskalibyśmy, stosując algorytm rekurencyjny podany wcześniej (dla odpowiednio wybranego pręta startowego i pręta docelowego). Wiedząc o tym, moglibyśmy posiłkować się rozwiązaniem zadania 2. Równość ciągów ruchów generowanych przez oba algorytmy nie jest jednak oczywista, szczególnie przy pierwszym zetknięciu z problemem wież z Hanoi i podanymi algorytmami. Dlatego poniżej prezentujemy rozwiązanie, w którym nie korzystamy z tych własności.

Mamy prześledzić działanie algorytmu dla parzystego n (n=4), przy którym nieparzyste ruchy (pierwszy, trzeci, piąty,…) będą polegały na przenoszeniu krążka nr 1 o jedną pozycję na le-

wo (ruchy A⇒C, B⇒A lub C⇒B), natomiast pozostałe ruchy nie zmienią pozycji krążka numer 1. Z obserwacji tych wynika, że uzyskany ciąg ruchów można opisać jako:

A⇒C, ??, C⇒B, ??, B⇒A, ??, A⇒C, ??, C⇒B, ??, B⇒A, ??, …,

gdzie ?? odpowiada jednemu ruchowi, w którym nie zmieniamy pozycji krążka numer 1. Po przeniesieniu krążka numer 1 ruchem A⇒C krążek numer 1 znajdzie się na pręcie C, więc w następnym ruchu nie uczestniczy pręt C. Analogicznie po przeniesieniu krążka numer 1 ruchem C⇒B krążek numer 1 znajdzie się na pręcie B, więc w następnym ruchu nie uczestniczy pręt B. Podobna obserwacja zachodzi dla przeniesienia krążka 1 ruchem B⇒A. Pozwala to nieco uszczegółowić ciąg ruchów:

A⇒C, A ? B, C⇒B, A ? C, B⇒A, B ? C, A⇒C, A ? B, C⇒B, A ? C, B⇒A, B ? C, …,

gdzie x ? y oznacza ruch x⇒y lub y⇒x, w zależności od tego, który ruch w danym momencie jest dozwolony (pamiętajmy, że nie możemy kłaść krążka większego na mniejszy).

Stosując schemat oparty na powyższej regule, stwierdzimy, że wszystkie krążki zostaną przeniesione z pręta A na pręt B po następujących piętnastu ruchach:

A⇒C, A⇒B, C⇒B, A⇒C, B⇒A, B⇒C, A⇒C, A⇒B, C⇒B, C⇒A, B⇒A, C⇒B, A⇒C, A⇒B, C⇒B.

Zadanie 10.

10. 1.

Z treści zadania natychmiast otrzymujemy, że $n = 16$, a więc $m = 4$. Do rozwiązania zadania wystarczy zauważyć, że tablica *Z*[0 .. m-1] ma przechowywać potęgi x, x^2, x^4, x^8, gdzie $x = 2$.

10.2.

Rozwiązanie zadania wymaga przeprowadzenia krótkiej analizy algorytmu zapisanego w funkcji *F*. Można zauważyć, że funkcja *F* dokonuje podziału danej tablicy na połowy, o ile $n > 1$, a następnie wywołuje się rekurencyjnie dla obu części. Dodatkowo na końcu wykonywane jest jeszcze jedno mnożenie. Stąd natychmiast otrzymujemy wzór rekurencyjny na liczbę $f(n)$ operacji mnożenia wykonywanych przez funkcję *F* dla tablicy *n*-elementowej:

$$f(n) = 2 \cdot f(n/2) + 1.$$

Dysponując powyższym wzorem, możemy łatwo znaleźć wyniki, które należy wpisać do tabelki. Ponadto warto zauważyć, że rozwiązaniem powyższej rekurencji jest funkcja

$$f(n) = n - 1 \quad (n = 2^m).$$

10.3.

Najpierw wyznaczymy $g(n)$ — liczbę wywołań rekurencyjnych, jaka zostanie wykonana przez funkcję *F* przy obliczaniu wartości wielomianu $P(x)$ stopnia $n - 1$, gdzie n jest potęgą dwójki. Po przeprowadzeniu krótkiej analizy algorytmu możemy zauważyć, że jeśli $n = 1$, to funkcja *F* nie wykona żadnych dodatkowych wywołań rekurencyjnych, a więc

$$g(1) = 1.$$

Natomiast dla $n > 1$ można zauważyć, że funkcja wykonuje dodatkowo dwa wywołania rekurencyjne dla tablic o połowę mniejszych, co oznacza, że

$$g(n) = 2g(n/2) + 1.$$

Uczeń może tutaj zauważyć podobieństwo do poprzedniego zadania. Jedyną różnicą jest to, że w powyższej rekurencji mamy warunek początkowy $g(1) = 1$. W zadaniu jednak należy znaleźć wzór ogólny dla $g(n)$. Aby go wyprowadzić, warto wyznaczyć kilka początkowych wartości funkcji $g(n)$, mianowicie mamy:

$$\begin{aligned} g(1) &= 1 \\ g(2) &= 2 \cdot g(1) + 1 = 3, \\ g(4) &= 2 \cdot g(2) + 1 = 7, \\ g(8) &= 2 \cdot g(4) + 1 = 15, \\ g(16) &= 2 \cdot g(8) + 1 = 31. \end{aligned}$$

Stąd można wywnioskować, że $g(n) = 2n - 1$. Można to udowodnić za pomocą indukcji matematycznej.

10.4.

W zadaniu rozważane są wielomiany stopnia $n - 1$, gdzie tym razem n jest potęgą trójki. Do rozwiązania problemu obliczania wartości wielomianu stosuje się również technikę *dziel i zwyciężaj*. Zaproponowana funkcja *G* dokonuje podziału tablicy współczynników na trzy równe części, a następnie, po rozwiązaniu trzech podproblemów obliczania $A(x), B(x), C(x)$, funkcja oblicza swój wynik, wykonując dodatkowo 2 mnożenia. W zadaniu należy obliczyć łączną liczbę mnożeń (oznaczmy ją przez $h(n)$) wykonywanych przez funkcję *G* dla problemów o pewnych rozmiarach. Przeprowadzając analizę podobną jak w zadaniu 2, otrzymujemy następujący związek rekurencyjny:

$$h(n) = 3h(n/3) + 2$$

z warunkiem początkowym $h(1) = 0$. Powyższy wzór właściwie wystarczy do rozwiązania całego zadania. Warto jednak dla dodatkowego ćwiczenia wykazać, że rozwiązaniem powyższej rekurencji jest funkcja

$$h(n) = n - 1.$$

Porównując to z wynikiem zadania 2, widzimy, że funkcje *F* i *G* wykonują dokładnie tę samą liczbę mnożeń.

Zadanie 11.

11.1.

Rozwiązanie zadania ułatwia fakt, że coraz bardziej złożone wyrażenia pojawiające się w kolejnych wierszach tabeli składają się z wyrażeń, które występują w wierszach wcześniejszych. W szczególności wyrażenie z ostatniego wiersza

((4 + 3) ∗ 2) – (5 ∗ (7 – 6))

można przedstawić jako $W_1 – W_2$ dla wyrażeń

W_1 = ((4 + 3) ∗ 2),

W_2 = (5 ∗ (7 – 6)),

które znajdują się odpowiednio w wierszu trzecim i czwartym tabeli. A zatem

ONP**(**((4 + 3) ∗ 2) – (5 ∗ (7 – 6))**)** = ONP(W_1) ONP(W_2)-

dla powyższych W_1 i W_2. Wypełniając kolejne wiersze tabeli, możemy zatem korzystać z wartości zapisanych w wierszach wcześniejszych. Poniżej prezentujemy końcowe rozwiązanie:

$W = W_1$ op W_2	W_1	W_2	op	ONP(W)
4 + 3	4	3	+	4 3 +
(4 + 3) ∗ 2	**(4+3)**	**2**	**∗**	**4 3 + 2 ∗**
5 ∗ (7 – 6)	**5**	**(7 – 6)**	**∗**	**5 7 6 – ∗**
((4 + 3) ∗ 2) – (5 ∗ (7 – 6))	**((4 + 3) ∗ 2)**	**(5 ∗ (7 – 6))**	**–**	**4 3 + 2 ∗ 5 7 6 – ∗ –**

11.2.

Zadanie to można rozwiązać, śledząc działanie algorytmu na podanych danych.

W algorytmie korzystamy ze stosu reprezentowanego przez tablicę T. Wyrażenie wartościujemy, czytając sekwencyjnie („od lewej do prawej”) jego kolejne elementy i wykonując następujące akcje:

- jeśli przeczytany element jest liczbą, umieszczamy go na szczycie stosu;
- jeśli przeczytany element jest operatorem *op* (należącym do zbioru {+, –, ∗}), zdejmujemy kolejno dwie liczby ze szczytu stosu (nazwijmy je odpowiednio a i b) oraz umieszczamy na szczycie stosu wartość b op a.

W naszym rozwiązaniu stos jest reprezentowany przez tablicę T, zawartość stosu to $T[1]$ $T[2]\ldots T[k-1]$, gdzie $T[1]$ jest elementem znajdującym się na dnie stosu, a $T[k-1]$ jest elementem ze szczytu stosu. Dlatego też wartość zmiennej k zmienia się odpowiednio, gdy zdejmujemy elementy ze stosu lub umieszczamy elementy na jego szczycie.

Wykorzystując powyższy opis, podajemy poniżej rozwiązanie dla wszystkich wierszy, dodając dla czytelności kolumnę z kolejnymi wczytywanymi znakami wyrażenia w postaci ONP.

i	$X[i]$	Wartość zmiennej k po i-tym przebiegu pętli	Zawartość tablicy $T[1..k-1]$ po i-tym przebiegu pętli
1	9	2	9
2	7	3	9, 7
3	+	2	16
4	3	3	16, 3
5	□□	2	48
6	5	3	48,5
7	4	4	48, 5, 4
8	–	3	48, 1
9	2	4	48, 1, 2
10	□□	3	48,2
11	–	2	46

11.3.

Podobnie jak w zadaniu 2 rozwiązanie można uzyskać, śledząc działanie algorytmu na podanych danych wejściowych. Pozostawiając takie rozwiązanie czytelnikowi, postaramy się opisać kryterium rozstrzygające, czy dany tekst X jest poprawnym wyrażeniem ONP, na którym opiera się podany algorytm.

Działanie algorytmu można streścić w następujący sposób:

- czytając kolejne elementy *X*, liczymy różnicę między liczbą przeczytanych dotychczas liczb a liczbą przeczytanych dotychczas operatorów; różnicę tę zachowujemy w zmiennej licznik;
- jeśli po przeczytaniu kolejnego znaku różnica między liczbą przeczytanych dotychczas liczb a liczbą przeczytanych dotychczas operatorów (zmienna *licznik*) jest mniejsza niż 1, uznajemy, że *X* nie jest poprawnym wyrażeniem w postaci ONP;
- jeśli po przeczytaniu ostatniego znaku różnica między liczbą przeczytanych liczb a liczbą przeczytanych operatorów jest równa 1, uznajemy, że *X* jest poprawnym wyrażeniem w postaci ONP; w przeciwnym razie uznajemy, że *X* nie jest poprawnym wyrażeniem w postaci ONP.

Stosując powyższe obserwacje, łatwo uzupełnić tabelę:

Napis	**Wartość zmiennej *licznik* po zakończeniu**	**Czy to poprawne wyrażenie w ONP?**
1 2 + ∗	1	NIE
1 2 + 3 4 – 5 ∗ 7 8 + 9	4	NIE
1 2 3 4 5 + + + +	1	TAK
1 2 3 4 5 + + + + + +	1	NIE
1 2 3 4 5 + + + + +	1	NIE
1 2 + 2 3 – 3 4 ∗ 4 5 + – – –	1	TAK
1 2 + 2 3 – 3 4 ∗ 4 5 + – – – – –	1	NIE
1 2 + 3 4 – 5 ∗ 7 8 + 9 + + +	1	TAK

11.4.

Zauważmy, że $X = (X_1\ op_9\ 10)$, gdzie

X_1= ((((((((1 op_1 2) op_2 3) op_3 4) op_4 5) op_5 6) op_6 7) op_7 8) op_8 9).

Zatem ONP(X) = ONP(X_1) 10 op_9. Regularność wyrażenia X pozwala na wyciągnięcie wniosku, że ONP(X) uzyskamy, wypisując kolejno od prawej strony 10 op_9, 9 op_8, 8 op_7 itd.:

ONP(X): 1 2 op_1 3 op_2 4 op_3 5 op_4 6 op_5 7 op_6 8 op_7 9 op_8 10 op_9.

Analogicznie Y=(1 op_1 Y_1), gdzie

Y_1= (2 op_2 (3 op_3 (4 op_4 (5 op_5 (6 op_6 (7 op_7 (8 op_8 (9 op_9 10)))))))).

Zatem ONP(Y) = 1 ONP(Y_1) op_1. Regularność wyrażenia Y pozwala na wyciągnięcie wniosku, że ONP(Y) uzyskamy, wypisując kolejno od lewej strony 1, 2, 3 itd. oraz od prawej strony op_1, op_2, op_3 itd.:

ONP(Y): 1 2 3 4 5 6 7 8 9 10 op_9 op_8 op_7 op_6 op_5 op_4 op_3 op_2 op_1.

Zadanie 12.

Zaprezentowany szyfr Beauforta jest prostym szyfrem wieloalfabetycznym, w którym każda z liter tekstu jest szyfrowana według jednego z 26 podobnych szyfrów przestawieniowych. Szyfry te różnią się przesunięciem szyfrowanej litery o wartość *k*. Początkowa wartość *k* określa, który z 26 szyfrów został wyborany do zastosowania w przypadku pierwszej litery. Dalsze postępowanie (zmiany parametru *k* dla kolejnych liter / wybór szyfru dla kolejnej litery) jest zdefiowane w algorytmie i zależne jedynie od wartości *k* dla pierwszej litery.

W podanym algorytmie szyfrowanie każdej litery realizowane jest kolejno poprzez: odjęcie kodu litery od 90 (inaczej mówiąc, obliczenie kolejnego od końca alfabetu numeru litery szyfrującej), dodanie do wyniku liczby *k* (przesunięcie o *k* liter w prawo), a następnie obliczenie reszty z dzielenia dotychczasowgo wyniku (czyli jeżeli przesunięcie miałoby spowodować „wyjście poza alfabet", to będziemy odliczali to przesunięcie ponownie od pierwszej litery alfabetu). Po tych wszystkich obliczeniach do otrzymanej wartości dodawane jest 65, aby otrzymać kod ASCII szukanej litery.

12.1.

W kolejnych iteracjach wartość zmiennej *i* zmienia się o 1. Po każdym zwiększeniu wartości zmiennej *i* o 1 następuje zwiększenie zmiennej *k* o bieżącą wartość zmiennej *i*. Zatem uzupełniona tabela wygląda następująco:

lp	*i*	*k*
1	0	20
2	**1**	**21**
3	**2**	**23**
4	**3**	**26**
5	**4**	**30**
6	**5**	**35**

12.2.

Ponieważ po dodaniu *k* obliczana jest reszta z dzielenia przez 26, to zwiększenie *k* o 26 lub wielokrotność tej liczby da identyczny wynik. Istnieje zatem jedynie 26 wartości początkowych *k* dających różne szyfrogramy.

12.3.

Szyfrowanie słowa "MAPA" wygląda następująco:

1. Litera "M" (kod 77) jest zastępowana literą "B"
   (90-77+1) mod 26 + 65 daje w wyniku 66, czyli "B"
   - wartość *i* jest zwiększana o 1 (do 1)
   - wartość *k* jest zwiększana o i (do 2)
2. Litera "A" (kod 65) jest zastępowana literą "O"
   (90-65+2) mod 26 + 65 daje w wyniku 79, czyli "O"
   - wartość *i* jest zwiększana o 1 (do 2)
   - wartość *k* jest zwiększana o i (do 4)
3. Litera "P" (kod 80) jest zastępowana literą "B"
   (90-80+4) mod 26 + 65 daje w wyniku 66, czyli "B"
   - wartość *i* jest zwiększana o 1 (do 3)
   - wartość *k* jest zwiększana o i (do 7)
4. Litera "A" (kod 65) jest zastępowana literą "T"
   (90-65+7) mod 26 + 65 daje w wyniku 84, czyli "T"

Aby odszyfrować słowo "DAN", wykonujemy operację odwrotną do tej w algorytmie szyfrującym. W poniższych obliczeniach *x* jest kodem bieżącej litery.

1. W pierwszym kroku musimy znaleźć literę, która w wyniku szyfrowania została zastąpiona przez "D" (kod 68). Wiemy, że (90–*x*+14) mod 26 + 65 dało w wyniku 68, czyli (90–*x*+14) mod 26 dało 3. Wiemy zatem, że (90–*x*+14) było równe 3 (lub większe o wielokrotność 26). Wiemy również, że *x* miało wartość z zakresu 65-90. Łatwo zatem zauważyć, że wartością *x* powinno być 75, co odpowiada literze "K".
2. W drugim kroku musimy znaleźć literę, która w wyniku szyfrowania została zastąpiona przez "A" (kod 65). Wiemy, że (90–*x*+15) mod 26 + 65 dało w wyniku 65. Wynika z tego, że (90–*x*+15) jest równe 0 (lub większe o wielokrotność 26). Biorąc pod uwagę możliwe wartości *x*, łatwo można wyliczyć, że *x* jest równe 79, czyli odpowiada literze "O".
3. W trzecim kroku musimy znaleźć literę, która w wyniku szyfrowania została zastąpiona przez "N" (kod 78). Wiemy, że (90–*x*+17) mod 26 + 65 dało w wyniku 78. Wynika z tego, że (90–*x*+17) jest równe 13 (lub większe o wielokrotność 26). Biorąc pod uwagę możliwe wartości *x*, łatwo można wyliczyć, że *x* jest równe 68, czyli odpowiada literze "D".

12.4.

Algorytm szyfrujący wykonuje dla każdej kolejnej litery *i* następujące działania (w kolejnych krokach dodany fragment przekształcenia jest zaznaczony pogrubieniem):

- przekształca kod ASCII na numer w alfabecie <1; 26>
  (**Tekst[i]-64**);
- oblicza numer litery w alfabecie, licząc go od jego końca (w „odwróconym alfabecie")
  (**26-**(Tekst[i]-64));
- zwiększa znaleziony numer litery o *k*
  (26-(Tekst[i]-64**)+k**);
- przekształca tak obliczony numer na numer litery w „odwróconym" alfabecie <1; 26>
  ((26-(Tekst[i]-64)+k) **mod 26 + 1**);
- przekształca ten numer na kod ASCII
  ((26-(Tekst[i]-64)+k) mod 26 + 1 **+ 64**)**.**

Algorytm deszyfrujący powinien te operacje wykonać w odwrotnej kolejności:

- przekształcić kod ASCII na numer w „odwróconym" alfabecie
  (**Szyfrogram[i]-64**);
- zmniejszyć znaleziony numer litery o *k*
  ((Szyfrogram[i]-64)**-k**);
- przekształcić tak obliczony numer na numer litery w odwróconym alfabecie <1; 26>
  (((Szyfrogram[i]-64)-k) **mod 26 + 1**);
- obliczyć numer litery w „zwykłym" alfabecie
  (**26-**(((Szyfrogram[i]-64)-k) mod 26 + 1));
- przekształcić numer w alfabecie na kod ASCII
  ((26-(((Szyfrogram[i]-64)-k) mod 26 + 1))**+64**)**.**

Przekształcając tak uzyskany wzór, uzyskamy:

```
(26-(((Szyfrogram[i]-64)-k) mod 26 + 1))+64
(26-((Szyfrogram[i]-64-k) mod 26 + 1))+64
26-((Szyfrogram[i]-64-k) mod 26 + 1)+64
(26-(Szyfrogram[i]-64-k)) mod 26 + 1 + 64
(26-(Szyfrogram[i]-64-k)) mod 26 + 65
(26-Szyfrogram[i]+64+k) mod 26 + 65
(90-Szyfrogram[i]+k) mod 26 + 65
```

Pozostałe fragmenty algorytmu (pętla pobierająca kolejne litery, zmiana wartości klucza) nie ulegają zmianie. Cały algorytm będzie więc wyglądał następująco:

```
i ← 0
dopóki (i<n) wykonuj
     Tekst[i] ← (90- Szyfrogram[i]+k) mod 26 + 65
     i ← i+1
     k ← k+i
```

Warto zauważyć, że algorytm deszyfrujący jest praktycznie taki sam jak szyfrujący.

Zadanie 13.

13.1.

Podany algorytm każdy kolejny punkt przekształca na inny: współrzędne (*PX[i], PY[i]*) zamieniają się na (*ax – (PX[i] – ax), ay – (PY[i] – ay)*). Następnie przekształcony punkt jest wyszukiwany wśród podanych.

Algorytm poda wynik TAK, jeśli każdy z punktów podanego zbioru zamieni się na punkt należący również do zbioru. Jeśli którykolwiek punkt po przekształceniu nie daje się odnaleźć w tablicach PX, PY, algorytm natychmiast kończy działanie z wynikiem NIE.

W pierwszym przykładzie sprawdzamy bezpośrednio, że punkt P_1 przechodzi na P_6, P_2 na P_5, P_3 na P_4, P_4 na P_3, P_5 na P_2 oraz P_6 na P_1.

W drugim okazuje się, że punkt P_3 (-2,2) przechodzi na punkt (0,0), którego nie ma w zbiorze.

W trzecim przykładzie P_1 przechodzi na P_3, P_2 na P_4, P_3 na P_1, P_4 na P_2, zaś punkt P_5 po przekształceniu pozostaje tym samym punktem P_5 (a zatem również da się odnaleźć w zbiorze).

13.2.

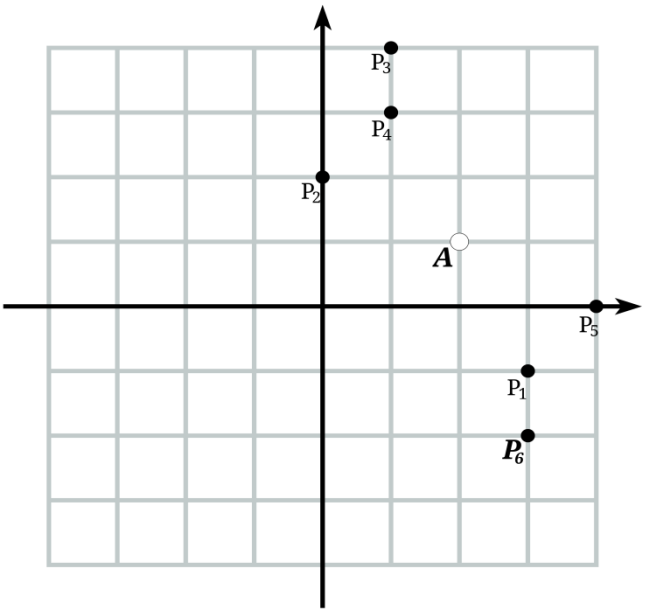
Analizując algorytm z pomocą rysunków, dochodzimy do wniosku, że każdy z punktów P_i jest przenoszony na przeciwległy (położony symetrycznie) punkt względem A. Istotnie, zmienne *roznicax* i *roznicay* mierzą różnicę między punktami P_i oraz A, kolejne ich odjęcie od współrzędnych A spowoduje więc odejście od punktu A w tym samym kierunku i na taką samą odległość, jaka była między P_i i A.

Zatem algorytm zwraca TAK, jeśli dla każdego punktu P_i da się znaleźć w zbiorze inny (lub ten sam) punkt położony symetrycznie do niego względem A. Warto przy okazji nadmienić, że w geometrii nazywa się w takiej sytuacji punkt A *środkiem symetrii* zbioru $\{P_1, P_2,..., P_n\}$, a sam zbiór — *środkowosymetrycznym*.

Jeśli rozumiemy, jak działa algorytm, pozostała nam geometryczna część zadania: musimy uzupełnić podane punkty P_1, P_2,..., P_5 jeszcze jednym punktem P_6, a także znaleźć punkt A tak, aby był on środkiem symetrii zbioru $\{P_1, P_2,..., P_n\}$. Trzeba tak dobrać punkt P_6, aby był

on symetryczny do jednego ze znanych już punktów, zaś pozostałe połączyły się w pary symetryczne względem A.

Wybieramy zatem punkt A (2,1) taki, aby punkt P_1 przechodził na P_4 (i odwrotnie), a P_2 na P_5. Punkt P_6 musi więc być symetryczny do P_3, a zatem mieć współrzędne (3,-2).



Zadanie 14.

14.1.

Zadanie polega na analizie podanego algorytmu i zaobserwowaniu, że do obliczenia pola powierzchni pewnego wielokąta wystarczy obliczyć wielkości, które zasugerowano w tabeli. Do rozwiązania zadania wystarczy wykonać elementarne obliczenia arytmetyczne w celu wyznaczenia wartości $x_{i+1} - x_i$ oraz $(x_{i+1} - x_i) \cdot y_i$ dla kolejnych wartości $i = 1,2,\ldots,5$. Do rozwiązania drugiej części zadania, a więc do wyznaczenia pola powierzchni, wystarczy wykorzystać podany w zadaniu wzór

$$pole = \frac{1}{2}\sum_{i=1}^{n}(x_{i+1} - x_{i-1}) \cdot y_i.$$

Ostatecznie odpowiedź uzyskujemy, dodając wszystkie liczby w kolumnie zawierającej wartości $(x_{i+1} - x_i) \cdot y_i$, a następnie dzieląc uzyskaną sumę przez 2.

14.2.

W zadaniu należy podać wzór na pole trójkąta przy zadanych współrzędnych jego wierzchołków. Do tego celu można wykorzystać podany algorytm geodety. Ponieważ w zadaniu jest dodatkowe założenie, że dane są współrzędne wierzchołków odwiedzanych w kolejności zgodnej z ruchem wskazówek zegara, więc do rozwiązania zadania można bezpośrednio wykorzystać wzór dany w treści zadania. Mając trzy punkty $P_1 = A = [x_1, y_1]$, $P_2 = B = [x_2, y_2], P_3 = C = [x_3, y_3]$, przyjmujemy zatem $n = 3$ oraz, wobec kroku 3 schematu geodety (podanego w treści zadania), $P_4 = P_1 = [x_1, y_1]$, a więc $x_4 = x_1$. Oznacza to, że otrzymujemy następujący wzór na pole trójkąta

$$pole = \frac{1}{2}\sum_{i=1}^{4}(x_{i+1} - x_{i-1}) \cdot y_i = \frac{(x_2 - x_0) \cdot y_1 + (x_3 - x_1) \cdot y_2 + (x_4 - x_2) \cdot y_3}{2},$$

a wobec tego, że w algorytmie przyjmujemy $x_0 = x_n = x_3$, otrzymujemy wynik.

14.3.

Aby wyznaczyć liczbę operacji arytmetycznych (dodawanie, odejmowanie, mnożenie) wykonywanych przez algorytm, wystarczy zauważyć, że wszystkie tego typu operacje wykonywane są jedynie w pętli

dla $i = 1,2, \dots, n$ **wykonuj**

$pole \leftarrow pole + (x_{i+1} - x_{i-1}) \cdot y_i$

Ponieważ wewnętrzna instrukcja wykonuje po jednej operacji każdego typu, zaś pętla wykonuje tę instrukcję n razy, łączna liczba operacji dodawania i odejmowania wynosi $2n$, a operacji mnożenia n.

Zadanie 15.

15.1.

Z zasad tworzenia zbiorów Cantora kolejnych rzędów wynika, że z każdego odcinka zbioru Cantora rzędu *n* powstają dwa odcinki w zbiorze Cantora rzędu *n*+1. Zatem z jednego odcinka w zbiorze Cantora rzędu 0 powstają dwa odcinki w zbiorze Cantora rzędu 1, następnie z dwóch odcinków w zbiorze Cantora rzędu 1 otrzymujemy cztery odcinki w zbiorze Cantora rzędu 2. Kontynuując to rozumowanie, w zbiorze rzędu 4 mamy 2*8=16 odcinków, w zbiorze rzędu 5 mamy 2*16=32 odcinków, w zbiorze rzędu 6 mamy 2*32=64 odcinków itd. Ogólnie liczba odcinków w zbiorze Cantora rzędu *n*+1 jest dwa razy większa niż liczba odcinków w zbiorze Cantora rzędu *n*. Stąd łatwo możemy wyznaczyć ogólny wzór określający liczbę odcinków w zbiorze Cantora rzędu i: $C(n)=2^n$.

15.2.

W zbiorach Cantora kolejnych rzędów (począwszy od zbioru Cantora rzędu 1) zachodzi następująca prawidłowość: długość odcinka w zbiorze rzędu *n*+1 jest równa jednej trzeciej długości odcinka ze zbioru rzędu *n*. Zatem długość odcinka w zbiorze rzędu 1 równa jest jednej trzeciej długości jednego odcinka w zbiorze rzędu 0 i wynosi $\frac{1}{3} \cdot 1 = \frac{1}{3}$, długość odcinka w zbiorze rzędu 2 równa jest jednej trzeciej długości jednego odcinka w zbiorze rzędu 1 i wynosi $\frac{1}{3} \cdot \frac{1}{3} = \frac{1}{9}$. Z kolei w zbiorze Cantora rzędu 3 długość odcinka wynosi $\frac{1}{3} \cdot \frac{1}{9} = \frac{1}{27}$, w zbiorze rzędu 4 długość odcinka wynosi $\frac{1}{81}$ itd. Na tej podstawie łatwo jest wyznaczyć ogólny wzór określający długość jednego odcinka w zbiorze Cantora rzędu *n*: $D(n)=\frac{1}{3^n}$.

15.3.

Z omówienia poprzedniego zadania wiemy, że każdy odcinek [x; y] zbioru Cantora rzędu *n* ma długość $y - x = \frac{1}{3^n}$. Uzyskujemy z niego odcinki [x; $x+\frac{1}{3^{n+1}}$] i [$x+\frac{2}{3^{n+1}}$;y], należące do zbioru Cantora rzędu *n*+1. Korzystając z tej zależności, wyznaczymy odcinki zbiorów Cantora kolejnych rzędów:

n	długości odcinków	odcinki w zbiorze Cantora rzędu *n*
0	1	[0; 1]
1	$\frac{1}{3}$	[0; $\frac{1}{3}$], [$\frac{2}{3}$; 1]
2	$\frac{1}{9}$	[0; $\frac{1}{9}$], [$\frac{2}{9}$;$\frac{1}{3}$], [$\frac{2}{3}$;$\frac{7}{9}$], [$\frac{8}{9}$;1]
3	$\frac{1}{27}$	[0; $\frac{1}{27}$], [$\frac{2}{27}$; $\frac{1}{9}$], [$\frac{2}{9}$; $\frac{7}{27}$], [$\frac{8}{27}$; $\frac{1}{3}$], [$\frac{2}{3}$; $\frac{19}{27}$], [$\frac{20}{27}$; $\frac{7}{9}$], [$\frac{8}{9}$; $\frac{25}{27}$], [$\frac{26}{27}$; 1].

Znając końce odcinków podane w postaci ułamków zwykłych, możemy zastosować algorytm konwersji liczby do jej reprezentacji w systemie pozycyjnym o określonej podstawie (w naszym przypadku podstawą systemu jest 3).

Spróbujmy jednak znaleźć ogólną prawidłowość, która zachodzi dla trójkowych reprezentacji końców odcinków w zbiorach Cantora kolejnych rzędów.

Zauważmy, że tworząc zbiór Cantora rzędu 1, usuwamy ze zbioru Cantora rzędu 0 liczby z przedziału ($\frac{1}{3}$,$\frac{2}{3}$), czyli dokładnie zbiór tych liczb, których zapis w systemie trójkowym ma na pierwszym miejscu po przecinku cyfrę 1; wyjątkiem od tej reguły jest liczba $\frac{1}{3}$ o zapisie trójkowym 0,1, czyli liczba, w której 1 jest ostatnią cyfrą. W efekcie uzyskujemy przedziały [0; 0,1], [0,2; 1].

Podobnie, tworząc zbiór Cantora rzędu 2, usuwamy ze zbioru Cantora rzędu 1 liczby, których zapis w systemie trójkowym ma na drugim miejscu po przecinku cyfrę 1; wyjątkiem są liczby o zapisie trójkowym 0,01 i 0,21, w których 1 jest ostatnią cyfrą. W efekcie uzyskujemy przedziały: [0; 0,01], [0,02; 0,1], [0,2; 0,21] i [0,22; 1]. Można zauważyć, że początki przedziałów to liczby, których rozwinięcie trójkowe:

- ma co najwyżej dwie cyfry po przecinku,
- składa się tylko z cyfr 0 i 2.

Analogicznie początki odcinków wchodzących w skład zbioru Cantora rzędu *n* to liczby, których rozwinięcie trójkowe:

- ma co najwyżej *n* cyfr po przecinku,
- składa się tylko z cyfr 0 i 2.

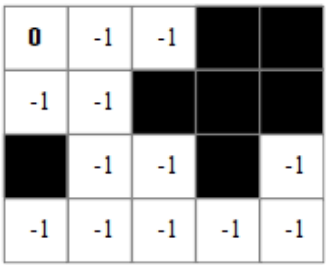
Uwzględniając powyższą obserwację oraz fakt, że odcinki w zbiorze Cantora rzędu 3 mają długość $\frac{1}{27}$=$0{,}001_{(3)}$, uzyskujemy następujące trójkowe reprezentacje odcinków tworzących zbiór Cantora rzędu 3:

[0; 0,001],[0,002;0,01], [0,02; 0,021], [0,022; 0,1],
[0,2; 0,201],[0,202;0,21], [0,22;0,221], [0,222; 1].

Zadanie 16.

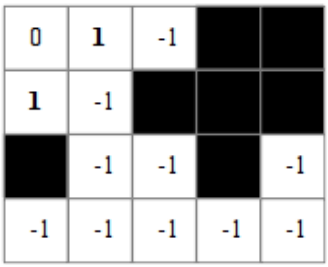
16.1.

Przeanalizujmy krok po kroku działanie algorytmu na podanej planszy. Na początku we wszystkie białe pola wpisana jest liczba -1, zaś w pole (1,1) — liczba 0:

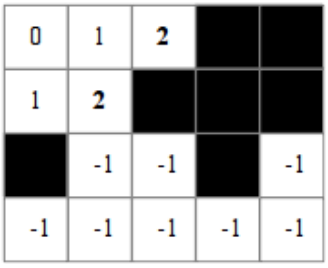


Kiedy rozpoczyna się główna pętla, zmienna *k* jest równa *0*. Algorytm wykona kolejno następujące czynności:

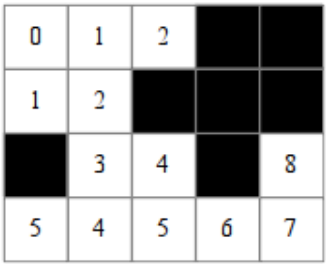
- sprawdzi, czy są takie pola, których odległość od lewego górnego rogu równa jest *k* (w tym wypadku *0*) — jest jedno takie pole na planszy, więc algorytm jeszcze się nie kończy.
- dla każdego takiego pola będzie szukał wszystkich sąsiednich, po czym jeśli jest na nich liczba -1, zastąpi ją przez *k+1*, czyli przez *1*. W efekcie otrzymamy następującą planszę:



Teraz zmienna *k* przybiera wartość 1 i powtarzane są te same instrukcje: szukamy pól z zapisaną liczbą 1 — są dwa takie na planszy — i wszystkim z nimi sąsiadującymi, które miały wartość –1 zmieniamy ich wartości na 2:



Teraz w pola sąsiadujące z dwójkami, w których wciąż jest -1, wpisujemy 3, potem sąsiadom trójek wpisujemy 4 i tak dalej. Wreszcie w ósmym okrążeniu wpiszemy liczbę 8 w ostatnie możliwe pole:



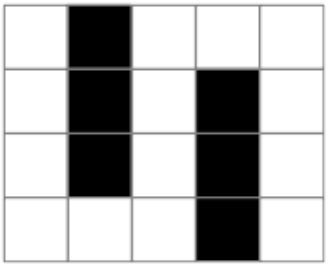
W następnej iteracji nie będzie żadnego pola, na którym byłaby wartość -1, tak więc na planszy nie pojawi się żadne pole z liczbą 9. Teraz wartość zmiennej *k* zwiększa się do 9, po czym algorytm sprawdza, czy są na planszy pola z wartością 9, a ponieważ ich nie ma, wykonywanie algorytmu się kończy.

Spróbujmy dojść do tego, jakie jest znaczenie liczb napisanych na planszy. Pole z liczbą 0, od którego zaczęliśmy wykonywanie algorytmu, nazwijmy *startowym*. Pola z liczbą 1 są sąsiadami pola startowego — doszliśmy do nich w jednym kroku algorytmu. Pola z liczbą 2 to sąsiedzi pól z jedynkami, a więc pola, do których dało się dojść z pola startowego w dwóch krokach, a nie dało się w jednym.

Można naszą obserwację sformułować ogólnie tak: jeśli ustawimy pionek na polu startowym i pozwolimy mu w jednym ruchu przejść z pola na dowolne sąsiednie (w wyłączeniem pól wyciętych), to liczba na polu będzie oznaczała **odległość tego pola od startu**: liczbę ruchów, w których pionek mógłby do niego dojść. Użyjemy tej obserwacji do rozwiązania pozostałych zadań.

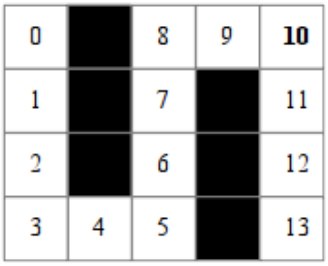
16.2.

To zadanie jest nietypowe ze względu na wiele możliwości poprawnej odpowiedzi: musimy po prostu skonstruować planszę, na której pionek będzie potrzebował aż 10 ruchów, aby dojść do jakiegoś pola. Na całkowicie pustej planszy — co łatwo sprawdzić — wystarczyłoby mu 7 ruchów dla każdego pola. Konieczne jest więc umieszczenie przeszkód: wycięcie niektórych pól tak, aby droga do niektórych pól się wydłużyła. Przypomina to znane łamigłówki, polegające na szukaniu drogi w labiryntach, tutaj jednak to my musimy zaprojektować odpowiedni



labirynt. Dobrym rozmieszczeniem przeszkód jest na przykład takie:

Widać, że tutaj pionek będzie musiał przejść dłuższą drogę do pól w prawej dolnej części planszy. Wynik algorytmu będzie na takiej planszy następujący:



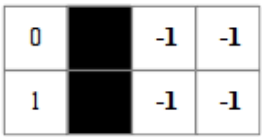
Możliwych jest bardzo wiele rozwiązań — zachęcamy do znalezienia kilku przykładów mniejszych lub większych plansz, na których droga okaże się wyjątkowo długa.

16.3.

Tutaj musimy skonstruować planszę, na której wykonywanie algorytmu zakończy się mimo tego, że na niektórych polach wciąż pozostanie wartość -1. Zgodnie z naszą interpretacją wartości na polach planszy na każdym z nich, do którego można dojść z pola startowego (1,1), pojawi się liczba oznaczająca odległość tego pola od startu.

Musimy zatem na planszy umieścić pola, które nie są wycięte, ale do których w ogóle nie da się dojść. Umieszczone przeszkody (wycięte pola) muszą odciąć część planszy od pola startowego. Tak jak w poprzednim zadaniu, i w tym jest bardzo wiele możliwości rozwiązania. Jedno z najprostszych wygląda następująco:

Żadnego pola z prawej części planszy nie da się osiągnąć z pola startowego. Wynik działania algorytmu na takiej planszy przedstawiony jest poniżej:



Zadanie 17.

17.1.

Zgodnie z opisem

- „uruchomienie hamowania następuje, jeśli odległość od najbliższego pojazdu jest **mniejsza niż** 15 metrów **oraz** prędkość samochodu, w którym działa system, wynosi **mniej niż** 30 km/h”;
- „hamulce mają być włączone do osiągnięcia przez samochód **prędkości 0 km/h**”.

Zatem poprawne rozwiązanie zadania wygląda następująco:

Błędny zapis w pseudokodzie	**Poprawny zapis w pseudokodzie**
jeżeli *prędkość samochodu* <30 **lub** *odległość między samochodami* <15	**jeżeli** *prędkość samochodu* <30 **i** *odległość między samochodami* <15
powtarzaj dopóki *prędkość samochodu* = 100:	**powtarzaj dopóki** *prędkość samochodu* > 0:

17.2.

Zauważmy, że dane brzegowe występują wówczas, gdy jedna z wartości (prędkość lub odległość) może spowodować włączenie lub wyłączenie systemu hamowania, czyli gdy zachodzi jakiś z warunków:

- prędkość jest równa 0 km/h (wyłączenie systemu);
- prędkość jest równa 29 km/h oraz odległość jest mniejsza niż 15 m (włączenie systemu);
- prędkość jest mniejsza niż 30 km/h oraz odległość jest równa 14 m (włączenie systemu).

Z danymi niezgodnymi ze specyfikacją mamy do czynienia, gdy prędkość przekracza 350 km/h (w treści zadania założono, że taka sytuacja nie występuje). W pozostałych sytuacjach mamy do czynienia z danymi standardowymi.

Powyższe obserwacje prowadzą do następującego rozwiązania:

Dane testowe	**Typ danych testowych**
***prędkość samochodu* — 29 km/h,** ***odległość między samochodami* — 1 m**	**brzegowe**
prędkość samochodu — 400 km/h, *odległość od najbliższego pojazdu — 0 m*	niezgodne
prędkość samochodu — 13 km/h, *odległość od najbliższego pojazdu — 8 m*	standardowe
prędkość samochodu — 45 km/h, *odległość od najbliższego pojazdu — 17 m*	standardowe
prędkość samochodu — 0 km/h, *odległość od najbliższego pojazdu — 17 m*	brzegowe

17.3.

Zasadniczym celem testowania oprogramowania jest sprawdzenie, czy działa ono zgodnie z przyjętymi założeniami, a także czy oprogramowanie działa zgodnie z formalną specyfikacją. Cele te zostały wyrażone w zdaniach:

- „upewnienie się, że system poradzi sobie z odczytem parametrów z czujników i temp jego reakcji pozwoli uniknąć zderzenia”, co oznacza sprawdzenie, czy system realizuje ogólne założenia (uniknięcie zderzenia);
- „sprawdzenie, czy system działa zgodnie z podaną specyfikacją”, co oznacza sprawdzenie, czy system realizuje warunki określające moment włączenia i wyłączenia systemu hamowania na podstawie odczytu prędkości i odległości od najbliższego pojazdu.

Długość kodu programu nie jest optymalizowana **na etapie** testowania, tym bardziej że krótszy kod oznacza często mniejszą czytelność programu.

Ponadto pomiary prędkości i odległości są efektem działania układów niezależnych od systemu hamowania i samo testowanie systemu hamowania nie ma na celu weryfikacji działania tych układów.

17.4.

Jeśli uwzględnimy fakt, że wartości sąsiednich pomiarów prędkości i odległości mogą się różnić o co najwyżej (odpowiednio) 1 km/h i 1m, wartości pomiarów gwarantujące **włączenie** systemu hamowania niezależnie od pomiarów poprzednich będą następujące: odległość mniejsza niż 15 m i prędkość mniejsza niż 30 km/h.

Z drugiej strony, jeśli uwzględnimy fakt, że po włączeniu systemu hamowania prędkość samochodu się zmniejsza, **wyłączenie** systemu hamowania niezależnie od pomiarów poprzednich jest możliwe, gdy prędkość równa jest co najmniej 30 km/h lub równa jest 0 km/h. (Zwróćmy uwagę, że odległość między pojazdami może w trakcie hamowania zarówno rosnąć, jak i maleć, gdyż zależy ona od prędkości obu pojazdów).

Wartości pomiarów niespełniające warunków podanych w dwóch poprzednich akapitach oznaczają, że stan systemu hamowania (włączony/wyłączony) zależny jest zarówno od aktualnych, jak i poprzednich wyników pomiarów.

Powyższe obserwacje oznaczają, że poprawnie wypełniona tabelka wygląda następująco:

Prędkość samochodu	**Odległość między samochodami**	**Stan automatycznego hamowania**
poniżej 30 km/h	poniżej 15m	włączony
poniżej 30 km/h	równa15m	nieustalony
równa 30 km/h	poniżej 15m	wyłączony
równa 30 km/h	równa 15m	wyłączony
powyżej 30 km/h	dowolna	wyłączony

Zadanie 20.

20.1.

Do rozwiązania zadania wystarczy zauważyć, że mimo brakującej cyfry znamy długość (liczbę cyfr) szukanej liczby narcystycznej. W pierwszych dwóch przykładach szukana jest liczba 3-cyfrowa, a w ostatnim 5-cyfrowa. W pierwszym przykładzie szukana liczba x jest więc w postaci

$$x = 300 + c \cdot 10 + 1,$$

gdzie c to brakująca cyfra. Zgodnie z definicją podaną w treści zadania, aby liczba x była liczbą narcystyczną, to musiałoby być prawdziwe równanie

$$x = 3^3 + c^3 + 1.$$

Odejmując równania stronami, otrzymujemy $c^3 - 10c = 273$, a stąd — że $c = 7$. Pozostałe przykłady można rozwiązać dokładnie w taki sam sposób. Warto tutaj zauważyć, że zawsze szukamy $c \in \{0,1,2, \dots ,9\}$. Zwykle większość cyfr z tego zbioru można natychmiast odrzucić.

20.2.

Do rozwiązania zadania wystarczy jedynie znajomość sposobu zamiany liczby z systemu dziesiętnego na system o podstawie *B*. Ponieważ wszystkie przykłady w zadaniu rozwiązujemy dokładnie w taki sam sposób, więc skupimy się tylko na pierwszym z nich. Po zamianie liczby 3433 na system o podstawie 6 otrzymujemy liczbę 5-cyfrową $(23521)_6$. Teraz wystarczy sprawdzić, że $3433 = 2^5 + 3^5 + 5^5 + 2^5 + 1^5$, a więc liczba 3433 jest 6-narcystyczna.

20.3.

1. Zgodnie z podaną w treści zadania definicją, algorytm sprawdzania, czy dana liczba *x* jest *B*-narcystyczna, można zrealizować w następujących dwóch krokach: Znajdź reprezentację liczby *x* w systemie o podstawie *B*. Powiedzmy, że jest ona postaci
$$x = a_{n-1}B^{n-1} + a_{(n-2)}B^{n-2} + \cdots + a_1 B + a_0.$$
2. Sprawdź, czy
$$a_{n-1}^n + a_{n-2}^n + \cdots + a_1^n + a_0^n = x.$$

Pierwszy krok można rozwiązać, korzystając z tego, że cyfra a_0 jest resztą z dzielenia liczby x przez B. Następne cyfry można wyznaczyć rekurencyjnie, po wykonaniu operacji $x \leftarrow \text{div}(x, B)$. W ten sposób otrzymamy wszystkie cyfry w kolejności od a_0 do a_{n-1}. Rekurencję tę należy zakończyć, gdy $B = 0$. W ten sposób otrzymamy również liczbę n, tzn. długość zapisu liczby x w systemie o podstawie B. Mając te informacje, przechodzimy do drugiego kroku. Jedną z możliwych jego realizacji jest napisanie do obliczania potęgi z^k pomocniczej funkcji *potega*(*z*, *k*). Wówczas obliczanie sumy $a_{n-1}^n + a_{n-2}^n + \cdots + a_1^n + a_0^n$ można zrealizować za pomocą zwykłego sumowania kolejnych składników. Zauważmy, że sumowanie to można zorganizować w kolejności od a_0^n do a_{n-1}^n, co oznacza postępowanie podobne do tego, jak w przypadku algorytmu wyznaczania reprezentacji liczby *x* w systemie o podstawie *B*:

- wyznaczać kolejne cyfry $a_0, a_1, \ldots, a_{n-1}$,
- dla każdej cyfry a_i obliczać a_i^n, korzystając z pomocniczej funkcji potega; sumując zarazem tak wyznaczone wartości n-tych potęg cyfr a_i.

Poniżej prezentujemy fragment pseudokodu realizujący naszą strategię.

```
suma ← 0;
dopóki m>0 wykonuj
        suma ← suma + potega( m mod B, n );
        m ← m div B;
```

Po obliczeniu tej sumy wystarczy sprawdzić, czy *suma=x*.

Zadanie 21.

Zauważmy, że opisany algorytm potęgowania wykorzystuje binarne rozwinięcie wykładnika, w którym jest ono przeglądane od **najbardziej znaczącego bitu** (mówimy o metodzie „od lewej do prawej"). Bardziej rozpowszechniony jest algorytm, w którym rozwinięcie binarne wykładnika jest przeglądane od najmniej znaczącego bitu (metoda „od prawej do lewej").

21.1.

Zadanie jest poprzedzone przykładem, który powinien ułatwić rozwiązanie postawionego problemu. Zgodnie z opisem w pierwszym kroku przyjmujemy, że wynik jest równy *x*. W kolejnych krokach podnosimy ten wynik do kwadratu, a w przypadku gdy kolejny bit rozwinięcia jest równy 1, wykonujemy dodatkowe mnożenie uzyskanego wyniku przez *x*. Rozwinięcie binarne liczby z zadania jest równe $k = 38 = (100110)_2$. Kolejno obliczane potęgi przedstawione zostały w tabelce. Zauważmy, że w przypadku gdy bit rozwinięcia jest równy 1, otrzymujemy dwie kolejne potęgi.

i	k_i — kolejny bit rozwinięcia	obliczona potęga
5	1	X
4	0	x^2
3	0	x^4
2	1	x^8, x^9
1	1	x^{18}, x^{19}
0	0	x^{38}

21.2.

W zadaniu trzeba obliczyć liczbę operacji wykonywanych w algorytmie. Należy pamiętać o tym, że w przypadku gdy bit rozwinięcia binarnego jest równy 1, zawsze dodatkowo wykonujemy jedno mnożenie.

Dla $k = 4$ reprezentacja binarna jest równa 100, co daje kolejne potęgi równe:

x (pierwszy bit równy 1),

$x^2 = x*x$ (drugi bit równy 0) ,

$x^4 = x^2*x^2$ (trzeci bit równy zero),

stąd liczba mnożeń jest równa 2.

Dla $k = 5 = (101)_2$ kolejne potęgi liczby x byłyby równe:

x (pierwszy bit równy 1),

$x^2 = x*x$ (drugi bit równy zero),

$x^4 = x^2* x^2$, $x^5 = x^4 * x$ (dodatkowe mnożenie, gdyż trzeci bit równy 1),

zatem otrzymujemy 3 mnożenia.

Oczywiście można dalej postępować analogicznie: po zamianie liczby k na system binarny wypisać kolejne potęgi liczby x i sprawdzać, ile mnożeń zostało wykonanych. Można jednak to zadanie wykonać sprytniej. Łatwo zauważyć, że każde 0 w zapisie binarnym daje nam jedno mnożenie, każda jedynka w zapisie binarnym (z wyjątkiem najbardziej znaczącej) daje nam 2 mnożenia. W ten sposób dla $k = 6 = (110)_2$ mamy 2+1 = 3 mnożenia, dla $k=7 = (111)_2$ wykonujemy 2+2 = 4 mnożenia itd. W ten sposób wypełnienie tabeli jest bardzo proste.

21.3.

W tym zadaniu trzeba skonstruować algorytm, który korzystając z binarnej reprezentacji wykładnika, obliczy wartość x^k metodą opisaną w zadaniu. Jak się do tego zabrać? Za wartość początkową wyniku przyjmujemy liczbę x. Następnie przeglądamy kolejne bity z binarnej reprezentacji wykładnika, od n-1 do 0. W każdym kroku podnosimy aktualny wynik do kwadratu. Dodatkowo, gdy $k_i = 1$, mnożymy wynik przez wartość x. Do rozwiązania zadania można zastosować zarówno pętlę for (**dla każdego** $i = n$-1, n-2, ..., 0), jak i pętlę while (**dopóki** $i \geq 0$ **wykonuj ...**). W tym ostatnim wypadku należy pamiętać o tym, aby zmniejszać wartość i o 1 po wykonaniu obliczeń. Przedstawione rozumowanie prowadzi do następującego rozwiązania:

```
p ← x
i ← n – 1
dopóki i ≥ 0 wykonuj
        p ← p * p
        jeżeli k_i = 1
                p ← p * x
        i ← i – 1
```

Zadanie 22.

22.1.

Do rozwiązania zadania potrzebne jest tylko zrozumienie specyfikacji podanego schematu Hornera. Ponieważ dany wielomian jest piątego stopnia, więc należy przyjąć *n*=5. Natomiast liczby rzeczywiste $a_0, a_1, \ldots, a_5$ mają być kolejnymi współczynnikami wielomianu $P(x)$. Wartość parametru x otrzymujemy natychmiast, jako że w zadaniu żąda się obliczenia wartości $P(6)$.

22.2.

Przypomnijmy, że wiersz (*) jest częścią pętli:

```
                dla k = n − 1, n − 2, ... ,0 wykonuj
(*)                 w ← x · w + a_k
```

Instrukcja w wierszu (*) wykonuje jedno dodawanie i jedno odejmowanie. Ponieważ jest ona wykonywana *n* razy, więc każda z tych operacji zostanie wykonana *n* razy.

22.3.

Z treści zadania natychmiast otrzymujemy:

$$a_0 = 9, a_1 = 7, a_2 = -5, a_3 = 2, a_4 = 0, a_5 = -3, a_6 = 4 \text{ oraz } x = 2.$$

Wartość współczynnika a_4 wynika z tego, że w podanym wzorze nie występuje x^4. Do rozwiązania zadania wystarczy zatem przeanalizować działanie danego schematu Hornera dla powyższych danych. Na początku algorytmu obliczana jest wartość $w = a_6 = 4$. Następnie dla k=5,4,3,2,1,0, algorytm, w linii (*), zmienia wartość zmiennej w. Otrzymujemy zatem dla k = 5 obliczoną wartość $w = x * 4 - 3 = 5$. W następnym kroku, tj. dla k = 4, będzie $w = x * 5 + 0 = 10$. W ten sposób otrzymujemy kolejne wyniki.

22.4.

Wskazówką do rozwiązania zadania jest obserwacja, że $R(x) = P(y)$, gdzie $y = x^2$ oraz

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0.$$

Oznacza to, że wyznaczenie wartości $R(x)$ można zrealizować bardzo efektywnie za pomocą schematu Hornera (zastosowanego do powyższego wielomianu $P(x)$), ale nie dla wyjściowej wartości x, lecz dla x^2. Rozwiązanie uzyskujemy zatem, modyfikując podany schemat Hornera (obliczania wartości $P(x)$) poprzez dodanie na początku instrukcji

$$x \leftarrow x \cdot x.$$

Zadanie 23.

23.1.

W pierwszym wierszu tabeli wpisujemy wartości początkowe nadane zmiennym *i*, *a*, *r* w algorytmie. Następnie wyliczamy wartości tych zmiennych w kolejnych wykonaniach pętli.

Wartość *i*	Wartość *a*	Wartość *r*
0	0	1
1	a ← 0+1	r ← 1+2
2	a ← 1+3	r ← 3+2
3	a ← 4+5	r ← 5+2
4	a ← 9+7	r ← 7+2
5	a ← 16+9	r ← 9+2

Warto zauważyć, że ostatni raz algorytm rozpocznie wykonanie pętli, gdy wartość *a* będzie równa 16. Ostatni wiersz tabeli prezentuje wartości zmiennych po wyjściu z pętli.

Zadanie 23.

23.1.

W pierwszym wierszu tabeli wpisujemy wartości początkowe nadane zmiennym *i*, *a*, *r* w algorytmie. Następnie wyliczamy wartości tych zmiennych w kolejnych wykonaniach pętli.

Wartość *i*	Wartość *a*	Wartość *r*
0	0	1
1	a ← 0+1	r ← 1+2
2	a ← 1+3	r ← 3+2
3	a ← 4+5	r ← 5+2
4	a ← 9+7	r ← 7+2
5	a ← 16+9	r ← 9+2

Warto zauważyć, że ostatni raz algorytm rozpocznie wykonanie pętli, gdy wartość *a* będzie równa 16. Ostatni wiersz tabeli prezentuje wartości zmiennych po wyjściu z pętli.

23.2.

Analiza zmian wartości zmiennych występujących w algorytmie pokazuje, że:

- zmienna *a* przyjmuje jako wartości kwadraty kolejnych liczb naturalnych,
- zmienna *r* przyjmuje wartości kolejnych liczb nieparzystych.

Wartość obliczona przez algorytm jest równa dokładnie wartości pierwiastka z liczby *x* tylko wtedy, gdy *x* jest kwadratem liczby naturalnej.

Po wyjściu z pętli wartość *i* jest równa liczbie powtórzeń pętli i jest o 1 większa niż wartość całkowitego pierwiastka z liczby *x*.

23.3.

Dla danej wartości *n* możemy wyznaczyć element x_n w pętli:

```
xi ← x / 2
dla i=1,2,...,n wykonuj
    xi ← (xi + x / xi) / 2
```

otrzymując wartość x_n w zmiennej *xi*.

Z treści zadania wiemy, że ciąg x_n jest zbieżny do wartości pierwiastka kwadratowego liczby *x*. Jeśli pierwiastek z *x* nie jest liczbą całkowitą, wówczas zbieżność ciągu x_n do $\sqrt{x}$ gwarantuje, że dla odpowiednio dużych *n* wartość x_n znajduje się pomiędzy zaokrągleniem $\sqrt{x}$ w dół do liczby całkowitej a zaokrągleniem $\sqrt{x}$ w górę do liczby całkowitej. Wykorzystując tę obserwację, rozwiązanie moglibyśmy uzyskać w poniższy sposób:

```
xi ← x / 2
dopóki prawda wykonuj
    xi ← (xi + x / xi) / 2
    p ← część_całkowita(xi)
    jeżeli p · p ≤ x oraz (p+1)· (p+1) >x
        zwróć p i zakończ
```

Rozwiązanie takie może jednak nie zadziałać poprawnie w sytuacji, gdy $\sqrt{x}$ **jest liczbą całkowitą**. (A przynajmniej poprawność powyższego rozwiązania nie wynika z tego, że ciąg x_n zbiega do $\sqrt{x}$). W takiej sytuacji możliwe jest, że $x_n<\sqrt{x}$ dla wszystkich dużych *n* (dających dobre przybliżenie $\sqrt{x}$). To z kolei oznacza, że dla *p* równego zaokrągleniu x_n w dół mamy $(p+1)\cdot(p+1)\leq x$. W rezultacie nigdy nie będzie spełniony warunek zakończenia powyższej pętli ($p \cdot p \leq x$ **oraz** $(p+1)\cdot(p+1) > x$). Dla ilustracji tej sytuacji wyobraźmy sobie, że ciąg x_n dla $x=16$ zbiega od dołu do 4, tzn. jego kolejne wartości są coraz bliższe 4, ale mniejsze niż 4. (W rzeczywistości taka sytuacja nie ma miejsca dla $x=16$, ale w oparciu o wiedzę szkolną nie możemy wykluczyć podobnej sytuacji dla większego *x*, będącego kwadratem liczby naturalnej). Na przykład $x_n = 4 - 1/n$. Wówczas w każdym obrocie powyższej pętli **dopóki** mamy p=3 i warunek $p \cdot p \leq x$ **oraz** $(p+1)\cdot(p+1) > x$ nie jest spełniony (mamy wtedy $p=3$, $x=16$).

Poniżej prezentujemy pełny algorytm uwzględniający sytuację, gdy $\sqrt{x}$ jest liczbą całkowitą.

```
xi ← x / 2
kontynuacja ← prawda
dopóki kontynuacja wykonuj
    xi ← (xi + x / xi) / 2
    p ← część_całkowita(xi)
    jeżeli p · p ≤ x oraz (p+1)· (p+1) >x
        kontynuacja ← fałsz
    jeżeli (p – 1)· (p – 1) ≤ x oraz p · p >x
        p ← p – 1
        kontynuacja ← fałsz
zwróć p i zakończ
```

Zadanie 24.

24.1.

Przeanalizujemy działanie funkcji F dla kolejnych przykładów oraz dla danych z zadania:

$p = 1$, $k = 5$, $e = 10$.

Dla $T = [3, 4, 6, 8, 9]$:

p	k	$k = p$	s	$T[s]$	$T[s] > 10$
1	5	nie	3	6	nie
4	5	nie	4	8	nie
5	5	tak	–	–	–

Ponieważ $T[5] \leq 10$, funkcja F zwraca wartość $p + 1 = 6$.

Dla $T = [15, 16, 18, 22, 24]$:

p	k	$k = p$	s	$T[s]$	$T[s] > 10$
1	5	nie	3	18	tak
1	3	nie	2	16	tak
1	2	nie	1	15	tak
1	1	tak	–	–	–

Ponieważ $T[1] > 10$, funkcja F zwraca wartość $p = 1$.

Dla $T = [2, 10, 16, 24, 26]$:

p	k	$k = p$	s	$T[s]$	$T[s] > 10$
1	5	nie	3	16	tak
1	3	nie	2	10	nie
3	3	tak	–	–	–

Ponieważ $T[3] > 10$, funkcja F zwraca wartość $p = 3$.

Dla $T = [1, 3, 10, 10, 18]$:

p	k	$k = p$	s	$T[s]$	$T[s] > 10$
1	5	nie	3	10	nie
4	5	nie	4	10	nie
5	5	tak	–	–	–

Ponieważ $T[5] > 10$, funkcja F zwraca wartość $p = 5$.

24.2.

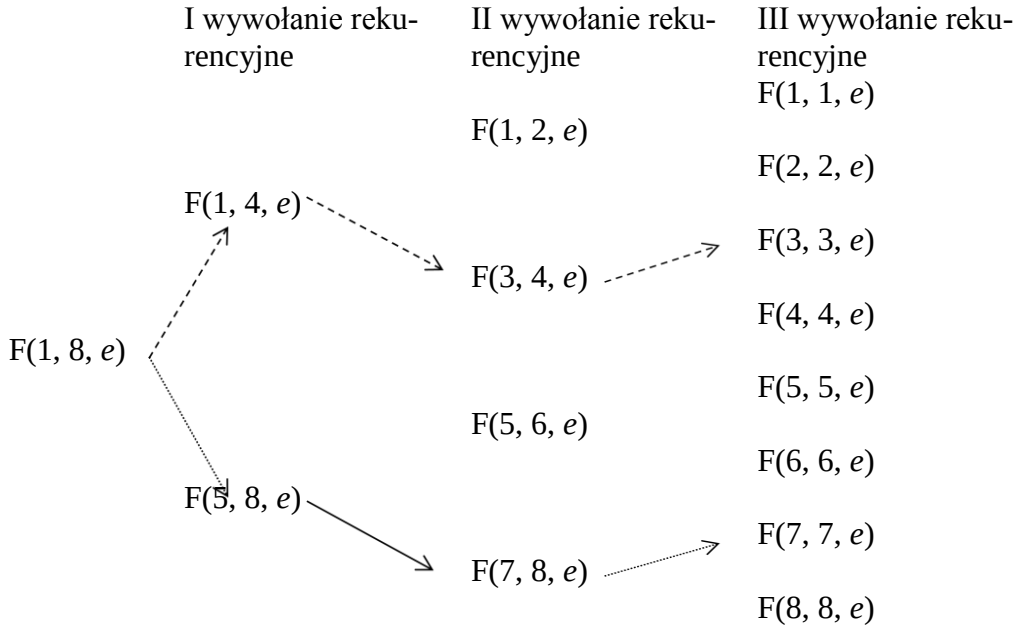
Dane są uporządkowane. W funkcji F stosujemy wywołania rekurencyjne. W każdym wywołaniu rekurencyjnym ciąg danych jest redukowany o połowę. Algorytm przypomina przeszukiwanie binarne, opiera się na tej samej zasadzie, zatem poprawną odpowiedzią jest strategia „dziel i zwyciężaj”.

24.3.

Rozważamy liczbę n, która jest potęgą dwójki. Zauważmy, że liczba wywołań rekurencyjnych jest równa liczbie wykonań instrukcji $s \leftarrow (p+k)$ div 2. Te instrukcje wykonują się, gdy $k \neq p$, czyli gdy w ciągu danych jest więcej niż 1 element. Musimy się zastanowić, ile razy odrzucamy połowę ciągu, tak aby w ciągu pozostał 1 element. Dla $n = 8 = 2^3$ pierwsze wywoła-

nie rekurencyjne zredukuje zakres poszukiwań do ciągu 4-elementowego (elementy ciągu od pierwszego do czwartego lub od piątego do ósmego). Po drugim wywołaniu analizujemy ciąg dwuelementowy. W trzecim wywołaniu sprawdzamy jeden element ciągu. Analogiczne rozważanie można przeprowadzić dla $n = 16 = 2^4$ (wywołujemy funkcję F czterokrotnie), dla $n = 32 = 2^5$ (wywołujemy funkcję F pięciokrotnie). Dla $n = 2^k$ funkcja F będzie wywołana k razy, $k = \log_2 n$.

Dla lepszego zrozumienia poniżej przedstawiono możliwe wywołania funkcji F(1, 8, *e*) oraz przykładowe ścieżki wywołań rekurencyjnych.



24.4.

Zauważmy, że funkcja F(*p, k, e*) znajduje pierwszą pozycję w posortowanej tablicy *T* [*p..k*], na której wartość jest większa od danej liczby *e*. Wynikiem wywołania F(1, *n, b*) będzie pierwsza pozycja w tablicy *T*, dla której wartość jest większa od liczby *b*. Zatem będzie to liczba elementów tablicy mniejszych bądź równych *b*. Analogicznie wywołanie funkcji F(1, *n, a*) zwróci najmniejszy indeks *i* w tablicy *T*, taki że *T* [*i*] > *a*. Zatem wartość różnicy F(1, n, b) – F(1, n, a) jest równa liczbie elementów tablicy, które należą do przedziału [*a*+1, b]. Trzeba więc jeszcze osobno zliczyć elementy, które są równe *a*, co prowadzi do złożoności liniowej algorytmu:

prawy ← F(1, *n*, *b*)
lewy ← F(1, *n*, *a*)
i ← *lewy* – 1
dopóki (*i*>0 **oraz** *T* [*i*] = *T* [*lewy*])
    i ← *i* – 1
lewy ← *i*+1
w ← *prawy* – *lewy*

Zauważmy jednak, że osobne obliczanie elementów równych *a* jest zbędne. Wywołanie funkcji F(1, *n*, *a* – 1) zwraca najmniejszy indeks *i*, taki że *T* [*i*] >*a* – 1. Prowadzi to do rozwiązania o złożoności logarytmicznej:

prawy ← F(1, *n*, *b*)
lewy ← F(1, *n*, *a* – 1)
w ← *prawy* – *lewy*

Zadanie 25.

25.1.

Zauważmy, że liczba powtórzeń pętli oznaczonej (*) dla słowa o ustalonej długości zależy od tego, czy i jak szybko napotkamy znaki różniące się od siebie. Maksymalna liczba powtórzeń dla słowa, które nie jest palindromem, jest wykonywana wtedy, gdy niezgodność zostanie wykryta dla ostatniej pary porównywanych znaków. Najpierw porównywane są pary znaków najbliżej „środka" słowa, a potem przesuwamy się do pozycji coraz bliżej początku i końca słowa. Zatem poprawną odpowiedzią jest słowo, które ma różne znaki na pierwszej i ostatniej pozycji (a po usunięciu tych dwóch znaków jest palindromem).

25.2.

W naszym rozwiązaniu porównujemy w pętli pary znaków w zdaniu, począwszy od pierwszego i ostatniego znaku. Indeksy wskazujące porównywane znaki przyjmują wartości początkowe:

i ← 1
j ← *długość*(*Zdanie*)

W kolejnych przebiegach pętli zwiększamy indeks *i* oraz zmniejszamy indeks *j* aż do momentu, gdy *i* ≥ *j*.

dopóki *i* <*j* **wykonuj**
……
i ← *i* + 1
j ← *j* – 1

Słowa w zdaniu są rozdzielone co najmniej jedną spacją. Poniższe pętle służą do pomijania znaków spacji :

dopóki *Zdanie*[*i*] =' ' **wykonuj**
  i ← *i* +1
dopóki *Zdanie*[*j*] =' ' **wykonuj**
  j ← *j* – 1

Zwróćmy uwagę, że powyższe pętle zakończą swoje działanie dzięki założeniu, że w tekście występuje co najmniej jedna litera.

Porównanie znaków i działanie algorytmu kończymy po napotkaniu rozbieżności:

jeżeli *Zdanie*[*i*] ≠ *Zdanie*[j]
  zwróć NIE i **zakończ**

Zadanie 26.

26.1.

Ustalmy najpierw zbiory liter występujących w poszczególnych słowach:

X	*Y*	Zbiór liter w *X*	Zbiór liter w *Y*
HHGGFFEEDDCCBBAA	*ABCDEFGH*	{A, B, C, D,E, F, G, H}	{A, B, C, D, E, F, G, H}
DCBADCBA	*FGHABCJD*	{A, B, C, D}	{A, B, C, D, F, G, H, J}
ABCDE	*ABCCBAE*	{A, B, C, D, E}	{A, B, C, E}
AAAAA	*AA*	{A}	{A}
ABA	*ACA*	{A, B}	{A, C}
ACEGJ	*ABCDEFGHJ*	{A, C, E, G, J}	{A, B, C, D, E, F, G, H, J}

W podanych powyżej zbiorach podkreślone zostały elementy z różnicy symetrycznej zbioru liter występujących w *X* i zbioru liter występujących w *Y*. Korzystając z powyższej tabeli i definicji *k-podrzędności*, otrzymujemy następujące rozwiązanie zadania:

X	*Y*	Podrzędność
HHGGFFEEDDCCBBAA	*ABCDEFGH*	0
DCBADCBA	*FGHABCJD*	4
ABCDE	*ABCCBAE*	NIE
AAAAA	*AA*	0
ABA	*ACA*	NIE
ACEGJ	*ABCDEFGHJ*	4

26.2.

Efektem działania dwóch pierwszych pętli jest utworzenie reprezentacji zbioru liter występujących w słowie *Y* w postaci tak zwanego „wektora charakterystycznego”. Dokładniej, po pętli

dla *i* = 1,2,…,*d* **wykonuj**
    lit ← *Y*[*i*]
    Czyjest[kod(*lit*)] ← prawda

zachodzi warunek: *Czyjest*[*i*]=prawda, gdy litera o kodzie *i* występuje w słowie *Y*, oraz *Czyjest*[*i*] = fałsz — w przeciwnym wypadku. W ostatniej pętli

czyp ← prawda
dla *i*=1,2,…,*d* **wykonuj**
    lit ← *X*[*i*]
    czyp ← *czyp* **i** *Czyjest*[kod(*lit*)]

sprawdzamy, czy w słowie *X* występuje litera, której nie ma w *Y*. Dokładniej, zmiennej *czyp* nadajemy wartość koniunkcji zdań: „czy *i*-ta litera słowa *X* występuje w *Y*”. Tak więc po zakończeniu tej pętli *czyp* ma wartość „prawda” dokładnie wtedy, gdy słowo *X* jest podrzędne względem *Y*.

Specyfikacja algorytmu A wygląda więc następująco:

Specyfikacja

Dane:

X, *Y* — słowa, w których występują tylko litery ze zbioru {*A, B, C, D, E, F, G, H, I, J*}

Wynik:

1 — gdy *X* jest słowem podrzędnym względem *Y*,

0 — gdy *X* nie jest słowem podrzędnym względem *Y*.

Tabelkę podaną w treści zadania możemy wówczas uzupełnić w oparciu o to, czy *X* jest podrzędne względem *Y*:

X	*Y*	wynik algorytmu A
HHGGFFEEDDCCBBAA	*ABCDEFGH*	1
DCBADCBA	*FGHABCJD*	1
ABCDE	*ABCCBA*	0
AAAAA	*AA*	1
AA	*AAAAA*	1
ACEGJ	*ABCDEFGH*	0

26.3.

Podobnie jak w algorytmie A analizowanym w zadaniu 2 dwie pierwsze pętle tworzą reprezentacje zbiorów liter ze słów *X* i *Y* w tablicach *Czy_x* i *Czy_y*. Dokładniej, po wykonaniu pętli

```
dx ← dlugosc(X)
dla i=1,2,…,dx wykonuj
        lit ← X[i]
        Czy_x[kod(lit)] ← prawda
dy ← dlugosc(Y)
dla i=1,2,…,dy wykonuj
        lit ← Y[i]
        Czy_y[kod(lit)] ← prawda
```

zachodzą warunki: *Czy_x*[*i*]=prawda wtedy i tylko wtedy, gdy litera o kodzie *i* występuje w słowie *X*, *Czy_y*[*i*]=prawda wtedy i tylko wtedy, gdy litera o kodzie *i* występuje w słowie *Y*.

Definicje *k-podrzędności* i podrzędności możemy wówczas wyrazić w odniesieniu do *X* i *Y* w następujący sposób:

a) *X* nie jest podrzędne względem *Y*, gdy dla pewnego $i \in [1,10]$ zachodzi: *Czy_y*[*i*]=fałsz **oraz** *Czy_x*[*i*]=prawda

b) *X* jest *k*-podrzędne względem *Y*, gdy warunek *Czy_y*[*i*]=prawda **oraz** *Czy_x*[*i*]=fałsz zachodzi dla *k* różnych wartości $i \in \{1,2,\ldots,10\}$ oraz nie zachodzi warunek a)

Oznacza to, że ostatni fragment programu należy uzupełnić w następujący sposób:

```
jeżeli Czy_y[i]=prawda oraz Czy_x[i]=fałsz
                k← k+1
jeżeli Czy_y[i]=fałsz oraz Czy_x[i]=prawda
```

26.4.

Aby uwzględnić częstość wystąpień poszczególnych liter, zmodyfikujemy ideę wektora charakterystycznego z algorytmów A, B zaprezentowanych w zadaniach 2 i 3. Utworzymy tablice liczb całkowitych *Ile_x*[1..10] i *Ile_y*[1..10]. Docelowo *Ile_x*[*i*] dla $i \in [1,10]$ ma być równe liczbie wystąpień litery o kodzie *i* w słowie *X*, podobnie *Ile_y*[*i*] dla $i \in [1,10]$ ma być równe liczbie wystąpień litery o kodzie *i* w słowie *Y*. Poniżej prezentujemy fragment pseudokodu dający taki rezultat:

```
dla i=1,2,…,10 wykonuj
        Ile_x[i] ← 0,
        Ile_y[i] ← 0
dx ← dlugosc(X)
dla i=1,2,…,dx wykonuj
        lit ← X[i]
        Ile_x[kod(lit)] ← Ile_x[kod(lit)] + 1
dy ← dlugosc(Y)
dla i=1,2,…,dy wykonuj
        lit ← Y[i]
        Ile_y[kod(lit)] = Ile_y[kod(lit)] + 1
```

Przy tablicach *Ile_x* i *Ile_y* wypełnionych zgodnie z powyższym opisem sprawdzenie równoważności *X* i *Y* sprowadza się do zweryfikowania, czy *Ile_x*[*i*]=*Ile_y*[*i*] dla każdego $i \in [1,10]$:

```
dla i=1,2,…,10 wykonuj
        jeżeli Ile_x[i] ≠ Ile_y[i]
                zwróć 0 i zakończ
zwróć 1
```

Zadanie 27.

27.1.

Miejsca wystąpień wzorca zaznaczone są w poniższej tabeli.

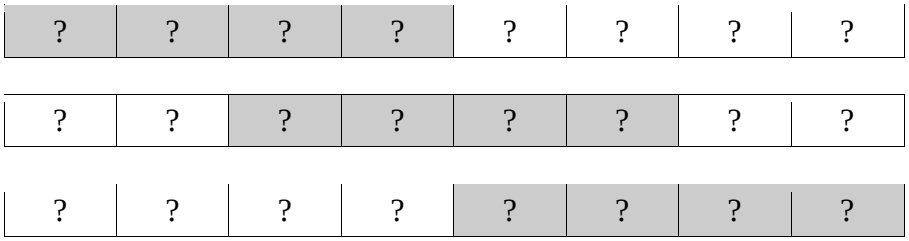
Wzorzec	**Tekst**	**Występuje?**
para	*opera*	z błędem
para	*aparat*	dokładnie
kran	*karawana*	nie
sport	*bezspornie*	z błędem
ryt	*zakryty*	dokładnie
sofa	*solanka*	z błędem

27.2.

Oczywiste jest, że trzy słowa długości 4 nie zmieszczą się w słowie o długości 8, jeśli nie będą się częściowo nakładały. Wyobraźmy sobie zatem — nie znając jeszcze jego liter — tekst o długości 8, w którym musi wystąpić trzy razy wzorzec:

?	?	?	?	?	?	?	?

Wybierzmy miejsca, w które chcemy wpasować wzorzec, na przykład w ten sposób:

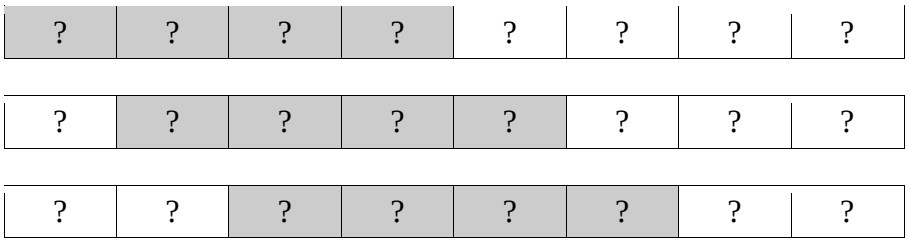


Widzimy, że skoro ten sam wzorzec ma wystąpić na miejscach 1 – 4 oraz 3 – 6, to pierwsza i trzecia litera tekstu muszą być takie same (są pierwszą literą wzorca), podobnie trzecia i piąta oraz piąta i siódma.

Analogicznie takie same muszą być litery 2, 4, 6 i 8, a to znaczy, że możliwą odpowiedzią do zadania jest na przykład:

Wzorzec *abab* występuje w nim trzy razy, tak jak zaplanowaliśmy.

Równie dobrze można jednak zamiast liter *a* i *b* wybrać na przykład *x* i *y* albo nawet *a* i *a* — otrzymalibyśmy wtedy tekst *aaaaaaaa* i wzorzec *aaaa*, będące również prawidłowym rozwiązaniem (wzorzec występuje w tekście aż pięć razy). Moglibyśmy też inaczej zaplanować wystąpienia wzorca w tekście:



W tym wypadku podobne rozumowanie prowadzi do wniosku, że litery pierwsza, druga itd. aż do szóstej, muszą być wszystkie identyczne, za to dwie ostatnie mogą być absolutnie dowolne. Otrzymamy w ten sposób kolejne możliwe rozwiązania: wzorzec *aaaa* i tekst *aaaaaabc*.

27.3.

Potrzebne nam będzie najpierw rozwiązanie bardzo prostego problemu: mając dane dwa słowa tej samej długości *n*, chcemy sprawdzić, czy są one identyczne. Odpowiedni algorytm powinien wyglądać podobnie do następującego:

Dane:

słowa A[1..n] i B[1..n]

Wynik:

„TAK”, jeśli są identyczne, „NIE”, jeśli są różne.

```
identyczne ← prawda
dla i = 1,2,..., n
        jeżeli A[i] ≠ B[i] wykonuj
                identyczne ← fałsz
jeżeli identyczne = prawda
```

```
        wypisz „TAK”
w przeciwnym razie
        wypisz „NIE”
```

Algorytm sprawdza, czy kolejne litery obu słów są zgodne. W razie znalezienia niezgodności przestawia zmienną *identyczne* na wartość ***fałsz***, którą zachowa ona do końca działania.

Za pomocą tej prostej procedury możemy teraz zrealizować wyszukiwanie w tablicy znaków *tekst[1..n]* słowa *wzorzec[1..m]* — na razie dokładnie, bez dozwolonego jednego błędu. Aby to osiągnąć, sprawdzamy najpierw, czy *wzorzec[1..m]* jest identyczny z fragmentem *tekst[1..m]*. Jeśli tak, znaleźliśmy wystąpienie wzorca. Jeśli nie, sprawdzamy fragment *tekst[2..m+1]*, a potem, być może, *tekst[3..m+2]* i tak aż do końca tekstu, czyli do ostatniego możliwego fragmentu *tekst[n-m+1..n]*. Odpowiednia pętla wygląda następująco:

```
dla i = 1, 2, ..., n–m+1 wykonuj
        identyczne ← prawda
        dla j = 1, 2, ..., m wykonuj
                jeżeli wzorzec[j] ≠ tekst[i+j-1]
                        identyczne ← fałsz
        jeżeli identyczne = prawda
                wypisz „TAK”
                zakończ algorytm
wypisz „NIE”
```

Zmienna *i* wskazuje pierwszy znak sprawdzanego fragmentu — porównujemy w każdym momencie *wzorzec[1..m]* oraz *tekst[i..i+m-1]*.

Ostatnim krokiem prowadzącym do rozwiązania zadania jest uwzględnienie jednego dozwolonego błędu w porównywaniu. Zauważmy, że dotychczas już przy pierwszej niezgodności znaków przestawialiśmy zmienną *identyczne* na **fałsz** do końca sprawdzania. Zamiast tego policzymy niezgodne znaki w osobnej zmiennej *błędy* podczas porównywania, za każdą różnicę dodając do niej 1. Odpowiednia zmiana w algorytmie jest bardzo niewielka:

```
dla i = 1, 2, ..., n–m+1 wykonuj
        bledy ← 0
        dla j = 1, 2, ..., m wykonuj
                jeżeli wzorzec[j] ≠ tekst[i+j-1]
                        błędy ← błędy + 1
        jeżeli błędy ≤ 1
                wypisz „TAK”
                zakończ wykonywanie algorytmu
wypisz „NIE”
```

Takie rozwiązanie, podobnie jak poprzednie, porównuje wszystkie możliwe fragmenty *tekstu z wzorcem*, ale teraz dla każdego z nich oblicza, ile znaków jest niezgodnych. Algorytm odpowiada TAK, jeśli którykolwiek fragment ma tych znaków nie więcej niż jeden. Jeśli wszystkie fragmenty mają co najmniej dwa niezgodne znaki, odpowiedzią jest NIE.

Zadanie 28.

28.1.

Rozważmy osobno każde ze słów AAAABBBB, ABBBABAA, ABAAAABA i AAAAAAAAA, dla których należy sprawdzić słabą A-palindromiczność oraz prawdziwość warunków 2a i 2b:

- Słowo AAAABBBB:

  W tym słowie nie jest spełniony warunek 2a, gdyż ostatnia litera słowa jest różna od pierwszej litery. Zatem AAAA BBBB nie jest słabym A-palindromem. Jednocześnie AAAA jest słabym A-palindromem, zatem warunek 2b jest spełniony.

- Słowo ABBBABAA:

  Pierwsza i ostatnia litera słowa jest równa A, zatem warunek 2a jest spełniony. Jednocześnie ABAA jest słabym A-palindromem, co oznacza, że spełniony jest warunek 2b oraz badane słowo jest słabym A-palindromem.

- Słowo ABAAAABA:

  Pierwsza i ostania litera słowa jest równa A, zatem warunek 2a jest spełniony. Jednocześnie zarówno ABAA jak i AABA są słabymi A-palindromami, co oznacza, że spełnieny jest warunek 2b.

- Słowo AAAAAAAAAA:

  Pierwsza i ostatnia litera słowa jest równa A, zatem warunek 2a jest spełniony. Połowa długości słowa to 5, zatem do spełnienia 2b konieczne jest, aby AAAAA było słabym A-palindromem. Jednak AAAAA nie jest słabym A-palindromem, tak jak nie jest nim żadne słowo o nieparzystej długości większej od 1 (do spełnienia warunku 2. konieczna jest parzysta długość słowa).

Zatem poprawnie wypełniona tabela wygląda następująco:

słowo	**2a**	**2b**	**Czy słowo jest słabym A-palindromem?**	**uzasadnienie**
AABA ABAA	tak	tak	tak	AABA
AAAB BAAA	tak	nie	nie	
AAAB BAAB	nie	nie	nie	
AAAA BBBB	**nie**	**tak**	**nie**	
ABBB ABAA	**tak**	**tak**	**tak**	**ABAA**
ABAA AABA	**tak**	**tak**	**tak**	**ABAA lub AABA**
AAAAA AAAAA	**tak**	**nie**	**nie**	

28.2.

Można zauważyć, że słaby A-palindrom długości 8 o najmniejszej liczbie liter *A* można uzyskać na dwa sposoby:

- łącząc (dowolny) słaby A-palindrom długości 4 o najmniejszej liczbie liter *A* ze słowem *BBBA* lub
- łącząc słowo *ABBB* z (dowolnym) słabym A-palindromem długości 4 o najmniejszej liczbie liter *A*.

W efekcie słabe A-palindromy długości 8 o najmniejszej liczbie liter *A* to:

ABBBABAA, **ABBB**AABA, ABAA**BBBA**, AABA**BBBA**.

Zauważmy, że każdy z tych słabych A-palindromów ma o jedną literę *A* więcej niż słabe A-palindromy długości 4 z najmniejszą liczbą liter *A*.

Uogólniając powyższe rozumowanie, słaby A-palindrom długości 2^k o najmniejszej liczbie liter *A* ma jedną z postaci:

- ABB…B x,
- x BB…BA.

gdzie x jest słabym A-palindromem długości 2^{k-1} o najmniejszej liczbie liter A, a ciąg liter B poprzedzający/następujący po x ma długość $2^{k-1} - 1$. Oznaczmy przez F_k najmniejszą liczbę liter A słabego A-palindromu długości 2^k. Z powyższej dyskusji uzyskujemy zależność:

- $F_1 = 2$,
- $F_2 = 3$,
- $F_k = F_{k-1}+1$,

co prowadzi do wniosku, że $F_k = k+1$, czyli że najmniejsza liczba liter A słabego A-palindromu długości 2^k to $k+1$.

Przykładowa poprawna zawartość pierwszej tabeli może być następująca.

m	***najmniejsza liczba liter A słabego A-palindromu o długości m***	***słaby A-palindrom o długości m i najmniejszej liczbie liter A***
2	2	AA
4	3	**ABAA**
8	**4**	**ABBB ABAA**

A zawartość drugiej tabeli jest następująca:

m	***najmniejsza liczba liter A słabego A-palindrom o długości m***
16	**5**
32	**6**
2^{10}	**11**
2^{20}	**21**

28.3.

Najprościej będzie zastosować funkcję rekurencyjną opartą na definicji słabego A-palindromu:

```
funkcja CzySlabe (W)
(1)     jeżeli W jest słowem pustym
(2)         zwróć „Nie” i zakończ
(3)     jeżeli W=”A”
(4)         zwróć „Tak” i zakończ
(5)     m ← dlugosc(W)
(6)     jeżeli m mod 2=1
(7)         zwróć „Nie” i zakończ
(8)     w przeciwnym razie
(9)         jeżeli W[1]≠W[m]
(10)            zwróć „Nie” i zakończ
(11)        r1 ← CzySlabe(W[1, m/2] )
(12)        jeżeli r1 = „Nie”, to
(13)            zwróć CzySlabe(W[m/2+1,m] )
(14)        w przeciwnym razie zwróć „Tak”
```

W przedstawionym rozwiązaniu uwzględnione zostały słowa o długości 0 (które zgodnie z definicją nie są słabymi A-palindromami, ale ten przypadek może umknąć przy „dosłownym" tłumaczeniu definicji na funkcję rekurencyjną).

Ponadto wykonaliśmy drobną optymalizację przy sprawdzaniu alternatywy:

„$W[1,m/2]$ jest słabym A-palindromem **lub** $W[m/2+1, m]$ jest słabym A-palindromem"

Zauważmy, że jeśli $r1$="Tak" po wykonaniu wiersza (11), to pierwszy składnik powyższej alternatywy jest prawdziwy. Nie trzeba więc sprawdzać drugiego składnika. Dlatego w kroku (13) sprawdzamy, czy $W[m/2+1,m]$ jest słabym A-palindromem tylko w sytuacji, gdy $r1$="Nie".

Alternatywą dla rozwiązania rekurencyjnego może być podejście wstępujące (programowanie dynamiczne):

- najpierw sprawdzamy i zapamiętujemy, które 1-literowe podsłowa W są słabymi A-palindromami,
- wykorzystując powyższe informacje, sprawdzamy, które ze słów $W[1,2]$, $W[3,4]$, $W[5,6]$, … są słabymi A-palindromami,
- wykorzystując informacje o słabych A-palindromach długości 2, sprawdzamy, które ze słów $W[1,4]$, $W[5,8]$, $W[9,12]$,… są słabymi A-palindromami,
- itd.

Rozwiązanie takie wymaga jednak dużo większej sprawności w programowaniu, wykraczającej poza wymagania szkoły ponadgimnazjalnej.

29.1.

W pierwszym zadaniu należy zdekodować zapis:

❍❍❍❍❍●●●●●❍●❍❍●●❍❍❍❍❍❍●●●❍❍❍❍●●●●

Korzystając z informacji o zakończeniu kodowania każdej litery dwiema czarnymi kulkami, można podzielić zapis na:

❍❍❍❍❍●● ●● ●❍●❍❍●● ❍❍❍❍❍❍●● ●❍❍❍❍●● ●●

Używając tabeli umieszczonej w treści zadania, można zdekodować tę wiadomość jako słowo „MATURA".

29.2.

W związku z tym, że każdy znak jest kodowany przynajmniej dwiema kulkami, a dwie ostatnie z nich są czarne, algorytm powinien zawsze analizować kolor dwóch kolejnych kulek i przerwać pobieranie, gdy natrafi kolejno na dwie czarne kulki.

29.3.

Aby wypełniać swoje zadanie, algorytm powinien cyklicznie:

- sprawdzać, czy w ciągu są jeszcze kulki,
- pobierać kulki odpowiadające jednej literze,
- dopisywać odpowiednią literę do budowanej wiadomości.

Zadanie 30.

30.1.

Argumentami funkcji *szyfruj* są znak, który chcemy zaszyfrować, i klucz (liczba wskazująca, o ile pozycji w alfabecie przesunięty jest w stosunku do znaku jawnego znak zaszyfrowany). Ponieważ *k* jest dowolną liczbą całkowitą nieujemną, operacja *k* modulo 26 pozwala obliczyć rzeczywiste przesunięcie w 26-znakowym alfabecie. Jeżeli każdej literze alfabetu jednoznacznie przypiszemy jej numer (dziesiętny kod ASCII), to do numeru znaku należy dodać klucz *k* modulo 26. W przypadku gdy liczba *kod*(zn)+*k* przekracza zakres kodów ASCII przypisanych wielkim literom, należy przejść na początek alfabetu (odjąć 26). Cykliczność rozwiązuje następująca instrukcja:

```
jeżeli kod_zn> 90
        kod_zn←kod_zn -26
```

30.2.

W zadaniu należy zaszyfrować słowo INFORMATYKA kluczami: k=1 dla pierwszej litery słowa, k=2 dla drugiej litery, … , k=11 dla ostatniej litery słowa.

W drugim wierszu tabeli należy odszyfrować podany napis, zaszyfrowany opisanym powyżej sposobem, co powinno przygotować zdającego do rozwiązania zadania 3.

30.3.

Zadanie wymaga podania algorytmu deszyfrującego słowo zaszyfrowane sposobem opisanym w zadaniu 2. Ponieważ szyfr jest symetryczny, wykonujemy operację odwrotną do szyfrowania: dla każdej litery w słowie od jej kodu ASCII odejmujemy klucz k. W przypadku gdy wartość tego wyrażenia jest mniejsza niż 65 (kod litery A), dodajemy 26. Tak zapisaną funkcję deszyfrującą należy wywołać dla każdej litery słowa, gdzie drugim argumentem funkcji będzie pozycja tej litery w słowie.

Zadanie 31.

Algorytm podany przed treścią zadań działa w taki sposób, że najpierw wypisuje na wejściu znaki $W[1]$, $W[1+m]$, $W[1+2m]$, …, potem znaki $W[2]$, $W[2+m]$, $W[2+2m]$, … itd. aż do znaków $W[m-1]$, $W[m-1+m]$, $W[m-1+2m]$, … Działanie algorytmu można też opisać inaczej:

- kolejne litery słowa W wpisujemy w kolejnych kolumnach tabeli składającej się z m wierszy (gdzie m to zaokrąglenie n/k w górę do liczby całkowitej); najpierw wypełniamy pierwszą kolumnę (od góry do dołu), potem drugą itd.
- następnie wypisujemy na wyjściu kolejne wiersze tabeli.

Interpretację tą wykorzystamy w rozwiązaniach zadań. (Jak wykażemy później, dobór wartości m zapewnia, że tekst będzie umieszczony w dokładnie k kolumnach tabeli, czyli tworzy tabelę rozmiaru $m \times k$.)

31.1.

Długość tekstu ZADANIE1JESTŁATWE jest równa 17. Mamy więc n=17 i k=3. Ponieważ 17 div 3=5 i 17 mod 3≠0, wartość m to 6. Zgodnie z powyższą interpretacją algorytmu, słowo umieścimy w następującej tabeli

Z	E	Ł
A	1	A
D	J	T
A	E	W
N	S	E
I	T	

Wynik uzyskamy wypisując zawartość tabeli wierszami: ZEŁA1ADJTAEWNSEIT

Analogicznie możemy zakodować drugi tekst o długości n=18, przyjmując m=1+18 div 4=5, gdyż 18 mod 4≠0:

Z	I	S	S
A	E	T	T
D	1	P	E
A	J	R	
N	E	O	

Wynik uzyskamy wypisując zawartość tabeli wierszami: ZISSAETTD1PEAJRNEO.

31.2

Znając długość n=14 tekstu UDOMEWIKAEOĆMD, uzyskujemy m=1+14 div 3=5. Zatem zakodowana postać tekstu została uzyskana poprzez wypisanie kolejnych wierszy tabeli 5-wierszowej, która została wypełniona poprzez wpisanie oryginalnego tekstu do kolejnych kolumn (od góry do dołu).

Zauważmy, że tekst o długości n=14 wypełni w całości 14 div 5=2 kolumny tabeli 5-wierszowej i zajmie 14 mod 5=4 pola trzeciej kolumny. Oznacza to, że zakodowana postać tekstu wypełnia kolejne wiersze tabeli w poniższy sposób

U	D	O
M	E	W
I	K	A
E	O	Ć
M	D	

Odczytując tekst kolumnami, uzyskamy wynik: UMIEMDEKODOWAĆ.

Analogicznie dla k=4 i zakodowanej postaci DRJTOZEBES mamy n=10, m=1+10 div 4=3. Tekst o długości n=10 wypełni w całości 10 div 3=3 kolumny tabeli 3-wierszowej i zajmie 10 mod 3=1 pole trzeciej kolumny. Zakodowana postać tekstu wypełnia kolejne wiersze tabeli w poniższy sposób:

D	R	J	T
O	Z	E	
B	E	S	

Odczytując tekst kolumnami, uzyskamy wynik: DOBRZEJEST.

31.3.

Najpierw pokażemy, że liczba kolumn tablicy, do której algorytm kodujący „wpisuje” tekst jest równa k.

Dla k podzielnego przez n sytuacja jest prosta: jeśli liczba wierszy to $m=n/k$ to liczba kolumn jest równa $n/m=k$, wtedy tablica pomieści dokładnie $m \cdot k=n$ liter.

Gdy k nie jest podzielne przez n, to liczba kolumn jest równa najmniejszej liczbie naturalnej p spełniającej warunek $m \cdot p > n$ (wynikowa tablica o m wierszach i p kolumnach powinna mieć więcej pól niż liter tekstu, czyli więcej niż n). Wybór m jako zaokrąglenia w górę wartości n/k oznacza, że $m \cdot k > n$ oraz $m \cdot (k-1) < n$. Zatem k spełnia warunki nałożone na p, tablica ma k kolumn.

Przypomnijmy, że kodując umieszczamy pierwsze m liter tekstu w pierwszej kolumnie naszej poglądowej tabeli. Skoro tabela ma rozmiar $m \times k$, litery z pierwszej kolumny są w zakodowanym tekście (zapisanym wierszami) na pozycjach 1, $1+k$, $1+2k$, … Analogicznie kolejne m liter tekstu tworzy drugą kolumnę, zapisaną w zakodowanym tekście na pozycjach 2, $2+k$, $2+2k$,… Poniżej prezentujemy algorytm oparty na tej obserwacji. Zewnętrzna pętla „dla $i=1,2,\ldots,k$” przebiega kolejne kolumny naszej poglądowej tabeli ilustrującej działanie kodowania. Wewnętrzna pętla „dopóki $j \leq n$” wypisuje zawartość i-tej kolumny, czyli litery na pozycjach i, $i+k$, $i+2k$,… w zakodowanej postaci tekstu.

Algorytm:

$n \leftarrow dlugosc(X)$
dla $i=1,2,\ldots,k$ **wykonuj**
    $j \leftarrow i$
    dopóki $j \leq n$ **wykonuj**
        wypisz $X[j]$
        $j \leftarrow j+k$

31.4.

Wartość m wyznaczyć możemy tak samo, jak zostało to zrobione w algorytmie podanym w treści zadania:

$m \leftarrow n$ div k
jeżeli n mod $k \neq 0$
    $m \leftarrow m+1$

Ponieważ tabela ma k wierszy i m kolumn (a tekst wpisujemy do kolejnych wierszy), pierwsza kolumna tabeli składa się z liter na pozycjach 1, $1+m$, $1+2m$, itd. dopóki nie przekroczymy wartości n. Ogólnie, i-ta kolumna tabeli dla $1 \leq i \leq m$ składa się z liter na pozycjach i, $i+m$, $i+2m$,… Ostatnia pozycja w i-tej kolumnie jest równa $i+(k-1) \cdot m$ gdy $i+(k-1) \cdot m \leq n$ oraz $i+(k-2) \cdot m$ gdy $i+(k-1) \cdot m > n$.

Korzystając z powyższych obserwacji, zawartość i-tej kolumny możemy wypisać z góry na dół w następujący sposób:

$j \leftarrow i$
dopóki $j \leq n$ **wykonuj**
    wypisz $W[j]$
    $j \leftarrow j+m$

Natomiast z dołu do góry zawartość i-tej kolumny wypisujemy tak:

$j \leftarrow i+(k-1) \cdot m$
jeżeli $j > n$
    $j \leftarrow j-m$
dopóki $j > 0$ **wykonuj**
    wypisz $W[j]$
    $j \leftarrow j-m$

Poniższy algorytm „przebiega” kolejne kolumny i wypisuje ich zawartość z góry na dół lub z dołu do góry, w zależności od tego czy *i* (czyli numer kolumny) jest liczbą parzystą czy nieparzystą.

```
n ← dlugosc(W)
m ← n div k
jeżeli n mod k≠0
        m←m+1
dla i=1,2,…,m wykonuj
        jeżeli (i mod 2=1)
                j ← i
                dopóki j ≤ n wykonuj
                        wypisz W[j]
                        j ← j+m
        w przeciwnym razie
                j ← i+(k – 1) · m
                jeżeli j>n
                        j ← j – m
                dopóki j > 0 wykonuj
                        wypisz W[j]
                        j ← j – m
```

Zadanie 32.

32.1.

Rozwijamy napisy zgodnie z definicją: na przykład fragment *(cd)* oznacza *cdcd*, a zatem *a(cd)a* to *acdcda*. Analogicznie postępujemy z pozostałymi przykładami.

32.2.

Aby *napis[1..n]* był dwukrotnym powtórzeniem, liczba n musi być parzysta, a pierwsza połowa napisu (czyli *napis[1..n/2]*) musi być identyczna z drugą połową (*napis[n/2+1..n]*). Pierwsza część algorytmu może zatem wyglądać następująco:

```
możliwe ← fałsz
jeśli n mod 2 = 0 wykonaj
        możliwe ← prawda
        dla i = 1, 2, ..., n/2
                jeśli napis[i] ≠ napis[i+n/2]
                        możliwe ← fałsz
```

W pierwszym wierszu ustawiamy zmienną *możliwe* na wartość **fałsz**. Jeśli n okaże się nieparzyste, taka wartość pozostanie już do końca algorytmu — nie wykona się żadna inna instrukcja. Jeśli n jest parzyste, ustawiamy zmienną *możliwe* na **prawdę**, ale następnie porównujemy ze sobą każdą literę z pierwszej połowy napisu (*napis[i]*) z odpowiednią literą z drugiej połowy (*napis[n/2+i]*). Jeśli wszystkie pary są identyczne, zmienna *możliwe* zachowa wartość **prawda**, w razie zaś jakiejkolwiek różnicy przyjmie wartość **fałsz.**

Widzimy zatem, że zmienna *możliwe* będzie **prawdą** dokładnie wtedy, kiedy napis będzie składać się z dwóch identycznych fragmentów. W takim wypadku musimy jeszcze wypisać wynik: wypisujemy lewy nawias, potem pierwszą połowę tablicy *napis*, a kończymy nawiasem zamykającym.

```
jeśli mozliwe wykonaj
        wypisz '('
        dla i = 1, 2, ..., n/2
                wypisz napis[i]
        wypisz ')'
```

32.3.

Załóżmy, że analizujemy kolejne litery napisu *napis[1..n]*. Dopóki nie napotkamy żadnego nawiasu, możemy te litery wypisywać na wyjście bez żadnych zmian:

```
p ← 1
dopóki p < n wykonuj
        jeśli napis[p] jest literą
                wypisz napis[p]
                p ← p+1
        ....
```

Co się jednak dzieje, kiedy napotkamy nawias? Oznacza on, że pewien fragment zaczynający się tym nawiasem, a kończący najbliższym prawym nawiasem (na przykład fragment *bar* w słowie skompresowanym *ra(bar)*), musi być wypisany dwukrotnie. W tym celu trzeba:

- znaleźć w napisie najbliższy prawy nawias (wiemy, że na pewno wystąpi, co upraszcza szukanie),
- fragment napisu między nawiasami wypisać dwukrotnie,
- przeskoczyć już wypisany fragment, a dalszą analizę tekstu rozpocząć od pierwszego znaku po prawym nawiasie.

Pierwszy punkt zrealizujemy prostą pętlą **dopóki***:*

```
p ← 1
dopóki p ≤ n wykonuj
        jeśli napis[p] jest literą
                wypisz napis[p]
                p ← p+1
        jeśli napis[p] = '('
                k ← p+1
                dopóki napis[k] ≠ ')'
                        k ← k+1
                ...
```

Wiemy, że pod *napis[p]* znajduje się znak '('. Ustawiamy zmienną *k* na *p+1*, a następnie zwiększamy ją aż pod *napis[k]* pojawi się ')'. Teraz fragmentem do dwukrotnego wypisania jest *napis[p+1..k-1]*:

```
p ← 1
dopóki p ≤ n wykonuj
        jeśli napis[p] jest literą
                wypisz napis[p]
                p ← p+1
        jeśli napis[p] = '('
                k ← p+1
                dopóki napis[k] ≠ ')'
                        k ← k+1
```

```
        powtórz 2 razy:
            dla j = p+1, ..., k-1
                wypisz napis[j]
        ...
```

Po dwukrotnym wypisaniu fragmentu miedzy nawiasami trzeba jeszcze przeskoczyć (czyli zwiększyć zmienną *p*) do pierwszego znaku po nawiasie, a więc znaku *napis[k+1]*. Kompletny algorytm ma więc następującą postać:

```
p ← 1
dopóki p ≤ n wykonuj
    jeśli napis[p] jest literą
        wypisz napis[p]
        p ← p+1
    jeśli napis[p] = '('
        k ← p+1
        dopóki napis[k] ≠ ')'
            k ← k+1
        powtórz 2 razy:
            dla j = p+1, ..., k-1
                wypisz napis[j]
        p ← k+1
```

Zadanie 33.

33.1.

Zauważmy, że wartości $B[0,k]$ i $B[n+1,k]$ są przez algorytm ustalane na 0 dla każdego $1 \le k \le n$, co oznacza, że wiersze indeksowane przez 0 i 6 można wypełnić wartościami 0. Z pierwszej pętli algorytmu widać też, że wartości w kolumnie o numerze 1 spełniają warunek: $B[i,1] = 1$, gdy $A[i,1] > 0$, oraz $B[i,1] = 0$, gdy $A[i,1] \le 0$. A zatem możemy być pewni wartości tablicy *B* przedstawionych na poniższym rysunku:

	1	2	3	4	5
0	0	0	0	0	0
1	0				
2	0				
3	1				
4	0				
5	0				
6	0	0	0	0	0

Wartość 1 na pozycji [3,1] wynika z tego, że A[3,1] jest (jedyną) dodatnią wartością w tablicy *A* w kolumnie o indeksie 1. Wewnętrzna pętla „**dla** i=1,2,…,n **wykonuj**” ustala wartości w *j*-tej kolumnie tablicy *B*. Dokładniej, wartość $B[i,j]$ jest ustawiana na 1 dla $1 \le i \le n$, jeśli choć jedna z wartości $B[i-1, j-1]$=1, $B[i, j-1]$=1, $B[i+1, j-1]$ również jest równa 1. Wypełniając w ten sposób po kolei kolumny o indeksach 2, 3, 4 i 5, uzyskujemy następującą zawartość tablicy *B*:

	1	2	3	4	5
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	1	0	0
3	1	0	0	1	0
4	0	0	0	0	1
5	0	0	0	0	0
6	0	0	0	0	0

Z ostatniej pętli algorytmu wynika jasno, że zwracana przez niego wartość jest równa 1 wtedy, gdy co najmniej jedna z wartości w kolumnie *n* tablicy *B* jest równa 1. Ponieważ warunek ten zachodzi w naszym przykładzie (*n*=5), wartość zwracana przez algorytm jest równa 1.

33.2.

Zaprezentowany algorytm jest przykładem zastosowania techniki nazywanej programowaniem dynamicznym. Z obserwacji poczynionych w omówieniu rozwiązania zadania 1 wiemy, że w kolumnie 1 tablicy *B* występuje wartość 1 dokładnie wtedy, gdy odpowiednia wartość w tablicy *A* jest dodatnia. Z drugiej strony w kolumnie *j*>1 na pozycji *i* pojawi się wartość 1 dokładnie wtedy, gdy choć jedna z wartości na pozycjach [*j* – 1, *i* – 1], [*j* – 1, *i*], [*j* – 1, *i* + 1] będzie równa 1. Po uwzględnieniu definicji wędrowców pozwala to na sformułowanie warunku: *B*[*i*,*j*]=1 po zakończeniu działania algorytmu dokładnie wtedy, gdy istnieje ścieżka wędrowca chodzącego, zaczynająca się w pierwszej kolumnie i kończąca w polu na pozycji [*i*, *j*], która przechodzi wyłącznie przez pola o wartościach dodatnich. Uzyskujemy w efekcie następującą specyfikację:

Specyfikacja

Dane:

n — liczba naturalna większa od 1,

A — plansza rozmiaru *n* × *n* wypełniona liczbami całkowitymi.

Wynik:

1 — jeśli istnieje trasa wędrowca typu **chodzącego**

i prowadząca tylko przez pola o wartościach **dodatnich,**

0 — w przeciwnym przypadku.

33.3.

Największą wartość trasy wędrowca typu skaczącego uzyskamy, wybierając z każdej kolumny największą wartość pola. Zadanie sprowadza się zatem do wyznaczenia maksimum w każdej kolumnie i zsumowania maksimów ze wszystkich kolumn. W poniższym pseudokodzie zmienna *suma* reprezentuje sumę maksymalnych wartości z kolumn 0, 1, …, *i* – 1, a w zmiennej *mx* wyznaczamy największą wartość w kolumnie *i*.

```
suma ← 0
dla j=1,2, …,n wykonuj
        mx ← A[1,j]
        dla i=2,3…,n wykonuj
            jeżeli A[i,j] >mx
                    mx ← A[i,j]
      suma ← suma + mx
zwróć suma
```

33.4.

W odniesieniu do wędrowca spadającego będziemy mówić, że pola w wierszu 1 są najwyżej, w wierszu *n* najniżej — i ogólnie wiersz *i* jest wyżej od wiersza *j* dla $i<j$.

Pole $A[i,j]$ nazywać będziemy *dopuszczalnym,* jeśli istnieje trasa wędrowca spadającego zaczynająca się w pierwszej kolumnie, przechodząca wyłącznie przez pola o wartościach nieujemnych i kończąca się w polu $A[i,j]$. Aby sprawdzić, czy istnieje trasa spełniająca warunki zadania, spróbujemy ustalić dla każdej kolumny j (zaczynając od $j=1$) **najwyższe** pole dopuszczalne. Oznaczmy numer wiersza najwyższego pola dopuszczalnego w kolumnie j przez $W[j]$. Przyjmijmy też, że gdy w kolumnie j nie ma pola dopuszczalnego, to $W[j]= -1$. Zauważmy, że trasa wędrowca spadającego może przechodzić tylko przez pola dopuszczalne.

Zgodnie z zasadami przemieszczania się wędrowca spadającego, zachodzą też następujące własności:

1. Jeśli wszystkie wartości w pierwszej kolumnie tablicy A są ujemne, to $W[1]= -1$; w przeciwnym razie $W[1]$ jest równe najmniejszej wartości i, dla której $A[i,1]\geq 0$.
2. Niech $w=W[j-1]$. Wówczas:
    a) jeśli $w= -1$, to $W[j] = -1$;
    b) jeśli $w>0$ i wszystkie wartości wśród $A[w,j]$, $A[w+1,j],\ldots,A[n,j]$ są ujemne, wtedy również $W[j]= -1$;
    c) jeśli $w>0$ i wśród $A[w,j]$, $A[w+1,j],\ldots,A[n,j]$ jest co najmniej jedna liczba nieujemna, wówczas $W[j]$ jest równe najmniejszej wartości i spełniającej warunki: $i\geq w$ oraz $A[i,j]\geq 0$.

Powyższa własność 1. wynika wprost z definicji pola dopuszczalnego: dopuszczalne pola w pierwszej kolumnie to pola z nieujemną wartością.

Własność 2.a) opisuje sytuację, gdy nie ma pola dopuszczalnego w kolumnie $j-1$. Ponieważ do kolumny $j>1$ można dojść tylko z pola w kolumnie $j-1$, brak pola dopuszczalnego w kolumnie $j-1$ oznacza też brak pola dopuszczalnego w kolumnie j.

Własności 2.b) i 2.c) opierają się na obserwacji, że najwyższe pole dopuszczalne w kolumnie $j>1$ nie może się znajdować wyżej niż najwyższe pole dopuszczalne w kolumnie $j-1$. Jest to konsekwencją tego, że do pola dopuszczalnego w kolumnie j możemy dojść **tylko** z pola dopuszczalnego w kolumnie $j-1$. W połączeniu z faktem, że rozważamy wędrowca spadającego, uzyskujemy własności 2.b) i 2.c).

Zgodnie ze specyfikacją algorytm powinien zwrócić wartość 1 dokładnie wtedy, gdy $W[n]>0$. Nasz algorytm polegać więc będzie na wyznaczaniu po kolei $W[1]$, $W[2]$, … z zastosowaniem powyższych własności 1 i 2. Wykorzystamy w nim również następującą obserwację: po napotkaniu j takiego, że $W[j] = -1$, wiemy, że nie istnieje ścieżka spełniająca warunki zadania,

i możemy zwrócić wartość 0. Algorytm realizujący nasz sposób rozwiązania zadania może wyglądać następująco:

```
W[0] ← 1
dla j=1,2, …,n wykonuj
    i ← W[j – 1]
    W[j] ← -1
    dopóki i≤n i W[j]<0
        jeżeli A[i,j] > 0
            W[j] ← i
        w przeciwnym razie i ←i+1
    jeżeli W[j]<0
        zwróć 0
zwróć 1
```

W modelu odpowiedzi przedstawiliśmy nieco inny algorytm. Główna różnica między algorytmem podanym powyżej a algorytmem z modelu odpowiedzi opiera się na obserwacji, że do wyznaczania wartości $W[j]$ wystarczy znajomość $W[j–1]$ i nie jest konieczne przechowywanie $W[1],\ldots,W[j–2]$. Można zatem tablicę liczb W zastąpić odpowiednio aktualizowanymi dwoma zmiennymi prostymi reprezentującymi wartości $W[j–1]$ i $W[j]$.

Zadanie 36.

Tabele `Uczniowie` i `Oceny` połączone są relacją jeden do wielu poprzez klucz Id_u (`JOIN Oceny ON Uczniowie.Id_u = Oceny.Id_u`). Z tabeli `Uczniowie` należy wybrać tych uczniów, a dokładnie ich nazwiska (`SELECT Uczniowie.Nazwisko`), których średnia ocen wynosi co najmniej 4,5 (`Avg(Oceny.Wynik)>= 4.5`)

Zadanie 37.

Polecenie `ORDER BY Cena DESC` porządkuje produkty malejąco względem ceny. Na trzech pierwszych pozycjach (po uporządkowaniu) znajdą się winogrona, maliny i orzechy. W tabeli, w której należy zaznaczyć prawidłowe odpowiedzi nie wymieniono wszystkich owoców z tabeli `Produkty`. Znajdujemy w niej tylko dwie z trzech możliwych nazw: maliny i orzechy.

Zadanie 38.

Korzystając z tabeli `Agenci`, należy wybrać pole obszar działania (`obszar_dzialania`) i policzyć liczbę wierszy dla różnych obszarów działania (`COUNT(*)`), co będzie jednoznaczne z policzeniem liczby agentów przypisanych do danego obszaru.

Zadanie 39.

39.1.

Należy zauważyć, że rozmiar pliku `aquapark2.jpg` jest znacznie mniejszy. Ponieważ zdjęcie ma być umieszczone na stronie internetowej, należy wybrać zdjęcie `aquapark2.jpg`. Mniejszy rozmiar oznacza szybsze załadowanie zdjęcia ze strony WWW.

39.2.

Zadanie sprawdza znajomość cech charakterystycznych formatów graficznych. Format GIF ma paletę barw ograniczoną do 256 kolorów. Format JPEG co prawda daje dobrą kompresję zdjęć przy niewielkiej utracie jakości, jednak nie nadaje się do zastosowania podczas wydruku. Poprawną odpowiedzią jest format TIFF. Obrazy w tym formacie są przechowywane bez utraty jakości. Format ten używany jest standardowo przez firmy DTP, przygotowujące materiału do druku.

39.3.

Jest to zadanie rachunkowe. Ponieważ bitmapę pamiętamy w systemie RGB, na zapamiętanie 1 piksela potrzebujemy 3 x 8 = 24 bity. Aby zapamiętać całą bitmapę, potrzebujemy 1024 x 768 x 24 = 18 874 368 bitów = 18 874 368/8 bajtów = 2 359 296 bajtów (1 bajt to 8 bitów). Przy zamianie na kilobajty należy pamiętać, że 1 kilobajt = 1024 bajty, stąd otrzymujemy 2359296/1024 = 2304 kB.

Zadanie 40.

Rozwiązując zadanie, należy zwrócić uwagę na adresowanie zakresu komórek stanowiącego argument funkcji LICZ.JEŻELI. Chcemy, aby formuła wpisana do komórki G1:

- zliczała liczbę ocen bardzo dobrych w kolumnie A,
- po skopiowaniu do komórek H1, I1 i J1 zliczała liczbę ocen bardzo dobrych w kolumnach B, C i D.

Musimy więc zadbać, aby przy kopiowaniu formuły zmieniał się odpowiednio numer kolumny. Dlatego zmiana numeru kolumny nie powinna być blokowana w adresie zakresu. Blokowanie numeru wiersza nie ma znaczenia (kopiujemy wartości w obrębie tego samego wiersza, a wszystkie formuły dotyczą tego samego zakresu wierszy 1,…,200). Odpowiedzi poprawne to:

```
LICZ.JEŻELI(A$1:A$200;"=5")
LICZ.JEŻELI(A1:A200;"=5")
```

W formule LICZ.JEŻELI(B$1:B$200;"=5") błędnie wskazano zakres danych (z przesunięciem o jedną kolumnę), a w formule LICZ.JEŻELI($A$1:$A$200;"=5") została zablokowana zmiana numeru kolumny.

Zadanie 41.

Przy imporcie danych do arkusza kalkulacyjnego wszystkim kolumnom domyślnie nadawany jest format ogólny. Formatowanie ogólne działa w ten sposób, że zawartość komórki jest traktowana jako liczba, jeśli tylko wprowadzony ciąg znaków można zamienić na liczbę. Numer PESEL, jako ciąg cyfr, jest poprawnym zapisem liczby całkowitej. W zapisie liczby całkowitej przyjmujemy, że zera umieszczone na początku nie mają znaczenia dla wartości liczby i można je pominąć. Dlatego po zastosowaniu formatowania ogólnego do numeru PESEL rozpoczynającego się od cyfry 0 otrzymamy liczbę 10-cyfrową (pierwsza cyfra zostanie pominięta). Zmiana formatowania ogólnego dla tak przekształconej liczby na format specjalny „Numer PESEL” lub tekstowy nie spowoduje odzyskania utraconej informacji. Formatowanie specjalne „Numer PESEL” umożliwi wyświetlenie liczby 10-cyfrowej jako ciągu 11 cyfr (z dodanym zerem na początku), ale zawartość komórki nie ulegnie zmianie. Dlatego jedynym

poprawnym rozwiązaniem jest nadanie kolumnie zawierającej numery PESEL już od początku formatu tekstowego.

Zadanie 42.

W zadaniu należy przeanalizować działanie danego algorytmu dla przykładowych danych. Ponieważ w obu zadaniach dane tablice są takie same, więc przedstawiona poniżej analiza dotyczy jednocześnie zadania 1 i 2.

Dwa pierwsze przykłady zawierają na tyle mało liczb, że działanie algorytmu można zrealizować po prostu krok po kroku, zliczając po drodze liczbę wykonanych porównań w wierszach (*) i (**). Aby rozwiązać pozostałe dwa przykłady, należy dokonać pełnej analizy algorytmu. Na początku zauważmy, że w algorytmie wartość zmiennej *x* jest początkowo równa T[1], i nigdy potem nie jest zmieniana. Dla tablicy T = [1, 2, 3, ..., 100] wartość *x* będzie stale równa 1. Zauważmy, że konstrukcja algorytmu zawiera jedną główną pętlę, a w jej środku następujące dwie pętle:

```
pętla 1
        wykonuj
                j ← j-1
(*)     dopóki T[j] > x

pętla 2
        wykonuj
                i ← i+1
(**)    dopóki T[i] < x
```

W pierwszej pętli algorytm zmniejsza wskaźnik *j* tak długo, dopóki T[*j*]>*x*. Natomiast w drugiej pętli algorytm zwiększa wskaźnik *i* tak długo, dopóki T[*i*]<*x*.

Dla tablicy T=[1, 2, 3, ..., 100], tj. dla *n*=100 oraz *x*=1, pierwsza pętla wykonuje porównania „T[100]>*x*”, „T[99]>*x*”, ..., „T[1]>*x*”. Ostatnie z nich ma wartość logiczną `false`; wówczas pętla kończy się, a wartość *j* jest równa 1. W drugiej pętli algorytm wykona tylko jedno porównanie, mianowicie „T[1]<*x*”, po którym pętla zostanie przerwana. Następnie można zauważyć, że algorytm przerwie główną pętlę, gdyż nie jest prawdą, że „*i*<*j*”. Ostatecznie otrzymujemy 100 porównań wykonanych w wierszu (*) i 1 porównanie w wierszu (**). Widzimy zatem, że algorytm nie wykonał żadnej operacji zamiany.

Aby rozwiązać ostatni przykład, tj. dla T=[100,99,98...,1], zauważmy, że *n*=100 oraz *x*=100. Zatem w pierwszej pętli algorytm wykona tylko jedno porównanie „T[100]>*x*”, po czym przerwie pętlę z wartością *j*=100. W drugiej pętli algorytm również wykona tylko jedno porównanie „T[1]<*x*”, po którym pętla ta zostanie przerwana z wartością *i*=1. Ponieważ *i*<*j*, więc algorytm wykona zamianę T[1] ↔T[100] oraz przejdzie ponownie do wykonania głównej pętli. Za drugim razem w pierwszej pętli algorytm znów wykona tylko jedno porównanie, mianowicie „T[99]<*x*”. W drugiej pętli natomiast zostaną wykonane porównania „T[2]<*x*”, „T[3]<*x*”, ..., „T[100]<*x*”, po których pętla zostanie przerwana z wartością *i*=100. W tym momencie zostanie przerwana również główna pętla. Oznacza to, że algorytm wykona w sumie 2 porównania w wierszu (*) i 100 w wierszu (**). Liczba zamian natomiast będzie równa 1.

Dla uzupełnienia warto jeszcze dodać, że podany algorytm tak naprawdę realizuje podział tablicy na pewne dwie części (niekoniecznie równoliczne). W pierwszej części będą znajdowały się wszystkie liczby mniejsze od *x*, a w drugiej te większe lub równe *x*. Podobny podział jest podstawą *szybkiego algorytmu sortowania* (ang. *quick-sort*).

Zadanie 43.

43.1.

Najpierw wskażmy aplikacje, dla których wykorzystanie informacji o położeniu geograficznym („lokalizacji") jest przydatne w sposób oczywisty:

- Serwis pogodowy: informacja o położeniu daje możliwość automatycznego wyboru prognozy pogody dla aktualnej lokalizacji użytkownika.
- Serwis informacji turystycznej Wrocławia: informacja o aktualnej lokalizacji użytkownika pomoże ustalić najbliższe atrakcje turystyczne oraz wskazać drogę do wybranych atrakcji.
- Wyszukiwanie optymalnej trasy samochodowej: to najlepiej znane zastosowanie systemów lokalizacji geograficznej; występuje w nim swoiste sprzężenie zwrotne: system wskazuje użytkownikowi optymalną drogę do celu, a jednocześnie dane o położeniu użytkowników pozwalają oszacować natężenie ruchu, co z kolei pozwala wybierać drogę omijającą tzw. korki samochodowe.

Pomysłowy czytelnik może zapewne wyobrazić sobie zastosowania lokalizacji w innych aplikacjach, np.:

- Znajomość lokalizacji pozwoli zasugerować język używany w edytorze tekstu czy systemie komputerowego składu tekstu (np. w Polsce spodziewamy się korzystania z języka polskiego, a w Argentynie z języka hiszpańskiego).
- Lokalizacja pozwala zasugerować format danych w arkuszu kalkulacyjnym (np. format walutowy czy sposób prezentacji daty dopasowany do kraju).

Nietrudno jednak wskazać wady i ograniczoną przydatność powyższych funkcjonalności. Personalizacja urządzeń mobilnych powoduje, że używany w edytorze tekstu język raczej przypisujemy do użytkownika, a nie do kraju, w którym on się znajduje. Wydaje się więc, że trudno o przekonujące argumenty na rzecz istotnej przydatności lokalizacji w edytorze tekstu, systemie komputerowego składu tekstu lub arkuszu kalkulacyjnym.

Lokalizacja nie wydaje się też przydatna przy korzystaniu z katalogu książek biblioteki szkolnej. (Mogłaby być przydatna w przypadku biblioteki miejskiej czy uczelnianej: instytucje takie mają wiele oddziałów i aplikacja mogłaby znaleźć najbliższy oddział, w którym dostępna jest poszukiwana książka.)

Trudno też wyobrazić sobie istotne zastosowania lokalizacji w elektronicznym dzienniku lekcyjnym lub kompilatorze języka programowania.

43.2.

Urządzenia mobilne typu smartfon/tablet są wygodne przede wszystkim w sytuacjach, w których użytkownik „konsumuje" treści (czyta, słucha). Brak dużej ergonomicznej klawiatury oraz odpowiednio dużego monitora zazwyczaj utrudnia tworzenie treści (pisanie tekstów, obsługę systemów składu tekstu, projektowanie CAD/CAM, pisanie i testowanie programów komputerowych, edycję zestawień w arkuszu kalkulacyjnym itp.).

Uwzględniając powyższe czynniki, do aplikacji (z podanej listy), których przydatność jest istotnie ograniczona w sytuacji, gdy użytkownik ma jedynie dostęp do urządzenia mobilnego, zaliczyć można: system komputerowego składu tekstu, edytor tekstu, arkusz kalkulacyjny, kompilator języka programowania.

43.3.

Mapa bitowa (np. format BMP) nie wykorzystuje zależności między pikselami, które pozwalają zmniejszać pamięć potrzebną do cyfrowej reprezentacji zdjęć. Z kolei zastosowanie kodów korekcji CRC służy zwiększeniu odporności na błędy zapisu, odczytu lub transmisji danych kosztem **zwiększenia** rozmiaru danych.

Przez rozdzielczość zdjęcia rozumiemy liczbę wierszy oraz liczbę kolumn w tablicy pikseli, z których zdjęcie się składa. Zmniejszenie rozdzielczości oznacza zmniejszenie liczby kolumn i liczby wierszy w tej tablicy. Przy zdjęciach wysokiej jakości pozwala to znacznie ograniczyć rozmiar jego cyfrowej reprezentacji bez znacznej straty jakości zdjęcia.

Celem kompresji jest zmniejszenie rozmiaru danych. Standard JPEG opisuje sposób kompresji obrazów. Jest powszechnie stosowany w cyfrowym zapisie zdjęć. W kompresji JPEG możemy zmniejszać rozmiar pliku kosztem pogorszenia wizualnej jakości zdjęcia (jakości kompresji).

Podsumowując, prawdziwe jest zdanie drugie i czwarte, natomiast pierwsze i trzecie są fałszywe.

43.4.

Protokół FTP umożliwia przesyłanie plików poprzez sieć, nie zawiera jednak żadnych mechanizmów ochrony przed nieuprawnionym dostępem. Z kolei protokół SSL został zaprojektowany właśnie w celu umożliwienia transferu zapewniającego poufność i integralność przesyłanych danych, a także uwierzytelniania uczestników komunikacji (wykorzystuje się do tych celów odpowiednie mechanizmy kryptograficzne); narzędzia te są w szczególności przydatne przy korzystaniu z usług bankowości internetowej (patrz przedrostek `https` w adresie internetowym).

Naturalnymi mechanizmami ochrony dostępu do indywidualnych kont są również hasła i PIN-y. Ich wzmocnieniem są często hasła jednorazowe tworzone dla potwierdzenia pojedynczych operacji/czynności.

Celem kompresji jest przede wszystkim zmniejszenie rozmiaru cyfrowej reprezentacji danych. Zazwyczaj nie wiąże się ono z ochroną prywatności, poufności bądź ochroną przed nieuprawnionym dostępem.

Powyższe uwagi prowadzą do następującego rozwiązania zadania:

	P	F
Protokół transferu plików FTP (File Transfer Protocol).		x
Protokół SSL (Secure Socket Layer).	x	
Uwierzytelnianie użytkownika przy pomocy hasła lub PIN.	x	
Kompresja dysku twardego.		x
Hasła jednorazowe generowane przez układy kryptograficzne i dostarczane kanałami informacyjnymi alternatywnymi dla Internetu.	x	

Zadanie 46.

Do głównych zadań systemu operacyjnego należy:

- przydział poszczególnym zadaniom czasu procesora (odpowiedź P w pierwszym zdaniu);
- przydział poszczególnym zadaniom pamięci operacyjnej (odpowiedź P w czwartym zdaniu);
- dostarczenie mechanizmów synchronizacji zadań i komunikacji między nimi;
- obsługiwanie sprzęt oraz zapewnia równolegle wykonywanym zadaniom oraz synchronizuje dostęp procesów do niego.

Dodatkowo system operacyjny może posiadać środowisko graficzne ułatwiające komunikację z użytkownikiem (w trzecim zadaniu właściwa jest odpowiedź F). Większość obecnie używanych systemów operacyjnych posiada graficzny interfejs użytkownika. Przykładami systemu operacyjnego bez środowiska graficznego są Microsoft DOS oraz GNU/Linux bez X-Windows.

Większość systemów operacyjnych posiada własny system plików, rozwijany równolegle z nim, ze względu na pewne specyficzne właściwości nadawane plikom (np. atrybut wykonywalności pliku). Z powodów użytkowych systemy operacyjne potrafią obsługiwać wiele systemów plików, w szczególności inny system do urządzeń wielokrotnego zapisu (w zależności do systemu operacyjnego może to być np. FAT/NTFS/ext3/HFS), a inny do płyt CD/DVD z ograniczonymi możliwościami ponownego zapisu, np. ISO9660 lub UDF (w drugim zadaniu właściwa jest odpowiedź F).

Zadanie 47.

NTFS to standardowy system plików począwszy od systemu Windows NT, poprzez jego kolejnych następców, aż do Windows 7 i Windows 8. Dla Linuxa to system plików obcego pochodzenia, ale po zainstalowaniu odpowiedniego sterownika Linux potrafi go obsłużyć.

NTFS przechowuje informację o rozmiarze, dacie utworzenia i modyfikacji pliku oraz ścieżce dostępu do pliku. NTFS nie ogranicza maksymalnego rozmiaru pliku do 4GB w przeciwieństwie do FAT32. W NTFS funkcjonuje tzw. lista kontroli dostępu (ACL), która umożliwia administratorowi nadawanie praw dostępu do plików i katalogów pojedynczym użytkownikom lub grupom użytkowników.

Zadanie 48.

Dla poprawnego udzielenia odpowiedzi trzeba ustalić adresy sieci, w których położone są cztery komputery. Na podstawie adresu IP i maski sieciowej można ustalić, że adresy te to:

- w przypadku komputera A: 10.20.0.0 / maska 255.255.0.0, w skrócie 10.20.0.0/16;
- w przypadku komputera B: 10.0.0.0 / maska 255.255.255.0; w skrócie 10.0.0.0/24;
- w przypadku komputera C: 1.2.3.0 / maska 255.255.255.0; w skrócie 1.2.3.0/24;
- w przypadku komputera D: 1.2.3.0 / maska 255.255.255.0; w skrócie 1.2.3.0/24.

Zdanie 1: Komputer A używa adresu IP z puli tzw. adresów prywatnych, a więc niedostępnych z sieci Internet. Komputer ten musi być podłączony do sieci Internet za pośrednictwem routera, który przy okazji może też przekierowywać ruch adresowany do pewnego publicznego adresu IP na ten właśnie komputer. (Zdanie jest prawdziwe).

Zdanie 2: Dwa z czterech komputerów używają adresów prywatnych, dwa — adresów publicznych. Nie ma to jednak znaczenia w przypadku dostępu do sieci Internet. Wszystkie wymienione komputery, niezależenie od adresu IP, jaki mają przypisany w sieciach lokalnych, mogą mieć dostęp do sieci Internet. (Zdanie jest fałszywe).

Zdanie 3: Komputer A jest w podsieci 10.20.0.0/16, a komputer B — w podsieci 10.0.0.0/24. Leżą zatem w różnych podsieciach. (Zdanie jest fałszywe).

Zdanie 4: Komputery C i D mają takie same adresy sieci (1.2.3.0/24) i są one publiczne, więc jest to ta sama podsieć. Nie ma to jednak znaczenia dla fizycznego położenia tych komputerów w jednym budynku, mogą się one znajdować w różnych budynkach. (Zdanie jest fałszywe).

Zadanie 49.

Zasada działania chmury obliczeniowej polega na przeniesieniu całego ciężaru świadczenia usług IT (danych, oprogramowania, mocy obliczeniowej) na serwer i umożliwienie stałego dostępu do niego komputerom klientów. Wystarczy zalogować się z jakiegokolwiek komputera z dostępem do Internetu, by zacząć korzystać z chmury obliczeniowej.

Chmura obliczeniowa jest szerokim pojęciem, zmierzającym w takim kierunku, aby całe oprogramowanie było instalowane na serwerze, zaś w komputerze klienta znajdował się tylko interfejs do komunikacji z tym oprogramowaniem. Oprogramowanie nie jest kopiowane i powielane w komputerze klienta, więc klient nie narusza praw autorskich i nie musi wykupować licencji, płaci jedynie za użytkowanie programów. Klient nie instaluje oprogramowania na serwerze.

Dokumenty i dane klienta przechowywane są na serwerze, na ich bezpieczeństwo nie mają wpływu ewentualne awarie komputera klienta.

Szybkość obliczeń zależy od mocy obliczeniowej serwera. Moc obliczeniowa (procesory, pamięć RAM, przestrzeń dyskowa, przepustowość łącza internetowego itp.) może być zwiększana lub zmniejszana na życzenie klienta, jest ograniczona tylko wielkością zasobów usługodawcy. Opłaty naliczane są za faktycznie wykorzystaną moc obliczeniową w danym czasie.

Zadanie 50.

HTTP Cookie jest krótkim tekstem (o rozmiarze maksymalnie 4 kB), który serwer witryny internetowej wysyła do przeglądarki klienta. Przy następnych odwiedzinach tej witryny przez klienta serwer próbuje odczytać swoje *cookie*. Odczyt *cookie* przez serwer jest możliwy tylko z domeny, która ustawiła to *cookie*, lub z jej subdomeny.

Cookie może mieć ustawiony czas życia, tzn. czas przechowywania w ustawieniach przeglądarki. Jeśli nie ma ustawionego czasu życia, jest przechowywane tylko do chwili zamknięcia okna przeglądarki.

W treści *cookie* może być zawarta informacja o preferencjach klienta: jego różnych wyborach podczas odwiedzin witryny. Z poziomu cookie nie ma możliwości wykonania poleceń, w szczególności dotyczących konfiguracji przeglądarki.

Przeglądarki umożliwiają kontrolę zapisanych *HTTP cookie* (przeglądanie, usuwanie) oraz zablokowanie samej obsługi *cookie*.

Witryny sklepów internetowych ustawiają *HTTP cookie* po zalogowaniu klienta (zapisują identyfikator sesji), aby rozpoznać tego klienta, gdy przechodzi on między różnymi stronami

serwisu sklepowego, wybierając towary do koszyka. Zablokowanie obsługi *cookie* w przeglądarce może uniemożliwić serwisowi opartemu na mechanizmie sesji rozpoznanie klienta i skompletowanie jego koszyka.

Uwaga: Istnieje także inny rodzaj cookie: *Flash Cookies*, znane jako *Local Shared Objects*, zapisywane przez przeglądarkę internetową (dokładniej: przez plugin *Flash*) po wejściu na stronę WWW zawierającą obiekt Flash, np. banner. Zawierają one także tekst, ale mogą osiągać rozmiar do 100 kB, nie mają określonego czasu wygaśnięcia (nie są automatycznie usuwane) i nie można ich przeglądać ani usunąć z poziomu ustawień przeglądarki. Można jedynie ustawić rozmiar przechowywanych danych w ustawieniach Adobe Flash Playera.

Zadanie 51.

Metoda *brute force* działa tylko na krótkich hasłach, najlepiej na takich, które złożone są z samych liter lub samych cyfr („1234", „axf").

Na atak słownikowy podatne są wszystkie hasła, które są trywialnymi ciągami kolejnych liter lub cyfr („abcdef", „1234", ale również kolejne litery na klawiaturze: „qwerty"), hasła będące prostymi słowami, zwłaszcza kojarzącymi się z hasłem („haslo", „password", „Ania", „komputer", „admin"), a także hasło identyczne z nazwą użytkownika.

Zadanie 52.

24-bitowy kolor oznacza, że na opis koloru pojedynczego piksela przeznacza się 24 bity, co oznacza, że mamy do dyspozycji paletę barw o 2^{24} =16 777 216 kolorów. 24 bity to 3 bajty. Oko człowieka nie dostrzega różnicy pomiędzy obrazem rzeczywistym a zapisanym przy pomocy 24-bitowej palety barw, dlatego taką paletę nazywa się true colour i stosuje się do zapisu fotografii. W przypadku obrazów rastrowych o rozdzielczości 300x300 pikseli mamy 90 000 pikseli opisanych przez 24 bity każdy.

90 000*24 bity = 2 160 000 bitów = 270 000 bajtów = 270 KB.

Zadanie 53.

53.1.

Na początku na podstawie zawartości zdjęć chcemy ustalić, jak w zegarze kodowane są bity 0 i 1 (który z nich kodowany jest białym kolorem, a który — czarnym), a także, czy liczby w kolumnach kodowane są „od góry do dołu" (najbardziej znaczący bit jest najwyżej, najmniej znaczący najniżej), czy też odwrotnie.

Z treści zadania wynika, że wyświetlane „metodą obrazkową" na kolejnych zdjęciach stany zegara oznaczają coraz późniejszą porę. W szczególności na zdjęciu 1 wartość godziny jest najmniejsza, a na zdjęciu 4 — największa. Przeanalizujmy kolumny z godzinami na czterech zdjęciach. Na pierwszym zdjęciu w pierwszej kolumnie występują tylko białe komórki, zatem biały kolor oznacza bit 0 (w przeciwnym razie pierwsza cyfra godziny byłaby równa 15, co oznacza kodowanie godziny niezgodne z przyjętym opisem). Na drugim i trzecim zdjęciu czarna jest pierwsza komórka od dołu w pierwszej kolumnie, zatem możemy rozważyć dwa sposoby kodowania: od góry (0001) i od dołu (1000). Wartość dziesiętna pierwszego zapisu wynosi 1, drugiego 8. Poprawnym wynikiem jest 1, gdyż pierwszą cyfrą godziny w układzie GG może być tylko 0, 1, 2. Zatem zapis binarny poszczególnych cyfr na zegarze ma najbardziej znaczący bit w pierwszym wierszu od góry. Stąd łatwo odczytać godziny na kolejnych zdjęciach:

Zdjęcie 1 — 09:20:15
Zdjęcie 2 — 11:47:51
Zdjęcie 3 — 18:33:59
Zdjęcie 4 — 23:10:36

53.2.

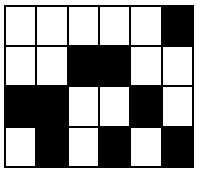
Aby zamalować w drugim zadaniu odpowiednie pola, musimy zamienić zapis poszczególnych cyfr z systemu dziesiętnego na binarny (możemy także wykorzystać informacje o sposobie kodowania z pierwszego zadania):

$2_{(10)} = 0010_{(2)}$
$3_{(10)} = 0011_{(2)}$
$4_{(10)} = 0100_{(2)}$
$5_{(10)} = 0101_{(2)}$
$2_{(10)} = 0010_{(2)}$
$9_{(10)} = 1001_{(2)}$

Przepisując zapis binarny (w ustalonym powyżej porządku) do planszy zegara, otrzymujemy:

0	0	0	0	0	1
0	0	1	1	0	0
1	1	0	0	1	0
0	1	0	1	0	1

Następnie zamieniając 1 na czarne pola, zaś 0 na pola białe, otrzymujemy rozwiązanie zadania:



Zadanie 54.

Stos jest liniową strukturą danych, w której są one dokładane na wierzch stosu i z wierzchołka stosu pobierane. Mamy możliwość dodania nowego elementu, bezpośredni dostęp do ostatnio zapisanego elementu oraz możliwość usunięcia najpóźniej dodanego elementu.

Zadanie 55.

55.1.

Należy zauważyć, że w każdym wywołaniu funkcji F(n) argument jest zmniejszany dwukrotnie. Pytanie o złożoność można zamienić na pytanie: Ile razy należy dzielić n na pół, aby uzyskać $n = 1$? Dla $n = 8 = 2^3$ kolejne wywołania funkcji to F(4), F(2), F(1), czyli dzielenie występuje 3 razy. Dla $n = 16 = 2^4$ kolejne wywołania to F(8), F(4), F(2), F(1), czyli dzielenie występuje 4 razy. Dla $n = 2^k$ takie dzielenie będzie występowało k razy, zatem złożoność algorytmu jest logarytmiczna.

55.2.

Zadanie można rozwiązać na 2 sposoby. Można przeanalizować działanie funkcji F dla podanych argumentów, np.

$$F(8) = F(4) + 1 = (F(2) + 1) + 1 = ((F(1) + 1) + 1) + 1 = \big((1 + 1) + 1\big) + 1 = 4;$$

$$F(12) = F(6) + 1 = (F(3) + 1) + 1 = \big((F(1) + 1) + 1\big) + 1 = ((1 + 1) + 1) + 1 = 4.$$

Można też zauważyć, że funkcja F oblicza liczbę bitów potrzebnych do zapamiętania liczby *n* w systemie binarnym; w ten sposób zadanie bardzo się upraszcza. I tak do zapamiętania *n* = 8, *n* = 9, *n* = 12 potrzebujemy 4 bitów, a dla *n* = 1 potrzebujemy 1 bitu.

Stąd łatwo uzyskujemy dwie pierwsze odpowiedzi. Przy udzielaniu odpowiedzi na kolejne pytania trzeba pamiętać o tym, że wartość wyrażenia *a* **lub** *b* jest fałszywa tylko w wypadku, gdy zarówno *a* = fałsz, jak i *b* = fałsz (w pozostałych przypadkach jest prawdziwa), zaś wartość wyrażenia *a* **oraz** *b* jest prawdziwa tylko wtedy, gdy zarówno *a* = prawda, jak i *b* = prawda (w pozostałych przypadkach jest fałszywa). Stąd otrzymamy:

F(1) = 0 **lub** F(9) = 4 ⇔ fałsz **lub** prawda ⇔ prawda,

F(1) = 1 **oraz** F(9) = 3 ⇔ prawda **oraz** fałsz ⇔ fałsz.

Zadanie 56.

Aby zgodnie z prawem korzystać z programu komputerowego, powinniśmy posiadać licencję na jego użytkowanie. W przeciwnym razie dopuszczamy się kradzieży, a odpowiedzialność za tego typu czyn przewiduje art. 278 par. 1kk, który mówi: "Kto zabiera w celu przywłaszczenia cudzą rzecz ruchomą podlega karze pozbawienia wolności od 3 miesięcy do 5 lat". A w odniesieniu do programu komputerowego par. 2 tegoż art. stwierdza: "Tej samej karze podlega, kto bez zgody osoby uprawnionej uzyskuje cudzy program komputerowy w celu osiągnięcia korzyści majątkowej".

Warunki licencji na użytkowanie programu precyzują, co można zrobić z posiadanym programem. Zazwyczaj licencje płatne nie dopuszczają odstępowania programu innym osobom. Warunki licencji freeware dopuszczają rozpowszechnianie programu, ale bez możliwości czerpania korzyści finansowych z tego tytułu.

Wersje demo programów można z zasady instalować na komputerze, nie posiadając licencji na dany program. Celem udostępniania wersji demo jest zachęcenie (potencjalnego) klienta do zakupu licencji.

Zadanie 57.

Poprawne odpowiedzi do zadania wynikają bezpośrednio z treści zapisów Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz.U. 1994 nr 24 poz. 83):

„Rozdział 10

Ochrona wizerunku, adresata korespondencji i tajemnicy źródeł informacji

Art. 81. 1. Rozpowszechnianie wizerunku wymaga zezwolenia osoby na nim przedstawionej. W braku wyraźnego zastrzeżenia zezwolenie nie jest wymagane, jeżeli osoba ta otrzymała umówioną zapłatę za pozowanie.

2. Zezwolenia nie wymaga rozpowszechnianie wizerunku:

1) osoby powszechnie znanej, jeżeli wizerunek wykonano w związku z pełnieniem przez nią funkcji publicznych, w szczególności politycznych, społecznych, zawodowych;

2) osoby stanowiącej jedynie szczegół całości takiej jak zgromadzenie, krajobraz, publiczna impreza.”

Zadanie 60.

60.1.

Liczbę liczb mniejszych niż 1000 przechowujemy w zmiennej *licznik.* Z każdą taką napotkaną liczbę zwiększamy licznik o 1.

Dodatkowo przechowujemy (w zmiennych *przedostatnia* i *ostatnia*) dwie ostatnie napotkane liczby mniejsze niż1000. Przy znalezieniu kolejnej takiej liczby (nazwijmy ją *a*) staje się ona *ostatnią*, a *ostatnia — przedostatnią:*

```
int przedostatnia = 0, ostatnia = 0;
int licznik = 0;
for(int i=0; i<200; i++)
{
     int a;
     in >> a;
     if (a<1000)
     {
          przedostatnia = ostatnia;
          ostatnia = a;
          licznik++;
      }
}
```

Można łatwo znaleźć inne sposoby rozwiązania tego zadania: na przykład zapisywać wszystkie odpowiednio małe liczby w tablicy i wypisać dwa ostatnie jej elementy.

60.2.

Istotą tego zadania jest sprawdzenie dla danej liczby *a*, ile ma ona dzielników, a jeśli ma ich dokładnie 18 — wypisanie ich wszystkich. Najprostszym pomysłem jest przejść w pętli wszystkie liczby od *1* do *a*, sprawdzając dla każdej, czy jest ona dzielnikiem *a.* Znalezione dzielniki zliczamy w zmiennej *ile_dzielnikow*:

```
int ile_dzielnikow = 0;
for(int i=1; i<=a; i++)
     if (a%i == 0)
          ile_dzielnikow++;
```

Jeśli okaże się, że dzielników jest 18, wypisujemy znalezioną liczbę *a*, a następnie wykonujemy jeszcze raz taką samą pętlę, tym razem wypisując dzielniki na wyjście:

```
if (ile_dzielnikow==18)
{
     out << a << endl;
     for(int i=1; i<=a; i++)
          if (a%i == 0)
               out << i << " ";
          out << endl;
}
```

Zauważmy, że w ten sposób dzielniki zostaną wypisane w prawidłowej kolejności rosnącej, jako że sprawdzamy podzielność od najmniejszych do największych liczb.

60.3.

W tym zadaniu musimy zapisać wszystkie liczby w tablicy (nazwijmy ją *liczby[1..200]*), jako że do każdej z nich będziemy musieli odwołać się wielokrotnie.

Konieczne jest również wielokrotne sprawdzanie największego wspólnego dzielnika dwóch liczb. Wygodnie jest wydzielić tę część programu do osobnej funkcji. Samo sprawdzanie zaimplementujemy bardzo znanym algorytmem, zwanym powszechnie *algorytmem Euklidesa:*

```
int nwd(int x, int y)
{
    if (x < y)
        return nwd(y,x);
    if (y == 0)
        return x;
    return nwd(y,x%y);
}
```

Funkcja *nwd()* będzie przyjmować dwa argumenty — liczby całkowite dodatnie — i zwracać ich największy wspólny dzielnik. Mając tę funkcję, iterujemy się po wszystkich liczbach i dla każdej z nich sprawdzamy jej względną pierwszość z pozostałymi:

```
for(int i=0; i<200; i++)
{
     bool ok = true;
     for(int j=0; j<200; j++)
          if (i != j && nwd(liczby[i],liczby[j])>1)
               ok = false;
     if (ok)
          (...)
}
```

Główna pętla przebiega po wszystkich liczbach — *liczba[i]* jest tą, którą właśnie sprawdzamy. Najpierw ustawiamy zmienną *ok* na ***true***, a następnie przebiegamy drugą pętlą (sterowaną zmienną *j*) po wszystkich danych liczbach. Jeśli napotkana *liczba[j]* (dla $i \neq j$, co koniecznie musimy sprawdzić!) ma wspólny dzielnik z *liczbą[i]* różny od 1, to *liczba[i]* nie może być odpowiedzią. W takim wypadku zmieniamy *ok* na ***false***. Liczby, dla których *ok* pozostanie na końcu z wartością ***true***, są względnie pierwsze z pozostałymi.

Do kompletnego algorytmu brakuje jeszcze wyznaczenia i przechowania największej liczby spełniającej nasz warunek — jest to już bardzo proste zadanie:

```
int najwieksza = 0;
for(int i=0; i<200; i++)
{
     bool ok = true;
     for(int j=0; j<200; j++)
          if (i != j && nwd(liczby[i],liczby[j])>1)
               ok = false;
      if (ok && liczby[i] > najwieksza)
           najwieksza = liczby[i];
}
```

Zadanie 61.

61.1.

Aby sprawdzić, czy dany ciąg jest arytmetyczny, najłatwiej obliczyć różnicę dwóch pierwszych wyrazów, a następnie sprawdzić, czy każda następna różnica dwóch kolejnych wyrazów jest taka sama:

```
in >> dlugosc;
for(int i = 0; i < dlugosc; i++)
     in >> ciag[i];
int roznica = ciag[1] - ciag[0];
bool arytm = true;
for(int i = 0; i+1 < dlugosc; i++)
     if (ciag[i+1] - ciag[i] != roznica)
     {
          arytm = false;
          break;
     }
```

Zmienną *arytm* ustawiamy na **true** i jeżeli któraś z różnic okaże się inna niż pierwsza obliczona, zmieniamy jej wartość z powrotem na **false** (i możemy już przerwać pętlę instrukcją **break**). Jeśli po zakończeniu pętli zmienna *arytm* wciąż ma wartość **true**, ciąg jest arytmetyczny. W takim wypadku trzeba zwiększyć o 1 odpowiedni licznik (utrzymywany w zmiennej *ilearytm*) oraz sprawdzić, czy nowa różnica nie jest większa od dotychczas znalezionych (największą dotychczas znalezioną przechowujemy w zmiennej *maxroznica*):

```
if (arytm)
{
     ilearytm++;
     if (roznica>maxroznica)
     maxroznica = roznica;
}
```

Oczywiście musimy tak uczynić dla każdego ciągu, w odpowiedniej pętli.

61.2.

Skoncentrujmy się wyłącznie na zagadnieniu: Jak sprawdzić, czy podana liczba jest sześcianem? Przechowywanie największego znalezionego sześcianu będziemy realizować podobnie jak w zadaniu 1. Jest kilka sposobów rozwiązania tego problemu, wzorcowe korzysta z tego, że skoro liczby w zadaniu są dodatnie i nie przekraczają 1 000 000, to mogą być sześcianami tylko liczb między 1 a 100. Sprawdzamy więc kolejno wszystkie takie liczby:

```
bool czy_szescian(int liczba)
{
    for(int x = 1; x <= 100; x++)
        if (x*x*x == liczba)
            return true;
    return false;
}
```

Alternatywnie można obliczyć przybliżony pierwiastek sześcienny z liczby za pomocą funkcji matematycznych (np. funkcja *pow* w C++), po czym zaokrąglić go do liczby całkowitej, podnieść do sześcianu i sprawdzić, czy wynik jest równy liczbie pierwotnej:

```
bool czy_szescian(int liczba)
{
    double pierwiastek = pow(liczba,1.0/3.0);
    int pz = (int)round(pierwiastek);
    if (pz*pz*pz==liczba)
        return true;
    else
        return false;
}
```

61.3.

To zadanie może sprawić kłopoty bardzo szczególnego rodzaju. Można do niego zastosować pomysł trochę podobny jak w zadaniu 1: obliczamy różnice między każdymi dwoma kolejnymi wyrazami, a potem sprawdzamy, czy wszystkie są równe pierwszej. Jeśli któraś jest inna, odpowiedni wyraz jest tym błędnym, który mieliśmy znaleźć:

```
in >> dlugosc;
for(int i = 0; i < dlugosc; i++)
     in >> ciag[i];
for(int i = 0; i+1 < dlugosc; i++)
     roznice[i] = ciag[i+1] - ciag[i];
```

(...)

```
for(int i = 0; i+1 < dlugosc; i++)
if (roznice[i] != roznice[0])
{
     out << ciag[i+1] << endl;
     break;
}
```

Niestety, ten sposób zawodzi w dwóch przypadkach: kiedy błędny jest pierwszy wyraz oraz kiedy błędny jest drugi wyraz ciągu[5]. Aby nasz algorytm był kompletny, musimy te dwa przypadki rozpatrzyć osobno, i to zanim użyjemy podanego wyżej sposobu.

Najpierw spróbujmy wykryć błąd na pierwszym wyrazie ciągu. Jako że wpływa on tylko na pierwszą różnicę, w takim wypadku wszystkie różnice z wyjątkiem pierwszej będą sobie

5 Rozwiązanie, które nie uwzględnia tego szczególnego przypadku, zaimplementowane jest w pliku `rozw3_zle.cpp` – można samodzielnie sprawdzić, że niektóre wyniki są dla tego rozwiązania błędne.

równe. Nie trzeba sprawdzać wszystkich: jeśli druga różnica będzie równa kolejnej, a pierwsza inna, od razu możemy stwierdzić, że odpowiedzialny za to jest pierwszy wyraz:

```
if (roznice[0]!=roznice[1] && roznice[1]==roznice[2])
{
     out << ciag[0] << endl;
     continue;
}
```

Jeśli błędny jest drugi wyraz, sytuacja jest podobna, choć nieco bardziej złożona: drugi wyraz wpływa na pierwszą i drugą różnicę, a wszystkie dalsze będą takie same. Wystarczy więc wtedy porównać trzecią i czwartą różnicę (muszą być równe) oraz sprawdzić, czy druga i pierwsza są inne:

```
if (roznice[0]!=roznice[2] && roznice[1]!=roznice[2] &&
     roznice[3]==roznice[2])
{
     out << ciag[1] << endl;
     continue;
}
```

Zadanie 62.

62.1.

Aby rozwiązać to zadanie, po wczytaniu każdej liczby należy porównać ją z dotychczas znalezionymi wartościami, najmniejszą i największą, i w razie potrzeby zapamiętać właśnie wczytaną liczbę. Jako początkowe wartości tych liczb należy przyjąć: 0 dla największej (każda liczba będzie od niej większa) i np. 1 000 000 dla najmniejszej (każda ósemkowa liczba sześciocyfrowa jest mniejsza od siedmiocyfrowej). Ponieważ wykonujemy na wczytanych liczbach tylko operacje porównywania, możemy je wczytywać i porównywać jak liczby zapisane dziesiętnie. Główna część kodu w języku C++ wygląda zatem następująco:

```
int minimum = 1000000;
int maximum = 0;

for(i=0; i<1000; i++) {
    cin >> liczba;

    if (minimum > liczba) {
        minimum = liczba;
    }
    if (maximum < liczba) {
        maximum = liczba;
    }
}
```

62.2.

Przy inicjacji za ostatnio wczytaną liczbę (*poprzednia*) można przyjąć 1000000, ponieważ każda liczba w ciągu jest od niej mniejsza. W ten sposób wykryjemy podciąg niemalejący, zaczynający się od pierwszej liczby w pliku. W rozwiązaniu tego zadania po wczytaniu każdej liczby trzeba określić, czy jest to początek (*pierwszy* element) nowego podciągu niemalejącego, czy też kontynuacja wcześniej wykrytego niemalejącego podciągu. W pierwszym przypadku wczytana liczba musi być mniejsza od *poprzedniej*, natomiast w drugim przypad-

ku nie mniejsza od poprzedniej. Trzeba pamiętać, żeby liczyć elementy podciągu niemalejącego.

```
int naj_dlugosc_podciagu = 0;
int naj_pierwszy; // pierwszy element najdłuższego podciągu
int poprzednia = 1000000;
int dlugosc_podciagu = 0; //długość aktualnie wykrywanego podciągu

for(i=0; i<1000; i++) {
    cin >> liczba;

    if (liczba < poprzednia) {
        // początek nowego podciągu, czyli zakończył się podciąg
        // niemalejący – należy porównać długość zakończonego podciągu
        // z dotychczas najdłuższymu
        if (dlugosc_podciagu > naj_dlugosc_podciagu) {
           naj_dlugosc_podciagu = dlugosc_podciagu;
           naj_pierwszy = pierwszy;
        }

        pierwszy = liczba;
        dlugosc_podciagu = 1;
    } else {
        // kontynuacja podciągu
        dlugosc_podciagu++;
    }
    // podmieniamy element „poprzedni” bieżącym
    poprzednia = liczba;
}
```

Na zakończenie trzeba jeszcze sprawdzić, czy ostatni podciąg (kończący się ostatnim elementem ciągu) nie jest najdłuższy.

Gorszym rozwiązaniem jest sprawdzanie po każdym elemencie, czy bieżący podciąg jest najdłuższy. Rozwiązanie takie jest nieoptymalne. Jego zaletą jest natomiast minimalne uproszczenie kodu i uniknięcie konieczności dodatkowego sprawdzania ostatniego podciągu.

62.3.

Najczęściej rozwiązywanie tego typu zadań polega na wczytywaniu liczb w zapisie ósemkowym jako napisów albo jako liczb dziesiętnych, a następnie konwertowaniu tych zapisów na właściwe wartości. Poniżej prezentujemy rozwiązanie, które wykorzystuje możliwość czytania liczb zapisanych ósemkowo z jednoczesną konwersją na wartości dziesiętne.

Poprawne wczytanie wartości dziesiętnej bardzo ułatwia rozwiązanie tego zadania. Jeżeli cyklicznie będziemy wczytywać liczbę w zapisie ósemkowym z pierwszego pliku oraz liczbę w zapisie dziesiętnym z drugiego pliku, to sprawdzenie warunkó•w z obu podpunktów zadania nie stanowi już żadnego problemu. Zapisany kod programu w języku C++ wygląda następująco:

```
for(i=0; i<1000; i++) {
    plik1 >> oct >> licz1;
    plik2 >> licz2;

    if (licz1 == licz2) {
        liczbar++;
    } else if (licz1 > licz2) {
        liczbaw++;
    }
}
```

Uwaga: W powyższym rozwiązaniu pokazany jest sposób czytania z plików liczb zapisanych inaczej niż dziesiętnie. W języku C++ możemy też wczytywać liczby zapisane szesnastkowo. Służy do tego manipulator `hex`. W podobny sposób możliwe jest wypisywanie liczb zarówno na ekran, jak i do pliku, np.:

```
cout << "Liczba 42 w systemie osemkowym: " << oct << 42 << endl;
```

62.4.

W tym zadaniu należy sprawdzić każdą cyfrę każdej liczby i zliczać te spośród nich, które mają wartość 6. Dodatkowo należy każdą liczbę przekształcić na liczbę ósemkową i w tak przekształconych liczbach w podobny sposób zliczyć cyfry 6.

Jeżeli liczby będziemy wczytywać z pliku do programu jako dziesiętne, to będą one tak zapamiętane. Należy wtedy odzyskać z wczytanych wartości cyfry w zapisach: dziesiątkowym i ósemkowym. Odzyskiwanie polega na dzieleniu całkowitym odpowiednio przez 10 i 8. Szukanymi cyframi są reszty z dzielenia.

Zapisany kod programu w języku C++ wygląda następująco:

```
// zliczamy '6' w zapisie dziesiętnym
temp = liczba;
while (temp) {
    if (temp%10 == 6) {
        cyfr6d++;
    }
    temp /= 10;
}

// zliczamy '6' w zapisie osemkowym
temp = liczba;
while (temp) {
    if (temp%8 == 6) {
        cyfr6o++;
    }
    temp /= 8;
}
```

Zadanie 63.

Zadanie rozpoczynamy od wczytania danych z pliku do np. tablicy napisów `ciagi`.

63.1.

Aby sprawdzić, czy ciąg jest dwucykliczny, wystarczy porównać ze sobą odpowiednie znaki napisu reprezentującego ten ciąg. Z definicji dwucykliczności wynika, że długość takiego napisu (oznaczymy ją przez d) musi być liczbą parzystą. Porównujemy ze sobą znaki napisu znajdujące się na pozycjach: 0, 1, ..., $d/2 - 1$ ze znakami znajdującymi się na pozycjach: $d/2$, $d/2+1$, ..., $d - 1$. Jeżeli na którejkolwiek parze pozycji (i, $d/2+i$) znajdują się różne znaki, to dany ciąg nie jest dwucykliczny. Poniżej przedstawiono funkcję, która sprawdza, czy ciąg jest dwucykliczny:

```
bool dwucykliczny(string ciag)
{  int d=ciag.length();
   if (d%2==1) return false;
   for (int j=0; j<d/2; j++)
       if (ciag[j]!=ciag[d/2+j]) return false;
   return true;
}
```

Alternatywnym sposobem rozwiązania jest skorzystanie z operacji na napisach (np. dzielimy napis o parzystej długości na połowy i sprawdzamy, czy są one takie same). Wymaga to jednak znajomości metody `substr()` z klasy `string`.

Aby wypisać wszystkie interesujące nas ciągi, można zastosować pętlę `for` i dla każdego napisu sprawdzić dwucykliczność przy pomocy funkcji `dwucykliczny`:

```
ofstream fout;
fout.open("wyniki_ciagi.txt");
for(int i=0; i<1000; i++)
   if (dwucykliczny(ciagi[i]))
       fout<<ciagi[i]<<endl;
```

63.2.

W tym zadaniu dla każdego ciągu sprawdzamy wszystkie pary kolejnych znaków: jeżeli któraś z par składa się z dwóch jedynek, to taki ciąg nie spełnia warunków zadania. Poniżej przedstawiamy funkcję realizującą takie sprawdzanie. Zwraca ona wartość `true` wtedy i tylko wtedy, gdy ciąg nie zawiera sąsiadujących jedynek.

```
bool dwie_jedynki(string ciag)
{for (int i=0; i<ciag.length()-1; i++)
   if  (ciag[i]=='1' && ciag[i+1]=='1') return false;
return true;
}
```

Do obliczenia liczby ciągów niezawierających dwóch sąsiadujących jedynek stosujemy pętlę, w której dla kolejnych $i = 0, 1, ..., 999$ wywołujemy funkcję `dwie_jedynki(ciag[i])`. Liczba tych ciągów zostanie zapamiętana w zmiennej `ile`. Oto fragment kodu źródłowego:

```
int ile=0;
for (int i=0; i<1000; i++)
    if (dwie_jedynki(ciagi[i])) ile++;
```

63.3.

To zadanie można podzielić na etapy. Najpierw obliczamy wartość liczby, której reprezentacja binarna jest zapamiętana w ciągu, a następnie sprawdzamy, czy liczba ta jest liczbą półpierwszą.

Krok pierwszy można wykonać na kilka sposobów. Jednym z nich jest wykorzystanie schematu Hornera, co realizuje poniższa funkcja:

```
long long int bin2wart(string bin)
{
long long int w=0;
for(int i=0; i<bin.length(); i++)
   w=w*2+(bin[i]-'0');
return w;
}
```

Zauważmy, że ciągi zostały wczytane jako napisy, zatem `bin[i]` jest zmienną typu znak, pamiętającą *i*-tą cyfrę rozwinięcia binarnego. Aby użyć tej zmiennej w obliczeniach wartości

liczby, musimy zamienić jej wartość (znakową) na wartość liczbową. Możemy to zrobić, odejmując od niej kod ASCII cyfry 0: `bin[i]-'0'` lub `bin[i]-48`.

Sprawdzanie, czy liczba jest półpierwsza, można także zrealizować na kilka sposobów. Pierwszy z nich korzysta z algorytmu obliczania czynników pierwszych liczby. Możemy napisać funkcję, która w pętli sprawdza kolejnych kandydatów na czynniki pierwsze: jeśli badana liczba dzieli się bez reszty przez kandydata, to zwiększamy o 1 liczbę znalezionych czynników i modyfikujemy badaną liczbę (dzielimy ją przez znaleziony czynnik). Oczywiście, gdy liczba znalezionych czynników przekroczy 2, możemy przerwać obliczenia — badana liczba nie jest liczbą półpierwszą. Jeśli obliczenia nie zostały przerwane, mamy dwie możliwości: albo badana liczba jest liczbą pierwszą (gdy liczba czynników jest równa 1), albo jest liczbą półpierwszą. Musimy sprawdzić, z którym przypadkiem mamy do czynienia. Poniżej przedstawiamy fragment kodu:

```
bool czy_polpierwsza(long long int x)
{
int ile=0;
int i=2;
while (x>1){
      if (x%i==0){
         ile++;
         x=x/i;
         if (ile>2) return false;
      }
      else i++;
}
if (ile==1) return false;
return true;
}
```

Innym sposobem jest skorzystanie wprost z definicji liczby półpierwszej: jest ona iloczynem dwóch liczb pierwszych. Dla badanej liczby sprawdzamy kolejnych kandydatów na czynniki iloczynu, zaczynając od dwójki. Jeżeli badana liczba dzieli się przez kandydata bez reszty oraz liczbami pierwszymi są zarówno kandydat, jak i wynik dzielenia badanej liczby przez kandydata, to taka liczba jest półpierwsza. Przykładem realizacji opisanej metoda jest poniższa funkcja:

```
bool czy_polpierwsza2(long long int x)
{
for (long long int j=2; j<=x; j++)
   if (x%j == 0 && czy_pierwsza(j) && czy_pierwsza(x/j))
       return true;
return false;
}
```

Zauważmy jednak, że obydwie metody mogą być bardzo czasochłonne. Przykładowo, jeśli badana liczba x jest liczbą pierwszą, to obydwie metody będą sprawdzać wszystkich kandydatów z przedziału $[2,x]$. Ponieważ x może być rzędu 2^{18}, liczba wykonywanych operacji może być nieakceptowalnie wielka.

Można to poprawić, modyfikując funkcję `czy_polpierwsza2()` tak, by nie sprawdzać kandydatów o wartości większej niż pierwiastek z x. Ponadto można usprawnić tę funkcję, likwidując niepotrzebne wywołania funkcji `czy_pierwsza()`. Zauważmy bowiem, że:

- jeśli liczba *x* jest liczbą pierwszą, to nie może być liczbą półpierwszą;
- jeśli liczba *x* jest parzysta, to wystarczy sprawdzić, czy liczba *x*/2 jest liczbą pierwszą;
- jeśli liczba *x* dzieli się bez reszty przez *dzielnik* nieparzysty, to wystarczy sprawdzić, czy liczba *x*/*dzielnik* jest liczbą pierwszą. Jest tak dlatego, że pierwszy napotkany przez algorytm dzielnik nieparzysty jest liczbą pierwszą.

Prowadzi to do następującej funkcji:

```
bool czy_polpierwsza2_v2(long long int x)
{    if (czy_pierwsza(x)) return false;
     if (x%2==0) return (czy_pierwsza(x/2));
     for (long long int j=3; j*j<=x; j+=2)
          if (x%j == 0) return (czy_pierwsza(x/j));
}
```

Funkcja `czy_pierwsza()`, która jest wywoływana powyżej, jest realizacją algorytmu sprawdzania, czy liczba jest pierwsza. Niech badaną liczbą będzie x. Jeżeli x jest równe 1, to ta liczba nie jest pierwsza. Jeżeli x dzieli się przez 2 bez reszty i jednocześnie nie jest równe 2, to badana liczba również nie jest pierwsza. Jeżeli x podzieli się przez jakąkolwiek spośród liczb: 3, 5, 7, ..., $\sqrt{n}$, to także nie będzie liczbą pierwszą. Algorytm ten realizuje poniższa funkcja:

```
bool czy_pierwsza(long long int x)
{
     if (x==1) return false;
     if (x%2 == 0 && x!=2) return false;
     for(long long int i=3; i*i<=x; i+=2)
        if (x%i == 0) return false;
     return true;
}
```

Dodatkowo należy wyznaczyć minimalną i maksymalną liczbę półpierwszą. Najpierw znajdziemy pierwszą liczbę półpierwszą w danych, przyjmując ją jako początkową wartość minimalnej i maksymalnej liczby półpierwszej:

```
long long int min, max, liczba;
int j=0;
bool wystepuje=false;
while(j<1000 && (!wystepuje)){
   liczba = bin2wart(ciagi[j]);
   if (czy_polpierwsza(liczba)){
      wystepuje=true;
      min=max=liczba;
   } else j++;
}
```

W wyniku wykonania powyższego fragmentu programu zmienna `j` wskazuje na pierwszą liczbę półpierwszą (jeśli `j<1000`) lub wskazuje na brak liczb półpierwszych (gdy `j=1000`). Ponadto `wystepuje=true` dokładnie wtedy, gdy w ciągu znaleźliśmy liczbę półpierwszą.

Następnie przeglądamy kolejne liczby (począwszy od `j`-tej), stosując klasyczny algorytm znajdowania minimum (lub maksimum) w nieuporządkowanym ciągu liczb. Pamiętamy jednak o tym, że minimum i maksimum wybieramy tylko spośród liczb półpierwszych. Dlatego

też wartość kolejnej liczby porównywana jest z `min` i `max` tylko wtedy, gdy dana liczba jest półpierwsza. Poniżej prezentujemy fragment kodu realizujący powyżej opisany algorytm:

```
ile=0;
for(int i=j; i<1000; i++)
{
   liczba = bin2wart(ciagi[i]);
   if (czy_polpierwsza(liczba))
   {
      ile++;
      if (liczba>max) max = liczba;
      if (liczba<min) min = liczba;
   }
}
```

Po wykonaniu powyższego fragmentu programu zmienna `wystepuje` wskazuje, czy w pliku wystąpiła choć jedna liczba półpierwsza. Gdy `wystepuje=true`, zmienne `min` i `max` zawierają odpowiednio wartość najmniejszej liczby półpierwszej i wartość największej liczby półpierwszej. Ponadto w zmiennej `ile` wyznaczona zostanie ilość liczb półpierwszych.

Zadanie 64.

Do reprezentacji jednego obrazka używać będziemy tablicy dwuwymiarowej rozmiaru 21×21. Zdefiniujemy w tym celu typ *obrazek*:

```
#define RZM 20
typedef short obrazek[RZM+1][RZM+1];
```

Piksele obrazka będziemy przechowywać na pozycjach [*i*][*j*] dla $0 \le i,j < RZM$, bit parzystości wiersza $0 \le i < RZM$ umieścimy na pozycji [*i*][RZM], natomiast bit parzystości kolumny $0 \le j < RZM$ — na pozycji [RZM][*j*].

Stosując standardowe metody wczytywania danych z pliku tekstowego, możemy umieścić kolejne obrazki w tablicy elementów typu obrazek bądź kolejne obrazki wczytywać do tej samej zmiennej w celu ich sekwencyjnego przetwarzania.

Zauważmy, że każde z zadań sprowadza się do zliczenia liczby obrazków spełniających podaną własność lub znalezieniu obrazka ekstremalnego (minimalnego lub maksymalnego) względem pewnej własności. Dlatego w omówieniu skoncentrujemy się na przetwarzaniu pojedynczego obrazka.

64.1.

Do rozwiązania zadania przydatna będzie funkcja, która zlicza liczbę czarnych pikseli w obrazku:

```
int ileCzarnych(obrazek pobraz){
    int czarne=0;
    for(int i=0;i<RZM;i++){
        for(int j=0;j<RZM;j++){
            czarne+=pobraz[i][j];
        }
    }
    return czarne;
}
```

Zgodnie z treścią zadania obrazek jest rewersem, jeśli wartość zwracana przez funkcję `ileCzarnych` jest większa od $RZM^2/2=200$. Korzystając z funkcji `ileCzarnych`, sprowadzamy zadanie do znalezienia największej wartości `ileCzarnych` zwracanej dla obrazków z pliku wejściowego oraz zliczenia liczby obrazków mających więcej niż 200 czarnych pikseli.

64.2.

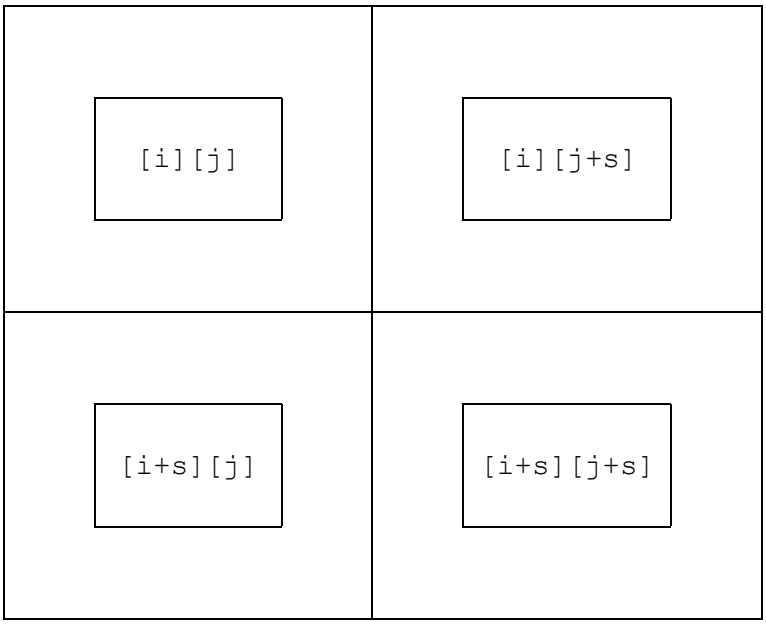
Rozważmy obrazek przechowywany w zmiennej `robr` typu `obrazek`. Obrazek taki jest rekurencyjny, gdy cztery „podobrazki" rozmiaru $\frac{n}{2}\times\frac{n}{2}$ o zakresach współrzędnych z poniższego rysunku są równe:

robr[0..RZM/2 – 1] [0..RZM/2–1]	robr[0..RZM/2–1] [RZM/2..RZM–1]
robr[RZM/2, RZM–1] [0..RZM/2–1]	robr[RZM/2, RZM–1] [RZM/2, RZM–1]

Własność powyższą możemy wyrazić inaczej: `robr` jest rekurencyjny, jeśli dla każdego $0\leq$`i,j`$<RZM/2$ zachodzi:

```
robr[i][j]=robr[i][j+s]=robr[i+s][j]=robr[i+s][j+s]
```

dla `s`=RZM/2.



Na tej obserwacji oparta jest podana poniżej funkcja `czyRek`, która zwraca wartość true dokładnie wtedy, gdy obrazek zapamiętany w zmiennej `robr` jest rekurencyjny:

```
bool czyRek(obrazek robr){
  int s=RZM/2;
  for(int i=0; i<RZM/2; i++)
    for(int j=0; j<RZM/2; j++){
      if (robr[i][j]!=robr[i+s][j]) return false;
      if (robr[i][j]!=robr[i][j+s]) return false;
      if (robr[i][j]!=robr[i+s][j+s])
        return false;
    }
  return true;
}
```

Mając do dyspozycji funkcję `czyRek`, zadanie 2 możemy sprowadzić do zliczania liczby elementów w ciągu spełniających podaną własność (czyli liczby obrazków rekurencyjnych) oraz wypisania pierwszego napotkanego elementu spełniającego tę własność (czyli pierwszego napotkanego obrazka rekurencyjnego).

64.3.

Aby rozwiązać zadanie 3, postaramy się najpierw ustalić liczbę błędnych bitów parzystości wierszy oraz liczbę błędnych bitów parzystości kolumn w obrazku. Zauważmy, że poprawna wartość bitu parzystości ciągu jest równa reszcie z dzielenia przez 2 sumy wszystkich elementów ciągu. Wykorzystując ten fakt, liczbę błędnych bitów parzystości wierszy w obrazku zapamiętanym w zmiennej `pobraz` typu `obrazek` możemy wyznaczyć przy pomocy następującego fragmentu programu (wynik w zmiennej `ileBlWiersz`):

```
int parzw, ileBlWiersz=0;
for(int i=0; i<RZM; i++){
    parzw=0;
    for(int j=0; j<RZM; j++)
        parzw=parzw+pobraz[i][j];
    if ((parzw%2)!=pobraz[i][RZM]) ileBlWiersz++;
}
```

Analogicznie możemy policzyć liczbę błędnych bitów parzystości kolumn (wynik w zmiennej `ileBlKol`):

```
int parzk, ileBlKol=0;
for(int i=0; i<RZM; i++){
    parzk=0;
    for(int j=0; j<RZM; j++)
        parzw=parzw+pobraz[j][i];
    if ((parzk%2)!=pobraz[RZM][i]) ileBlKol++;
}
```

Zgodnie z podanymi w treści zadania definicjami:

- obrazek jest poprawny, gdy `ileBlWiersz+ileBlKol==0;`
- obrazek jest naprawialny, gdy `ileBlWiersz≤1` oraz `ileBlKol≤1` (warto zwrócić uwagę, że warunek ten nie jest równoważny `ileBlWiersz+ileBlKol≤2`).

Ponieważ naszym zadaniem jest nie tylko wyznaczenie liczby obrazków poprawnych i naprawialnych, ale również ustalenie największej liczby błędnych bitów parzystości w obrazku, napiszemy funkcję `stanPoprawnosci` o nagłówku

```
int stanPoprawnosci(obrazek pobraz)
```

zwracającą:

- wartość 0, gdy `pobraz` jest poprawny;
- wartość 1, gdy `pobraz` jest naprawialny.
- liczbę błędnych bitów parzystości w przeciwnym przypadku.

Treść funkcji prezentujemy poniżej. Proces zliczania liczby błędnych bitów parzystości w wierszach i kolumnach zorganizowaliśmy trochę inaczej niż w powyższym opisie — realizujemy go w jednej pętli zagnieżdżonej, a nie w dwóch (prześledzenie szczegółów pozostawiamy czytelnikowi).

```
int stanPoprawnosci(obrazek pobraz){
    int parzw,parzk;
    int ileBlWiersz, ileBlKol;

    ileBlWiersz=0;
    ileBlKol=0;
    for(int i=0; i<RZM; i++){
        parzw=parzk=0;
        for(int j=0; j<RZM; j++){
            parzw=parzw+pobraz[i][j];
            parzk=parzk+pobraz[j][i];
        }
        if ((parzw%2)!=pobraz[i][RZM]) ileBlWiersz++;
        if ((parzk%2)!=pobraz[RZM][i]) ileBlKol++;
    }
    if (ileBlWiersz+ileBlKol==0) return 0;
    if (ileBlWiersz<=1 && ileBlKol<=1) return 1;
    return ileBlWiersz+ileBlKol;
}
```

Na koniec zwróćmy uwagę, że funkcja `stanPoprawnosci` zwraca wartość 1 zarówno dla tych obrazków naprawialnych, które mają tylko jeden błędny bit parzystości, jak i dla tych obrazków naprawialnych, w których występuje zarówno błąd w kolumnie, jak i w wierszu. Uruchamiając tę funkcję dla obrazków z pliku wejściowego, można się dowiedzieć, że w pliku są obrazki, które mają więcej błędów parzystości niż 2. Z tego powodu brak rozróżnienia między jednym a dwoma błędami przez funkcję `stanPoprawnosci` (dla obrazków naprawialnych) nie ma wpływu na ustalenie największej liczby błędnych bitów parzystości w jednym obrazku.

64.4.

Aby ustalić, które obrazki są naprawialne, możemy posłużyć się przedstawioną wcześniej funkcją `stanPoprawnosci`. Jeśli `stanPoprawnosci(pobraz)` zwróci wartość 1, `pobraz` jest naprawialny. Załóżmy więc, że dostajemy na wejściu obraz **naprawialny** (ale nie poprawny!) i powinniśmy jedynie ustalić pozycję (tj. numer wiersza i numer kolumny) wartości, którą wystarczy zmienić, aby uzyskać obrazek poprawny. W treści zadania opisany został sposób wyznaczenia tych numerów, podamy go jeszcze raz w odniesieniu do numeru wiersza, w nieco zmienio nej postaci:

(a) jeśli błąd parzystości wystąpił w wierszu i-tym, powinna zostać zmieniona wartość w wierszu i;

(b) jeśli w żadnym wierszu nie wystąpił błąd parzystości, należy zmienić wartość w ostatnim wierszu (tym, w którym umieszczone są bity parzystości kolumn).

Analogicznie można sformułować regułę dla numeru kolumny. Korzystając z tych własności, zaprogramujemy nasze rozwiązanie. Chcemy, aby zmienna `wynik` typu

```
struct korekta{
    int wiersz;
    int kolumna;};
```

na końcu przechowywała numer wiersza i numer kolumny pozycji, którą należy zmienić. Przy tym, zgodnie z treścią zadania, wynik będziemy reprezentować, przyjmując numerację wierszy/kolumn 1,2,3,..,21 a nie 0,1,2,..20. Zmienną `wynik` zainicjujemy tak, jakby w obrazku nie było żadnego błędu parzystości wiersza i kolumny:

```
    wynik.wiersz=RZM+1;
    wynik.kolumna=RZM+1;
```

Następnie w zagnieżdżonej pętli podobnej do tej, w której zliczaliśmy liczbę błędnych bitów parzystości (p. opis funkcji `stanPoprawnosci`), weryfikujemy poprawność kolejnych bitów parzystości wierszy i kolumn. Po napotkaniu pierwszego błędu w wierszu (lub kolumnie) zmieniamy odpowiednio wartość `wynik.wiersz` (lub `wynik.kolumna`). Brak błędu w wierszu (lub kolumnie) oznacza, że wartość `wynik.wiersz lub wynik.kolumna` pozostanie równa zainicjowanej wartości początkowej, ROZM+1, co jest zgodne z podanymi powyżej regułami (a) i (b). Poniżej podajemy pełną treść funkcji, która zwraca jako wynik strukturę typu `struct korekta`, opisującą pozycję w obrazku, którą należy zmienić:

```
struct korekta korygujObrazek(obrazek pobraz){
    int parzw,parzk;
    struct korekta wynik;

    wynik.wiersz=ROZM+1;
    wynik.kolumna=ROZM+1;
    for(int i=0; i<RZM; i++){
        parzw=parzk=0;
        for(int j=0; j<RZM; j++){
            parzw=parzw+pobraz[i][j];
            parzk=parzk+pobraz[j][i];
        }
        if ((parzw%2)!=pobraz[i][ROZM])
            wynik.wiersz=i+1;
        if ((parzk%2)!=pobraz[ROZM][i])
            wynik.kolumna=i+1;
    }
    return wynik;
}
```

Przypomnijmy, że powyższa funkcja nie sprawdza, czy obrazek jest naprawialny i dlatego rozwiązanie 4 uzyskamy, najpierw sprawdzając dla każdego obrazka, czy jest on naprawialny (opisana wcześniej funkcja `stanPoprawnosci`), a następnie uruchamiając funkcję `korygujObrazek` dla tych obrazków, które są naprawialne. Jako ćwiczenie pozostawiamy utwo-

rzenie rozwiązania, które integruje obie czynności (sprawdzanie naprawialności i znajdowanie pozycji, którą trzeba zmienić) bez dwukrotnego przeglądania zawartości obrazka.

Zadanie 65.

Stosując standardowe metody wczytywania danych z pliku tekstowego, możemy umieścić liczniki i mianowniki wszystkich ułamków w tablicach, na przykład:

```
int licz[1000], mian[1000];
```

Spostrzegawczy czytelnik zapewne zauważył, że wczytywanie danych do tablicy nie jest konieczne, gdyż w każdym z zadań wystarczy jednorazowo sekwencyjnie przejrzeć ciąg danych od początku do końca. Dla czytelności rozwiązania przyjmujemy jednak, że dane są wczytane do tablicy i każde z zadań rozwiązujemy osobno.

65.1.

Naturalnym sposobem rozwiązania zadania 1 wydaje się porównywanie **wartości** ułamków, zapamiętanych w zmiennych typu float lub double[6] w C/C++. Aby odróżnić ułamki o tej samej wartości, musimy wówczas porównywać ich mianowniki. Zadanie sprowadza się wówczas do zaimplementowania algorytmu szukania **minimum rozszerzonego**. Z dwóch ułamków o tej samej wartości, ale różnych mianownikach, za mniejszy uznajemy ten, który ma mniejszy mianownik.

Powyższe rozwiązanie obarczone jest jednak błędem zaokrągleń związanych z reprezentacją zmiennopozycyjną liczb. Może to prowadzić do sytuacji, w których porównanie dwóch ułamków o tej samej wartości nie wskaże na ich równość. Aby się przekonać o takiej możliwości, polecamy czytelnikowi sprawdzenie działania następującego fragmentu kodu programu:

```
double x,y,z,v, a,b;

x=1.0; y=3.0;
z=2.0; v=6.0;
b=x/y;
cout << "Roznica: " << z/v-b<< endl;
```

Dlatego w omawianym rozwiązaniu będziemy porównywać ułamki, sprowadzając je do wspólnego mianownika. Unikniemy wówczas operacji na liczbach w reprezentacji zmiennoprzecinkowej (float/double). Sprawdzenie, czy l_1 / m_1 jest mniejsze od (lub równe) l_2 / m_2, sprowadza się do sprawdzenia, czy $l_1 \cdot m_2$ jest mniejsze od (lub równe) $l_2 \cdot m_1$:

```
bool mniejszy(long l1, long m1, long l2, long m2){
    return (l1*m2<l2*m1);
}
bool rowny(long l1, long m1, long l2, long m2){
    return (l1*m2==l2*m1);
}
```

Ułamek o najmniejszej wartości i najmniejszym mianowniku nazwijmy *minimalnym*. Stosując powyższą funkcję, poszukujemy ułamka minimalnego w pętli, przechowując licznik i mianownik ułamka minimalnego spośród dotychczas przejrzanych w zmiennych `minL`

[6] Pamiętać przy tym należy, że np. w języku C podstawienie x=a/b dla zmiennych a i b typu int oraz x typu float da w wyniku zaokrąglenie w dół a/b do liczby całkowitej; aby uzyskać wynik dokładny, konieczne jest wymuszenie konwersji typu, np. przez zastąpienie instrukcji x=a/b przez x=1.0*a/b lub x=(float)a/(float)b.

i `minM`. Wartości te zmieniają się zawsze po napotkaniu ułamka o wartości mniejszej od `minL/minM` lub ułamka o wartości `minL/minM`, ale o mianowniku mniejszym od `minM`. Poniżej prezentujemy fragment kodu realizujący opisany sposób rozwiązania:

```
int minL, minM;

minL=licz[0];
minM=mian[0];
for(int i=1; i<1000; i++){
    if (mniejszy(licz[i],mian[i],minL,minM)){
        minL=licz[i]; minM=mian[i];
    }
    else if (rowny(licz[i],mian[i],minL,minM)){
            if (mian[i]<minM){
                minL=licz[i]; minM=mian[i];
            }
          }
}
```

65.2.

Wiadomo, że ułamek jest w postaci nieskracalnej wtedy i tylko wtedy, gdy największy wspólny dzielnik licznika i mianownika jest równy 1. Aby sprowadzić ułamek do postaci nieskracalnej, wystarczy zatem podzielić zarówno licznik, jak i mianownik ułamka przez ich największy wspólny dzielnik. Dlatego przydatnym narzędziem do rozwiązania zadania może być funkcja wyznaczająca największy wspólny dzielnik dwóch liczb. Poniżej podajemy taką funkcję, opartą na algorytmie Euklidesa:

```
int  nwd(int n, int m){

    if (m==0) return n;
    return nwd(m,n%m);
}
```

Przy skorzystaniu z powyższej funkcji zadanie sprowadza się do zliczenia liczby takich $i \in [0, 999]$, że `nwd(licz[i], mian[i])` jest równe 1. Poniżej podajemy przykładową implementację takiego zliczania:

```
int ileN=0;

for(int i=0; i<1000; i++){
    if (nwd(licz[i],mian[i])==1)
        ileN++;
}
```

Zwróćmy jednak uwagę, że zadanie to można rozwiązać, nawet nie wiedząc o zależności między największym wspólnym dzielnikiem licznika i mianownika a nieskracalnością ułamka. Dla ułamka o postaci a / b wystarczy sprawdzić dla każdej z liczb $i \in [2,\min(a,b)]$, czy i dzieli zarówno a, jak i b. Poniżej prezentujemy fragment takiego rozwiązania:

```
int mniejsza, j, ileN=0;
bool czyN;
```

```
for(int i=0; i<1000; i++){
    if (licz[i]<mian[i]) mniejsza = licz[i];
    else mniejsza = mian[i];
    j=2; czyN = true;
    while (j<=mniejsza){
        if ((licz[i]%j==0) && (mian[i]%j==0))
            czyN=false;
            j++;
    }
    if (czyN) ileN++;
}
```

Po wykonaniu powyższego kodu liczba ułamków w postaci nieskracalnej znajdzie się w zmiennej `ileN`.

Zwróćmy jeszcze na koniec uwagę na to, że powyższy algorytm można znacznie przyspieszyć, kończąc pętlę `while` po znalezieniu pierwszego wspólnego dzielnika licznika i mianownika, a więc wtedy, gdy wartość zmiennej `czyn` zmieni się na `false`.

65.3.

Zauważmy, że do uzyskania nieskracalnej postaci ułamka a / b wystarczy podzielić a oraz b przez `nwd(a,b), tj. przez` największy wspólny dzielnik a i b. Inaczej mówiąc, licznik nieskracalnej postaci to `a/nwd(a,b)`, a mianownik to `b/nwd(a,b)`. Na przykład

$$\frac{8}{12} = \frac{8/\text{nwd}(8,12)}{12/\text{nwd}(8,12)} = \frac{2}{3}$$

Naturalnym rozwiązaniem zadania jest więc wyznaczenie sumy wartości `licz[i]/nwd(licz[i],mian[i])` dla $i \in [0,999]$. Poniżej przykładowa implementacja takiego rozwiązania, gdzie wynik umieszczany jest w zmiennej `sumL`:

```
long d, sumL=0;
for(int i=0; i<1000; i++){
    d=nwd(licz[i],mian[i]);
    sumL = sumL + licz[i]/d;
}
```

65.4.

Wiedząc, że każdy mianownik jest dzielnikiem liczby b, możemy wszystkie ułamki przekształcić do postaci, w której mianownik jest równy b. Dokładniej, korzystamy z tożsamości: $\frac{x}{y} = \frac{x \cdot b / y}{b}$. Zadanie sprowadza się wówczas do zsumowania wartości `licz[i]·b/mian[i]` dla $i \in [0,999]$. Przy takim rozwiązaniu pojawia się jednak pewna pułapka, polegająca na tym, że niektóre wartości `licz[i]·b` dla pewnych i przekraczają zakres liczb typu long w popularnych kompilatorach C/C++. Dlatego konieczne jest użycie typu long long. Poniżej przykładowy fragment takiego rozwiązania, w którym `liczS/b` jest równe sumie ułamków `licz[0]/mian[0],…,licz[i]/mian[i]`:

```
long long mianS, liczS, nowyLicz;
liczS=0;
mianS=2*3*5*7*13;
```

```
        mianS=mianS*mianS/13;
        for(int i=0; i<ile; i++){
            nowyLicz = licz[i]*mianS/mian[i];
            liczS+= nowyLicz;
        }
```

Zadanie 66.

We wszystkich zadaniach będziemy czytać dane w pętli przebiegającej kolejne wiersze pliku i sprawdzać, czy liczby w bieżącym wierszu spełniają warunki podane w treści zadania. Zakładając, że dane pobierane są ze standardowego wejścia, pętlę wczytującą liczby można zapisać następująco:[7]

```
int i, liczba1, liczba2, liczba3;
 for (i=1; i<=1000; i++)
 {
  scanf("%i %i %i\n",&liczba1, &liczba2, &liczba3);
 }
```

66.1.

Aby znaleźć trójki liczb spełniające warunek opisany w zadaniu, potrzebujemy funkcji, która obliczy sumę cyfr liczby. Stosując operację obliczania reszty z dzielenia, możemy wydzielić kolejne cyfry i zsumować je. Posłuży do tego poniższa funkcja:

```
    int suma_cyfr(int a)
    {int suma=0;
     while (a>0)
          {suma=suma+a%10;a=a/10;}
     return(suma);
    }
```

Po sprawdzeniu, że suma cyfr pierwszych dwóch liczb w wierszu jest równa trzeciej liczbie, wypisujemy trójkę liczb na wyjściu:

```
if ((suma_cyfr(liczba1)+suma_cyfr(liczba2))==liczba3)
     printf("%i %i %i\n",liczba1, liczba2, liczba3);
```

66.2.

W zadaniu trzeba sprawdzać spełnienie następujących dwóch warunków dla każdego wiersza pliku: czy iloczyn dwóch pierwszych liczb w wierszu jest równy trzeciej liczbie i czy dwa pierwsze elementy wiersza są liczbami pierwszymi. Do sprawdzenia, czy liczba jest pierwsza, użyjemy poniższej funkcji:

```
bool pierwsza(int x)
{ if (x<2) return false;
  for (int k=2; k*k<=x; k++)
     if (x%k==0) return false;
  return true;
}
```

[7] Czytelnik powinien zapoznać się ze sposobami umożliwiającymi potraktowanie zawartości ustalonego pliku jako standardowego wejścia.

Do wypisania interesujących nas wierszy użyjemy zagnieżdżonych instrukcji warunkowych, w których najpierw sprawdzamy pierwszy warunek i dopiero gdy jest on spełniony, przechodzimy do sprawdzenia, czy dwa pierwsze elementy wiersza są liczbami pierwszymi.

```
if ((liczba1*liczba2)==liczba3)
 {if (pierwsza(liczba1)&&pierwsza(liczba2))
    printf("%i %i %i\n",liczba1, liczba2, liczba3);}
```

66.3.

Do sprawdzenia, czy trójka liczb umieszczona w wierszu reprezentuje długości boków trójkąta prostokątnego, użyjemy zależności wynikającej z twierdzenia Pitagorasa. Zwróćmy uwagę, że w zadaniu nie ustalono kolejności liczb w wierszu (w szczególności tego, czy są one uporządkowane). Sprawdzimy zatem trzy warunki, w każdym z nich przyjmujemy inną liczbę jako długość przeciwprostokątnej. Zapiszemy to w postaci funkcji:

```
int czy_prostokatny(int a, int b, int c)
{
return(((a*a+b*b)==c*c)||((a*a+c*c)==b*b)||((c*c+b*b)==a*a));
}
```

W zadaniu należy znaleźć pary sąsiadujących ze sobą wierszy, które spełniają opisany powyżej warunek. Dlatego w pętli wprowadzimy pomocnicze zmienne `pop1`, `pop2`, `pop3` do przechowywania wartości liczb z poprzedniego wiersza oraz zmienną `poprzedni`, w której pamiętamy wynik działania funkcji sprawdzającej, czy liczby `pop1`, `pop2`, `pop3` są bokami trójkąta prostokątnego. W każdym obrocie pętli czytającej dane z pliku wejściowego wykonujemy następujący ciąg instrukcji:

```
aktualny=czy_prostokatny(liczba1, liczba2, liczba3);
if (poprzedni&&aktualny)
 {printf("%i %i %i\n",pop1, pop2, pop3);
  printf("%i %i %i\n\n",liczba1, liczba2, liczba3);}
poprzedni=aktualny;
pop1=liczba1; pop2=liczba2; pop3=liczba3;
```

Aby uwzględnić to, że pierwszy wiersz pliku nie jest poprzedzony innym wierszem, zmiennej `poprzedni` początkowo nadajemy wartość zero. Zmienna `aktualny` przechowuje wynik działania funkcji `czy_prostokatny` dla liczb z bieżącego wiersza. Wiersz bieżący i poprzedni wypisujemy, gdy zmienne `aktualny` i `poprzedni` mają wartość 1.

66.4.

Najpierw utworzymy funkcję, która sprawdza, czy podane liczby a, b i c spełniają warunek trójkąta.

```
int czy_trojkat(int a, int b, int c)
{
    return((a+b>c)&&(b+c>a)&&(a+c>b));
}
```

Następnie w pętli czytamy kolejne trójki liczb z pliku wejściowego i jednocześnie:

- zwiększamy zmienną `licznik` (o początkowej wartości 0) przy każdym wierszu, w którym liczby spełniają warunek trójkąta;

- aktualizujemy największą długość ciągu trójkątnego, na końcu którego znajduje się bieżący wiersz (przyjmujemy przy tym, że jeśli liczby w bieżącym wierszu nie spełniają nierówności trójkąta, długość ciągu trójkątnego kończącego się na bieżącym wierszu jest równa zero); służy do tego zmienna `dlugosc`, która początkowo przyjmuje wartość zero, a następnie jest zerowana, gdy napotkamy koniec ciągu trójkątnego (czyli wiersz, w którym liczby nie spełniają warunku trójkąta), a zwiększana o 1, gdy napotkamy wiersz, w którym liczby spełniają warunek trójkąta.

Ponadto zmienna `max` (początkowo równa 0) służy do przechowania aktualnie największej długości ciągu trójkątnego. Poniższy fragment kodu realizuje opisane powyżej modyfikacje w sytuacji, gdy `liczba1`, `liczba2` i `liczba3` to elementy bieżącego wiersza:

```
if (czy_trojkat(liczba1, liczba2, liczba3))
    {dlugosc++; licznik++;}
else
     dlugosc=0;
if (dlugosc>max)max=dlugosc;
```

Zadanie 67.

Zestaw zadań rozwiążemy, pisząc program komputerowy, natomiast obraz fraktala w postaci białych i czarnych kwadratów utworzymy w arkuszu kalkulacyjnym.

67.1.

Wartości pierwszych 40 liczb Fibonacciego wygenerujemy za pomocą następującego programu:

```
const int n = 40;
    long int f1 = 1, f2 = 1, f;
    for (int i = 3; i <= n; i++) {
        f = f1 + f2;
        f1 = f2;
        f2 = f;
        if (i%10 == 0) cout << i << "\t" << f << endl;
}
```

Wypisujemy tylko F_{10}, F_{20}, F_{30}, F_{40}, czyli takie liczby, których indeksy są podzielne przez 10.

67.2.

Liczby pierwsze w ciągu liczb Fibonacciego F_1, F_2, … , F_{40} znajdziemy za pomocą prostej funkcji:

```
bool pierwsza(long a) {
   if (a==1) return false;
   if (a==2 || a==3) return true;
   for (long p=2; p<=sqrt((double)a); p++){
      if (a%p == 0) return false;
}
    return true;
}
```

Dla liczb mniejszych lub równych 3 funkcja wypisuje od razu prawidłowy wynik. Dla liczb większych niż 3 sprawdza, czy istnieją dzielniki liczby z zakresu od 2 do pierwiastka z tej

liczby. Jeśli napotka taki dzielnik, wynikiem funkcji jest FAŁSZ, w przeciwnym wypadku wynikiem jest PRAWDA.

Mając zapisaną funkcję *pierwsza*, możemy sprawdzać, czy liczba F jest liczbą pierwszą od razu po jej wygenerowaniu.

67.3.

Przy tworzeniu binarnego fraktala Fibonacciego czeka nas więcej pracy. Najpierw przekonwertujemy liczby Fibonacciego zapisane w systemie dziesiętnym na system dwójkowy, pamiętając o zachowaniu jednakowej długości wszystkich zapisów binarnych (dodaniu w odpowiednich miejscach zer wiodących).

Binarny zapis otrzymamy, korzystając z funkcji *bin*:

```
string bin(long a) {
    string b = "";
    while (a>0) {
        if (a%2 == 0) b = "0"+ b; else b = "1" + b;
    a /= 2;
    }
    return b;
}
```

Zapis binarny uzupełniamy wiodącymi zerami do długości zapisu binarnego F_{40}, czyli jeżeli *m* jest długością zapisu binarnego F_{40}, to:

```
while (b.length()<m) b = "0" + b;
```

Reprezentacja graficzna przekonująco wygląda w postaci czarnych i białych kwadracików. Aby ją uzyskać, wczytujemy binarne zapisy do arkusza (każdą cyfrę do osobnej komórki), a następnie korzystamy z formatowania warunkowego. Dla komórek zawierających wartość 1 ustawiamy tło czarne, a dla komórek zawierających wartość 0 — tło białe.

67.4.

Przechodzimy pętlą zewnętrzną przez wszystkie zapisy binarne, a pętlą wewnętrzną przez kolejne cyfry zapisu binarnego, zliczając dla każdego zapisu liczbę jedynek. Jeżeli ta liczba w pojedynczym zapisie wynosi dokładnie 6, dostajemy pozycję takiego zapisu binarnego w tablicy.

```
for (int i=1; i<=n; i++) {
       int licz1 = 0;
        for (int j=0; j<m; j++)
            if (b[j]=='1') licz1++;
        if (licz1 == 6) cout>>i>>endl;
       }
```

Na koniec wypisujemy zapisy binarne o indeksach i=14, i=21, i=24, i=25.

Zadanie 68.

Napisy umieścimy w tablicach, których elementy reprezentują typ string:

```
string s1[1000], s2[1000];
```

w ten sposób, że pierwszy napis z i-tego wiersza zostanie zapisany jako `s1[i]`, a drugi napis z i-tego wiersza jako `s2[i]`.

68.1.

Zauważmy, że napisy jednolite `x` i `y` są anagramami dokładnie wtedy, gdy `x` jest równe `y`. Zatem w zadaniu 1 wystarczy zliczyć liczbę takich i, dla których `s1[i]==s2[i]` oraz `s1[i]` jest napisem jednolitym. Wykorzystamy w tym celu poniższą funkcję:

```
bool jednolity(string s)
{  int d=s.length();
   for(int i=1; i<d; i++)
        if (s[i]!=s[i-1]) return false;
    return true;
}
```

Korzystając z funkcji `jednolity`, wynik uzyskamy w zmiennej `ileZ1` po wykonaniu poniższego fragmentu programu:

```
int ileZ1=0;
for (int i=0; i<1000; i++)
    if (jednolity(s1[i]))
        if (s1[i]==s2[i]) ileZ1++;
```

68.2.

Zauważmy, że napisy `x` i `y` są anagramami dokładnie wtedy, gdy liczba wystąpień każdej z liter A, B, C, D, E, F, G, H, I, J jest w obu napisach taka sama. Aby wykorzystać tę obserwację, utworzymy funkcję `zamien`, której celem jest zliczenie liczby wystąpień każdej litery w podanym słowie i zapamiętanie wyników w tablicy typu

```
int t[10];
```

w taki sposób, że liczba wystąpień A znajdzie się w `t[0]`, liczba wystąpień B znajdzie się w `t[1]` itd.

```
void zamien(string slowo, int t[10])
{   int d=slowo.length();
    for(int i=0; i<10; i++) t[i] = 0;
    for(int i=0; i<d; i++) t[slowo[i] - 'A']++;
}
```

Jeśli w tablicach `int tx[10]` i `ty[10]` zebraliśmy informacje o liczbie wystąpień liter w napisach `x` oraz `y`, to wówczas możemy sprawdzić, czy słowa `x` i `y` są anagramami, wykonując wywołanie `czyAnag(tx, ty)` funkcji `czyAnag` zdefiniowanej następująco:

```
bool czyAnag(int tx[10], int ty[10])
{
    for(int i=0; i<10; i++)
        if (tx[i]!=ty[i]) return false;
    return true;
}
```

Korzystając z funkcji `zamien`, chcemy zapamiętać informacje o liczbie wystąpień poszczególnych liter w podanych napisach w tablicach:

```
int c1[1000][10], c2[1000][10];
```

w taki sposób, że `c1[i][0], c1[i][1], …, c1[i][9]` będą reprezentowały liczby wystąpień liter w `s1[i]`; podobnie `c2[i][0], c2[i][1], …, c2[i][9]` mają reprezentować liczby wystąpień liter odpowiednio w `s2[i]`. Odpowiedź do zadania 2 uzyskamy wówczas w zmiennej `ileZ2` po wykonaniu następującego fragmentu programu:

```
int ileZ2 = 0;
for (int i=0; i<1000; i++)
{  zamien(s1[i],c1[i]);
   zamien(s2[i],c2[i]);
   if (czyAnag(c1[i],c2[i])) ileZ2++;
}
```

68.3.

Przyjmijmy, że informacje o liczbie wystąpień poszczególnych liter w słowach z pliku wejściowego zapisane są w tablicach `c1` i `c2`, tak jak opisaliśmy to w rozwiązaniu zadania 2. Zadanie 3 możemy wówczas rozwiązać, zliczając dla każdego `i` z przedziału [0,999]:

- liczbę takich j z zakresu [0,999], że `s1[i]` i `s1[j]` są anagramami lub `s1[i]` i `s2[j]` są anagramami;
- liczbę takich j z zakresu [0,999], że `s2[i]` i `s1[j]` są anagramami lub `s2[i]` i `s2[j]` są anagramami.

Następnie, wybierając największą z tak wyznaczonych wartości, uzyskamy rezultat wymagany w zadaniu 3. Stwórzmy w tym celu funkcję pomocniczą, która wyznaczy liczbę anagramów dla napisu, którego statystyki wystąpień liter umieściliśmy w tablicy `c`:

```
int zad3Pom(int c[10], int liczbaslow)
{       int ile=0,j;
        for(j=0; j< liczbaslow; j++){
            if (czyAnag(c,c1[j])) ile++;
            if (czyAnag(c,c2[j])) ile++;
        }
        return ile;
}
```

Dokładniej, powyższa funkcja wyznacza liczbę takich napisów wśród `s1[0],..,s1[999]` i `s2[0],..,s2[999]`, dla których funkcja `zamien` (patrz opis rozwiązania zadania 2) da w wyniku tablicę o takiej samej zawartości jak tablica `c`.

Korzystając z funkcji `zad3Pom`, rozwiązanie zadania uzyskamy jako wartość poniższej funkcji:

```
int zad3()
{   int i, ilePowt, maxPowt=0;
    string s;
    int i, j;

    for(i=0; i<1000; i++){
        ilePowt=zad3Pom(c1[i],1000);
        if (ilePowt>maxPowt) maxPowt=ilePowt;
        ilePowt=zad3Pom(c2[i],1000);
        if (ilePowt>maxPowt) maxPowt=ilePowt;
    }
    return maxPowt;
}
```

Przedstawione powyżej rozwiązanie można nieco usprawnić. Dla każdego $i \in [0,999]$ wystarczy sprawdzać liczbę anagramów słów `s1[i]` i `s2[i]` jedynie wśród `s1[0],…,s1[i]` oraz `s2[0],…,s2[i]`. Pozostawiamy czytelnikowi zastanowienie się nad tym, dlaczego nie zmieni to wyniku naszych obliczeń.

W przypadku bardzo obszernych danych można się pokusić o kolejne usprawnienie. (Zaznaczmy jednak, że rozmiar danych w treści zadania nie wymaga stosowania metod omówionych w dalszej części opisu.) Potraktujmy tablice `c1` i `c2` jako ciągi 10-elementowe, które mogą być porównywane leksykograficznie. Po posortowaniu ciągu `c1` (lub `c2`) zgodnie z porządkiem leksykograficznym wszystkie anagramy konkretnego napisu występujące na pierwszych pozycjach w wierszach reprezentowane są jako ciąg kolejnych elementów (o tej samej wartości) w tablicy `c1`. Analogiczna własność zachodzi dla anagramów konkretnego napisu występujących na drugich pozycjach w ciągu i tablicy `c2`. Pozostawiamy czytelnikowi zastanowienie się, jak z tych obserwacji można skorzystać, aby uzyskać szybki algorytm rozwiązujący zadanie 3.

Zadanie 69.

Przyjmijmy, że genotypy wczytane zostały już do tablicy stringów:

```
#define ROZ 1000
string genotypes[ROZ];
```

W opisie rozwiązania stosujemy notację a[i..j], która oznacza ciąg a[i], a[i+1], …, a[j].

69.1.

Z opisu danych wejściowych wiemy, że długość genotypu jest nie większa niż 500. Aby rozwiązać zadanie, utworzymy tablicę liczników

```
#define MAXL 500
int licz[MAXL+1];
```

W zmiennej `licz[i]` dla każdego i∈[1,500] wyznaczymy liczbę wystąpień genotypów o długości i, wcześniej inicjując wartości w tablicy `licz` zerami:

```
for(int i=1; i<=MAXL; i++) licz[i]=0;
for(int i=0; i<ROZ; i++){
    licz[genotypes[i].length()]++;
}
```

Po wykonaniu powyższego fragmentu programu liczba wartości różnych od zera wśród `licz[1..500]` będzie równa liczbie gatunków. Natomiast największa liczba osobników jednego gatunku będzie równa największej liczbie wśród `licz[1], licz[2], …, licz[500]`. Obserwacje te wykorzystujemy w poniższym fragmencie programu:

```
ileg=najw=0;
for(int i=1; i<=MAXL; i++){
    if (licz[i]>0) ileg++;
    if (licz[i]>najw) najw=licz[i];
}
```

Zmienne `ileg` i `najw` odpowiadają po zakończeniu pętli `for` odpowiednio liczbie niezerowych wartości wśród `licz[1], licz[2], …, licz[500]`(liczbie gatunków) oraz największej liczbie wśród `licz[1], licz[2], …, licz[500]` (największej liczbie osobników tego samego gatunku).

69.2.

Zauważmy, że w zadaniu nie wystarczy sprawdzić, w ilu napisach z tablicy `genotypes` występuje fragment BCDDC. Mutacja pojawia się tylko wówczas, gdy fragment BCDDC występuje w genie. Musimy zatem odróżniać w każdym genotypie fragmenty należące do genów od pozostałej części. Z uwagi na to, że również w zadaniach 3 i 4 potrzebne jest odróżnienie fragmentów genotypu tworzących geny od części niekodującej, napiszemy funkcję wyznaczającą kolejne geny danego genotypu. Funkcja ta będzie miała nagłówek:

```
Gen nextGen(string s, int i)
```

gdzie typ `Gen` zdefiniowany jest następująco

```
typedef struct{
    int bg;
    int end;
}Gen;
```

Oznaczmy długość napisu `s`, będącego parametrem funkcji `nextGen`, przez `len`. Chcemy, aby funkcja `nextGen` zwracała jako wartość strukturę typu `Gen`, taką że pierwszy gen w napisie `s[i]s[i+1]… s[len-1]` zaczyna się na pozycji `bg`, a kończy na pozycji `end`. W przypadku gdy w napisie `s[i]s[i+1]… s[len-1]` nie ma genu, ustalimy wartość `bg` na `len` (czyli długość `s`). Poniżej podajemy kod funkcji `nextGen`:

```
Gen nextGen(string s, int i)
{   int len=s.length();
    Gen pom;
    pom.bg=pom.end=len;
    while (i<len-4 && (s[i]!='A' || s[i+1]!='A'))
        i++;
    if (i>len-4) // za mało miejsca na cały gen
        return pom;
    pom.bg=i;
    while (i<len-1 && (s[i]!='B' || s[i+1]!='B'))
        i++;
    if (i==len-1) // za mało miejsca na cały gen
        { pom.bg=len; return pom;}
    pom.end=i+1;
    return pom;
}
```

Celem pierwszej pętli w funkcji `nextGen` jest znalezienie początku genu (AA). Natomiast druga pętla służy ustaleniu pozycji końca genu (BB).

Do sprawdzenia, czy w genie zaczynającym się w napisie `s` na pozycji `bg` i kończącym na pozycji `end` występuje podana mutacja, użyjemy poniższej funkcji:

```
bool czyMutacjaGen(string s, int bg, int end)
{   string mut="BCDDC";
    int len = mut.length();
    while (bg<=end-len+1) {
        if (mut==s.substr(bg,len))
            return true;
        bg++;
    }
    return false;
}
```

Aby sprawdzić, czy mutacja występuje w genotypie, wyszukujemy kolejne występujące w nim geny i do każdego z nich stosujemy funkcję `czyMutacjaGen`. Czynimy to tak długo, aż napotkamy gen z mutacją lub sprawdzimy wszystkie geny.

```
bool czyMutacjaGenotyp(string s)
{   int lenMut= mut.length();
    Gen  pom;
    bool czyM=false;
    pom.bg=0;
    int i=0;
    do{
        pom = nextGen(s,pom.bg);
        if (czyMutacjaGen(s,pom.bg,pom.end))
            return true;
        pom.bg=max(pom.bg,pom.end)+1;
    } while(pom.bg<s.length());
    return false;
}
```

Liczbę osobników z podaną mutacją możemy wyznaczyć, zliczając liczbę napisów g wśród genotypes[0…999], takich że czyMutacjaGenotyp(g) zwraca wartość true.

Na koniec zaznaczmy, że efektywniejszym czasowo rozwiązaniem zadania 2 byłoby sprawdzenie, gdzie występują geny oraz czy mutacja występuje w genach w jednym przebiegu przez kolejne znaki genotypu. Zaprezentowaliśmy rozwiązanie wykorzystujące funkcję wyznaczającą kolejne geny, gdyż funkcja taka będzie przydatna także w kolejnych zadaniach.

69.3.

Do zliczenia liczby genów w jednym genotypie zastosujemy funkcję `nextGen`:

- do wyznaczenia pierwszego genu w genotypie `g` uruchamiamy `nextGen(g,0);`
- po znalezieniu genu zaczynającego się na pozycji `bg` i kończącego na pozycji `end` następnego genu szukamy, wywołując `nextGen(g,end+1)`.

Wykorzystamy też fakt, że w przypadku braku genu w części napisu `g` zaczynającej się na pozycji `i` funkcja `nextGen(g,i)` zwróci strukturę typu `Gen`, której pole `bg` jest równe długości napisu `g`. Poniżej prezentujemy funkcję `ileGenow`, zliczającą liczbę genów w napisie `s`:

```
int ileGenow(string s)
{   Gen pom;
    int len=s.length();
    int i=0;
    int ileg=0;
    while (i<s.length()){
        pom=nextGen(s,i);
        if (pom.bg<len) ileg++;
        i=pom.end+1;
    }
    return ileg;
}
```

Aby ustalić największą liczbę genów u jednego osobnika, wyznaczymy maksymalną wartość powyższej funkcji dla słów `genotypes[0...999]`. W podobny sposób można wykorzystać funkcję `nextGen` do utworzenia funkcji o nagłówku

```
int najdluzszyGen(string s)
```

wyznaczającej największą długość genu w napisie `s`. Implementację takiej funkcji pozostawiamy czytelnikowi. Korzystając z tej funkcji, największą długość genu w całym pliku, wyznaczymy, znajdując największą liczbę wśród liczb `najdluzszyGen(genotypes[i])` dla $i \in [0,999]$.

69.4.

Aby ustalić, czy dany genotyp jest odporny, napiszemy funkcję `czescKodujaca`, która zwraca jako wynik część kodującą genotypu `s`:

```
string czescKodujaca(string s)
{   Gen pom;
    int len=s.length();
    string rob="";
    int i=0;
    while (i<len){
        pom=nextGen(s,i);
        if (pom.bg<len)
          rob=rob+s.substr(pom.bg,pom.end-pom.bg+1);
        i=pom.end+1;
    }
    return rob;
}
```

Aby porównać część kodującą genotypu czytanego od strony lewej do prawej i czytanego od strony prawej do lewej, potrzebujemy jeszcze funkcji „odwracającej" napis:

```
string odwroc(string s)
{
    string rob="";
    int len=s.length()
    for(int i=0;i<len;i++)
        rob=s[i]+rob;
    return rob;
}
```

Jeżeli dysponujemy powyższymi funkcjami, ustalenie, czy `genotypes[i]` jest odporny, sprowadza się do sprawdzenia prawdziwości warunku

```
czescKodujaca(genotypes[i])==czescKodujaca(odwroc(genotypes[i]))
```

W efekcie zadanie możemy rozwiązać, zliczając liczbę napisów `genotypes[i]` dla $i \in [0,999]$ spełniających powyższy warunek oraz liczbę palindromów wśród `genotypes[0..999]`.

Zadanie 70.

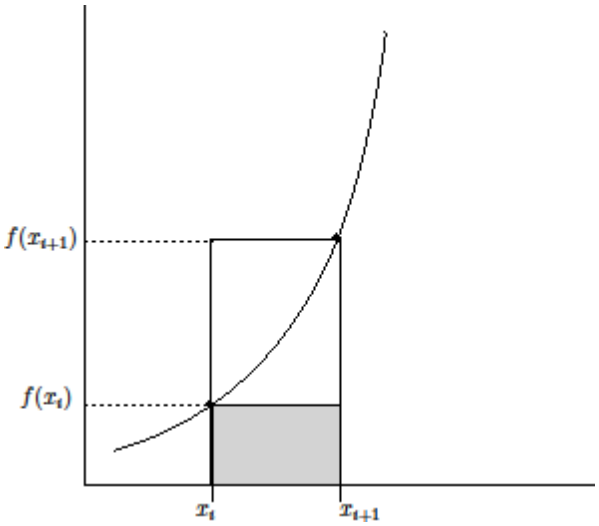
W treści nie występuje polecenie „Napisz program", zatem zadanie można wykonać zarówno w arkuszu kalkulacyjnym, jak i za pomocą programu komputerowego.

70.1.

Zadanie sprowadza się do obliczenia sumy dwóch pól: pola powierzchni pomiędzy wykresem funkcji $f(x)$ i osią X oraz pola pomiędzy wykresem funkcji $g(x)$ i osią OX. Uwzględniając fakt, że $f(x)$ jest rosnąca w przedziale [2,10], pole obszaru ograniczonego osią OX oraz wykresem funkcji $f(x)$ w zadanym przedziale możemy szacować z góry oraz z dołu przy pomocy pól obszarów złożonych z prostokątów. Przedział [2, 10] dzielimy na n równych części. Oznaczmy prawe końce tych części przez x_i (i = 1, ..., n), przyjmijmy też x_0=2.

W części ograniczonej punktami x_i i x_{i+1}budujemy (p. rysunek):

- przy ograniczeniu z dołu: prostokąt o bokach $h = \frac{10-2}{n} = \frac{8}{n}$ i $f(x_i)$;
- przy ograniczeniu z dołu: prostokąt o bokach $h = \frac{10-2}{n} = \frac{8}{n}$ i $f(x_{i+1})$.



Ograniczenie z dołu na pole obszaru pomiędzy wykresem $f(x)$ i osią OX jest równe:

$$P_d=h\cdot(f(x_0)+f(x_1)+\ldots+f(x_{n-1}))$$

a ograniczenie z góry na pole obszaru pomiędzy wykresem $f(x)$ i osią OX jest równe:

$$P_g=h\cdot(f(x_1)+f(x_2)+\ldots+f(x_n)).$$

Przybliżając pole przez $(P_d+P_g)/2$, wymaganą dokładność uzyskamy, gdy $P_g - P_d \leq 1/1000$. Analogiczny algorytm możemy zastosować do obliczenia pola pomiędzy wykresem funkcji $g(x)$ a osią OX. Ponieważ funkcja $g(x)$ przyjmuje wartości ujemne, boki i-tego prostokąta są równe $h = \frac{10-2}{n} = \frac{8}{n}$ oraz $-g(x_i)$ (przy szacowaniu pola od dołu) lub $-g(x_{i+1})$.

Pamiętać przy tym należy, że docelowy wynik stanowi suma obu pól. Aby błąd wyniku był mniejszy niż 0.001, wyznaczymy oba pola z dokładnością $\frac{1}{2} \cdot 0.001$.

Pisząc program komputerowy, musimy też uważnie zaprogramować dwie funkcje: (`f()` oraz `g()`), które będą wykorzystywane przy wyznaczaniu pola. W szczególności, jeśli używamy C++, należy pamiętać o tym, że wynikiem dzielenia liczb całkowitych jest liczba całkowita (część ułamkowa wyniku jest odrzucana), co może prowadzić do bardzo niedokładnych obliczeń. Stąd we fragmentach kodu podanych poniżej stosujemy np. zapis `3.0/250`, a nie `3/250.`

```
double f(double x)
{
   return (x*x*x*x/500-x*x/200-3.0/250);
}

double g(double x)
{
   return (-x*x*x/30+x/20+1.0/6);
}
```

Obliczając pole ograniczone przez krzywą $f(x)$ lub $g(x)$, testować będziemy różne wartości `n` w opisanej powyżej metodzie, zaczynając od `n=10` i zwiększając `n` dwukrotnie w kolejnych krokach. Wynik uznamy za poprawny, gdy różnica między oszacowaniem pola z góry i oszacowaniem pola z dołu będzie mniejsza od wartości zmiennej `blad`. Algorytm ten realizuje podana poniżej funkcja `pole`. W funkcji tej w zmiennych `pfgora` i `pfdol` wyznaczamy przybliżone ograniczenia z góry i z dołu na pole pomiędzy osią OX a wykresem $f(x)$

w przedziale [*a*, *b*]. Zgodnie z wcześniejszym opisem po uzyskaniu `pfgora` i `pfdol` różniących się o mniej niż `blad` za końcowe pole przyjmujemy średnią arytmetyczną z wartości `pfgora` i `pfdol`. Analogicznie wyznaczamy pole pomiędzy osią OX a wykresem *g*(*x*), pamiętając o tym, że funkcja *g*(*x*) przyjmuje w rozważanym przedziale wartości ujemne.

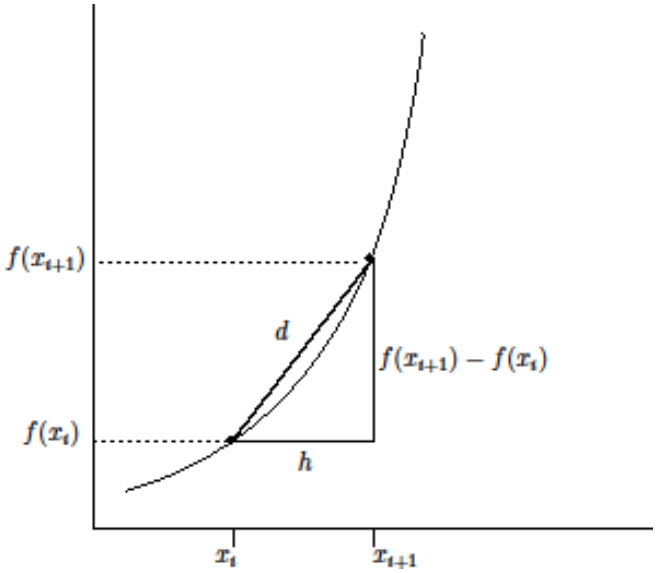
```
double pole(double a, double b, double blad)
{
double pfgora=0, pfdol=0, pf;
double pggora=0, pgdol=0, pg;
double x,h;
int n=10;
//pole f
do{
  h=(b-a)/n; x=a;
  pfdol=pfgora=0;
  for(int i=0; i<n; i++)
  {
      pfdol=pfdol+f(x);
      x=x+h;
      pfgora=pfgora+f(x);
  }
  pfgora=pfgora*h;
  pfdol=pfdol*h;
  n=2*n;
}while (pfgora-pfdol>blad/2);
pf=(pfgora+pfdol)/2;
//pole g
x=a; n=10;
do{
  h=(b-a)/n; x=a;
  pgdol=pggora=0;
  for(int i=0; i<n; i++)
  {
      pgdol=pgdol-g(x);
      x=x+h;
      pggora=pggora-g(x);
  }
  pggora=pggora*h;
  pgdol=pgdol*h;
  n=2*n;
} while (pggora-pgdol>blad/2);
pg=(pggora+pgdol)/2;
return (pf+pg);
}
```

Aby uzyskać wymagany rezultat, funkcję `pole(a,b,blad)` wywołujemy dla *a* = 2, *b* = 10 oraz `blad=0.001`.

70.2.

Zgodnie z opisem podanym w treści zadania sumujemy długości odcinków o końcach $(x_i, f(x_i))$ i $(x_{i+1}, f(x_{i+1}))$ dla $i = 0, 1, \ldots, 999$, gdzie $x_0 = 2$ oraz $x_i = 2+8i/1000$.

Długość odcinka o końcach $(x_i, f(x_i))$ i $(x_{i+1}, f(x_{i+1}))$ wyznaczamy, korzystając z twierdzenia Pitagorasa (zob. rysunek):



Następnie w pętli sumujemy długości kolejnych odcinków. Pamiętamy również o tym, żeby do długości krzywej dodać odpowiednie odcinki poziome i pionowe.

```
double dlugosc(double a, double b, double n)
{
double d=0;
d=d+16+f(10)-g(10);
double h = 1.0*(b-a)/n;
double x=a;
double tmp;
//dlugosc luku f
for (int i=0; i<n; i++)
{
       tmp=sqrt(h*h+(f(x+h)-f(x))*(f(x+h)-f(x)));
       d=d+tmp;//nowa dlugosc
       x=x+h; //zmiana x_i
}
//dlugosc luku g
x=a;
for (int i=0; i<n; i++)
{
       tmp=sqrt(h*h+(g(x+h)-g(x))*(g(x+h)-g(x)) );
       d=d+tmp;//nowa dlugosc
       x=x+h; //zmiana x_i
}
return d;
}
```

Ponieważ wynikiem ma być liczba całkowita, otrzymaną sumę zaokrąglamy do najbliższej liczby całkowitej w górę.

70.3.

W tym zadaniu przyjmujemy, że pasy są prostokątami. Pasy wycinamy od prawej strony materiału. Ponieważ szerokość pojedynczego pasa jest równa 0,25 m, to ostatni prostokąt rozpocznie się dla $x = 9{,}75$. Długość boku i-tego prostokąta jest równa $f(x_i) - g(x_i)$. Ponieważ ma być ona liczbą całkowitą, zaokrąglamy ją do najbliższej liczby całkowitej w dół. Na końcu sumujemy wszystkie obliczone długości boków prostokątów. Przykładowa realizacja w języku programowania znajduje się poniżej. Funkcja `floor(x)` z biblioteki `cmath` zwraca największą liczbę całkowitą nie większą niż x.

```
double zadanie3()
{
double d=0;
for (double x=2; x<=9.75; x=x+0.25)
    d=d+floor(f(x)-g(x));
return d;
}
```

Zadanie 71.

71.1.

Wobec definicji funkcji f, aby wyznaczyć wartość $f(1.5)$, należy obliczyć wartość $f_2(1.5)$. Zakładając, że współczynniki postaci potęgowej wielomianu f_2 pamiętamy w zmiennych (w kolejności od współczynnika stojącego przy najwyższej potędze): `A`, `B`, `C` i `D`, wartość $f_2(1.5)$ można obliczyć np. za pomocą schematu Hornera, tzn.

((A*x+B)*x+C)*x+D

gdzie zmiennej `x` należy uprzednio nadać wartość `1.5`.

71.2.

W rozwiązaniu zadania bardzo pomocna jest analiza wykresu funkcji załączonego do treści. Można zaobserwować, że wartość maksymalna funkcji f jest osiągana w przedziale $[4,5)$. Oznacza to, że wystarczy ograniczyć się tylko do wielomianu f_4. Niech w zmiennych `A`, `B`, `C` i `D` pamiętane będą współczynniki tego wielomianu (w kolejności od współczynnika stojącego przy najwyższej potędze).

Aby wyznaczyć maksymalną wartość $f_4(x)$ dla $x \in [4,5)$, można obliczać wartości $f_4(x_j)$ dla bardzo wielu punktów $x_j \in [4,5)$. Na przykład można obliczać wartości $f_4(x_j)$ początkowo dla $x_0 = 4$, a następnie zwiększać wartość argumentu o pewną małą liczbę h (krok), tzn. przyjąć $x_{j+1} = x_j + h$. W ten sposób obliczamy wartości $f_4(x_j)$, dopóki zachodzi nierówność $x_j < 5$. Aby wyznaczyć maksymalną wartość $f_4(x_j)$, należy zastosować klasyczny algorytm znajdowania elementu maksymalnego. Znalezienie wartości x_j,dla której $f_4(x_j)$ jest maksymalne, można zaimplementować, wykorzystując dwie zmienne globalne:

- `mx` (do zapamiętania x_j),
- `mfx` (do zapamiętania $f(x_j)$).

```
double mx, mfx;
void maksimum(double step)
{
   double x = 4.0;
   mx = x;
   mfx = ((A*x+B)*x+C)*x+D;
   for (x = 4.0; x<5.0; x+=step)
   {
        double fx = ((A*x+B)*x+C)*x+D;
        if ( fx > mfx )
        {
             mx  =  x;
             mfx = fx;
        }
   }
}
```

Powyższa procedura przyjmuje argument `step`, który oznacza wartość kroku h.

Stosując tę procedurę z krokiem $h = 0.001$, uzyskujemy wartości:

```
mx  = 4.092
mfx = 3.06495098
```

Aby jednak mieć pewność, że `3.06495` jest prawidłowym wynikiem tego zadania, tzn. jest maksymalną wartością funkcji f_4 z dokładnością do pięciu cyfr, wystarczy uruchomić powyższą procedurę ponownie, ale ze znacznie mniejszym krokiem, np. h=0.000001. Wówczas otrzymujemy:

```
mx  = 4.092275
mfx = 3.06495154
```

Zatem po zmniejszeniu kroku h uzyskaliśmy trochę większą wartość mfx. Można również sprawdzić, że dalsze zmniejszanie kroku nie zmienia w ogóle podanej powyższej przybliżonej wartości `mfx`. To oznacza, że wynik `3.06495` jest prawidłowo zaokrągloną do pięciu cyfr po przecinku maksymalną wartością funkcji $f_4(x)$. Jeśli chodzi o wartość x, zaokrągloną do trzech miejsc po przecinku, to można sprawdzić, że (prawe strony zostały podane z dokładnością do pięciu cyfr):

```
f(4.090)=3.06491
f(4.091)=3.06494
f(4.092)=3.06495
f(4.093)=3.06495
f(4.094)=3.06493
```

a więc obie odpowiedzi 4.092 i 4.093 są prawidłowe.

71.3.

Analizując dany w zadaniu wykres funkcji f, łatwo zauważyć, że ma ona cztery miejsca zerowe, odpowiednio w przedziałach (0,1), (2,3), (3,4) i (4,5). Oznacza to, że musimy wyznaczyć miejsca zerowe funkcji f_1, f_3, f_4, f_5. Do każdego z tych czterech zadań stosujemy ten sam algorytm, mianowicie metodę bisekcji. Dla przykładu ustalmy, że chcemy wyznaczyć miejsce zerowe funkcji $f_1(x)$. W metodzie bisekcji znajdowanie miejsca zerowego zaczyna się od podania przedziału, w którym się ono znajduje, takiego, na którego końcach wartości badanej

funkcji mają różne znaki. W rozważanym przykładzie przedziałem początkowym jest (0,1). Z analizy wykresu funkcji widzimy, że w lewym końcu przedziału funkcja ma wartość ujemną, a w prawym — dodatnią. W metodzie bisekcji następnie dzielimy przedział na dwie połowy i ograniczamy się do tej części, w której funkcja znów ma różne znaki na końcach przedziału. W ten sposób zawężamy przedział, w którym znajduje się szukane miejsce zerowe. Proces ten kończymy, gdy długość przedziału daje nam żądaną dokładność przybliżenia miejsca zerowego. Ponieważ w zadaniu należy wypisać odpowiedź z dokładnością do pięciu cyfr po przecinku, więc metodę bisekcji należy kontynuować, dopóki długość przedziału nie będzie mniejsza od pewnego maksymalnego dopuszczalnego błędu bezwzględnego `error`. W poniższym kodzie korzystamy z funkcji `f(x)` do obliczania wartości funkcji $f(x)$.

```
void bisekcja(double left, double right)
{
     double fl = f(left);
     while ( right-left > error )
     {
          double middle = (left+right)/2.0, fm = f(middle);
          if ( fl * fm < 0.0 )
               right = middle;
          else
               left = middle; fl = fm;
     }
     printf("Miejsce zerowe : %.5lf\n", (left+right)/2.0);
}
```

Zwróćmy uwagę, że warunek `( fl * fm < 0.0 )` sprawdzany w powyższej funkcji jest równoważny temu, że liczby `fl` (równa `f(left)`) i `fm` (równa `f(middle)`) mają przeciwny znak (jedna z nich jest ujemna, a druga dodatnia).

Ponieważ w zadaniu należy podać wynik z dokładnością do pięciu cyfr po przecinku, więc ustalimy próg błędu `error` na odpowiednio małą wartość. Zauważmy, że obranie `ror`=10^{-7} daje nam rozwiązanie x z błędem bezwzględnym spełniającym $|x - \alpha| \leq 10^{-7}$, co oznacza, że po zaokrągleniu go np. do pięciu cyfr po przecinku uzyskujemy wynik.

Zadanie 72.

72.1.

Za pomocą funkcji length() uzyskujemy długości obu napisów i porównujemy je ze sobą, pamiętając o tym, aby sprawdzić obie możliwości (albo pierwszy, albo drugi z nich może być tym dłuższym). Jeśli warunek podany w zadaniu jest spełniony, zwiększamy licznik przechowywany w zmiennej *wynik*:

```
int wynik = 0;
bool wypisalem = false;
for(int i=0; i<200; i++)
{
     string a,b;
     in >> a >> b;
     if (a.length()>=3*b.length() || a.length()*3 <=
b.length())
     {
          wynik++;
          (...)
     }
}
```

Konieczne jest jeszcze wypisanie pierwszej znalezionej pary. Po wypisaniu ustalamy wartość zmiennej logicznej *wypisalem* na *prawda*, co pozwoli nam uniknąć wypisania kolejnych par:

```
     if (a.length()>=3*b.length() || a.length()*3 <=
b.length())
     {
          wynik++;
          if (!wypisalem)
          {
               wypisalem = true;
               out << "Pierwsza para: " << endl;
               out << a << " " << b << endl;
          }
     }
```

72.2.

Dla dwóch zmiennych — łańcuchów znaków — chcemy sprawdzić, czy pierwszy z nich jest początkowym fragmentem drugiego. Pierwszy ma długość s = *a.length()*, zaś drugi *t* = *b.length()*. Wystarczy sprawdzić dwa warunki: czy *s* < *t* oraz czy pierwsze *s* znaków jest w obu napisach identyczne:

```
for(int i=0; i<N; i++)
{
     string a,b;
     in >> a >> b;
     if (a.length() >= b.length())
          continue;
     bool dobry = true;
     for(int j=0; j<a.length(); j++)
          if (a[j]!=b[j])
               dobry = false;
     (...)

}
```

Instrukcja:

```
if (a.length() >= b.length())
     continue;
```

powoduje natychmiastowe przejście do następnej iteracji pętli, jeśli drugi ciąg znaków jest za krótki (nie przewyższa pierwszego długością). Kolejne instrukcje porównują ze sobą kolejne znaki *a* i *b*. Pierwsza niezgodność powoduje ustawienie zmiennej *dobry* na *false*. Jeśli do końca algorytmu wartość tej zmiennej pozostanie *true*, musimy wypisać oba napisy, a także różniące je litery:

```
if (dobry)
{
     out << a << " " << b << " ";
     for(int j=a.length(); j<b.length(); j++)
          out << b[j];
     out << endl;
}
```

Litery, które różnią napisy, to te, znajdujące się na końcu dłuższego z nich (czyli *b*), licząc od miejsca, w którym skończył się krótszy (czyli od *a.length()*). Stąd wynika konstrukcja pętli taka jak powyżej.

72.3.

Napiszmy najpierw funkcję *koncowka(A,B)*, która przyjmując dwa napisy, określi, jak długa jest ich wspólna końcówka. Aby to określić, liczymy najpierw długości obu, odpowiednio *dl_A* i *dl_B*. Teraz trzeba sprawdzić, czy ostatnie litery *A* i *B* są równe (czyli czy *A[dl_A-1]* jest równe *B[dl_B-1]*), potem przedostatnie litery (*A[dl_A – 2]* oraz *B[dl_B – 2]*) i tak dalej:

```
int koncowka(string A, string B)
{
     int dl_A = A.length();
     int dl_B = B.length();
     int k = 0;
     while(k<dl_A && k<dl_B && A[dl_A-1-k]==B[dl_B-1-k])
          k++;
     return k;
}
```

Zmienna *k* przechowuje liczbę wcześniej już znalezionych wspólnych liter. Ważne jest, aby przerwać pętlę, kiedy *k* osiągnie jedną z liczb *dl_A*, *dl_B*, co będzie oznaczać, że sprawdziliśmy już jeden z napisów w całości. Gdybyśmy próbowali kontynuować, wyrażenie (na przykład) *dl_A -1-k* przyjęłoby wartości ujemne, a więc sprawdzalibyśmy nieistniejące wartości w zmiennej tablicowej *A*. Szczególnie w języku C++ to bardzo niebezpieczny rodzaj błędu, jego skutki są bowiem trudne do przewidzenia. Czasem taki program zostanie przerwany z błędem wykonania, ale zdarza się, że na przykład kontynuuje on działanie z nagle pojawiającymi się absurdalnymi wartościami zmiennych albo źle wykonuje niektóre instrukcje.

Skoro mamy już prawidłową funkcję *koncowka()*, reszta działań programu sprowadza się do tego, aby najpierw znaleźć najdłuższy wspólny końcowy fragment, a następnie wypisać wszystkie pary napisów, dla których wartość jest maksymalna. W tym celu użyjemy tablic przechowujących łańcuchy znaków: *pierwszy[200]*, *drugi[200]* oraz tablicy liczb *wspolny[200]*:

```
int dlugosc = 0;
string pierwszy[200],drugi[200];
int wspolna[200];
for(int i=0; i<N; i++)
{
     in >> pierwszy[i] >> drugi[i];
     wspolna[i] = koncowka(pierwszy[i],drugi[i]);
     if (wspolna[i] > dlugosc)
          dlugosc = wspolna[i];
}
out << "Maksymalna koncowka: " << dlugosc << endl;
for(int i=0; i<N; i++)
     if (wspolna[i]==dlugosc)
          out << pierwszy[i] << " " << drugi[i] << endl;
```

Pierwsza część programu (pętla *for*) zapisuje wczytane napisy w zmiennych *pierwszy[i]* oraz *drugi[i]*, a obliczoną końcówkę w zmiennej *wspolna[i]*. Najdłuższą dotychczas znalezioną końcówkę pamiętamy w zmiennej *dlugosc,* zmieniając jej wartość, kiedy trafimy na większą.

W drugiej części programu jeszcze raz sprawdzamy wszystkie znalezione końcówki — jeśli któraś z nich ma długość maksymalną, wypisujemy odpowiednie napisy na wyjście.

Zadanie 73.

73.1.

Po wczytaniu słowa przechodzimy pętlą przez wszystkie jego litery, porównując każdą z następną. Jeśli znajdziemy dwie identyczne, zwiększamy zmienną *licznik*, przechowującą liczbę odpowiednich słów:

```
int licznik = 0;
for(int i=0; i<1876; i++)
{
     string slowo;
     in >> slowo;
     for(int j=0; j<slowo.length()-1; j++)
     if (slowo[j]==slowo[j+1])
     {
          licznik++;
          break;
     }
}
```

Ważne są dwa szczegóły: po pierwsze, wewnętrzna pętla musi działać do przedostatniej litery, nie do ostatniej (*j < slowo.length()-1*), inaczej wykonalibyśmy porównanie z nieistniejącym elementem za tablicą. Po drugie, konieczna jest instrukcja *break* po znalezieniu dwóch kolejnych identycznych liter — bez niej może się zdarzyć, że znajdziemy w tym samym słowie jeszcze jakieś wystąpienie dwóch kolejnych liter, które nadmiarowo zwiększy *licznik*.

73.2.

Tworzymy tablicę *częstotliwosc[26]*, w której będziemy liczyć wystąpienia wszystkich liter: 'A' w *czestotliwosc[0]*, 'B' w *czestotliwosc[1]* itd. Napotykając literę, odejmujemy od niej kod

znaku 'A', aby uzyskać liczbę z przedziału [0, 25], a następnie zwiększamy wartość w odpowiedniej komórce tablicy *czestotliwosc*:

```
int czestotliwosc[26];
for(int j=0; j<26; j++)
     czestotliwosc[26] = 0;
int suma = 0;
for(int i=0; i<1876; i++)
{
     in >> slowo;
     for(int j=0; j<slowo.length(); j++)
          czestotliwosc[slowo[j]-'A']++;
     suma += slowo.length();
}
```

Podany fragment programu oblicza jeszcze jedną wartość: sumę długości wszystkich słów, czyli liczbę wystąpień wszystkich liter w pliku. Będzie ona potrzebna do policzenia procentowej częstotliwości wystąpień:

```
for(int j=0; j<26; j++)
{
     out << (char)('A'+j) << ": " << czestotliwosc[j];
     out << " (" << fixed << setprecision(2) <<
     100*czestotliwosc[j]/(double)suma << "%)" << endl;
}
```

Teraz dla *j = 0..25* wykonujemy następujące czynności:

- Wypisujemy j-ty znak, czyli *'A'+j*. Aby poinformować kompilator, że chcemy wypisać go jako znak (*char)*, a nie jako liczbę (*int*), zamieniamy go na odpowiedni typ instrukcją *(char)('A'+j)*, zwaną *rzutowaniem*.
- Wypisujemy liczbę wystąpień tego znaku
- Obliczamy częstotliwość procentową: *czestotliwosc[j]/suma*100%*. Musimy jeszcze zaznaczyć, że chcemy wyniku jako liczby rzeczywistej (zarówno *czestotliwosc[j]*, jak i *suma* to liczby całkowite), zatem przed dzieleniem zamienimy (znowu używając rzutowania) zmienną *suma* na typ rzeczywisty *double*.
- Ustalamy sposób wypisywania liczb: z dokładnie dwoma miejscami po kropce dziesiętnej. Służy do tego fragment instrukcji „*<< fixed << setprecision(2)*”.

73.3.

Zacznijmy od funkcji *spogloski(A)*, która w danym łańcuchu znaków *A* określi długość najdłuższego ciągu kolejnych spółgłosek:

```
int spolgloski(string A)
{
    int s = 0;
    int ciag = 0;
    for(int i=0; i<A.length(); i++)
    {
        if (A[i]=='A' || A[i]=='E' || A[i]=='O' || A[i]=='U'
               || A[i]=='I' || A[i]=='Y')
            s = 0;
        else
            s++;
        if (s > ciag)
            ciag = s;
    }
    return ciag;
}
```

Przechodzimy główną pętlą przez wszystkie litery słowa. Zmienna *s* przechowuje, ile (w danym momencie) ostatnich liter było spółgłoskami. Widać, że jeśli napotykamy kolejną spółgłoskę, to tę wartość należy zwiększyć o 1. Jeśli zaś kolejną literą będzie samogłoska, wartość *s* spada do 0.

Zmienna *ciag* przechowuje długość najdłuższego znalezionego do tej pory ciągu. Każdą wartość, jaką przybierze *s*, będziemy porównywać z wartością *ciag* i w razie potrzeby zwiększać tę ostatnią.

Teraz możemy już wczytać i sprawdzić wszystkie słowa:

```
int najdluzsze = 0;
int licznik = 0;
string odpowiedz;
for(int i=0; i<dlugosc; i++)
{
     string slowo;
     in >> slowo;
     int k = spolgloski(slowo);
     if (k>najdluzsze)
     {
          najdluzsze = k;
          licznik = 1;
          odpowiedz = slowo;
     }
     else if (k==najdluzsze)
          licznik ++;
}
```

Zmienna *najdluzsze* będzie przechowywać długość najlepszego ciągu spółgłosek, *licznik* — liczbę słów z takim ciągiem, a *odpowiedz* — pierwsze znalezione słowo. Dla każdego znalezionego słowa wywołujemy funkcję *spolgloski* — jeśli wynik okaże się lepszy od dotychczasowego rekordu, ustawiamy *najdluzsze* na nową wartość, *odpowiedz* na słowo, które właśnie znaleźliśmy, a *licznik* na 1.

Musimy jeszcze uwzględnić sytuację, kiedy nowe słowo ma tyle samo spółgłosek co najlepsze dotychczas znalezione — wtedy nie zmieniamy ani wartości *najdluzsze*, ani słowa przechowywanego jako *odpowiedz*, musimy jednak zwiększyć *licznik* o 1.

Zadanie 74.

74.1.

Na potrzeby rozwiązania zadania warto zaprogramować funkcję *czySameCyfry(S*), która zwraca wynik `true` lub `false` w zależności od tego, czy w napisie *S* znajdują się tylko i wyłącznie cyfry od 0 do 9, czy nie. Można najpierw napisać dodatkowo funkcję *czyCyfra*(*z*), która sprawdza, czy dany znak *z* jest cyfrą:

```
bool czyCyfra(char z)
{
  if ('0' <= z && z <= '9') return true;
  else                      return false;
}
```

Teraz implementacja funkcji *czySameCyfry*(*S*) to realizacja pętli, w której przeglądamy każdy znak napisu *S* i sprawdzamy, czy jest on cyfrą, np. w następujący sposób:

```
bool czySameCyfry(string S)
{
  int n = S.size();
  for (int i=0; i<n; i++)
    if ( czyCyfra(S[i])==false ) return false;
  return true;
}
```

Aby obliczyć wynik, wystarczy zliczać, dla ilu napisów w pliku `hasla.txt` funkcja `czySameCyfry(...)` zwróciła wartość `true`.

74.2.

Rozwiązanie zadania można łatwo uzyskać, o ile najpierw posortujemy wszystkie hasła z pliku `hasla.txt` w kolejności leksykograficznej. Do tego celu można na przykład zastosować algorytm sortowania bąbelkowego, gdyż liczba wszystkich haseł jest bardzo mała. Dla wygody warto najpierw napisać funkcję *mniejszy*(A,B), która sprawdzi, czy napis *A* jest mniejszy leksykograficznie od napisu *B*:

```
bool mniejszy(string A, string B)
{
    int n = A.size();
    int m = B.size();
    for (int i=0; i<min(n,m); i++)
    {
        if (A[i] < B[i]) return true;
        if (A[i] > B[i]) return false;
    }
    if (n<m) return true;
    else return false;
}
```

Powyżej wykorzystano funkcję *min*, która zwraca mniejszą z danych dwóch liczb. Funkcji *mniejszy* nie trzeba w ogóle pisać, jeśli wykorzystuje się typ *string*. Mianowicie dostępny jest operator mniejszości <, który porównuje leksykograficznie dwa napisy. Niech tablica T[0..*n*] zawiera wszystkie wczytane hasła z pliku `hasla.txt` (*n*=200). Sortowanie bąbelkowe tablicy T[0..*n*-1] można zrealizować następująco:

```
for (int i=0; i<n-1; i++)
  for (int j=1; j<n; j++)
    if (mniejszy(T[j],T[j-1])) swap(T[j],T[j-1]);
```

Sortowanie to korzysta z powyższej funkcji *mniejszy* oraz pomocniczej funkcji *swap*, która zamienia miejscami napisy na pozycjach *j-1* i *j* w tablicy *T*.

Teraz gdy wszystkie hasła w tablicy *T* mamy dane w kolejności leksykograficznej możemy łatwo znaleźć te, które się powtarzają, i wypisać je bez powtórzeń. Mianowicie wystarczy przeglądać tablicę *T* kolejno od pierwszego hasła do ostatniego, przy czym przy przechodzeniu do następnego hasła będziemy sprawdzali, ile następnych haseł jest takich samych jak aktualnie rozważane. Algorytm ten można zrealizować w następujący sposób:

```
for (int i=0; i<n;i++)
{
  int j = 0;
  while(i+j+1<n && T[i+j+1]==T[i]) j++;
  if ( j>0 )
  {
    cout << hasla[i] << endl;
    i += j;
  }
}
```

W powyższym algorytmie wykorzystano operator == do porównywania napisów. Jeśli w rozwiązaniu wykorzystuje się typ `string`, to taki operator jest automatycznie dostępny. Jeśli natomiast go nie mamy, to do porównania napisów możemy na przykład wykorzystać podaną wyżej funkcję *mniejszy*. Mianowicie `A==B`, wtedy i tylko wtedy, gdy `mniejszy(A,B)==false && mniejszy(B,A) == false`.

74.3.

Rozwiązanie zadania budujemy poprzez zliczenie wyników, dla których wykryto, że pojedyncze hasło spełnia podane w zadaniu warunki. Aby to sprawdzić, należy najpierw zauważyć, że hasło musi składać się z co najmniej 4 znaków. Następnie należy przejrzeć każdy fragment 4-literowy i sprawdzić, czy zawiera on kolejne znaki ASCII (w dowolnej kolejności). Procedurę przeglądania wszystkich fragmentów 4-literowych napisu S[0..*n-1*] można zrealizować za pomocą zwykłej pętli, w której wskaźnik *i* zmieniamy od czwartej do ostatniej litery, a następnie badamy fragment złożony z liter na pozycjach *i*-3, *i*-2, *i*-1, *i*.

```
for (int i=3; i<n; i++)
  if ( kolejne(S[i-3],S[i-2],S[i-1],S[i]) ) return true;
```

Do ostatecznego rozwiązania zadania pozostaje zatem napisać funkcję *kolejne*(*a,b,c,d*), która sprawdzi, czy zbiór {*a,b,c,d*} zawiera 4 kolejne znaki ASCII. Dla wygody możemy to zrealizować, wstawiając wszystkie 4 znaki *a,b,c,d* do tablicy *T*, a następnie posortować ją dowolnym algorytmem. Wówczas wystarczy tylko sprawdzić, czy T[0], T[1], T[2] i T[3] są kolejnymi znakami, np. za pomocą następującego warunku

(T[0]+1==T[1] && T[1]+1==T[2] && T[2]+1==T[3]).

74.4.

Rozwiązanie zadania budujemy poprzez zliczenie wyników, dla których wykryto, że pojedyncze hasło spełnia podane warunki w zadaniu. Aby sprawdzić, czy wszytkie warunki zachodzą jednocześnie dla pewnego hasła S[0..*n*-1], można wykonać pętlę, w której będziemy przeglądać hasło znak po znaku, sprawdzając, czy jest on cyfrą, czy literą małą lub dużą. Dla każdego rodzaju znaku będziemy pamiętać zmienną logiczną, która będzie informowała, czy taki znak wystąpił. Niech `CY, ML, DL` oznaczają te zmienne odpowiednio dla typu: cyfra, mała litera, duża litera. Pętlę tę możemy zapisać następująco:

```
bool CY=false, ML=false, DL=false;
for (int i=0; i<n; i++)
{
  char c = S[i];
  if      ( '0' <= c && c <= '9' ) CY = true;
  else if ( 'a' <= c && c <= 'z' ) ML = true;
  else if ( 'A' <= c && c <= 'Z' ) DL = true;
}
```

Po zakończeniu pętli wystarczy sprawdzić, czy w haśle wystąpił każdy rodzaj znaku, a więc wystarczy sprawdzić warunek: `(CY && ML && DL)`.

Zadanie 75.

75.1.

W tym zadaniu, poza wczytaniem danych, wystarczy sprawdzić bardzo prosty warunek na pierwszą i ostatnią literę słowa. W C++ można to robić na przykład tak:

```
for(int i=0; i<slowa; i++)
{
     in >> S;
     if (S[0]=='d' && S[S.length()-1]=='d')
     out << S << endl;
}
```

75.2.

Tutaj wystarczy zaszyfrować z osobna każde słowo o długości co najmniej 10:

```
for(int i=0; i<S.length(); i++)
{
     S[i] -= 'a';
     S[i] = S[i]*5 + 2;
     S[i] %= 26;
     S[i] += 'a';
}
```

Wykonujemy kolejno następujące operacje na każdym znaku:

- przez odjęcie kodu znaku 'a' zamieniamy go na liczbę z przedziału [0, 25];
- liczbę tę mnożymy przez 5 i dodajemy 2;
- bierzemy resztę z dzielenia przez 26;
- z powrotem zamieniamy liczbę na znak, dodając kod 'a'.

Taka implementacja jest wygodna, ma jednak pewną wadę: otóż zmienna *S[i]*, na której dokonujemy operacji, jest typu znakowego (*char* w C++), a typ ten przechowuje liczby z przedziału –128..127 albo 0..255. O ile można bezpiecznie pomnożyć liczbę z przedziału [0,25] przez 5 i dodać do niej 2 (nie przekroczymy 127), o tyle gdyby kluczem szyfrowania było na przykład (19, 15), uzyskalibyśmy w ten sposób błędne odpowiedzi. Lepiej przepisać na chwilę literę *S[i]* do zmiennej liczbowej (na przykład typu *int*):

```
int c = S[i] - 'a';
c = c*x + y;
c %= 26;
S[i] = c + 'a';
```

75.3.

Dla wygody utwórzmy procedurę *szyfruj*, która przyjmuje ciąg znaków oraz dwie liczby *x*, *y* i koduje zadany ciąg szyfrem afinicznym z kluczem (x,y). Zrobimy to analogicznie jak w zadaniu 2:

```
string szyfruj(string jawny, int x, int y)
{
    string wynik = jawny;
    for(int i=0; i<wynik.length(); i++)
    {
        int c = wynik[i] - 'a';
        c = c*x + y;
        c %= 26;
        wynik[i] = c + 'a';

    }
    return wynik;
}
```

Metoda znalezienia właściwego klucza szyfrującego (a także deszyfrującego) jest zaskakująco prosta: ponieważ obie lizcby A i B są z przedziału [0,25], wszystkich możliwości klucza jest 26*26 = 676. Dla komputera jest to bardzo niewielka liczba[8]. Sprawdźmy więc, podwójną pętlą, wszystkie możliwe klucze, dla każdego szyfrując słowo jawne i sprawdzając, czy będzie wówczas identyczne z zakodowanym:

8 Jest to dobra ilustracja tego, że pojęcie złożoności obliczeniowej warto stosować ostrożnie: są sytuacje, w których algorytm o największej złożoności okazuje się najlepszą możliwością! W tym wypadku dzieje się tak dlatego, że wielkość danych (w tym wypadku rozmiar alfabetu, który ma 26 liter) może być bardzo mała.

```
for(int i=1; i<=ile; i++)
{
     string jawne,zakodowane;
     in >> jawne;
     in >> zakodowane;
     for(int A=0; A<=25; A++)
          for(int B=0; B<=25; B++)
               if (szyfruj(jawne,A,B)==zakodowane)
                    out  << "Klucz szyfrujacy numer " << i <<
": ("                          << A << "," << B << ")" << endl;
}
```

Podane rozwiązanie znajduje tylko klucz szyfrujący. Aby znaleźć klucz szyfrujący, konieczna jest więc druga, analogiczna pętla:

```
for(int A=0; A<=25; A++)
     for(int B=0; B<=25; B++)
          if (szyfruj(zakodowane,A,B)==jawne)
               out << "Klucz deszyfrujacy numer " << i << ": ("
<<                        A << "," << B << ")" << endl;
```

Można zmodyfikować podane wyżej procedury tak, aby po znalezieniu właściwego klucza natychmiast przerywały pętlę. Nie jest to konieczne i skomplikuje to nieco program, ale taka praktyka na dłuższą metę często przynosi bardzo istotną oszczędność czasową. Aby to uczynić, zdefiniujmy zmienną *znalazlem*, która przyjmie wartość **true**, kiedy klucz zostanie znaleziony. Pętlę będziemy wykonywać tylko dopóki *znalazlem* nie zmieni wartości:

```
bool znalazlem = false;
for(int A=0; A<=25 && !znalazlem; A++)
     for(int B=0; B<=25 && !znalazlem; B++)
          if (szyfruj(zakodowane,A,B)==jawne)
          {
               out << "Klucz deszyfrujacy numer " << i << ": ("
<<                        A << "," << B << ")" << endl;
               znalazlem = true;
          }
```

Zapis:

```
for(int A=0; A<=25 && !znalazlem; A++)
{
     (...)
}
```

wymaga pewnych wyjaśnień: oznacza on wykonywanie pętli dopóki $A \leq 25$ oraz zmienna *znalazlem* jest równa **false**. Jeśli którykolwiek z tych warunków przestanie być prawdą, pętla się kończy.

Zadanie 76.

76.1.

Do rozwiązania zadania przydatne jest zapamiętanie klucza (z wiersza nr 7) w tablicy 50-elementowej. Niech `P[0]`, `P[1]`, ..., `P[49]` oznaczają kolejne liczby klucza. Ponieważ w zadaniu należy zaszyfrować 6 danych napisów za pomocą klucza `P`, więc wygodnie jest

napisać program szyfrujący jeden napis, a następnie uruchomić go dla wszystkich danych napisów. Należy zwrócić uwagę, że w języku C++ napisy często pamiętane są w tablicach znaków, w których pierwsza litera ma wskaźnik 0, druga — 1 itd. Niech `A[0..49]` będzie napisem, który chcemy zaszyfrować kluczem `P[0..49]`. Podkreślmy, że w tablicy P znajdują się wartości od 1 do 50, a więc musimy być ostrożni przy programowaniu procedury szyfrującej, tzn. musimy za każdym razem zamieniać literę `A[i]` z literą `A[P[i]-1]`. Procedurę tę warto zaprogramować w pętli, która wykona operacje przestawiania liter w takiej samej kolejności, jak to zostało podane w treści zadania, tj. od pierwszej do ostatniej litery słowa A. Ponieważ klucz jest tej samej długości co słowo A, więc wystarczy kolejno zamieniać literę `A[i]` z literą `A[P[i]-1]` dla kolejnych wartości `i`=0,1,...,49. Na końcu w słowie A uzyskujemy w zaszyfrowane słowo wyjściowe. Proces szyfrowania napisu A możemy zatem zapisać w następujący sposób, gdzie do zamiany wartości dwóch zmiennych użyta jest funkcja *swap*:

```
string szyfr(string A)
{
    for (int i=0; i<50; i++)
      swap(A[i], A[P[i]-1]);
    return A;
}
```

76.2.

Podobnie jak w poprzednim zadaniu klucz będziemy pamiętali w tablicy `P[0..14]`, a napis A w tablicy `A[0..49]`. Tym razem przy szyfrowaniu napisu musimy dodatkowo pamiętać, aby po zamianie 15. litery, czyli `A[14]`, z literą `A[P[14]-1]` kolejną literę słowa, czyli `A[15]`, zamieniać z literą `A[P[0]-1]`, a następną z literą `A[P[1]-1]` itd. Możemy to zapisać z użyciem operacji modulo, czyli obliczania reszty z dzielenia całkowitego. W naszym przypadku łatwo zauważyć, że będziemy zamieniali literę `A[i]` z literą `A[P[i%15]-1 ]`. Poniższa funkcja realizuje proces szyfrowania kluczem 15-elementowym:

```
string szyfr(string A)
{
    for (int i=0; i<A.size(); i++)
      swap(A[i], A[P[i%15]-1]);
    return A;
}
```

76.3.

W zadaniu mamy dane słowo B oraz klucz [6, 2, 4, 1, 5, 3]. Ponownie przyjmijmy, że słowo pamiętane jest w tablicy `B[0..49]`, a klucz — w tablicy `P[0..5]`. Aby znaleźć słowo A, które po zaszyfrowaniu będzie słowem B, należy dokonać analizy algorytmu szyfrowania podanego w treści. Oczywiście słowo B powstało ze słowa A poprzez wykonanie pięćdziesięciu operacji zamian pewnych par liter na słowo A. Można łatwo zauważyć, że chcąc odwrócić proces szyfrowania, tzn. zdeszyfrować słowo B, wystarczy wykonać te same operacje zamian liter, ale w odwrotnym porządku. Ponieważ w słowie A zamieniano litery `A[i]` z `A[P[i%6]-1]`, więc proces deszyfrowania słowa `B[0..49]` możemy zapisać za pomocą następującego programu

```
    for (int i=49; i>=0; i--)
        swap(B[i], B[P[i%6]-1]);
```

W ten sposób, po wykonaniu powyższej pętli, w słowie B otrzymamy słowo pierwotne, a więc wynik zadania.

Zadanie 77.

Przydatna będzie wiedza o sposobie kodowania wielkich liter w standardzie ASCII, w szczególności znajomość kodów kolejno od ‘A’→65 do ‘Z’→90 i umiejętność konwersji kodu znaku na graficzną postać znaku za pomocą funkcji lub rzutowania typu `char`(). Zauważ, że pozycję litery w alfabecie można wyznaczyć, odejmując wartość 65 od jej kodu ASCII.

Całą zawartość tekstu (jawnego lub zaszyfrowanego) z jednego wiersza pliku pobieramy do **jednej zmiennej** tekstowej (C++, Java) lub do **tablicy znaków** (C, a także Pascal, ponieważ szyfr o długości przekraczającej 255 znaków nie zmieści się w zmiennej typu *String*). Używamy odpowiedniej funkcji w wybranym języku programowania:

C++ *getline*(strumien z pliku, zmiennaTekstowa);

C *fgets* (tablica znaków, rozmiar tablicy, wskaźnik do pliku);

Pascal *Read*(plik, znak) w pętli aż do znaku nowej linii #10;

Java *s.nextLine*(), gdzie *s* — zmienna typu *Scanner*.

77.1.

Tworzymy łańcuch znaków *q* (lub tablicę znaków) o długości równej długości tekstu *t*. Wypełniamy go kolejnymi znakami klucza (cyklicznie) na pozycjach odpowiadających wielkim literom tekstu, zaś pozostałe pozycje możemy wypełnić dowolnym znakiem niebędącym wielką literą, np. przepisać z tekstu, z tej samej pozycji, znak odstępu lub znak przestankowy.

Tworzymy pusty łańcuch znaków *s* dla wyniku szyfrowania.

Następnie dla każdego **znaku w tekście *t***:

> jeśli znak **nie jest** wielką literą, przepisujemy go do łańcucha wynikowego, w przeciwnym razie wykonujemy następujące kroki:
>
> 1) wyznaczamy odpowiadający mu znak klucza *c*, znajdujący się w łańcuchu *q* na tej samej pozycji co bieżący znak tekstu *t*;
> 2) wyznaczamy pozycję *k* znaku *c* w alfabecie (możemy w tym celu przeszukać alfabet lub prościej: wykorzystać kody ASCII);
> 3) obliczamy kod znaku szyfru, **dodając** wartość *k* do kodu znaku tekstu;
> 4) jeśli obliczona wartość kodu **jest większa od 90, odejmujemy** od niej **26**, aby przesunąć ten kod do zakresu odpowiadającego wielkim literom;
> 5) dołączamy znak odpowiadający obliczonemu kodowi na końcu łańcucha *s*.

Wypisujemy łańcuch wynikowy *s*.

77.2.

Problem odszyfrowania jest podobny do problemu szyfrowania, który rozwiązaliśmy w zadaniu1, rozwiązania obu problemów różnią się tylko w krokach 3) i 4) algorytmu opisanego powyżej. Tekst mamy teraz zaszyfrowany. Od kodu każdej litery szyfru trzeba więc **odjąć** przesunięcie ***k*** (krok 3) i sprawdzić, czy obliczony kod szyfru nie jest mniejszy od 65, czyli

najniższej wartości kodu z zakresu wielkich liter. Jeśli jest mniejszy, to trzeba **powiększyć** go o wartość **26** (krok 4).

Aby uprościć rozwiązanie całej wiązki zadań, wygodnie jest zdefiniować jedną funkcję szyfrującą/deszyfrującą i wykorzystać ją w rozwiązaniu obu zadań: 1 i 2. Może to być na przykład:

Vigenere(wiadomosc, klucz, szyfruj),

gdzie: *wiadomosc* — łańcuch znaków zawierający szyfr lub treść do zaszyfrowania,

klucz — słowo używane jako klucz szyfru,

szyfruj — parametr typu logicznego: *true* — szyfrowanie, *false* — deszyfrowanie, który będzie sterował wykonaniem jednego z dwóch wariantów kroków 3) i 4) algorytmu opisanego w rozwiązaniu zadania 1.

Przykładowy kod w języku C++

```
string Vigenere(string tresc, string klucz, bool szyfruj) {
    int n = tresc.length();
    int nk = klucz.length();

    string q = "";                  // odwzorowanie klucza dla calej tresci
    int powtorzenia = 1;            // liczba powtorzen klucza
    int j = 0;                      // indeks znaku w kluczu
    for (int i=0; i<n; i++) {
        char c = tresc[i];
        if (c>='A' && c<='Z') {     // jesli znak jest wielka litera
            c = klucz[j];           // weź znak klucza
            j++;                    // przesun się w kluczu na nastepny znak
            if (j==nk) {            // gdy klucz się skończył,
                  j=0;              // czytaj klucz od poczatku
                  powtorzenia++;    // zwiększ licznik powtórzeń klucza
            }
        }
        q += c;
    }
    cout << powtorzenia <<" powtorzenia klucza\n";

    string t="";                    // kodowanie lub dekodowanie
    for (int i=0; i<n; i++) {
        char c = tresc[i];
        if (c>='A' && c<='Z') {     // jesli znak jest wielka litera
            int k = q[i]-int('A');
            if (szyfruj) {          // szyfruj
                c += k;
                if (c>'Z') c -=26;
            }
            else {                  // deszyfruj
                c -= k;
                if (c<'A') c +=26;
            }
        }
        t += c;
    }
    return t;
}
```

77.3.

a) Przygotujemy tablicę liczb całkowitych *L* o rozmiarze 26 (tyle, ile mamy liter w alfabecie), każdy element tablicy będzie licznikiem wystąpień innej litery: *L*[0] będzie zliczać wystąpienia litery ‘A’, *L*[1] — litery ‘B’, …, *L*[25] — litery ‘Z’. Na początku każdy licznik inicjujemy wartością 0.

Dla kolejnych znaków szyfru powtarzamy następujące kroki:

jeżeli znak jest wielką literą:

obliczamy indeks *i* licznika *L[i]*, odpowiadający pozycji tego znaku w alfabecie,

zwiększamy o 1 wartość licznika *L[i]*

Przykładowy kod w języku C++

```
int L[26];
for (int j=0; j<26; j++) L[j]=0;      // liczniki wystapien liter
for (int i=0; i<s.length(); i++) {    // dla każdego znaku szyfru s
    if (s[i]>='A' && s[i]<='Z') {     // jeżeli znak jest wielką literą
        int j = s[i] - 'A';           // wyznacz jej pozycję w alfabecie
        L[j]++;                       // zwiększ licznik jej wystąpień
    }
}
cout << "Liczniki wystapien liter:\n"; // wypisz wyniki
for (int j=0; j<26; j++)
    cout << char(j+'A') << " - " << L[j] << endl;;
```

b) W celu oszacowania długości klucza szyfru obliczamy sumę iloczynów *L[i]**(*L[i]* -1) dla *i*∈<0..25>. Liczbę wszystkich wystąpień liter możemy wyznaczyć, sumując liczniki poszczególnych liter. Następnie obliczamy szacunkową długość klucza według podanego wzoru.

Wynik zaokrąglamy do 2 cyfr dziesiętnych. Można zastosować w tym celu funkcję *round()*, lecz nie wszystkie kompilatory ją implementują, funkcja nie należy do standardu. Samodzielne zaokrąglenie wyniku do 2 cyfr dziesiętnych można wykonać według następujących kroków:

- dodaj do wyniku wartość 0.005,
- następnie pomnóż przez 100,
- obetnij do liczby całkowitej,
- i podziel przez 100.

Wypisujemy dla porównania długość klucza podanego w pliku *szyfr.txt*.

Przykładowy kod w języku C++

```
    int suma = 0;
    int n = 0;
    for (int j=0; j<26; j++) {
      suma += L[j]*(L[j]-1);
      n += L[j];
    }
    double ko = (double)suma/n/(n-1);
    double d = 0.0285/(ko-0.0385);

    d += 0.005;                    // zaokraglenie do 2 cyfr dziesietnych
    d = 0.01*int(100*d);
    cout << d << " - szacunkowa dlugosc klucza\n";
    cout << klucz.length() << " - dokladna dlugosc klucza\n";
```

Aby zapisać wyniki do pliku tekstowego, można utworzyć w kodzie programu strumień wyjściowy skojarzony z plikiem i skierować do niego wszystkie wyniki lub uruchomiając program wykonywalny w oknie konsoli, przekierować standardowe wyjście tak, by program wysyłał dane wprost do pliku tekstowego zamiast na konsolę.

Zadanie 78.

Pobieramy całą treść pojedynczej wiadomości do jednej zmiennej tekstowej (C++, Pascal, Java) lub do tablicy znaków (C). Pamiętamy, że będzie to łańcuch znaków zawierający spacje. Nie potrzebujemy przechowywać w pamięci wszystkich wiadomości na raz. Wystarczy przetwarzać je kolejno, wykorzystując tę samą zmienną tekstową.

Za względu na potrzebę powtórzenia operacji **tworzenia skrótu** dla wielu wiadomości wygodnie jest zaprogramować tę operację jako osobną funkcję, która będzie wywoływana w pętli programu głównego dla kolejnych wiadomości czytanych z pliku.

78.1

Poniżej podano przykładowy kod funkcji skrótu w C++, realizującej opisany algorytm.

Należy uzupełnić poniższy kod o polecenia wypisywania żądanych wyników do pliku.

```
string skrot(string t) {
    string S = "ALGORYTM";
    int d = S.length();

    // uzupelnij dlugosc wiadomosci do wielokrotnosci 8 znakow
    while (t.length()%d != 0) t += ".";
    int n = t.length();

    // przetwarzanie wiadomosci
    int p = 0;
    while (p<n) {
        for (int j=0; j<d; j++) S[j] = (S[j]+t[p+j])%128;
        p += d;
    }
    // rzutowanie elementów tablicy S na zakres wielkich liter
    for (int j=0; j<d; j++) S[j] = 65 + S[j]%26;
    return S;
}
```

Funkcja skrótu w języku Pascal

```
function skrot(t: String): String;
   var S: array[1..8] of Integer;
       wynik: String;
       i,p: Integer;
begin
  wynik := 'ALGORYTM';
  for i:=1 to 8 do S[i] := Ord(wynik[i]);
  while (Length(t) mod 8 <> 0) do t := t + '.';
  writeln(Length(t));
  p := 0;
  while (p<Length(t)) do
  begin
        for i:=1 to 8 do S[i] := (S[i] + Ord(t[p+i])) mod 128;
        p := p + 8;
  end;
  for i:=1 to 8 do wynik[i] := Char(65 + S[i] mod 26 );
  skrot := wynik;
end;
```

78.2

Deszyfracja e-podpisu w języku C++

```
int N = 200;
int D = 3;
string mojSkrot = "";
for (int i=0; i<8; i++) {
      we2 >> ep;              // czytanie z pliku liczby, elementu e-podpisu
      int w = ep * D % N;
      mojSkrot += char(w);
}
cout << "moj skrot = " << mojSkrot << endl;
```

78.3

Program główny w języku C++, weryfikujący wiarygodność wiadomości:

```
ifstream we1("wiadomosci.txt");
ifstream we2("podpisy.txt");

int main() {
    string t;                 // tresc wiadomosci
    int p;                    // jedna liczba z 8 liczb epodpi-
su
    int N = 200, D = 3;    // elementy klucza publicznego
    vector<int> W;            // wykaz numerów wiarygodnych in-
formacji

    for (int i=1; i<=11; i++) {

        getline(we1,t);
        cout << t << endl;
        string mojskrot = skrot(t);
```

```
        cout << "moj skrot  = " << mojskrot << endl;

        string twojskrot = "";
        for (int i=0; i<8; i++) {
            we2 >> p;
            int w = ep * D % N;;
            twojskrot += char(w);
        }
        cout << "twoj skrot = " << twojskrot << endl;
        if (mojskrot == twojskrot) {
                cout << i << " - wiarygodna\n";
                W.push_back(i);
        }
        else  cout << i << " - zmieniona\n";

     }
    we1.close();
    we2.close();
    cout << "\nNumery wiarygodnych wiadomosci: ";
    for (int i=0; i<W.size(); i++) cout << W[i] << ", ";
    cout << endl;

    return 0;
}
```

Wiarygodność każdej wiadomości może być oceniana na bieżąco, program od razu wypisze odpowiednią informację o tym, czy wiadomość jest wiarygodna, czy zmieniona..

Program może także wypisać numery wszystkich wiarygodnych wiadomości dopiero po zakończeniu przetwarzania wszystkich wiadomości. W tym celu należy na bieżąco zapamiętywać ich numery. Można to zrobić w zwykłej tablicy statycznej, ale ponieważ nie znamy z góry liczby jej elementów, a nie chcemy marnować pamięci na elementy, które nie są wykorzystywane, zaproponujemy tu strukturę danych o typie `vector`.

Wektor jest alternatywą dla tablicy statycznej, jest wieloelementową strukturą danych, zdefiniowaną w bibliotece STL. Aby móc go używać, należy dołączyć do kodu programu jego definicję dyrektywą `#include <vector>`. Wektor umożliwia dostęp do swoich elementów za pomocą indeksu, podobnie jak tablica, lecz może zmieniać rozmiar podczas pracy programu (deklaracja rozmiaru nawet nie jest konieczna) i posiada zestaw funkcji znacznie ułatwiających zarządzanie elementami. Szczegółów poszukaj w dokumentacji STL.

Zadanie 79.

79.1.

W celu rozwiązania zadania warto najpierw napisać program, który dla danego okręgu sprawdza, czy znajduje się on w całości w pewnej ćwiartce. Zauważmy, że na to, aby okrąg o środku w punkcie (x,y) i promieniu r zawierał się w pewnej ćwiartce, potrzeba i wystarcza, aby $r < \min(|x|, |y|)$. Intuicyjnie chodzi o to, aby promień okręgu był mniejszy niż odległość jego środka od najbliższej osi Ox lub Oy. Jeśli już sprawdzimy to, czy okrąg leży w całości w pewnej ćwiartce, to ustalenia numeru tej ćwiartki można dokonać sprawdzając znaki liczb

x i y. W poniższej funkcji zaprogramowano sprawdzanie, do której ćwiartki należy okrąg o środku w punkcie (x, y) i promieniu r.

```
int cwiartka(double x, double y, double r)
{
    if ( r >= fabs(x) || r >= fabs(y) ) return 0;
    if ( x > 0 && y > 0 ) return 1;
    if ( x < 0 && y > 0 ) return 2;
    if ( x < 0 && y < 0 ) return 3;
    return 4;
}
```

Powyżej wykorzystano funkcję `fabs` do obliczania wartości bezwzględnej liczby rzeczywistej (typu `double`). Wynikiem funkcji jest numer ćwiartki, a jeśli okrąg nie zawiera się w żadnej z nich, to funkcja zwraca wynik 0. Jeśli dysponujemy powyższą funkcją, do rozwiązania zadania wystarczy wywołać ją dla każdego okręgu z pliku `okregi.txt` oraz zliczać, np. w tablicy 5-elementowej, liczbę okręgów w każdej ćwiartce z osobna oraz liczbę okręgów, które nie zawierają się w żadnej ćwiartce.

79.2.

W celu rozwiązania zadania znów warto napisać osobny (pod)program do sprawdzenia, czy dwa okręgi tworzą lustrzaną parę. Rozpatrzmy dwa okręgi: pierwszy o środku w (x_1, y_1) i promieniu r_1, drugi o środku w (x_2, y_2) i promieniu r_2. Zauważmy, że jeśli okręgi te tworzyą lustrzaną parę, to promienie tych okręgów są jednakowe, tzn. spełniony jest warunek $r_1 = r_2$. Następnie należy sprawdzić, czy punkty (x_1, y_1) i (x_2, y_2) są symetryczne względem jednej z osi Ox lub Oy. Można to prosto sprawdzić w następujący sposób: `( x1==x2 && y1==-y2 ) || ( x1==-x2 && y1==y2 )`. Ostatecznie funkcję sprawdzającą, czy dwa okręgi tworzą lustrzaną parę, możemy zapisać następująco:

```
bool czyParaLustrzana(
  double x1, double y1, double r1,   double x2, double y2,
double r2
)
{
    if ( r1 != r2 ) return false;
    if ( x1 == x2 && y1 == -y2 ) return true;
    if ( y1 == y2 && x1 == -x2 ) return true;
    return false;
}
```

Aby prawidłowo zliczać liczbę wszystkich lustrzanych par okręgów spośród wszystkich okręgów danych w pliku `okregi.txt`, można zapamiętać ich środki i promienie w tablicach `x[0..N-1]`, `y[0..N-1]`, `r[0..N-1]`, a następne zrealizować zliczanie w następujący sposób:

```
  int pary = 0;
  for (int i=1; i<N; i++)
    for (int j=0; j<i; j++)
      if   (czyParaLustrzana(x[i],y[i],r[i],   x[j],y[j],r[j]))
pary++;
```

Poprawność powyższego kodu można uzasadnić następująco: w pierwszej (zewnętrznej) pętli przeglądamy okręgi o numerach i od 1 do N-1, czyli wszystkie oprócz pierwszego. Dla każ-

dego okręgu o środku w punkcie (x[i], y[i]) i promieniu r[i] sprawdzamy, które z okręgów o numerach od 0 do i-1 tworzą z nim parę lustrzaną. W ten sposób każdą parę lustrzaną policzymy jeden raz.

79.3.

Liczbę par będziemy zliczać w podobny sposób jak w poprzednim zadaniu:

```
 int pary = 0;
 for (int i=1; i<N; i++)
   for (int j=0; j<i; j++)
     if (czyParaProstopadla(x[i],y[i],r[i], x[j],y[j],r[j]))
pary++;
```

Do rozwiązania zadania będzie zatem potrzebna funkcja, która sprawdzi, czy dwa okręgi tworzą parę prostopadłą. Niech (x_1, y_1) i (x_2, y_2) oznaczają środki dwóch okręgów, a r_1 i r_2 — odpowiednio ich promienie. Po pierwsze, aby okręgi te tworzyły parę prostopadłą, ich promienie muszą być jednakowe, czyli $r_1 = r_2$. Po drugie, należy sprawdzić, czy ich środki tworzą parę prostopadłą, tzn. czy jeden z nich powstaje przez obrócenie drugiego o 90 stopni względem początku układu współrzędnych w kierunku zgodnym lub przeciwnym do ruchu wskazówek zegara. Warto tutaj podkreślić, że sprawdzanie obrotu w obu kierunkach jest istotne dla obliczenia prawidłowego wyniku. Sprawdzenie, czy środki (x_1, y_1), (x_2, y_2) tworzą parę prostopadłą, można wykonać na kilka sposobów. Na przykład można sprawdzić, czy punkty te wraz z początkiem układu współrzędnych tworzą trójkąt prostokątny (aby to wykonać, wystarczy skorzystać z twierdzenia Pitagorasa). Pokażemy jednak inny sposób rozwiązania tego problemu. Mianowicie można zauważyć, że obrót punktu (a, b) o 90 stopni w kierunku przeciwnym do ruchu wskazówek zegara daje punkt o współrzędnych $(-b, a)$. Z tej obserwacji natychmiast wynika poprawność poniższego programu sprawdzającego, czy dwa okręgi tworzą parę prostopadłą.

```
bool czyParaProstopadla(
   double x1, double y1, double r1,
   double x2, double y2, double r2)
{
  if ( r1 != r2 ) return false;
  double nx1 = -y1, ny1 = x1;
  double nx2 = -y2, ny2 = x2;
  if ( nx1 == x2 && ny1 == y2 ) return true;
  if ( nx2 == x1 && ny2 == y1 ) return true;
  return false;
}
```

79.4.

Podkreślmy, że w zadaniu należy rozpatrywać tylko pierwsze 1000 okręgów. Do rozwiązania zadania warto najpierw zaprogramować funkcję sprawdzającą, czy dane dwa okręgi mają co najmniej jeden wspólny punkt. Załóżmy, że funkcja

```
bool sprawdz(double x1, double y1, double r1,
             double x2, double y2, double r2)
{
  // return true/false jeśli okręgi mają / nie mają coś wspól-
nego
}
```

rozwiązuje ten problem. Wówczas iteracyjnie obliczamy długości wszystkich łańcuchów. Na początku zakładamy, że pierwszy okrąg tworzy łańcuch długości 1. Następnie sprawdzamy, czy kolejny okrąg ma wspólny punkt z ostatnim okręgiem aktualnego łańcucha. Jeśli tak, to wydłużamy aktualny łańcuch. Jeśli nie, to przerywamy go, wypisując jego długość jako wynik, a następnie rozpoczynamy nowy jednoelementowy łańcuch, który uznajemy za aktualny. Algorytm ten można zaprogramować za pomocą jednej pętli:

```
  int dlugosc = 1;
  for (int i=1; i<1000; i++)
  {
    if ( sprawdz( x[i],y[i],r[i], x[i-1],y[i-1],r[i-1] ))
      dlugosc = dlugosc + 1;
    else
    {
      cout << "Dlugosc kolejnego lancucha = " << dlugosc <<
"\n";
      dlugosc = 1;
    }
  }
  cout << "Dlugosc kolejnego lancucha = " << dlugosc << "\n";
```

Zauważmy, że po zakończeniu pętli w zmiennej `dlugosc` przechowujemy długość ostatniego łańcucha, którą również powinniśmy wypisać jako wynik. Aby wyznaczyć maksymalną napotkaną długość łańcucha, wystarczy zmodyfikować powyższy program w następujący sposób:

```
  int dlugosc = 1, max_dlugosc = 1;
  for (int i=1; i<1000; i++)
  {
    if ( sprawdz( x[i],y[i],r[i], x[i-1],y[i-1],r[i-1] ))
    {
      dlugosc = dlugosc + 1;
    }
    else
    {
      if ( max_dlugosc < dlugosc ) max_dlugosc = dlugosc;
      cout << "Dlugosc kolejnego lancucha = " << dlugosc <<
"\n";
      dlugosc = 1;
    }
  }
  cout << "Dlugosc kolejnego lancucha = " << dlugosc << "\n";
  if ( max_dlugosc < dlugosc ) max_dlugosc = dlugosc;
  cout << "Dlugosc najdluzszego lancucha = " << max_dlugosc <<
"\n";
```

W zmiennej `max_dlugosc` przechowujemy bowiem maksymalną długość napotkanego dotychczas łańcucha. Do całkowitego rozwiązania zadania pozostaje zatem napisanie funkcji `sprawdz(x1,y1,r1,x2,y2,r2)`. Niech *D* oznacza odległość pomiędzy środkami okręgów. Możemy ją obliczyć za pomocą wzoru

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)\text{^}2},$$

a w języku C++ — za pomocą funkcji sqrt(...):

```
double D = sqrt( (x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) );
```

Mając wartość *D*, możemy łatwo badać wzajemne położenie okręgów. W celu wykrycia, czy mają one punkt wspólny, warto rozpatrzyć dwa przypadki:

$D \leq r_1$, czyli gdy środek drugiego okręgu leży wewnątrz pierwszego okręgu lub na nim,

$D > r_1$.

W obu przypadkach, aby oba okręgi miały wspólny punkt, promień drugiego okręgu nie może być zbyt mały ani też nie może być zbyt duży. Łatwo sprawdzić, że w pierwszym przypadku, aby okręgi miały co najmniej jeden punkt wspólny, muszą zachodzić nierówności:

$$r_1 - D \leq r_2 \leq r_1 +.$$

Z kolei w drugim przypadku wystarczy sprawdzić nierówności:

$$D - r_1 \leq r_2 \leq r_1 +.$$

Zaprogramowanie sprawdzenia powyższych nierówności pozostawimy czytelnikowi jako ćwiczenie.

Zadanie 80.

Stosując standardowe metody wczytywania danych z pliku tekstowego, możemy umieścić liczby z pliku wejściowego w tablicy, na przykład w tablicy o poniższej deklaracji:

```
int boki[500];
```

W każdym z zadań uzyskanie rozwiązania wymaga sprawdzania, czy podane trzy liczby dodatnie mogą być bokami trójkąta lub trójkąta prostokątnego. Zacznijmy zatem od omówienia tego zagadnienia. Liczby a, b, c są długościami boków trójkąta, gdy suma każdych dwóch z nich jest większa od trzeciej liczby. Wykorzystamy tę własność w funkcji `trojkat`, zwracającej `true`, gdy argumenty a, b, c są możliwymi długościami boków trójkąta:

```
bool trojkat(int a, int b, int c){
    return ((a+b>c)&&(a+c>b)&&(b+c>a));
}
```

Dzięki twierdzeniu Pitagorasa wiemy, że liczby a, b, c są bokami trójkąta prostokątnego, gdy kwadrat największej z liczb jest równy sumie kwadratów dwóch pozostałych liczb. Skorzystamy z tej własności w następujący sposób:

- najpierw zmienimy kolejność liczb a, b i c tak, aby a była największa;
- następnie sprawdzimy, czy $a^2=b^2+c^2$.

Ideę tę można zaimplementować przy pomocy poniższej funkcji:

```
bool prostokatny(int a, int b, int c)
{   int d;
    if (a<b) {d=b; b=a; a=d;}
    if (a<c) {d=c; c=a; a=d;}
    return (a*a==c*c+b*b);
}
```

80.1.

Mając do dyspozycji funkcję `prostokątny`, nie będziemy mieli większego problemu z rozwiązaniem zadania 1. Ponieważ mamy sprawdzić jedynie trójki kolejnych liczb, wystar-

czy utworzyć pętlę sprawdzającą możliwość utworzenia trójkąta z boków o długościach boki[i], boki[i+1] i boki[i+2] dla i=0,1,…,997, jednocześnie wypisując wartości boki[i], boki[i+1] i boki[i+2] za każdym razem, gdy utworzenie trójkąta będzie możliwe. Poniżej podajemy implementację tej metody:

```
for(int i=0; i<998;i++)
    if (prost(boki[i],boki[i+1],boki[i+2])){
        out << boki[i]<< ", " << boki[i+1]<<", "<<bo-
ki[i+2]<<";  ";
    }
```

gdzie `out` oznacza obiekt typu ofstream (biblioteka fstream) zidentyfikowany z plikiem wynikowym.

80.2.

Aby znaleźć boki tworzące trójkąt o największym obwodzie, możemy sprawdzić wszystkie trzyelementowe podzbiory zbioru B=`{boki[0], boki[1],…,boki[999]}`. Wymaga to przejrzenia $\binom{n}{3}$ różnych trójek liczb dla n=1000. Tego typu rozwiązanie przeglądające wszystkie możliwości prześledzimy w omówieniu rozwiązania zadania 3. Natomiast trójkąt o największym obwodzie spróbujemy znaleźć sprytniej.

Wszystkie trójkąty, których długości boków są trzema różnymi liczbami ze zbioru B=`{boki[0],…,boki[999]}`, nazwijmy *trójkątami dopuszczalnymi*.

Ograniczmy na chwilę nasze rozważania do takich trójkątów dopuszczalnych, w których *najdłuższym* bokiem jest boki[i] dla pewnego ustalonego i. Zauważmy, że jeśli istnieje choć jeden trójkąt dopuszczalny, w którym boki[i] jest najdłuższym bokiem, wówczas największy obwód takiego trójkąta (tzn. trójkąta z najdłuższym bokiem boki[i]) uzyskamy dobierając oprócz boki[i] dwie największe liczby ze zbioru B spośród liczb mniejszych od boki[i]. Rzeczywiście, skoro jakieś dwie liczby x,y∈B mniejsze od boki[i] spełniają warunek x+y>boki[i], to suma dwóch największych liczb z B mniejszych od boki[i] również będzie większa od boki[i].

Z powyższej obserwacji wynika, że gdybyśmy posortowali wszystkie elementy zbioru B, wówczas trójkąt o największym obwodzie znaleźlibyśmy wśród trójkątów, których boki są trzema **sąsiednimi** liczbami w (posortowanym) ciągu. Poniżej prezentujemy rozwiązanie oparte na tej idei. Najpierw uporządkujemy liczby w tablicy `boki`, stosując proste sortowanie bąbelkowe:

```
int ile, pom;
ile=1000;
for (int i=ile-1; i>0; i--)
    for (int j=0; j<i; j++)
        if (boki[j]>boki[j+1]){
            pom=boki[j]; boki[j]=boki[j+1]; boki[j+1]=pom;
        }
```

Następnie zaczynając od największej liczby w B, znajdującej się w `boki[999]`, szukamy największej liczby, która może utworzyć długości boków trójkąta wraz z dwiema liczbami poprzedzającymi ją w (posortowanej) tablicy:

```
bool brak = true;
```

```
i=ile;
while(brak && i>1){
    i--;
    if (trojkat(boki[i],boki[i-1],boki[i-2]))
        brak=false;
}
maxObw = boki[i]+boki[i-1]+boki[i-2];
```

Obwód największego trójkąta zostanie w efekcie zapisany w zmiennej `maxObw`, o ile zmienna `brak` przyjmie wartość `false` (co ma miejsce dla danych z pliku `dane_trojkaty.txt`).

Rozwiązanie powyższe jest szybsze od przeglądania wszystkich trójek liczb z B, pomimo tego, że stosujemy stosunkowo wolną metodę sortowania. Wynika to z tego, że sortowanie wymaga uruchomienia dwóch zagnieżdżonych pętli i wykonania w związku z tym liczby operacji proporcjonalnej do n^2 dla n=1000. Natomiast liczba wszystkich podzbiorów trzyelementowych zbioru B jest proporcjonalna do n^3.

Na koniec podkreślmy, że maksymalną liczbę punktów można uzyskać również za rozwiązanie mniej efektywne (np. przeglądające wszystkie trzyelementowe podzbiory B). Rozmiar danych jest na tyle mały, że program implementujący takie rozwiązanie wykona się bardzo szybko na standardowym komputerze. Bardziej wydajne rozwiązanie podaliśmy, aby zwrócić uwagę na różnicę w efektywności różnych rozwiązań algorytmicznych tego samego problemu. Poza tym przy sprawdzaniu wszystkich trójek liczb też się łatwo pomylić (więcej o tym w opisie rozwiązania zadania 3).

80.3.

Przypomnijmy, że w pliku `dane_trojkaty.txt` żadna liczba nie występuje więcej niż jeden raz. Aby ustalić liczbę trójkątów, które można utworzyć z mających różne długości odcinków ze zbioru `{boki[0],…,boki[999]}`, zastosujemy proste rozwiązanie, w którym w trzech zagnieżdżonych pętlach sprawdzimy każdy podzbiór trzyelementowy zbioru `{boki[0],…,boki[999]}`.[9] Poniżej można zobaczyć odpowiedni fragment kodu programu:

```
int ileT=0,ile=1000;
for (int i=0; i<ile-2; i++)
    for (int j=i+1; j<ile-1; j++)
        for (int k=j+1; k<ile;k ++){
            if (trojkat(boki[i],boki[j],boki[k])){
                ileT++;
            }
        }
```

Po wykonaniu zagnieżdżonych pętli for zmienna `ileT` przechowywać będzie liczbę różnych (nieprzystających) trójkątów, których długości boków są różnymi elementami zbioru B. Wyjaśnijmy, że wartości początkowe zmiennych *j* i *k* w pętli for zostały dobrane w taki sposób, aby każda trójka liczb była rozważana tylko raz. (Gdyby wartości początkowe zmiennych *i, j, k* były równe 0, wówczas każdą trójkę liczb rozważalibyśmy sześć razy; co więcej dopuszcza-

[9] Podobnie jak w zadaniu 2 również tutaj możliwe jest rozwiązanie znacznie szybsze od takiego, w którym sprawdzamy każdą trójkę liczb z pliku wejściowego. Jednak rozwiązanie takie wykracza poza zakres wymagań maturalnych.

libyśmy użycie liczby więcej niż raz w tym samym trójkącie, co jest niezgodne z treścią zadania).

Zadanie 81.

Do rozwiązania polecenia z każdego punktu piszemy oddzielny program. W każdym z nich w pętli 100 razy czytamy 6 wartości całkowitych z pliku i wykonujemy obliczenia zależne od konkretnego zadania. We wszystkich podanych poniżej przykładach współrzędne te są wczytane odpowiednio do zmiennych: *xa*, *ya*, *xb*, *yb*, *xc* oraz *yc*.

81.1.

Aby policzyć w ilu przypadkach wszystkie punkty A, B i C leżą w I ćwiartce układu współrzędnych, należy sprawdzić, czy każda ze współrzędnych tych punktów jest większa od zera. Jeżeli ten warunek jest spełniony, zwiększamy licznik o 1.

Przykładowy fragment kodu programu do zliczania takich przypadków:

```
if ((xa>0) && (ya>0) && (xb>0) && (yb>0) && (xc>0) && (yc>0)) {
    liczba++;
}
```

81.2.

Punkty A, B, C leżą na jednej prostej dokładnie w podanej kolejności, jeżeli |AB| + |BC| = |AC|. Przekształcając powyższy warunek aby uwzględnić wszystkie możliwe kombinacje kolejności punktów, otrzymujemy: suma długości odcinków |AB|, |AC| i |BC| musi być równa podwojonej długości najdłuższego z tych odcinków.

Przykładowy fragment kodu programu zliczający wiersze z punktami na jednej prostej:

```
dl1 = sqrt((xc-xb)*(xc-xb)+(yc-yb)*(yc-yb));
dl2 = sqrt((xc-xa)*(xc-xa)+(yc-ya)*(yc-ya));
dl3 = sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb));
if (dl1>dl2) {
    maks = dl1;
} else {
    maks = dl2;
}
if (dl3>maks) {
    maks = dl3;
}
if (2*maks == dl1+dl2+dl3) {
    liczba++;
}
```

Z teoretycznego punktu widzenia algorytm jest poprawny. W praktyce jednak ze względu na sposób zapisu liczb rzeczywistych w komputerze (prawie zawsze niedokładny) rozwiązanie to nie zawsze będzie działało poprawnie.

W podanym pliku `wspolrzedne.txt` jest wiersz (99):

-98 2 -86 10 -41 40

Wykonując obliczenia według powyższego algorytmu na 32 bitowych zmiennych rzeczywistych (typu `double`), otrzymamy następujące długości boków:

dl1 = 54,083269131959838
dl2 = 68,505474233815804
dl3 = 14,422205101855956

Suma dwóch krótszych (*dl1+dl3*) jest zatem równa 68,505274233815794 i jest różna od najdłuższego (*dl2*) o 0,00000000000001 czyli (10^{-14}).

Współliniowość można też stwierdzić, sprawdzając współczynniki kierunkowe prostych przechodzących przez punkty A i B oraz B i C, czyli sprawdzając równość $\frac{yb-ya}{xb-xa}=\frac{yc-yb}{xb-xa}$. Wzór ten jednak nie nadaje się do wykorzystania, gdy jedna z prostych jest pionowa, ponieważ w tym wzorze otrzymujemy zero w mianowniku. Odpowiedni do naszych celów jest wzór w postaci przekształconej: $(yb-ya)\cdot(xc-xb)=(yc-yb)\cdot(xb-xa)$. Zatem odpowiedni fragment programu mógłby wyglądać następująco:

```
if ((yb-ya)*(xc-xb) == (yc-yb)*(xb-xa)) {
    liczba++;
}
```

W przypadku obliczania współczynników nachylenia prostych błędy mogą się pojawić w wierszach 36 (prosta *x=36*):

36 24 36 -24 36 0

Oraz 88 (prosta (*y=-25*):

27 -25 9 -25 -9 -25

81.3.

Aby znaleźć trójkąt o największym obwodzie, obliczamy kolejno obwód każdego trójkąta i porównujemy z najdłuższym dotychczas znalezionym. Jeżeli nowo obliczony jest większy, to zapamiętujemy go.

```
obwod = sqrt((xc-xb)*(xc-xb)+(yc-yb)*(yc-yb))
       +sqrt((xc-xa)*(xc-xa)+(yc-ya)*(yc-ya))
       +sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb));
if (obwod>najdluzszy) {
    najdluzszy = obwod;
}
```

81.4.

Aby policzyć trójkąty prostokątne, wykorzystujemy twierdzenie Pitagorasa (kwadrat długości najdłuższego boku jest równy sumie kwadratów długości dwóch pozostałych). Ponieważ nie wiemy, który z boków jest najdłuższy, sprawdzamy trzy podobne warunki, przyjmując za najdłuższy za każdym razem inny bok. Ponieważ mamy do czynienia ze stosunkowo małymi liczbami (najdłuższy możliwy bok ma długość mniejszą niż 300), możemy do obliczeń stosować liczby całkowite. Nie mamy też potrzeby obliczać pierwiastków z sumy pól kwadratów.

Takie rozwiązanie może wyglądać następująco:

```
kw1 = (xc-xb)*(xc-xb)+(yc-yb)*(yc-yb);
kw2 = (xc-xa)*(xc-xa)+(yc-ya)*(yc-ya);
kw3 = (xa-xb)*(xa-xb)+(ya-yb)*(ya-yb);
if ((kw1==kw2+kw3) || (kw2==kw1+kw3) || (kw3==kw1+kw2)) {
    liczba++;
}
```

Gdybyśmy porównywali długości boków z pierwiastkami sumy kwadratów pozostałych boków, to ze względu na sposób zapisu w komputerze liczb rzeczywistych uzyskamy na ogół błędne wyniki.

81.5.

Możemy zauważyć, że miejsce przecięcia przekątnych równoległoboku ABCD dzieli je na równe części, inaczej mówiąc, środki obydwu przekątnych są w tym samym punkcie. Zatem $xa + xc = xb + xd$ oraz $ya + yc = yb + yd$. Obliczamy zatem $xd = xa + xc - xb$ i $yd = ya + yc - yb$. Następnie, aby sprawdzić, czy tak wyliczony punkt D leży na prostej $y=x$, porównujemy ze sobą współrzędne punktu D.

Odpowiedni fragment programu mógłby wyglądać następująco:

```
xd = xa+xc-xb;
yd = ya+yc-yb;
if (xd == yd) {
    liczba++;
}
```

Zadanie 84.

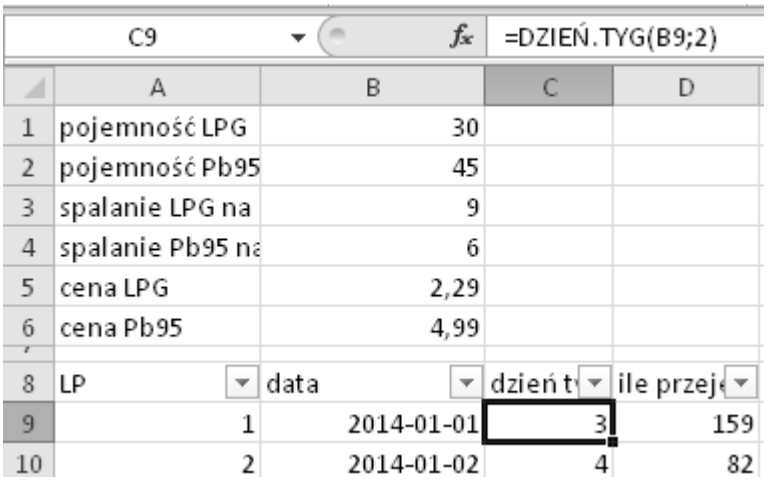
Zadanie to należy do typowych zadań symulacyjnych. Najtrudniejszą częścią takich zadań jest prawidłowe zasymulowanie powtarzających się zdarzeń, które są ze sobą powiązane. Zadanie można rozwiązać zarówno poprzez napisanie odpowiedniego programu, jak i za pomocą arkusza kalkulacyjnego. Poniżej przedstawimy wskazówki do rozwiązania zadania w arkuszu.

Rozwiązanie rozpoczynamy od wczytania pliku z danymi (dat oraz liczby przejechanych km) oraz od wpisana do arkusza stałych elementów zadania tzn. pojemności zbiornika na paliwo LPG, pojemności zbiornika na paliwo Pb95, spalania paliwa LPG, spalania paliwa Pb95 oraz ceny jednego litra LPG i jednego litra Pb95.

Ponieważ w zadaniu pojawiają się dni tygodnia, dobrze jest ustalić jedną kolumnę przechowującą dzień tygodnia. W każdym dniu symulacji zachodzi wiele możliwych zdarzeń (np. zużywanie paliwa, tankowanie), dlatego wskazane jest rozpisanie tego, co dzieje się podczas całego dnia. Bardzo ważne jest przeczytanie całej treści zadania, żeby wiedzieć, jaka jest kolejność zdarzeń oraz jakie elementy będą przydatne przy udzielaniu odpowiedzi na pytania.

Można ustalić, że kolejne kolumny w arkuszu to na przykład: data (wczytana z pliku z danymi), dzień tygodnia, liczba przejechanych kilometrów (wczytana z pliku z danymi), poranny stan paliwa w zbiorniku LPG, poranny stan paliwa w zbiorniku Pb95, liczba spalonych litrów (odpowiednio LPG i Pb95), stan w obu zbiornikach po podróży, tankowanie LPG i Pb95 oraz stan wieczorny w obu zbiornikach (po ewentualnym tankowaniu). Być może jest to rozbudowane, ale ułatwia kontrolowanie symulacji.

Aby wyznaczyć dzień tygodnia odpowiadający danej dacie, korzystamy z funkcji `DZIEŃ.TYG`, która zwraca liczbę od 1 do 7 na podstawie danej daty. Należy pamiętać, że drugi parametr pozwala dokładnie sprecyzować, w jaki sposób numerować dni. W naszym przypadku parametr ten jest równy 2, co odpowiada numeracji od poniedziałku (= 1) do niedzieli (= 7).



C9	fx	=DZIEŃ.TYG(B9;2)	
	A	B	C	D
1	pojemność LPG	30		
2	pojemność Pb95	45		
3	spalanie LPG na	9		
4	spalanie Pb95 na	6		
5	cena LPG	2,29		
6	cena Pb95	4,99		
8	LP	data	dzień t	ile przej
9	1	2014-01-01	3	159
10	2	2014-01-02	4	82

W pierwszym dniu rano w zbiorniku z LPG znajdowało się 30 litrów paliwa, a w zbiorniku z Pb95 znajdowało się 45 litrów paliwa. Takie liczby wpisujemy w arkuszu w komórkach `E9` oraz `F9`. Następnym elementem symulacji jest podróż pana Binarnego (wypełniamy komórki `F9` oraz `G9`): jeżeli w zbiorniku z LPG było więcej niż 15 litrów, to do jazdy wykorzystywał on tylko paliwo LPG, w przeciwnym razie połowę trasy pokonywał, korzystając z paliwa Pb95, a połowę — z paliwa LPG. Zauważmy, że te wyniki mają być zaokrąglone do dwóch miejsc po przecinku. Liczba spalonego paliwa LPG odpowiada formule:

```
=ZAOKR(JEŻELI(E9>15;$B$3*D9/100;$B$3*D9/200);2)
```

zaś liczba spalonego paliwa Pb95 odpowiada formule:

```
=ZAOKR(JEŻELI(E9>15;0;$B$4*D9/200);2)
```

Formuły te uwzględniają adresowanie bezwzględne, co znacznie ułatwia symulację. W komórce `I9` obliczamy ilość paliwa LPG pozostałego po przejechaniu trasy danego dnia: `=E9-G9`, analogicznie ilość paliwa pozostałego w zbiorniku Pb95 obliczamy komórce `J9`: `=F9-H9`.

Następny krok to tankowanie samochodu. Paliwo LPG było tankowane do pełna wtedy, gdy w zbiorniku LPG znajdowało się go mniej niż 5 litrów, i było niezależne od dnia tygodnia. Odpowiada to formule wpisanej w komórkę `K9`: `=JEŻELI(I9<5;$B$1-I9;0)`.

Tankowanie paliwa Pb95 odbywało się w każdy czwartek, jeśli w zbiorniku znajdowało się mniej niż 40 litrów paliwa. Tankowanie również odbywało się do pełna. Tutaj trzeba zwrócić uwagę na połączenie dwóch warunków w instrukcji `JEŻELI` za pomocą funkcji `ORAZ`: `=JEŻELI(ORAZ(C9=4;J9<40);$B$2-J9;0)`

Aby dokończyć symulację pojedynczego dnia, należy ustalić wieczorny stan paliwa w obu zbiornikach, tzn. uzupełnić komórki `M9`: `=I9+K9` oraz `N9`: `=J9+L9`, a na koniec przenieść te wartości na poranny stan zbiorników, czyli uzupełnić komórki `E10` i wpisać `=M9`, a w `F10` wpisać `=N9`. Teraz wystarczy skopiować formuły we wszystkich kolumnach aż do dnia 31 grudnia 2014 r. Tak przygotowany arkusz pozwoli odpowiedzieć na pytania postawione w zadaniach.

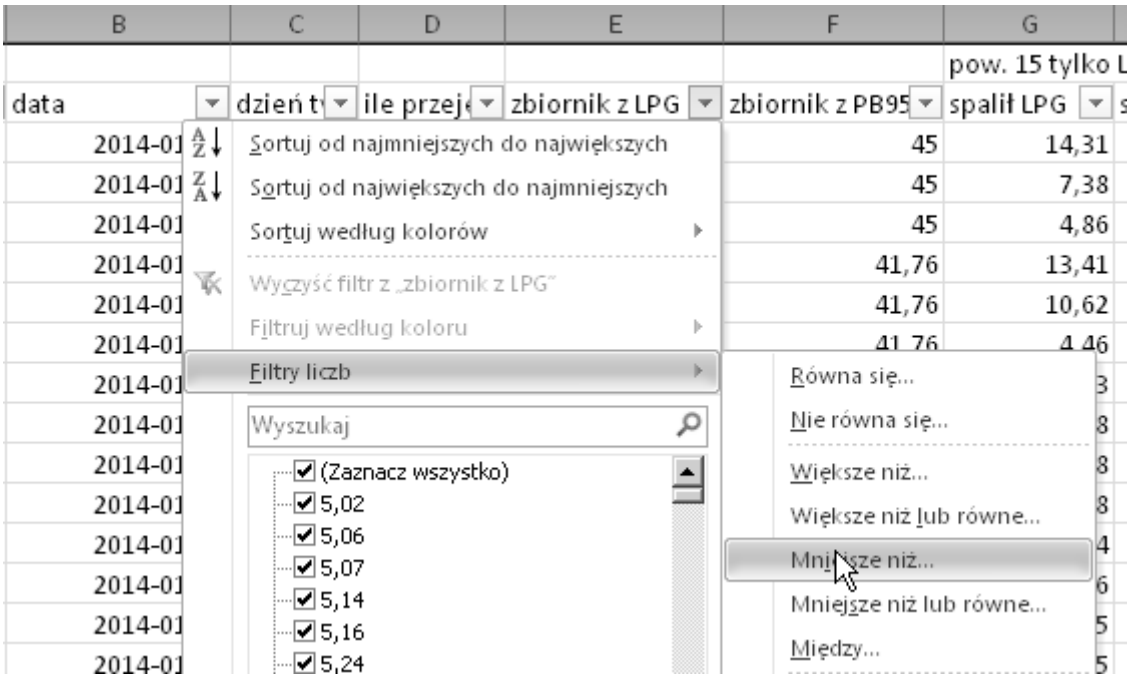
84.1.

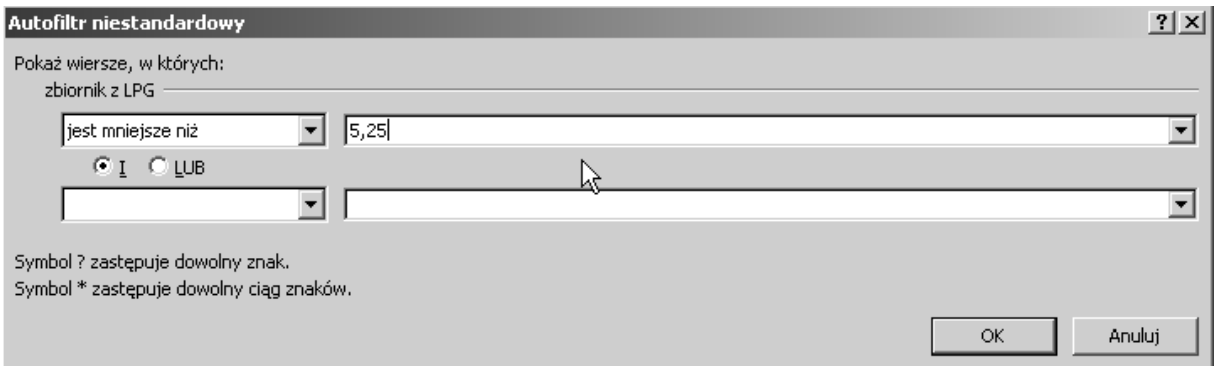
Aby wykonać to zadanie, możemy skorzystać z funkcji `LICZ.JEŻELI`. Dla liczby tankowań LPG zliczamy, ile komórek w kolumnie `K` jest większych od zera: `=LICZ.JEŻELI(K9:K373;">0")`. Analogicznie obliczamy liczbę tankowań Pb95.

Liczba dni, w które pan Binarny podczas jazdy korzystał wyłącznie z paliwa LPG, jest równa liczbie dni, w które Pan Binarny w ogóle nie korzystał z paliwa Pb95. Te dane znajdują się w kolumnie `H`, zatem zastosowanie ma np. `=LICZ.JEŻELI(dane!H9:H373;0)`. Pamiętajmy o tym, że jest to tylko jedna z wersji rozwiązania tego zadania.

84.2.

To zadanie można rozwiązać również na wiele sposobów. Jednym z nich jest filtrowanie danych (w karcie *Dane* wybieramy narzędzie *Filtruj*). Następnie w kolumnie `E`, która przechowuje stan paliwa LPG rano, rozwijamy opcję filtrowania. Wybieramy *Filtry liczb*, a potem opcję *Mniejsze niż*. W oknie autofiltru niestandardowego podajemy zadaną liczbę (5,25).



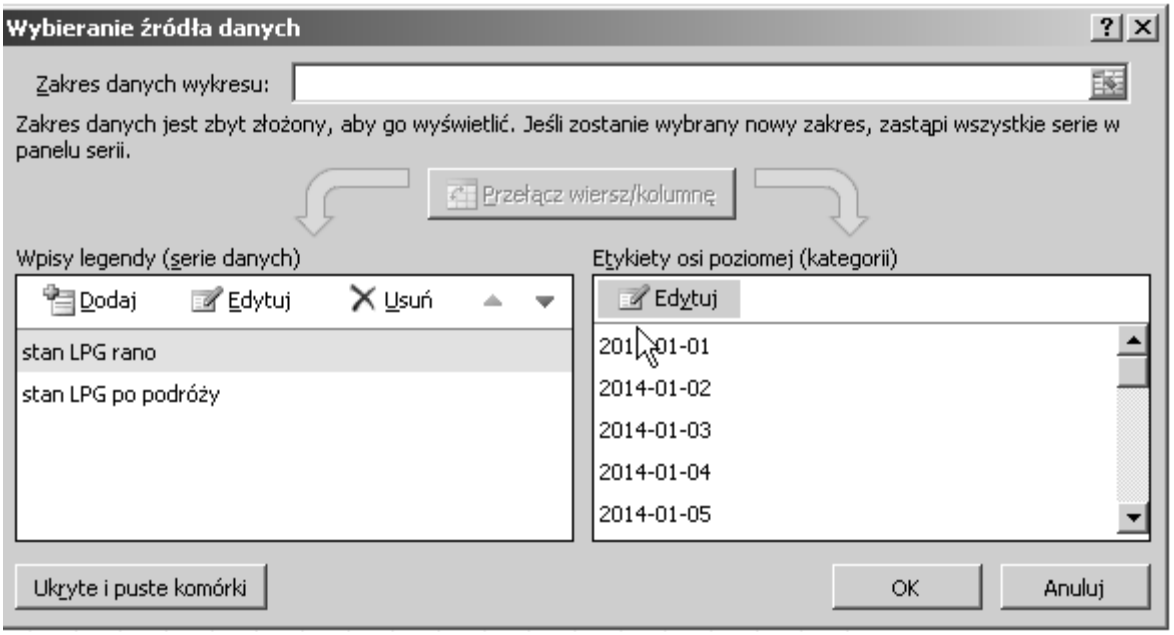


W rezultacie otrzymujemy wszystkie wiersze, w których wartość w kolumnie `E` jest mniejsza niż 5,25. Ponieważ mamy podać pierwszy dzień, w którym rano w zbiorniku LPG było mniej niż 5,25 litra, jest to dzień znajdujący się w pierwszym wierszu otrzymanego zestawienia.

	data	dzień t	ile przej	zbiornik z LPG	zbio
26	2014-01-26	7	93	5,24	
38	2014-02-07	5	35	5,16	
47	2014-02-16	7	97	5,06	
120	2014-04-30	3	35	5,14	
176	2014-06-25	3	97	5,07	
212	2014-07-31	4	41	5,02	
221	2014-08-09	6	30	5,06	

84.3.

Wykres tworzymy korzystając z karty *Wstawianie* → *Wykresy* → *Kolumnowy*. Trzeba pamiętać o dobraniu właściwych danych do wykresu (kolumny i zakres wierszy). W tym przypadku będą to komórki: `$E$9:$E$38` (stan zbiornika LPG przed podróżą) oraz `$I$9:$I$38` (stan zbiornika LPG po podróży, ale przed tankowaniem). Wykres powinien być czytelny, co oznacza, że etykiety osi X powinny być datami. Dane ustawiamy w następujący sposób: klikamy prawym przyciskiem myszy na oś poziomą i z rozwijanego menu wybieramy opcję *Zaznacz dane*. Następnie przechodzimy do edycji etykiet osi poziomej, których zakres możemy podać wprost z arkusza (komórki `$B$9:$B$39`).

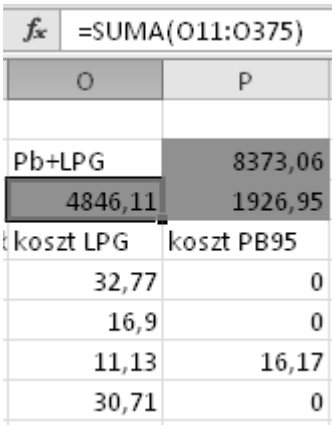


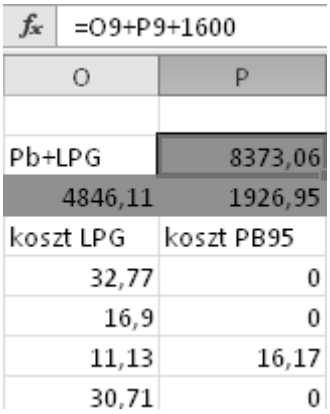
Pamiętajmy również o opisie kolumn — w narzędziach wykresu można wybrać odpowiedni typ wykresu. Przygotowany wykres zapisujemy w odpowiednim pliku i oczywiście pozostawiamy w arkuszu jako realizację komputerową rozwiązania zadania.

84.4.

Podczas obliczania kosztu eksploatacji z wykorzystaniem instalacji gazowej obliczenia możemy prowadzić w kolejnych kolumnach arkusza. Przypomnijmy, że koszty obu paliw zostały wpisane już do arkusza do komórek `B5` oraz `B6` jako elementy stałe symulacji. Ponieważ wszystkie wyniki mają być zaokrąglane do dwóch miejsc po przecinku, będziemy korzystać z funkcji `ZAOKR()`, której drugim argumentem jest liczba cyfr, do której ma zostać zaokrąglona liczba podawana jako pierwszy argument tej funkcji.

Koszt paliwa LPG dla pojedynczego dnia obliczamy w kolumnie `O` formułą `=ZAOKR(G11*$B$5;2)`, a koszt paliwa Pb95 — w kolumnie `P` analogiczną formułą `=ZAOKR(H11*$B$6;2)`. Te formuły kopiujemy do komórek, odpowiednio `O12:O375`, `P12:P375`, a następnie obliczamy odpowiednie sumy kosztów: `SUMA(O11:O375)` oraz `SUMA(P11:P375)`. Aby obliczyć roczny koszt eksploatacji, dodajemy do siebie obie obliczone sumy oraz koszt instalacji gazowej.





Podczas obliczania w rozważanym okresie kosztów eksploatacji samochodu wykorzystującego tylko paliwo Pb95 sumujemy liczbę przejechanych kilometrów. Kilometry przechowujemy w kolumnie `D`, stąd np. w komórce `D7` formuła `=SUMA(D11:D375)`. W komórce `D8` obliczamy, ile litrów paliwa zostałoby spalonych (zgodnie ze wzorem podanym w treści): `=ZAOKR(D7*B4/100;2)`. Na końcu obliczamy koszt paliwa: `=ZAOKR(D8*B6;2)`.

Zadanie 85.

W zadaniu symulacyjnym rozpoczniemy od rozplanowania pól arkusza.Najlepiej zrobić to dosyć szczegółowo.W odpowiednich komórkach utworzymy odpowiednie formuły odzwierciedlające opisane w zadaniu wydarzenia. Dla stałych wykorzystywanych w formułach, będziemy w formułach stosować adresy bezwzględne.

Zadanie dotyczy okresu od 23.04.2014 do 29.09.2014. Wypiszemy wszystkie daty w kolejnych wierszach i ponumerujemy dni wypasu owiec, a w komórce nad tabelą zapiszemyliczbę owiec w stadzie.

85.1.

Aby podać liczbę litrów mleka, jaką uzyskał baca od swojego stada owiec w okresie od 23.04.2014 do 29.09.2014, ustawimy wartość początkową na 0,50 l, a następnie utworzymy formułę, za pomocą której obliczymy ilość mleka od jednej owcy w kolejnych dniach, uwzględniając jej przyrost w kolejnych 7-dniowych okresach do dnia 24 czerwca włącznie:

```
=JEŻELI(MOD(A5;7)=1;ZAOKR(1,04*D4;2);D4)
```

Od 25 czerwca ilość mleka będzie co 7 dni zmniejszać się o 10%:

```
=JEŻELI(MOD(A67;7)=1;ZAOKR(0,9*D66;2);D66)
```

D5 =JEŻELI(MOD(A5;7)=1;ZAOKR(1,04*D4;2);D4)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		liczba owiec		600									
2													
3	dni wypasu	data	miesiąc	mleko dzienne 1 owcy	mleko	dzień tyg	popyt na sery	sery wyrobione z mleka udojonego poprzedniego dnia	sery uwędzone	gotowe sery	stan	sprzedane	czy dość
4	1	2014-04-23	4	0,50	300,00	3	0	0	0	0	0	0	1
5	2	2014-04-24	4	0,50	300,00	4	0	50	0	0	0	0	1
6	3	2014-04-25	4	0,50	300,00	5	0	50	0	0	0	0	1
7	4	2014-04-26	4	0,50	300,00	6	0	50	0	0	0	0	1
8	5	2014-04-27	4	0,50	300,00	7	0	50	0	0	0	0	1
9	6	2014-04-28	4	0,50	300,00	1	0	50	50	0	0	0	1
10	7	2014-04-29	4	0,50	300,00	2	36	50	50	50	50	36	1
11	8	2014-04-30	4	0,52	312,00	3	36	50	50	50	64	36	1

W kolumnie `E` znajdzie się ilość mleka otrzymanego w danym dniu od całego stada (`=D$1*D4`). Na koniec podsumowujemy cały okres wypasu: `=SUMA(E4:E163)`.

85.2.

Popyt turystów na ser zależy od dnia tygodnia: w sobotę i w niedzielę jest popyt na 100 serów, zaś w pozostałe dni tygodnia — na 36 serów.

W kolumnie `F` wyznaczamy dla każdej wypisanej daty dzień tygodnia (=DZIEŃ.TYG(B10;2)), a następnie w kolumnie `G` opisujemy popyt na sery: =JEŻELI(F10<6;36;100). I znowu należy podsumować kolumnę G: =SUMA(G4:G163).

85.3.

Baca wytwarza z każdych 6 litrów mleka jeden oscypek: `=ZAOKR.DO.CAŁK(E9/6)`. Proces wytwarzania sera trwa 6 dni, czyli sery z udoju z dnia 23.04 są gotowe do sprzedaży 29.04, w tym dniu pojawiają się pierwszy raz turyści. Baca gromadzi wyprodukowane sery, a turyści je regularnie kupują. Stan serów (kolumna `L`) obliczamy, odejmując od stanu z poprzedniego dnia sery zakupione i dodając do tego stanu sery tego dnia (gotowe do sprzedaży): `=L9-K9+J10`. Liczba sprzedanych serów zależy od popytu turystów i liczby serów gotowych do sprzedaży w bacówce (`stan`): `=JEŻELI(G10<=L10;G10;L10)`.

Jeżeli popyt na sery (`G10`) jest mniejszy lub równy liczbie gotowych serów w bacówce (`L10`), to sprzedawana jest taka liczba serów, na jaką jest popyt turystów, w przeciwnym razie sprzedawana jest liczba serów będących w bacówce na stanie.

W kolumnie `M` sprawdzimy, kiedy baca miał po raz pierwszy w sezonie mniej gotowych serów niż chcieli kupić turyści: `=JEŻELI(G10<=L10;1;0)`. I już łatwo policzyć w sezonie liczbę dni, w których baca miał mniej gotowych serów niż chcieli kupić turyści, licząc liczbę zer: `=LICZ.JEŻELI(M4:M163;0)`.

85.4.

W zadaniu wykonamy zestawienie, w którym dla każdego miesiąca podamy liczbę sprzedanych oscypków w każdym miesiącu (od kwietnia do września włącznie). Symulację dotyczącą liczby sprzedanych serów (kolumna `K`) mamy gotową. Teraz wybierzemy z daty miesiąc: `=MIESIĄC(B4)` i za pomocą sum częściowych podsumujemy dla każdego miesiąca liczbę serów. Do tak wykonanego zestawienia tworzymy wykres kolumnowy.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	liczba owiec			600,0000									
2													
3	data	miesiąc	dni wypas	mleko dzienne 1 owcy	mleko	dzień tyg	popyt na sery	wytworzone	sery uwędzone	gotowe sery	stan	sprzedane	czy dość
12		**4 Suma**										72	
44		**5 Suma**										1626	
75		**6 Suma**										1656	
107		**7 Suma**										1628	
139		**8 Suma**										1243	
169		**9 Suma**										586	
170					41406,00		8360						
171		**Suma końcowa**										6811	

85.5.

Aby podać najmniejszą liczbę owiec, jaką musiałby posiadać baca, aby każdego dnia przez cały okres wypasania owiec zaspokajać popyt turystów na oscypki, będziemy tak zmieniać liczbę owiec w stadzie, aby w kolumnie `M` (czy dosyć) pojawiły się tylko jedynki.

85.6.

Liczbę serów gotowych, przy założeniu, że ser tworzony byłby tylko z mleka owczego, już mamy. Po usunięciu zbędnych kolumn liczby serów wyrabianych tylko z mleka owczego znajdą się w kolumnie `F`, wystarczy więc podsumować kolumnę `F`. Ilość mleka uzyskiwana od stada zapisana jest w kolumnie E, liczba wyrabianych serów z 80% mleka owczego i 20% mleka krowiego to: `I5=ZAOKR.DO.CAŁK(E4/0,8/6)`, zaś z 60% mleka owczego i 20% mleka krowiego to: `L5=ZAOKR.DO.CAŁK(E4/0,6/6)`. Teraz zachowujemy odpowiednie przesunięcie pomiędzy datami, kiedy sery były wyrobione i gotowe do sprzedaży, na koniec sumujemy kolumny `K` i `N`.

Zadanie 86.

Zaprezentujemy rozwiązanie zadania z pomocą programu napisanego w języku C/C++. Dane wejściowe wczytamy do tablic `naz` i `tab`, wprowadzając wcześniej stałe `nokr` (liczba okręgów) i `nkom` (liczba komitetów):

```
#define nokr 20
#define nkom 5
string naz[nokr];
int tab[nokr][nkom]
```

Po wczytaniu danych `naz[i]` będzie zawierać nazwę okręgu *i*, a `tab[i][j]` — liczbę głosów oddanych w tym okręg na komitet *Kj*.

Rzeczywiste nazwy okręgów zapamiętamy w tablicy o deklaracji:

```
string naz[nokr];
```

Z uwagi na indeksowanie tablic od zera komitety oznaczać będziemy *K0, K1,...,K4* (przy wypisywaniu odpowiedzi do zadań uwzględnimy to, że rzeczywista numeracja to K1,…,K5).

86.1.

Dla zapisanych zgodnie z powyższym opisem danych wejściowych liczbę głosów w okręgu *i* możemy wyznaczyć jako `tab[i][0]+tab[i][1]+...tab[i][4]`. Wartości te możemy wyznaczyć w następującej pętli:

```
for(int i=0; i<nokr; i++) tabsum[i]=0;
for(int i=0; i<nokr; i++)
    for(int j=0; j<nkom; j++)
        tabsum[i]+= tab[i][j];
```

gdzie `tabsum[i]` reprezentuje liczbę głosów oddanych w okręgu *i*. Przypomnijmy: rzeczywistą nazwę okręgu *i* przechowujemy w `naz[i]`. Wykres sporządzimy natomiast w arkuszu kalkulacyjnym, wartości z tablicy `tabsum` można zresztą uzyskać, importując dane wejściowe do arkusza kalkulacyjnego i wyznaczając sumy w wierszach uzyskanego zestawienia. W niniejszym opisie koncentrujemy się na rozwiązaniu przy pomocy programu komputerowego, gdyż rozwiązywanie zadania 3 (a także zadań 4 i 5) w arkuszu kalkulacyjnym wydaje się bardzo pracochłonne.

86.2.

Na podstawie opisanej powyżej reprezentacji danych w tablicy `tab` poparcie komitetu *Kj* w okręgu *i* wyznaczymy jako

$P_{i,j}$ = 100% · tab[i][j] / (tab[i][0]+ tab[i][1]+…+ tab[i][4])

Zadanie sprowadza się więc do wyznaczenia dla każdego *j*=0,1,..,4 największej wartości spośród $P_{0,j}$, $P_{1,j}$,…, $P_{19,j}$. W naszym rozwiązaniu zadeklarujemy tablice

```
int maxnumer[nkom];
double maxproc[nkom];
```

Poniżej prezentujemy rozwiązanie, w którym przeglądamy wyniki w kolejnych okręgach (pętla zewnętrzna), przechowując w `maxproc[j]` największe poparcie procentowe wyznaczone dotychczas dla komitetu *Kj*, a w `maxnumer[j]` numer okręgu, w którym to poparcie zostało uzyskane:

```
double akt, sum;
for(int i=0; i<nkom; i++) {
        maxproc[i]=0.0; maxnumer[i]=0;
 }
 for(int i=0; i<nokr; i++){
    sum=0.0;
    for(int j=0; j<nkom; j++) sum+=tab[i][j];
    for(int j=0; j<nkom; j++){
        akt=tab[i][j]/sum;
        if (akt>maxproc[j]){
            maxproc[j]=akt;
            maxnumer[j]=i;
        }
    }
 }
```

Zauważmy, że w zmiennej `sum` wyznaczać będziemy

`(tab[i][0]+tab[i][1]+…+tab[i][4])`,

a w zmiennej `akt` wartość $P_{i,j}$. Nazwa okręgu będącego matecznikiem komitetu *Kj* będzie po wykonaniu powyższego fragmentu programu równa `naz[maxnumer[j]]`. Uwzględniając różnice w indeksowaniu komitetów, odpowiedź do zadania moglibyśmy wypisać na standardowym wyjściu np. tak:

```
cout << "Zadanie 2: " << endl;
for(int i=0; i<nkom; i++)
    cout << "K" << i+1 << ": " << naz[maxnumer[i]] << endl;
```

86.3.

Na użytek rozwiązań zadań 3, 4 i 5 zaimplementujemy wyznaczanie liczby mandatów zdobytych przez różne komitety wyborcze metodą Sainte-Laguë. Ponieważ będziemy stosować tę metodę wielokrotnie, utworzymy funkcję `fmandaty`, która dla wartości wejściowych:

- `ilek` — liczba komitetów wyborczych,
- `ilem` — liczba mandatów do podziału,
- `glosy[i]` — liczba głosów oddanych na komitet *Ki* dla $i \in [0, ilek - 1]$

umieści w `s[i]` liczbę mandatów przyznanych *i*-temu komitetowi dla $i \in [0, ilek - 1]$. Wartości $w_0,\ldots,w_{ilek-1}$ wyznaczane w metodzie Sainte-Laguë przechowywać będziemy jako `x[0],…,x[ilek-1]`. Przydzielając kolejne mandaty, aktualizować będziemy wartości współczynników w_i w tablicy `x` i liczbę przyznanych mandatów w tablicy `s`. Treść naszej funkcji jest następująca:

```
void fmandaty(int glosy[], int s[], int ilem, int ilek){
int indmax;
double x[nkom];

 for(int i=0; i<ilek; i++) {
    s[i]=0; x[i]=(double)glosy[i];
 }
 for(int i=0; i<ilem; i++){
    indmax=0;
    for(int j=1; j<ilek; j++){
        if (x[j]>x[indmax]) indmax=j;
    s[indmax]++;
    x[indmax]=glosy[indmax]/(2.0*s[indmax]+1);
 }
}
```

Podsumowując,

- pierwsza pętla „`for(int i=0;i<ilek;i++)`” inicjuje wartości współczynników w_i w tablicy `x` i liczby przyznanych mandatów w tablicy `s`;
- w pętli zewnętrznej „`for(int i=0;i<ilem;i++)`” przyznawane są kolejne mandaty;
- pętla wewnętrzna „`for(int j=1;j<ilek;j++)`” służy wyznaczeniu (w zmiennej `indmax`) numeru komitetu, któremu przyznany będzie kolejny mandat, za pętlą aktualizowane są odpowiednio tablice `s` i `x`.

Skoncentrujmy się teraz na rozwiązaniu zadania 3, którego dokonamy, z wykorzystaniem funkcji `fmandaty`. Do gromadzenia informacji o mandatach przyznanych we wszystkich okręgach wyborczych utworzymy tablicę

```
int wyniki[nokr][nkom];
```

Docelowo `wyniki[i][j]` powinno być równe liczbie mandatów przyznanych komitetowi *Kj* w okręgu o numerze *i*.

Korzystając z funkcji `fmandaty`, podział mandatów w okręgu *i* (czyli wartości `wyniki[i][j]` dla *j*=0,1,..,4) wyznaczymy, wywołując

```
fmandaty(tab[i],wyniki[i],20,5).
```

Wykonując takie wywołanie dla każdego $i \in [0,19]$, wypełnimy odpowiednio tablicę `wyniki`. Rozwiązanie zadania 3 sprowadza się wówczas wyznaczenia największej liczby wśród `wyniki[0][j],…,wyniki[19][j]` dla każdego $j \in [0,4]$. Napisanie odpowiednich fragmentów programu pozostawiamy czytelnikowi.

86.4.

Zaglądając do pliku wejściowego `dane_wybory.txt`, możemy zauważyć, że nazwy kolejnych okręgów to *A1,…,A5*; *B1,…,B5; C1,…,C5 oraz D1,…,D5*. A zatem:

- okręgi *A1,…,A5* mają numery: 0,..,4; okręgi *B1,…,B5* mają numery: 5,…,9; okręgi *C1,…,C5* mają numery: 10,…,14 , a okręgi *D1*,…,*D5* mają numery: 15,…,19.

Odpowiednie liczby mandatów możemy wyznaczyć z pomocą funkcji `fmandaty`.

Dla standardowego podziału mandatów możemy wykorzystać tablicę `wyniki` o zawartości opisanej w omówieniu rozwiązania zadania 3 (`wyniki[i][j]` to liczba mandatów przyznanych komitetowi *Kj* w okręgu *i*): liczba mandatów przyznanych komitetowi *Kj* w całym kraju jest równa sumie `wyniki[0][j]+…+wyniki[19][j]` .

Na potrzeby regionalnego podziału mandatów najpierw ustalimy liczbę głosów uzyskanych przez każdy z komitetów w poszczególnych regionach, a wyniki umieścimy w tablicy `tabR`:

```
#define nreg 4
int pocz,tabR[nreg][nkom];
for(int i=0; i<nreg; i++){
    for(int j=0; j<nkom; j++) tabR[i][j]=0;
    pocz=i*5;
    for(int k=0; k<nkom; k++)
        for(int j=pocz; j<pocz+5; j++)
           tabR[i][k]+=tab[j][k];
}
```

Następnie wyznaczymy liczbę mandatów w poszczególnych regionach za pomocą funkcji `fmandaty`:

```
int wynReg[nreg][nkom];
for(int i=0; i<nreg; i++){
    fmandaty(tabR[i],wynReg[i], 100, 5);
}
```

Na koniec pozostaje zsumowanie liczby mandatów dla każdego komitetu; liczba mandatów zdobytych przez *Ki* będzie równa `wynReg[0][i]+...+wynReg[nreg - 1][i]`.

86.5.

Oznaczmy komitet *Q* przez *K0*, a komitet *R* przez *K1*. Przyjmijmy, że liczba oddanych głosów i liczba mandatów do podziału są ustalone. Aby uzyskać odpowiedź na postawione pyta-

nie (najmniejsza liczba głosów, jaką musi uzyskać *K0*, aby zdobyć połowę mandatów), przeprowadzimy symulację wyników wyborów dla różnych wartości liczby głosów zdobytych przez komitet *K0*, zmieniających się od 1 do połowy liczby wszystkich oddanych głosów. Najmniejsza liczba głosów, przy której *K0* otrzyma połowę mandatów, wyznaczy odpowiedź na postawione pytanie. Gdyby metoda ta wymagała zbyt długich obliczeń, moglibyśmy ją zoptymalizować, znajdując odpowiedź metodą wyszukiwania binarnego (binarnie wyszukujemy najmniejszej liczby głosów, przy której *K0* uzyska połowę mandatów: wiedząc że wartość ta jest w przedziale [a,b], sprawdzamy, czy wystarczająca jest liczba głosów (a+b)/2, i w zależności od wyniku ograniczamy obszar poszukiwań do [1+(a+b)/2,b] lub [a, (a+b)/2]). Przy danych z zadania nie jest to jednak konieczne — obliczanie przydziału mandatów dla konkretnej liczby głosów wykonuje się bardzo szybko. Poniżej prezentujemy rozwiązanie, w którym:

- liczba oddanych głosów znajduje się w zmiennej `glosy`;
- pętla zewnętrzna przebiega przez trzy warianty zadania odpowiadające różnym liczbom mandatów do podziału (liczby te zapisane są w tablicy `lmandatow`);
- liczby głosów oddane na komitety *K0* i *K1* zapisujemy (przed wywołaniem `fmandaty`) w tablicy `testin`, a przyznane liczby mandatów funkcja `fmandaty` zapisuje w tablicy `testout`.

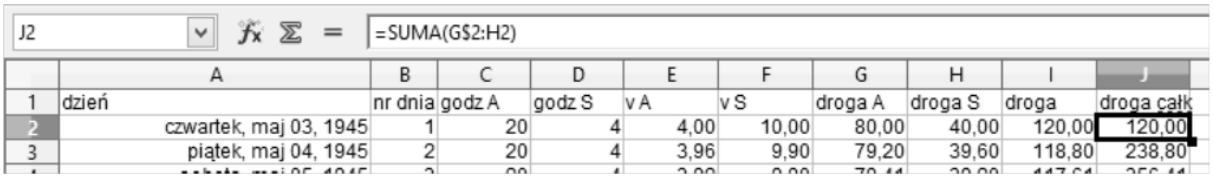
```
void zad5(){
 int testin[2], testout[2];
 int  glosow=100000, mandatow=20;
 int lmandatow[3]={20,40,100};

 cout << "Zadanie 5: " << endl;
 for(int j=0; j<3; j++){
    mandatow=lmandatow[j];
    for(int i=1; i<=glosow/2; i++){
        testin[0]=i;
        testin[1]=glosow-i;
        fmandaty(testin,testout, mandatow, 2);
        if (testout[0]==testout[1]){
            cout << "Dla m="<< mandatow/2 <<": " << i <<
endl;
            break;
        }
    }
 }
}
```

Zadanie 87.

Rozwiązanie rozpoczynamy od przygotowania tabeli, w której kolejnych kolumnach umieszczamy: datę (`dzień`), numer kolejnego dnia (`nr dnia`), informacje dotyczące danego dnia rejsu: liczbę godzin rejsu na silniku elektrycznym (`godz A`), liczbę godzin rejsu na silniku spalinowym (`godz S`), średnią prędkość okrętu na silniku elektrycznym (`v A`), jego średnią prędkość na silniku spalinowym (`v S`), drogę przebytą na silniku elektrycznym (`droga A`), drogę przebytą na silniku spalinowym (`droga S`), drogę przebytą danego dnia (`droga`)

oraz sumaryczną drogę przebytą od początku rejsu (`droga całk`). Początek takiej tabeli



	A	B	C	D	E	F	G	H	I	J
1	dzień	nr dnia	godz A	godz S	v A	v S	droga A	droga S	droga	droga całk
2	czwartek, maj 03, 1945	1	20	4	4,00	10,00	80,00	40,00	120,00	120,00
3	piątek, maj 04, 1945	2	20	4	3,96	9,90	79,20	39,60	118,80	238,80

będzie wyglądał następująco:

W wierszu opisanym „14 lipca” w kolumnach z liczbą godzin rejsu umieszczamy dwa zera, a w kolejnych dniach odpowiednio 0 i 11, odpowiadające dziennej liczbie godzin rejsu na silnikach elektrycznym i spalinowym:

73	piątek, lipiec 13, 1945	72	20	4	1,96	4,90	39,19	19,60	58,79	6180,10
74	***sobota, lipiec 14, 1945***	***73***	***0***	***0***	***1,94***	***4,85***	***0,00***	***0,00***	0,00	6180,10
75	niedziela, lipiec 15, 1945	74	0	11	1,92	4,80	0,00	52,82	52,82	6232,92
76	poniedziałek, lipiec 16, 1945	75	0	11	1,90	4,75	0,00	52,29	52,29	6285,21

87.1.

W tak skonstruowanej tabeli, informacji o tym, którego dnia średnia prędkość na silniku elektrycznym spadła poniżej 3 węzłów, szukamy w kolumnie `E`. W tej kolumnie wartość mniejsza niż 3 jest po raz pierwszy w wierszu 31, odpowiadającym 1 czerwca 1945.

87.2.

W tej samej tabeli na końcu kolumny z sumaryczną przebytą drogą (`J`) znajdujemy informację o przebytej podczas rejsu drodze: 7701,94.

87.3.

Na podstawie danych zawartych w kolumnach z datą (`A`) oraz drogą przebytą danego dnia (`I`) tworzymy wykres kolumnowy. Pierwszą kolumnę wykorzystujemy do opisania osi X wykresu. Pamiętamy o podpisaniu samego wykresu.

87.4.

Kopiujemy poprzednią tabelę z wyliczeniami. Zmieniamy w niej, poczynając od 15 lipca 1945, liczby godzin rejsu na silniku elektrycznym i na silniku spalinowym (odpowiednio na 20 i 4). Następnie w kolumnie z sumaryczną przebytą drogą szukamy pierwszej wartości nie większej bądź równej tej, wyliczonej w ostatnim wierszu poprzedniej tabeli. Wynika z niej, że do 14 sierpnia włącznie okręt mógł przepłynąć ponad 7722 Mm, czyli płynąc tym samym kursem dopłynąłby do Mar del Plata.

87.5.

Ponownie kopiujemy tabelę początkową do nowego arkusza. Tym razem dla wszystkich dni ustawiamy ten sam czas rejsu na silniku spalinowym, wpisując odpowiednią formułę: `=JEŻELI(DZIEŃ.TYG(A118;2)=7;18;20)`, gdzie w komórce `A118` jest podana data danego dnia. Formuła ta liczy dzienny czas rejsu na silniku elektrycznym (uzależniony od dnia tygodnia), ale z wyłączeniem pierwszej niedzieli. Zmieniamy też obliczenia dotyczące utraty średniej dobowej prędkości (0,997 zamiast 0,9 w odpowiedniej formule) oraz wykorzystania silników (90% możliwości).

Początek tej tabeli będzie wyglądał następująco:

J220 =SUMA(G$114:H220)

	A	B	C	D	E	F	G	H	I	J
113	dzień	nr dnia	godz A	godz S	v A	v S	droga A	droga S	droga	droga całk
114	czwartek, maj 03, 1945	1	20	4	3,60	9,00	72,00	36,00	108,00	108,00
115	piątek, maj 04, 1945	2	20	4	3,59	8,97	71,78	35,89	107,68	215,68
116	sobota, maj 05, 1945	3	20	4	3,58	8,95	71,57	35,78	107,35	323,03
117	niedziela, maj 06, 1945	4	20	4	3,57	8,92	71,35	35,68	107,03	430,06
118	poniedziałek, maj 07, 1945	5	20	4	3,56	8,89	71,14	35,57	106,71	536,77

Z ostatniego wiersza takiej tabeli odczytujemy długość rejsu: 9811,73 Mm.

Zadanie 88.

Rozwiązanie rozpoczynamy od przygotowania tabeli, w której kolejnych kolumnach umieszczamy: dzień (numer dnia miesiąca), wielkość dostawy, liczbę odpowiadającą 90% choinek zaokrągloną w dół do liczby całkowitej, zapotrzebowanie na choinki, rzeczywistą sprzedaż, liczbę choinek pozostałych na placu wieczorem oraz informację, czy danego dnia zabrakło choinek, aby wypełnić zapotrzebowanie.

Początek takiej tabeli będzie wyglądał następująco:

E5 =ZAOKR.DÓŁ((A5*A5-40*A5-50)/-10;0)

	A	B	C	D	E	F	G	H
3	dzień	dostawa	rano na placu	maks sprz	zapotrzeb.	sprzedane	pozostałe	zabrakło
4	1	50	50	45	8	8	42	FAŁSZ
5	2		42	37	12	12	30	FAŁSZ
6	3		30	27	16	16	14	FAŁSZ
7	4	50	64	57	19	19	45	FAŁSZ
8	5		45	40	22	22	23	FAŁSZ

Formuły wpisane w tym arkuszu wyglądają następująco:

	A	B	C	D	E	F	G	H
3	dzień	dostawa	rano na placu	maks sprz	zapotrzeb.	sprzedane	pozostałe	zabrakło
4	1	50	=B4	=ZAOKR.DÓŁ(C4*0,9;0)	=ZAOKR.DÓŁ((A4*A4-40*A4-50)/-10;0)	=MIN(D4:E4)	=C4-F4	=D4<E4
5	2		=G4+B5	=ZAOKR.DÓŁ(C5*0,9;0)	=ZAOKR.DÓŁ((A5*A5-40*A5-50)/-10;0)	=MIN(D5:E5)	=C5-F5	=D5<E5
6	3		=G5+B6	=ZAOKR.DÓŁ(C6*0,9;0)	=ZAOKR.DÓŁ((A6*A6-40*A6-50)/-10;0)	=MIN(D6:E6)	=C6-F6	=D6<E6
7	4	50	=G6+B7	=ZAOKR.DÓŁ(C7*0,9;0)	=ZAOKR.DÓŁ((A7*A7-40*A7-50)/-10;0)	=MIN(D7:E7)	=C7-F7	=D7<E7

Istotne jest też to, żeby w obliczeniach użyć funkcji `ZAOKR.DÓŁ`, a nie jedynie wyświetlać zaokrąglony wynik.

88.1.

Liczbę sprzedanych choinek obliczamy, sumując komórki z kolumny z wielkością sprzedaży (`F`), zaś liczbę choinek pozostałych na placu znajdujemy w ostatnim kolumny G, który odpowiada dacie 24 grudnia.

88.2.

Na podstawie kolumn `dzień` (`A`), liczba sprzedanych drzewek (`F`) i liczba drzewek pozostałych na placu (`G`) wykonujemy wykres kolumnowy skumulowany (aby było na nim dodatkowo widać liczbę choinek znajdujących się rano na placu).

88.3.

W kolumnie z informacją o braku choinek (H) szukamy pierwszego wystąpienia wartości „PRAWDA”. Odpowiedzią jest numer dnia w wierszu, w którym ta wartość wystąpiła.

88.4.

18	50	=P20+K21
19	50	=P21+K22
20		=P22+K23
21	50	=P23+K24
22	50	=P24+K25
23	50	=P25+K26
24	50	=P26+K27
	=ILE.NIEPUSTYCH(K4:K27)	

Całe zestawienie kopiujemy do nowego arkusza. Jeżeli w którymś dniu zabrakło choinek do sprzedaży, wyznaczamy dodatkową dostawę w tym dniu lub w jednym z poprzednich dni. Jeżeli w ostatniej komórce kolumny, z liczbą choinek pozostałych na placu, znalazła się liczba większa niż 49, to znaczy, że przynajmniej jedna dostawa była zbędna. W takim wypadku należy usunąć odpowiednią liczbę poprzedzających ten dzień dostaw w kolejności od najpóźniejszej, tak aby liczba pozostałych choinek zmalała poniżej 50 sztuk. W kolumnie zawierającej liczbę sprzedanych choinek (`F`) obliczamy sumę. Na podstawie zawartości kolumny z dostawami (`B`) wymieniamy dni, w których dostawy są realizowane, oraz podajemy liczbę dostaw. Możemy to zrobić jedna z funkcji: `LICZ.JEŻELI` lub `ILE.NIEPUSTYCH` (patrz ilustracja).

88.5.

dzień	dostawa	rano na placu	n
1	35	=K35	
2	=$K$35	=P35+K36	
3	=$K$35	=P36+K37	
4	=$K$35	=P37+K38	
5	=$K$35	=P38+K39	
6	=$K$35	=P39+K40	

Ponownie kopiujemy tabelę do nowego arkusza. W kolumnie z wielkością dostaw (`B`) we wszystkich wierszach wpisujemy tę samą wartość. Następnie zmniejszamy tę wartość cyklicznie o 1 i szukamy sytuacji, w której któregoś dnia zabraknie choinek na placu. Wtedy wracamy do wartości o jeden większej, która jest odpowiedzią na to zadanie.

Aby nie zmieniać za każdym razem zawartości 24 komórek, we wszystkich z nich poza pierwszą wstawiamy formułę pobierającą wartość z pierwszej, np. `=$K$35` (patrz ilustracja).

Zadanie 91.

Przy wczytywaniu danych z pliku należy pamiętać, aby kolumnie zawierającej numery PESEL nadać format tekstowy. Dane umieszczamy w następujących kolumnach:

	A	B	C
1	PESEL	Nazwisko	Imie
2	08242501475	Micun	Krzysztof
3	08242809191	Jablonski	Nikodem
4	08242912835	Leoniuk	Marcel
5	08250606999	Kurasik	Marcin

91.1.

Zgodnie z opisem zawartym w zadaniu, aby wyznaczyć płeć na podstawie numeru PESEL, należy wyodrębnić z niego cyfrę na 10. pozycji. Dla numeru PESEL, traktowanego jako dana typu tekstowego, można zastosować funkcje tekstowe. Wyodrębnimy 10. cyfrę, stosując funkcję `FRAGMENT.TEKSTU`. Wynikiem działania tej funkcji jest także tekst. Aby wyodrębnioną cyfrę potraktować jako liczbę, której parzystość powinniśmy zbadać, zastosujemy funkcję `WARTOŚĆ`. Poniżej prezentujemy formułę wyodrębniającą cyfrę oznaczającą płeć, którą zapisujemy w kolumnie `D`:

```
=WARTOŚĆ(FRAGMENT.TEKSTU(A2;10;1))
```

Dla przejrzystości rozwiązania w kolumnie `E` umieścimy formułę, której wynikiem będzie litera „k” (dla kobiet) lub „m” (dla mężczyzn). Posłuży nam do tego celu formuła warunkowa, w której odwołujemy się do wartości liczbowej umieszczonej w kolumnie `D`:

```
=JEŻELI(MOD(D2;2)=0;"k";"m")
```

W kolejnej kolumnie umieścimy formułę warunkową, której wartość wynosi 1, gdy spełnione są jednocześnie dwa warunki: płeć wyznaczona na podstawie numeru PESEL jest oznaczona literą „k” i ostatnia litera imienia jest różna od „a”. Do wyznaczenia ostatniej litery stosujemy funkcję `PRAWY`.

```
=JEŻELI(ORAZ(E2="k";PRAWY(C2)<>"a");1;0)
```

Po odfiltrowaniu wierszy, w których formuła ta przyjmuje wartość 1, otrzymamy rozwiązanie zadania.

1	Imie	Płeć	Zadanie 1
76	Doris	k	1
115	Ines	k	1
299	Beatrycze	k	1

91.2.

Rozwiązanie tego zadania rozpoczniemy od posortowania danych alfabetycznie według nazwiska i imienia. W kolumnie `D` umieścimy formułę warunkową, która przyjmuje wartość 1, gdy nazwiska i imiona w bieżącym wierszu oraz w następnym wierszu są takie same.

```
=JEŻELI(ORAZ(B2=B3;C2=C3);1;0)
```

Po odfiltrowaniu wierszy, w których formuła ta przyjmuje wartość 1, dostaniemy wykaz powtarzających się imion i nazwisk:

	A	B	C	D
1	PESEL	Nazwisko	Imie	Zadanie 2
193	59031152059	Kowalczyk	Mateusz	1
198	09211700664	Kozlowska	Malgorzata	1
199	09313003607	Kozlowska	Malgorzata	1
267	08322201772	Michalak	Krzysztof	1
455	09321501177	Wizniewski	Andrzej	1

Pamiętajmy, że należy wypisać wszystkie osoby spełniające warunki zadania, czyli również te, które na uporządkowanej alfabetycznie liście następują po osobach, dla których formuła warunkowa daje wartość 1. Na przykład pojawienie się wiersza 193 w powyższym zestawieniu świadczy o tym, że w rozwiązaniu należy umieścić osobę z wiersza 193 oraz osobę z wiersza 194.

Czytelnikowi pozostawiamy zastanowienie się nad rozwiązaniem zadania, które odfiltruje wszystkie wiersze, które powinny znaleźć się w rozwiązaniu zadania.

91.3.

Najpierw wyodrębnimy liczbę porządkową, stosując funkcję `FRAGMENT.TEKSTU`. Wynikiem działania tej funkcji jest także tekst. Aby wyodrębniony fragment tekstu potraktować jako liczbę, zastosujemy funkcję `WARTOŚĆ`. W kolumnie o nagłówku `Liczba porządkowa` zapiszemy formułę:

```
=WARTOŚĆ(FRAGMENT.TEKSTU(A2;7;3))
```

Następnie posortujemy zestawienie (zawierające imiona, nazwiska i liczby porządkowe) rosnąco według liczb porządkowych. Jako wynik odczytamy imię i nazwisko na pierwszej oraz ostatniej pozycji.

91.4.

Rozpoczniemy od wyodrębnienia cyfr miesiąca z numeru PESEL i zamiany tekstu na wartość liczbową. Zastosujemy funkcje `FRAGMENT.TEKSTU` oraz `WARTOŚĆ`, a formułę `=WARTOŚĆ(FRAGMENT.TEKSTU(A2;3;2))` umieścimy w kolumnie `D`. Następnie obliczymy rzeczywisty numer miesiąca. Można zauważyć, że:

- jeśli wyodrębniony z numeru PESEL numer miesiąca jest większy od 20, to rzeczywisty numer miesiąca uzyskamy zmniejszając wyodrębniony numer o 20;
- w przeciwnym razie rzeczywisty numer miesiąca jest równy wartości wyodrębnionej w kolumnie D.

Stosując te obserwacje, numer miesiąca wyznaczymy za pomocą następującej formuły:

```
=JEŻELI(D2>20;D2-20;D2)
```

Na podstawie numeru PESEL oraz rzeczywistego numeru miesiąca tworzymy tabelę przestawną, w której etykiety wierszy to rzeczywiste numery miesięcy, a wartości to liczby różnych numerów PESEL odpowiadających poszczególnym miesiącom. Czytelnik może też się zastanowić nad uzyskaniem podobnej tabeli, stosując funkcję `LICZ.JEŻELI` zamiast tabeli przestawnej. Sortujemy tabelę według numerów miesięcy.

Etykiety wierszy	Liczba z PESEL
1	68
2	33
3	9
4	16
5	13
6	15
7	19
8	22
9	32
10	67
11	99
12	101
Suma końcowa	**494**

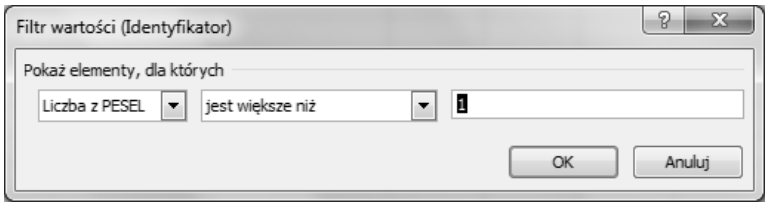
Uzyskane zestawienie kopiujemy w inne miejsce arkusza, dodajemy nagłówek, a numery miesięcy zastępujemy ich nazwami. Można to zrobić ręcznie lub wykorzystać tworzenie serii danych w arkuszu kalkulacyjnym Excel. Po wpisaniu nazw „styczeń” i „luty” w jednej kolumnie można przez przeciągnięcie myszą utworzyć listę nazw miesięcy dla całego roku. Poniżej prezentujemy gotowe zestawienie, na podstawie którego można utworzyć wymagany w zadaniu wykres kolumnowy:

	Liczba osób urodzonych w miesiącu
styczeń	68
luty	33
marzec	9
kwiecień	16
maj	13
czerwiec	15
lipiec	19
sierpień	22
wrzesień	32
październik	67
listopad	99
grudzień	101

91.5.

Aby utworzyć identyfikator, wykorzystamy funkcje `LEWY`, `PRAWY`, `FRAGMENT.TEKSTU`, `ZŁĄCZ.TEKSTY`. W kolumnie o nagłówku `Identyfikator` umieścimy formułę:

```
=ZŁĄCZ.TEKSTY(LEWY(C2);FRAGMENT.TEKSTU(B2;1;3);PRAWY(A2))
```



Następnie na podstawie identyfikatora oraz numeru PESEL tworzymy tabelę przestawną, w której etykiety wierszy to identyfikatory, a wartości to liczba numerów PESEL dla każdego identyfikatora. Sortujemy wiersze tabeli alfabetycznie według identyfikatora i filtrujemy je tak, aby widoczne były tylko wartości większe od 1. Do filtrowania należy wykorzystać polecenie `Filtry wartości`, dostępne w menu w pierwszej kolumnie tabeli przestawnej.

Etykiety wierszy	Liczba z PESEL
AWie3	2
AWit4	2
AWoj0	2
AWoj2	2
AWoj8	2
BWas9	2
JPod4	2
KMic2	2
LMar4	2
MKoc9	2
MKor0	2
MKow4	4
MLub7	2
NJak2	2
NJan3	2
NJan6	2
SCie9	2
SDab7	2
ZAda1	2
Suma końcowa	**40**

Uzyskane zestawienie identyfikatorów stanowi rozwiązanie zadania.

Zadanie 92.

Zadania rozwiążemy z pomocą MS Excel, najpierw wczytując dane do arkusza tak, aby wiersz 1 zawierał nagłówki kolumn, a kolumna A nazwy państw.

92.1.

Rozwiązanie zadania rozpoczynamy od wstawienia dwóch kolumn i zsumowania w nich liczby medali zdobytych na olimpiadach letnich (kolumna `G`) oraz liczby medali zdobytych na olimpiadach zimowych (kolumna `L`). To pozwoli nam zbudować funkcję logiczną sprawdzającą, które państwa spełniają warunki zadania. Zauważmy, że można pominąć sprawdzenie udziału w co najmniej jednej olimpiadzie letniej, ponieważ zdobycie co najmniej jednego medalu na takiej olimpiadzie daje nam pewność jego uczestnictwa w co najmniej jednej takiej olimpiadzie (a rozważamy tylko państwa, które zdobyły medale na olimpiadach letnich).

W tym celu dodajemy do zestawienia kolumnę `Czy spełnione są 3 warunki?` i w komórce `M2` budujemy formułę: `=JEŻELI(ORAZ(G2>0;H2>0;L2=0);1;0)`, którą kopiujemy do pozostałych wierszy zestawienia. Sumując otrzymane wartości w kolumnie `M`, otrzymamy poszukiwaną liczbę państw.

=SUMA(M2:M139)

Panstwo	Kontynent	OL_letnie	Zloty	Srebrny	Brazowy	Suma medali - OL	OL_zimowe	Zloty	Srebrny	Brazowy	Suma medali - OL	Czy spełnione są 3 warunki?		Ile Państw
Afganistan	Azja	13	0	0	2	2	0	0	0	0	0	0		54
Algieria	Afryka	12	5	2	8	15	3	0	0	0	0	1		
Antyle Holenderskie	Ameryka Płd.	13	0	1	0	1	2	0	0	0	0	1		
Arabia Saudyjska	Azja	10	0	1	2	3	0	0	0	0	0	0		
Argentyna	Ameryka Płd.	23	18	24	28	70	18	0	0	0	0	1		
Armenia	Azja	5	1	2	9	12	6	0	0	0	0	1		

Drugą część zadania możemy rozwiązać na dwa sposoby:

1. Dodajemy kolejną kolumnę, w której budujemy formułę: `=JEŻELI(M2=1;G2;0)`, dzięki której otrzymamy w kolumnie liczbę medali zdobytych na olimpiadach letnich przez państwa spełniające warunki. A suma tych medali jest odpowiedzią do drugiej części zadania.
2. Korzystając z wbudowanej funkcji `suma.jeżeli` (sumuje ona te wartości z zakresu, które spełniają określone kryteria), budujemy formułę, która sumuje liczbę medali zdobytych na olimpiadach letnich przez państwa spełniające podane warunki `=(SUMA.JEŻELI(M2:M139;1;G2:G139)`.

=SUMA.JEŻELI(M2:M139;1;G2:G139)

Srebrny	Brazowy	Suma medali - OL	OL_zimowe	Zloty	Srebrny	Brazowy	Suma medali - OL	Czy spełnione są 3 warunki?		
0	2	2	0	0	0	0	0	0		Ile Państw
2	8	15	3	0	0	0	0	1		54
1	0	1	2	0	0	0	0	1		Ile medali w letniej zdobyły Państwa spełniające warunki
1	2	3	0	0	0	0	0	0		
24	28	70	18	0	0	0	0	1		
2	9	12	6	0	0	0	0	1		
153	177	468	18	5	3	4	12	0		1218

92.2.

W celu rozwiązania zadania tworzymy tabelę przestawną, ustalając następujące wartości:

- etykiety wierszy: `Kontynent`,
- etykiety wartości: `OL_letnie` i `OL_zimowe`.

Dla obu wartości wybieramy funkcję `suma` i ostatecznie otrzymujemy potrzebne do sporządzenia wykresu zestawienie:

Etykiety wierszy	Suma z OL_letnie	Suma z OL_zimowe
Afryka	297	30
Ameryka Pld.	218	52
Ameryka Pln.	236	88
Australia i Oc.	55	34
Azja	422	177
Europa	682	571

Dla tak przygotowanych danych możemy utworzyć wykres przestawny, wybierając odpowiedni jego typ, albo skopiować zestawienie i sporządzić wykres w tradycyjny sposób, dostępny w grupie *Wykresy*, na karcie *Wstawianie*.

92.3.

Rozwiązanie zadania rozpoczynamy od zsumowania dla każdego państwa liczby poszczególnych typów medali zdobytych przez nie na obu olimpiadach.

	A	B	C	D	E	F	G	H	I	J	K	L	M
	Panstwo	Kontynent	OL_letnie	Zloty	Srebrny	Brazowy	OL_zimowe	Zloty	Srebrny	Brazowy	Razem złotych	Razem srebrnych	Razem brązowych
	Kamerun	Afryka	13	3	1	1	1	0	0	0	3	1	1
	Burundi	Afryka	5	1	0	0	0	0	0	0	1	0	0
	Zjednoczone Emiraty Arabskie	Azja	8	1	0	0	0	0	0	0	1	0	0
	StanyZjednoczone	Ameryka Pln.	26	976	758	666	22	96	102	83	1072	860	749
	ZSRR	Europa	9	395	319	296	9	78	57	59	473	376	355
	Wielka Brytania	Europa	27	236	272	272	22	10	4	12	246	276	284
	Francja	Europa	27	202	223	246	22	31	31	47	233	254	293

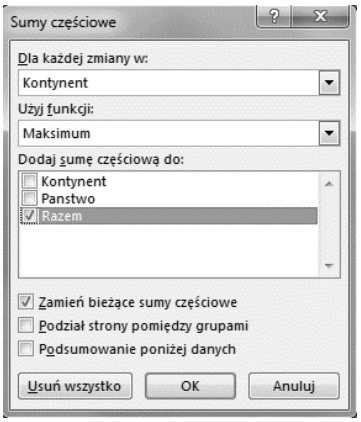
To pozwoli nam zbudować funkcję logiczną sprawdzającą, które państwa zdobyły więcej medali złotych niż (łącznie) medali srebrnych i brązowych. Dodajemy do zestawienia kolumnę `N`, w której sprawdzamy powyższy warunek, i w komórce `N2` budujemy formułę: `=JEŻELI(K2>(L2+M2);1;0)`, którą kopiujemy do pozostałych wierszy zestawienia. Sortując zestawienie malejąco według kolumny `L`, otrzymamy w pierwszych wierszach szukane państwa.

92.4.

Wyznaczenie dla każdego kontynentu państwa, które zdobyło łącznie największą liczbę medali na wszystkich olimpiadach, oraz ich liczby rozpoczynamy od zsumowania wszystkich medali dla każdego państwa. Następnie sortujemy otrzymane zestawienie leksykograficznie według nazwy kontynentu oraz malejąco według sumy medali. W efekcie pierwszy wiersz każdego kontynentu zawiera nazwę państwa z największą liczbą medali na tym kontynencie.

	A	B	C	D
	Kontynent	Panstwo	Razem	
	Afryka	Kenia	86	
	Afryka	Republika Poludniowej Afryki	76	
	Afryka	Etiopia	45	
	Afryka	Egipt	26	

Aby wyodrębnić tylko te wiersze, które nas interesują, możemy skorzystać z narzędzia *Sumy częściowe* (dostępnego w grupie *Konspekt*, na karcie *Dane*); dla każdego kontynentu wybierzemy maksymalną wartość wyliczonej sumy medali.

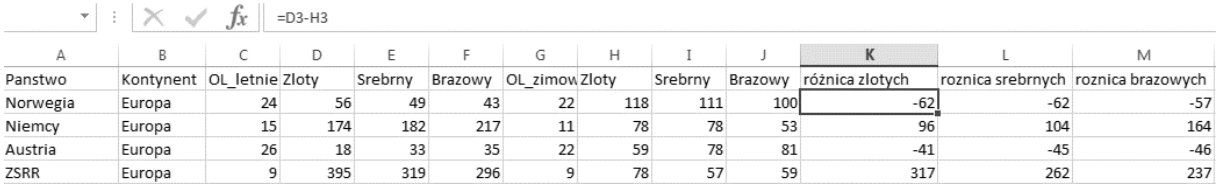


W obrębie danego kontynentu na pierwszej pozycji znajdzie się kraj z największą liczbą medali.

	A	B	C
1	Kontynent	Panstwo	Razem
2	**Maks. całkowite**		2681
3	**Afryka Maksimum**		86
30	**Ameryka Pld. Maksimum**		108
31	Ameryka Pld.	Brazylia	108
32	Ameryka Pld.	Argentyna	70
33	Ameryka Pld.	Kolumbia	19
34	Ameryka Pld.	Chile	13
35	Ameryka Pld.	Wenezuela	12

92.5.

Rozwiązanie ostatniego zadania (piątego) rozpoczynamy od wyznaczenia różnicy liczby medali każdego rodzaju zdobytych w olimpiadach letnich oraz liczby medali zdobytych w olimpiadach zimowych. Wartości te wyznaczamy dla każdego kraju.



Panstwo	Kontynent	OL_letnie	Zloty	Srebrny	Brazowy	OL_zimow	Zloty	Srebrny	Brazowy	różnica zlotych	roznica srebrnych	roznica brazowych
Norwegia	Europa	24	56	49	43	22	118	111	100	-62	-62	-57
Niemcy	Europa	15	174	182	217	11	78	78	53	96	104	164
Austria	Europa	26	18	33	35	22	59	78	81	-41	-45	-46
ZSRR	Europa	9	395	319	296	9	78	57	59	317	262	237

Kraj, dla którego wszystkie różnice są mniejsze od zera jest krajem zimowym (`=JEŻELI(ORAZ(K3<0;L3<0;M3<0);1;0)`), jeżeli zaś wszystkie różnice są dodatnie, kraj jest krajem letnim (`=JEŻELI(ORAZ(K3>0;L3>0;M3>0);1;0)`).

Zadanie 93.

Zadania rozwiążemy z pomocą MS Excel, najpierw wczytując dane do arkusza tak, aby wiersz 1 zawierał nagłówki kolumn, a kolumna A zawierała imiona osób.

93.1.

Rozwiązanie możemy uzyskać za pomocą funkcji `LICZ.JEŻELI`, wyliczając w kolumnie `G`, w każdym wierszu, liczbę wyjazdów danego pracownika:

```
=LICZ.JEŻELI($B$2:$B$1001;B2).
```

Zakres danych to komórki z nazwiskami pracowników (`$B$2:$B$1001`), zaś kryterium funkcji jest komórka z nazwiskiem pracownika w danym wierszu, tutaj: nazwisko z komórki `B2`.

Zwróćmy uwagę na sposób adresowania w formule z komórki `G2`:

- adres `$B$2` gwarantuje, że po skopiowaniu formuły początek zakresu zawsze będzie zaczynał się od drugiego wiersza, zaś adres `$B$1001` — że koniec zakresu będzie się kończył w wierszu 1001 (poniżej pierwszych kilka wierszy).

G
Ile wyjazdów
=LICZ.JEŻELI($B$2:$B$1001;B2)
=LICZ.JEŻELI($B$2:$B$1001;B3)
=LICZ.JEŻELI($B$2:$B$1001;B4)
=LICZ.JEŻELI($B$2:$B$1001;B5)
=LICZ.JEŻELI($B$2:$B$1001;B6)
=LICZ.JEŻELI($B$2:$B$1001;B7)
=LICZ.JEŻELI($B$2:$B$1001;B8)
=LICZ.JEŻELI($B$2:$B$1001;B9)
=LICZ.JEŻELI($B$2:$B$1001;B10)

Następnie, sortując zestawienie rosnąco według kolumny G, otrzymujemy poprawny wynik w pierwszym wierszu otrzymanego zestawienia.

G2 =LICZ.JEŻELI($B$2:$B$1001;B2)

	A	B	C	D	E	F	G
1	Imie	Nazwisko	Miasto	D_wyj	D_powr	Koszt_wyj	Ile wyjazdów
2	Ewelina	Adamska	Katowice	2014-11-04	2014-11-06	892,7	1
3	Janusz	Jurkicz	Malbork	2014-01-15	2014-01-17	1102	5
4	Piotr	Malski	Radom	2014-01-15	2014-01-16	302,5	5
5	Piotr	Malski	Kielce	2014-02-19	2014-02-22	841,7	5

93.2.

W drugim zadaniu musimy obliczyć, ile dni trwał każdy wyjazd, a następnie zsumować otrzymane wyliczenia dla każdego pracownika. W tym celu dodajemy do zestawienia kolumnę `Ile_dni_delegacji`, w której wyznaczamy, ile dni trwała każda delegacja, jednocześnie budując w komórce `H2` formułę: `=E2-D2+1` (liczba 1 dodana do różnicy to doliczenie do różnicy dat daty wyjazdu, która dla delegacji jednodniowych jest równa dacie powrotu) i kopiując ją do kolejnych wierszy arkusza (poniżej kilka pierwszych wierszy),

H
Ile_dni_delegacji
=E2-D2+1
=E3-D3+1
=E4-D4+1
=E5-D5+1
=E6-D6+1
=E7-D7+1
=E8-D8+1
=E9-D9+1
=E10-D10+1

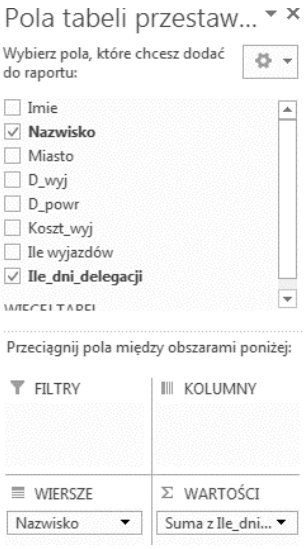
otrzymujemy dla każdego nazwiska liczbę dni trwania każdej delegacji (poniżej kilka pierwszych wierszy).

	A	B	C	D	E	F	G	H
1	Imie	Nazwisko	Miasto	D_wyj	D_powr	Koszt_wyj	Ile wyjazdów	Ile_dni_delegacji
2	Ewelina	Adamska	Katowice	2014-11-04	2014-11-06	892,7	1	3
3	Patrycja	Andrycz	Lublin	2014-01-03	2014-01-04	439,7	12	2
4	Patrycja	Andrycz	Kalisz	2014-01-12	2014-01-12	442	12	1
5	Patrycja	Andrycz	Kutno	2014-01-14	2014-01-17	665,8	12	4
6	Patrycja	Andrycz	Katowice	2014-02-19	2014-02-23	1290,7	12	5

Dla tak przygotowanych danych tworzymy tabelę przestawną i sortujemy ją z jednym kryterium grupowania, ustalając następujące wartości:

- etykiety wierszy: `Nazwisko`,
- etykiety wartości: wyliczona liczba dni delegacji.

W obszarze wartości wybieramy funkcję `suma`.



93.3.

Rozwiązanie zadania trzeciego rozpoczynamy od wyliczenia kosztu każdej delegacji w kolumnie `I` (`Koszt_delegacji`).

Wyliczając koszt, możemy wykorzystać przy budowaniu formuły wyliczoną w poprzednim zadaniu liczbę dni delegacji: `=JEŻELI(H2=1;F2+30;30+(H2-1)*24+F2)` albo zbudować formułę odwołującą się bezpośrednio do obu dat (wyjazdu i powrotu): `=F2+30+(E2-D2)*24`. Kopiując ją do pozostałych wierszy (poniżej kilka pierwszych wierszy),

	I
	Koszt_delegacji
	=F2+30+(E2-D2)*24
	=F3+30+(E3-D3)*24
	=F4+30+(E4-D4)*24
	=F5+30+(E5-D5)*24
	=F6+30+(E6-D6)*24
	=F7+30+(E7-D7)*24
	=F8+30+(E8-D8)*24
	=F9+30+(E9-D9)*24
	=F10+30+(E10-D10)*24

otrzymujemy zestawienie (poniżej kilka pierwszych wierszy):

	A	B	C	D	E	F	G	H	I
1	Imie	Nazwisko	Miasto	D_wyj	D_powr	Koszt_wyj	Ile wyjazdów	Ile_dni_delegacji	Koszt_delegacji
2	Ewelina	Adamska	Katowice	2014-11-04	2014-11-06	892,7	1	3	970,70
3	Patrycja	Andrycz	Lublin	2014-01-03	2014-01-04	439,7	12	2	493,70
4	Patrycja	Andrycz	Kalisz	2014-01-12	2014-01-12	442	12	1	472,00
5	Patrycja	Andrycz	Kutno	2014-01-14	2014-01-17	665,8	12	4	767,80
6	Patrycja	Andrycz	Katowice	2014-02-19	2014-02-23	1290,7	12	5	1416,70

I podobnie jak w poprzednim zadaniu tworzymy oraz sortujemy tabelę przestawną, w której ustalamy:

- etykiety wierszy: `Miasto`,
- etykiety wartości: wyliczona liczba dni delegacji.

W obszarze wartości ponownie wybieramy funkcję `suma`.

93.4.

W następnym zadaniu, korzystając z wbudowanej funkcji `=MIESIĄC(D2)`, wyłuskujemy dla każdej daty wyjazdu (`D_wyj`) numer miesiąca. Poniżej kilka pierwszych wierszy.

J
Miesiąc wyjazdu
=MIESIĄC(D2)
=MIESIĄC(D3)
=MIESIĄC(D4)
=MIESIĄC(D5)
=MIESIĄC(D6)
=MIESIĄC(D7)
=MIESIĄC(D8)

	A	B	C	D	E	F	G	H	I	J
1	Imie	Nazwisko	Miasto	D_wyj	D_powr	Koszt_wyj	Ile wyjazdów	Ile_dni_delegacji	Koszt_delegacji	Miesiąc wyjazdu
2	Ewelina	Adamska	Katowice	2014-11-04	2014-11-06	892,7	1	3	970,70	11
3	Patrycja	Andrycz	Lublin	2014-01-03	2014-01-04	439,7	12	2	493,70	1
4	Patrycja	Andrycz	Kalisz	2014-01-12	2014-01-12	442	12	1	472,00	1
5	Patrycja	Andrycz	Kutno	2014-01-14	2014-01-17	665,8	12	4	767,80	1
6	Patrycja	Andrycz	Katowice	2014-02-19	2014-02-23	1290,7	12	5	1416,70	2
7	Patrycja	Andrycz	Bydgoszcz	2014-05-05	2014-05-06	654,4	12	2	708,40	5

Zliczenia liczby wyjazdów w poszczególnych miesiącach można dokonać na kilka sposobów. Zapewne najprostszym rozwiązaniem jest posortowanie dat wyjazdów według miesięcy (wtedy nie trzeba nawet wyłuskiwać z daty numeru miesiąca) i „ręczne” zliczenie liczby wyjazdów w danym miesiącu. Można również skorzystać z funkcji arkusza kalkulacyjnego: `LICZ.JEŻELI()` albo `LICZ.WARUNKI()`.

J	K	L	M
Miesiąc wyjazdu		numer miesiąca	Liczba wyjazdów w miesiącu
=MIESIĄC(D2)		1	=LICZ.JEŻELI($J$2:$J$1001;L2)
=MIESIĄC(D3)		2	=LICZ.JEŻELI($J$2:$J$1001;L3)
=MIESIĄC(D4)		3	=LICZ.JEŻELI($J$2:$J$1001;L4)
=MIESIĄC(D5)		4	=LICZ.JEŻELI($J$2:$J$1001;L5)
=MIESIĄC(D6)		5	=LICZ.JEŻELI($J$2:$J$1001;L6)
=MIESIĄC(D7)		6	=LICZ.JEŻELI($J$2:$J$1001;L7)
=MIESIĄC(D8)		7	=LICZ.JEŻELI($J$2:$J$1001;L8)
=MIESIĄC(D9)		8	=LICZ.JEŻELI($J$2:$J$1001;L9)
=MIESIĄC(D10)		9	=LICZ.JEŻELI($J$2:$J$1001;L10)
=MIESIĄC(D11)		10	=LICZ.JEŻELI($J$2:$J$1001;L11)
=MIESIĄC(D12)		11	=LICZ.JEŻELI($J$2:$J$1001;L12)
=MIESIĄC(D13)		12	=LICZ.JEŻELI($J$2:$J$1001;L13)

Na podstawie otrzymanego zestawienia:

L	M
numer miesiąca	Liczba wyjazdów w miesiącu
1	130
2	68
3	63
4	32
5	31
6	63
7	70
8	54
9	121
10	103
11	135
12	130

wstawiamy wykres kolumnowy:

93.5.

Dodajemy kolumnę `Liczba noclegów` i wyliczamy w niej, ile noclegów przypada na każdą delegację, budując w komórce `K2` formułę: `=E2-D2` i kopiując ją do kolejnych wierszy arkusza (poniżej kilka pierwszych wierszy):

K
Liczba noclegów
=E2-D2
=E3-D3
=E4-D4
=E5-D5
=E6-D6
=E7-D7
=E8-D8
=E9-D9

Otrzymamy zestawienie (poniżej kilka pierwszych wierszy):

D	E	F	G	H	I	J	K
D_wyj	D_powr	Koszt_wyj	Ile wyjazdów	Ile_dni_dele	Koszt_delegacj	Miesiąc wyjazdu	Liczba noclegów
2014-11-04	2014-11-06	892,7	1	3	970,70	11	2
2014-01-03	2014-01-04	439,7	12	2	493,70	1	1
2014-01-12	2014-01-12	442	12	1	472,00	1	0
2014-01-14	2014-01-17	665,8	12	4	767,80	1	3
2014-02-19	2014-02-23	1290,7	12	5	1416,70	2	4
2014-05-05	2014-05-06	654,4	12	2	708,40	5	1

Średnią liczbę noclegów pracowników we wszystkich delegacjach, wyliczamy korzystając z funkcji `średnia` i zaokrąglając otrzymany na jej podstawie wynik do dwóch miejsc po przecinku (zgodnie z poleceniem w zadaniu): `=ZAOKR(ŚREDNIA($K$2:$K$1001);2)`.

W kolumnie K wyliczona została liczba noclegów niezależnie od długości trwania delegacji i dla delegacji jednodniowych otrzymaliśmy 0 noclegów. Aby wyliczyć średnią liczbę noclegów w delegacjach co najmniej 2-dniowych, korzystamy z wbudowanej funkcji obliczającej średnią z komórek spełniających dany warunek (lub kryteria) i podobnie jak w poprzednim obliczeniu jej wynik zaokrąglamy do dwóch miejsc po przecinku:

`=ZAOKR(ŚREDNIA.JEŻELI($K$2:$K$1001;">0");2)`.

Zadanie 94.

94.1.

Największy procentowy przyrost długości ciała wyrażony w procentach iloraz różnicy wzrostu w wieku lat 19 i długości ciała w dniu urodzin do długości ciała w dniu urodzin.

W2 fx =(V2-C2)/C2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ID_dziecka	plec	dlugosc_ur	1rok	2lata	3lata	4lata	5lat	6lat	7lat	8lat	9lat	10lat	11lat	12lat	13lat	14lat	15lat	16lat	17 lat	18lat	19lat	przyrost%
2	718	ch	46	68	82	91	97	103	109	114	119	124	128	134	139	146	153	158	162	166	168	170	270%
3	1943	ch	46	69	84	92	99	105	111	116	121	126	131	136	141	148	155	161	165	167	168	169	267%
4	576	ch	45	68	82	90	96	102	108	113	118	123	126	132	138	144	151	155	160	163	164	165	267%
5	284	ch	46	68	83	91	98	104	109	115	120	125	130	135	140	147	154	160	163	166	167	168	265%
6	847	ch	46	68	83	91	98	104	109	115	120	125	130	135	140	147	154	160	163	166	167	168	265%

Po obliczeniu procentowego przyrostu posortujemy dane według kolumny `przyrost%` w porządku malejącym. Pierwszy rekord spełnia kryteria zadania: dziecko o identyfikatorze 718 osiągnęło największy przyrost długości ciała: 270%.

94.2.

Do obliczenia średniej arytmetycznej wzrostu dziewcząt i chłopców stosujemy funkcję `ŚREDNIA.JEŻELI` o argumentach: zakres, kryteria i średnia_zakres. Zakres to kolumna `B` (`płeć`), kryteria to komórka z oznaczeniem płci ("d" lub "ch"), zaś średnia_zakres to kolumny ze wzrostem dzieci.

W przypadku, gdy nie znamy funkcji `ŚREDNIA.JEŻELI` lub brak jej w starszej wersji arkusza można zastosować funkcję `SUMA.JEŻELI`, a otrzymany wynik podzielić przez liczbę dzieci określonej płci.

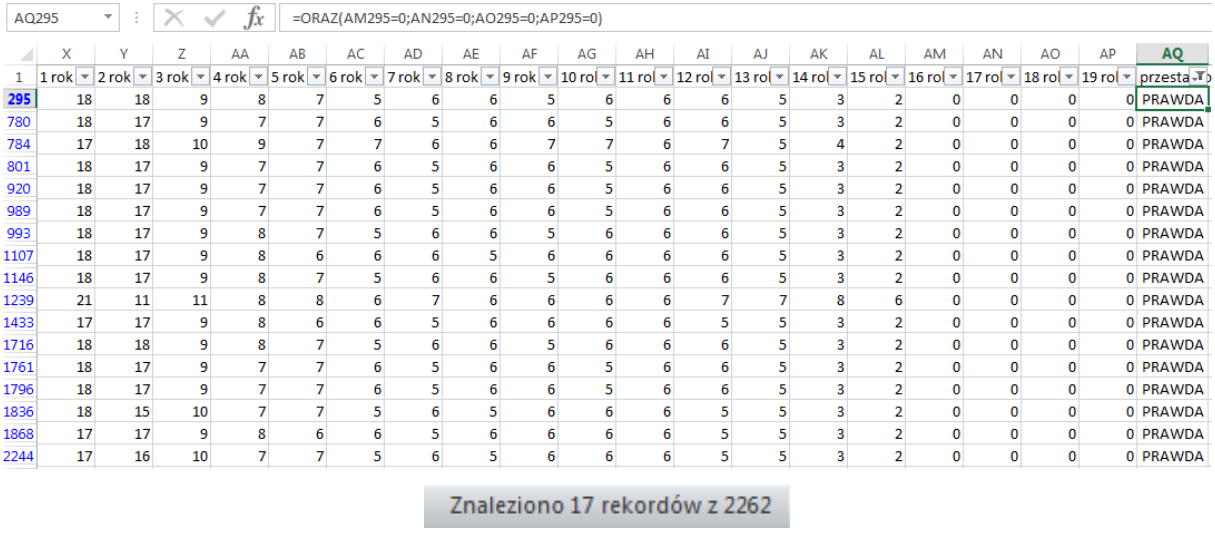
C2267 fx =ŚREDNIA.JEŻELI($B$2:$B$2263;$B2267;C$2:C$2263)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
2265																						
2266	średnia ary	plec	dlugosc_ur	1rok	2lata	3lata	4lata	5lat	6lat	7lat	8lat	9lat	10lat	11lat	12lat	13lat	14lat	15lat	16lat	17 lat	18lat	19lat
2267		d	51,430108	69,7331	86,196	95,68	103,34	110,12	115,9	121,63	127,45	133,46	139,63	146,06	152,35	157,63	161,02	162,92	163,79	164,15	164,39	164,5
2268		ch	52,668281	74,004	87,004	96,001	103,24	109,92	115,9	121,69	127,19	132,51	137,71	143,04	149,05	155,94	163,19	168,92	172,9	175,22	176,2	176,67
2269	różnica ch-dz		1,2381733	4,2709	0,8085	0,3205	-0,101	-0,207	0,0066	0,0609	-0,264	-0,947	-1,914	-3,019	-3,298	-1,69	2,1628	5,9986	9,1117	11,068	11,814	12,174

Po wykonaniu zestawienia średniego wzrostu dzieci należy obliczyć różnicę między średnim wzrostem chłopców a średnim wzrostem dziewcząt i wyróżnić różnice mniejsze lub równe -1, np. za pomocą formatowania warunkowego.

94.3.

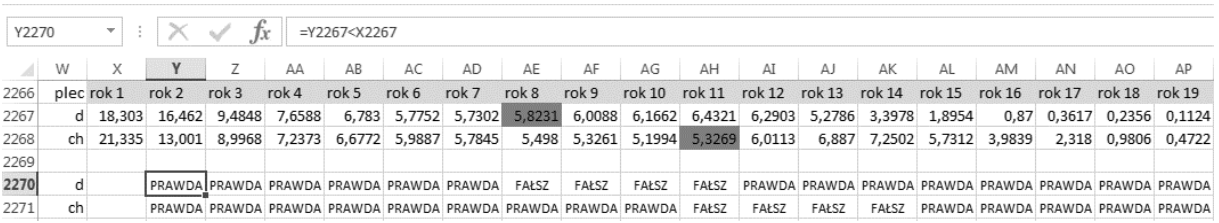
Aby sprawdzić przyrost długości ciała w danym roku, obliczamy różnicę pomiędzy wzrostem w roku bieżącym a wzrostem w roku poprzednim. Następnie odfiltrowujemy przyrosty równe 0 po 15 roku życia.



	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP	AQ
1	1 rok	2 rok	3 rok	4 rok	5 rok	6 rok	7 rok	8 rok	9 rok	10 ro	11 ro	12 ro	13 ro	14 ro	15 ro	16 ro	17 ro	18 ro	19 ro	przesta
295	18	18	9	8	7	5	6	6	5	6	6	6	5	3	2	0	0	0	0	PRAWDA
780	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
784	17	18	10	9	7	7	6	6	7	7	6	7	5	4	2	0	0	0	0	PRAWDA
801	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
920	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
989	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
993	18	17	9	8	7	5	6	6	5	6	6	6	5	3	2	0	0	0	0	PRAWDA
1107	18	17	9	8	6	6	6	5	6	6	6	6	5	3	2	0	0	0	0	PRAWDA
1146	18	17	9	8	7	5	6	6	5	6	6	6	5	3	2	0	0	0	0	PRAWDA
1239	21	11	11	8	8	6	7	6	6	6	6	7	7	8	6	0	0	0	0	PRAWDA
1433	17	17	9	8	6	6	5	6	6	6	6	5	5	3	2	0	0	0	0	PRAWDA
1716	18	18	9	8	7	5	6	6	5	6	6	6	5	3	2	0	0	0	0	PRAWDA
1761	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
1796	18	17	9	7	7	6	5	6	6	5	6	6	5	3	2	0	0	0	0	PRAWDA
1836	18	15	10	7	7	5	6	5	6	6	6	5	5	3	2	0	0	0	0	PRAWDA
1868	17	17	9	8	6	6	5	6	6	6	6	5	5	3	2	0	0	0	0	PRAWDA
2244	17	16	10	7	7	5	6	5	6	6	6	5	5	3	2	0	0	0	0	PRAWDA

94.4.

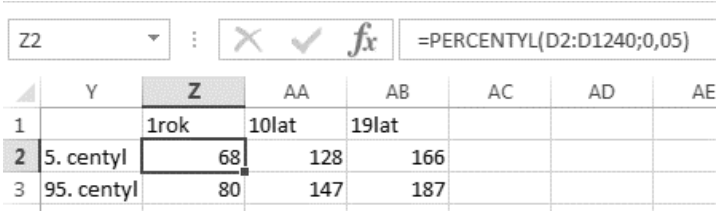
Po obliczeniu przyrostów długości ciała w kolejnych latach wyznaczamy średnie dla każdej płci osobno, a następnie sprawdzamy kierunek trendu w zakresie przyrostu w kolejnych latach: czy kolejny przyrost jest mniejszy od aktualnego. Pojawienie się wartości fałsz wyznacza moment, w którym trend się zmienia.



	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH	AI	AJ	AK	AL	AM	AN	AO	AP
2266	plec	rok 1	rok 2	rok 3	rok 4	rok 5	rok 6	rok 7	rok 8	rok 9	rok 10	rok 11	rok 12	rok 13	rok 14	rok 15	rok 16	rok 17	rok 18	rok 19
2267	d	18,303	16,462	9,4848	7,6588	6,783	5,7752	5,7302	5,8231	6,0088	6,1662	6,4321	6,2903	5,2786	3,3978	1,8954	0,87	0,3617	0,2356	0,1124
2268	ch	21,335	13,001	8,9968	7,2373	6,6772	5,9887	5,7845	5,498	5,3261	5,1994	5,3269	6,0113	6,887	7,2502	5,7312	3,9839	2,318	0,9806	0,4722
2269																				
2270	d		PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	FAŁSZ	FAŁSZ	FAŁSZ	FAŁSZ	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA
2271	ch		PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA	FAŁSZ	FAŁSZ	FAŁSZ	FAŁSZ	PRAWDA	PRAWDA	PRAWDA	PRAWDA	PRAWDA

94.5.

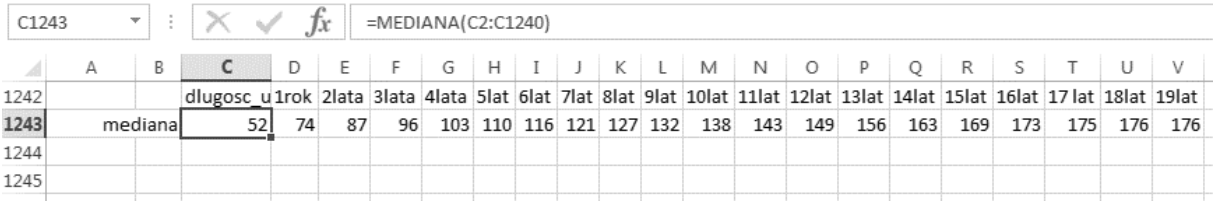
Centyle 5. i 95. obliczamy za pomocą funkcji `PERCENTYL` lub etapami: najpierw wyznaczamy pozycję 5. centyla, czyli 5% liczby wszystkich badanych chłopców (5%*1239), i w uporządkowanym szeregu sprawdzamy wartość na tej pozycji, a następnie postępujemy identycznie dla 95. centyla.



	Y	Z	AA	AB	AC	AD	AE
1		1rok	10lat	19lat			
2	5. centyl	68	128	166			
3	95. centyl	80	147	187			

94.6.

Mediana to inaczej środkowy wyraz uporządkowanego ciągu. Medianę wyznaczymy funkcją `MEDIANA` lub wybierając środkowy wzrost (w zadaniu mamy dane dotyczącd wzrostu 1239 chłopców, więc należy odczytać 620. pozycję) w uporządkowanym zestawieniu.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1242			dlugosc_u	1rok	2lata	3lata	4lata	5lat	6lat	7lat	8lat	9lat	10lat	11lat	12lat	13lat	14lat	15lat	16lat	17 lat	18lat	19lat
1243	mediana		52	74	87	96	103	110	116	121	127	132	138	143	149	156	163	169	173	175	176	176
1244																						
1245																						

Na podstawie uzyskanego zestawu danych tworzymy wykres liniowy, w którym na osi OX znajdzie się wiek chłopca, zaś na osi OY — jego wzrost.

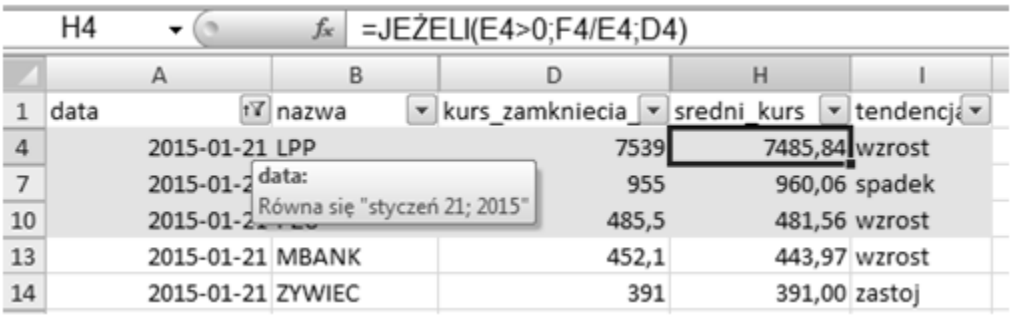
Zadanie 95.

Rozwiążemy zadania za pomocą arkusza kalkulacyjnego, każde zadanie w osobnym arkuszu. W tym celu po zaimportowaniu danych do arkusza utworzymy kopie tego arkusza.

95.1.

Średni kurs akcji każdej spółki w danym dniu możemy obliczyć, wykorzystując funkcję warunkową `JEŻELI`. Gdy obrót spółki i jej wolumen są większe od 0, funkcja powinna zwrócić wartość `obrot/wolumen`, w przeciwnym razie — `kurs_zamkniecia`.

Aby znaleźć spółki o najwyższym średnim kursie dnia, sortujemy wiersze tabeli malejąco według obliczonego średniego kursu dnia. Następnie nakładamy filtr na kolumnę *data*, aby wybrać tylko wiersze z dnia 2015-01-21. Trzy pierwsze wiersze tabeli są wynikami zadania.

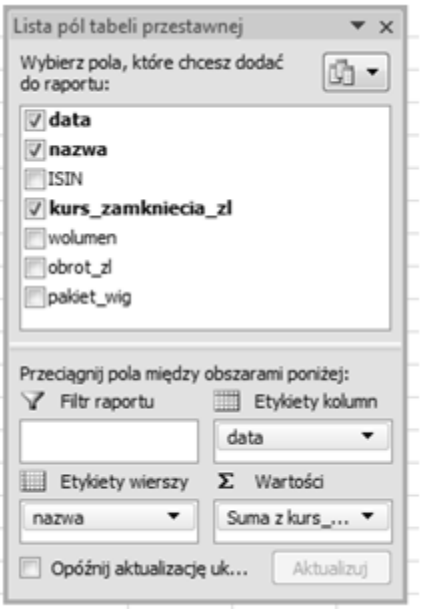


	A	B	D	H	I
1	data	nazwa	kurs_zamkniecia_	sredni_kurs	tendencja
4	2015-01-21	LPP	7539	7485,84	wzrost
7	2015-01-2		955	960,06	spadek
10	2015-01-2		485,5	481,56	wzrost
13	2015-01-21	MBANK	452,1	443,97	wzrost
14	2015-01-21	ZYWIEC	391	391,00	zastoj

95.2.

Przegrupujemy dane źródłowe, aby w jednym wierszu wystąpiły nazwa spółki oraz jej kursy zamknięcia sesji z trzech kolejnych dni. W tym celu wstawiamy tabelę przestawną. W nagłówkach wierszy umieszczamy pole `nazwa` spółki, w nagłówkach kolumn — pole `data`, a w polu wartości — `kurs_zamknięcia`.

3	Suma z kurs_zamkniecia_zl			
4		2015-01-21	2015-01-22	2015-01-23
5	06MAGNA	2,09	2,26	2,14
6	08OCTAVA	0,79	0,79	0,79
7	4FUNMEDIA	5,8	5,85	6,1
8	ABCDATA	3,37	3,43	3,4
9	ABMSOLID	0,3	0,3	0,3
10	ABPL	32,5	34,99	35,48
11	ACAUTOGAZ	27,5	27,51	27,6
12	ACE	8,24	8	8,79
13	ACTION	44,89	45,85	45,2
14	ADVADIS	0,01	0,01	0,01
15	AGORA	7,95	8,1	8,35
16	AGROTON	1,37	1,41	1,43
17	AGROWILL	1	1	1
18	ALCHEMIA	5,08	5,08	5,05
19	ALIOR	79,79	84	84,77
20	ALMA	14,14	14,15	14,65
21	ALTA	2,1	2,08	2,09



W nowej kolumnie obok tabeli przestawnej, w każdym wierszu, umieszczamy formułę, która obliczy względną dzienną zmianę kursu: podzieli kurs zamknięcia w dniu 2015-01-23 przez kurs zamknięcia w dniu 2015-01-22 i odejmie 1. Wynik wyrażamy w formacie procentowym z żądaną dokładnością. Aby znaleźć spółkę o najwyższej zmianie kursu, sortujemy całą tabelę malejąco według zmiany kursu.

D2 =C2/B2-1

	A	B	C	D	E
1		2015-01-22	2015-01-23	d	
2	MILKILAND	1,58	2,02	27,85%	
3	IDMSA	0,11	0,13	18,18%	
4	STARHEDGE	2,39	2,7	12,97%	

95.3.

Rodzaj spółki (”krajowa” lub ”zagraniczna”) wyznaczamy w każdym wierszu na podstawie pola *ISIN*. Jeśli pierwsze dwa znaki w tym polu są równe ”PL”, to spółka jest krajowa, w przeciwnym razie jest zagraniczna.

Do wycięcia dwóch znaków z lewej strony tekstu w polu *ISIN* stosujemy funkcję `LEWY`. Porównanie wyniku tej funkcji z tekstem ”PL” umieszczamy jako warunek wewnątrz funkcji warunkowej `JEŻELI`. Wyniki funkcji warunkowej umieszczamy w nowej kolumnie o nazwie `rodzaj`.

H2 =JEŻELI(LEWY(C2;2)="PL";"krajowa";"zagraniczna")

	A	B	C	D	E	F	G	H
1	data	nazwa	ISIN	kurs_zamkn	wolumen	obrot_zl	pakiet_wig	rodzaj
2	2015-01-21	06MAGNA	PLNFI0600010	2,09	9	18	6496000	krajowa

a) Zliczamy wystąpienia wszystkich słów ”krajowa” i słów ”zagraniczna” w polu `rodzaj`. Można to zrobić na dwa sposoby, używając:

- funkcji warunkowej `LICZ.JEŻELI`,

- narzędzia z karty Dane — `Sumy częściowe`: dla każdej zmiany w polu `rodzaj` używamy funkcji `licznik` w polu `nazwa` (w Excelu tabelę trzeba wcześniej posortować według rodzaju spółki).

Wystarczy zliczyć spółki tylko w jednym dniu. Jeśli nie chcemy filtrować danych dotyczących jednego wybranego dnia, zliczamy wystąpienia w całej kolumnie tabeli i wynik dzielimy przez 3 (liczba dni).

b) Łączne obroty spółek krajowych i spółek zagranicznych wyznaczamy w podobny sposób, używając jednego z dwóch narzędzi:

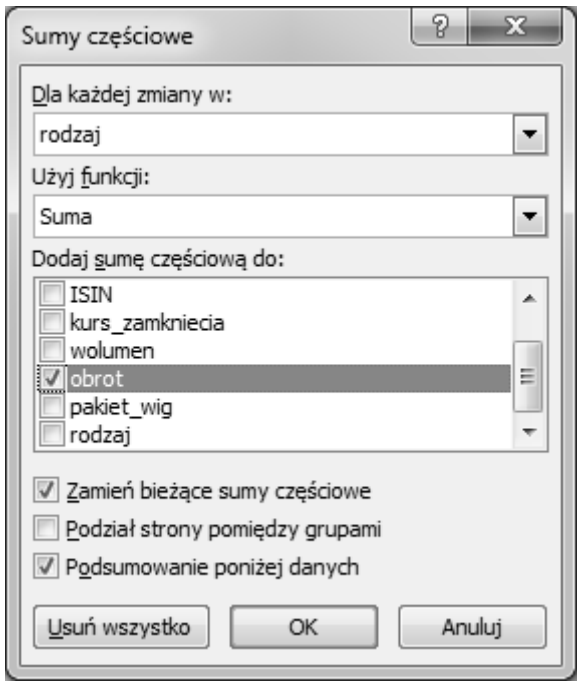
- funkcji `SUMA.JEŻELI`,
- narzędzia z karty Dane — `Sumy częściowe`: dla każdej zmiany w polu `rodzaj` używamy funkcji `suma` w polu `obrot` (w Excelu tabelę trzeba wcześniej posortować według rodzaju spółki).

Procentowy udział spółek krajowych w obrotach giełdy obliczamy, dzieląc obroty spółek krajowych przez sumę obrotów wszystkich spółek. Wynik wyrażamy w formacie procentowym z żądaną dokładnością.

Sposób 1 — funkcje warunkowe:

J	K	L	M
rodzaj spółki	liczba spółek	obroty spółek	% udział obrotów
krajowa	=LICZ.JEŻELI(H$2:H$1411;J2)/3	=SUMA.JEŻELI(H$2:H$1411;J2;F$2:F$1411)	=L2/L$4
zagraniczna	=LICZ.JEŻELI(H$2:H$1411;J3)/3	=SUMA.JEŻELI(H$2:H$1411;J3;F$2:F$1411)	=L3/L$4
razem	=SUMA(K2:K3)	=SUMA(L2:L3)	

Sposób 2 — przykładowo: wstawianie sum częściowych zawierających sumę obrotów poszczególnych rodzajów spółek.



95.4.

Obliczenie WIG zaczynamy od wyznaczenia w każdym wierszu wartości rynkowej pakietu spółki w danym dniu. W nowej kolumnie umieszczamy nagłówek: `wartość_rynkowa`, a w kolejnych komórkach tej kolumny formułę, która mnoży liczbę akcji (pole `pakiet_wig`) przez wartość akcji (pole `kurs_zamkniecia`) .

Dla każdego z trzech danych dni:

obliczamy M — sumę wartości rynkowych w tym dniu (możemy zastosować funkcję `SUMA.JEŻELI` z kryterium ustawionym dla daty lub ręcznie zaznaczyć zakres sumy lub wykorzystać narzędzie `Sumy częściowe`),

obliczamy indeks WIG według podanego wzoru, używając obliczonej powyżej wartości M oraz stałych wartości M_b, W_b, K, podanych w treści zadania.

D	E	F	G	H	J	K	L
kurs_zamkniecia_zl	wolumen	obrot_zl	pakiet_wig	wartość rynkowa	Mb =	57140000	
2,09	9	18	6496000	=G2*D2	Wb =	1000	
0,79	25	21	22309000	=G3*D3	K =	96,48213739	
5,8	1090	6270	1852000	=G4*D4			
3,37	10129	34090	48206000	=G5*D5	data	M	WIG
0,3	0	0	0	=G6*D6	42025	=SUMA.JEŻELI(A$2:A$1411;J6;H$2:H$1411)	=K6*K$2/K$1/K$3
32,5	894	29050	13122000	=G7*D7	42026	=SUMA.JEŻELI(A$2:A$1411;J7;H$2:H$1411)	=K7*K$2/K$1/K$3
27,5	718	19710	8143000	=G8*D8	42027	=SUMA.JEŻELI(A$2:A$1411;J8;H$2:H$1411)	=K8*K$2/K$1/K$3

95.5

Celem podjęcia decyzji: ”kup”, ”sprzedaj” lub ”obserwuj” wykonamy zestawienie, w którym dla każdej spółki umieścimy w jednym wierszu kursy zamknięcia z trzech kolejnych dni. Możemy wykorzystać tabelę przestawną, wykonaną na początkowym etapie rozwiązywania zadania 2 (skopiować ją do nowego arkusza, najlepiej wklejając wartości, a nie formuły) lub wykonać tę tabelę ponownie, korzystając z danych źródłowych.

W dwóch nowych kolumnach obliczamy dla każdej spółki różnice kursów zamknięcia akcji w kolejnych dniach:

`zmiana1` = kurs_zamknięcia z dnia 2015-01-**22** – kurs_zamknięcia z dnia 2015-01-**21**
`zmiana2` = kurs_zamknięcia z dnia 2015-01-**23** – kurs_zamknięcia z dnia 2015-01-**22**.

W następnej kolumnie umieścimy pole `decyzja`, które będzie zawierać słowa: ”kup”, ”sprzedaj” lub ”obserwuj”, zależnie od wartości w polach `zmiana1` i `zmiana2` w tym samym wierszu arkusza:

”kup” — gdy pierwsza zmiana jest dodatnia i druga zmiana jest większa od pierwszej,

”sprzedaj” — gdy pierwsza zmiany jest ujemna i druga zmiana jest mniejsza od pierwszej.

Jeśli żadna z powyższych sytuacji nie zachodzi, to warto spółkę tylko obserwować.

Wykorzystamy funkcję warunkową `JEŻELI` i funkcję logiczną `ORAZ`, która utworzy nam iloczyn logiczny dwóch warunków.

=JEŻELI(ORAZ(E2>0;F2>E2);"kup"; JEŻELI(ORAZ(E2<0;F2<E2);"sprzedaj";"obserwuj"))

	A	B	C	D	E	F	G
1	**Etykiety wierszy**	**2015-01-21**	**2015-01-22**	**2015-01-23**	zmiana1	zmiana2	decyzja
2	06MAGNA	2,09	2,26	2,14	0,17	-0,12	obserwuj
3	08OCTAVA	0,79	0,79	0,79	0	0	obserwuj

Następnie zliczymy wystąpienia słów: „kup”, „sprzedaj” i ”obserwuj” za pomocą funkcji warunkowej `LICZ.JEŻELI`

H	I	J
		ile
	kup	=LICZ.JEŻELI(G$2:G$471;I2)
	sprzedaj	=LICZ.JEŻELI(G$2:G$471;I3)
	obserwuj	=LICZ.JEŻELI(G$2:G$471;I4)

Zadanie 96.

96.1.

W zadaniu należy porównać strukturę wiekową ludności w dwóch latach kalendarzowych: 2013 i 2050, przedstawiając ją na wykresie punktowym.

W każdym wierszu tabeli źródłowej obliczamy łączną liczbę ludności: sumę liczby kobiet i mężczyzn z miast oraz wsi. Następnie kopiujemy liczby ludności dotyczące roku 2050 i umieszczamy je obok wyników z roku 2013, tak aby w jednym wierszu znalazły się liczby ludności z roku 2013 i 2050 dla wszystkich grup wiekowych od 0 do 100.

	A	B	G	H
1	rok	wiek	rok 2013	rok 2050
2	2013	0	=SUMA(C2:F2)	=G3739
3	2013	1	=SUMA(C3:F3)	=G3740
4	2013	2	=SUMA(C4:F4)	=G3741
5	2013	3	=SUMA(C5:F5)	=G3742

Tak przygotowane dane umożliwiają utworzenie wykresu, obejmującego dwie serie danych: liczbę ludności w roku 2013 i 2050 w tym samym zakresie kategorii: w wieku od 0 do 100.

96.2.

Należy obliczyć stosunek liczby ludności miast do liczby ludności wsi w roku 2013 i w roku 2050.

Do tabeli źródłowej dodajemy dwie nowe kolumny: `miasto` i `wieś`. W komórkach tych kolumn, w każdym wierszu, sumujemy liczby kobiet i mężczyzn, odpowiednio z miast i wsi. Następnie trzeba obliczyć sumę liczb ludności miast i wsi w każdym roku uwzględniającą wszystkie grupy wiekowe.

Wybierzemy narzędzie **sumy częściowe**: dla każdego roku zastosujemy funkcję `Suma` w polu `miasto` i w polu `wieś`. Ukryjemy wiersze składające się na sumę częściową roku (wybierając przycisk 2 na listwie z lewej strony ekranu).

Zamiast sum częściowych można po prostu użyć funkcji `SUMA` i ręcznie zaznaczać zakres sumowania dla ludności miast i ludności wsi z roku 2013 i roku 2050, lecz jest to bardziej pracochłonne.

W dwóch wierszach, przedstawiających sumy dla roku 2013 i 2050, obliczamy iloraz: suma ludności z miast przez sumę ludności wsi. Żądaną dokładność wyników zapewniamy, wybierając 2 miejsca dziesiętne w formacie liczbowym.

	A	G	H	I
1	rok	miasto	wies	miasto/wies
2	2013	=C2+D2	=E2+F2	
100	2013	=C100+D100	=E100+F100	
101	2013	=C101+D101	=E101+F101	
102	2013	=C102+D102	=E102+F102	
103	**2013 Suma**	=SUMY.CZĘŚCIOWE(9;G2:G102)	=SUMY.CZĘŚCIOWE(9;H2:H102)	=G103/H103
205	**2014 Suma**	=SUMY.CZĘŚCIOWE(9;G104:G204	=SUMY.CZĘŚCIOWE(9;H104:H204)	=G205/H205

96.3.

Należy obliczyć średnią ważoną wieku mężczyzny w mieście.

Obok tabeli źródłowej tworzymy nową kolumnę o nazwie `licznik`. W jej komórkach obliczymy kolejne elementy licznika ze wzoru podanego w treści zadania, w każdym wierszu iloczyn: `wiek` * liczba osób w tym wieku.

Następnie zastosujemy narzędzie **sumy częściowe**: dla każdego `roku` użyjemy funkcji `Suma` do pola `m_miasto` oraz do pola `licznik`.

Ukryjemy wiersze składające się na sumy częściowe (za pomocą przycisku 2 na listwie z lewej strony ekranu). W wierszach dotyczących roku 2013 i roku 2050 podzielimy wynik sumy częściowej pola `licznik` przez wynik sumy częściowej pola `m_miasto`. Końcowe wyniki sformatujemy z żądaną dokładnością.

96.4.

Dla każdego roku od 2013 do 2050 należy znaleźć najniższy wiek, w jakim kobiety przeważają liczebnie nad mężczyznami.

Do źródłowej tabeli danych dodajmy 4 nowe kolumny o następujących nazwach: `kobiety`, `mężczyźni`, `k>m`, `zmiana`.

W kolumnach `kobiety` i `mężczyźni` obliczymy w każdym wierszu liczbę kobiet (łącznie z miast i wsi) oraz liczbę mężczyzn (łącznie z miast i wsi).

W kolumnie `k>m` użyjemy funkcji warunkowej `JEŻELI`. Jeżeli w danym wierszu liczba kobiet jest większa od liczby mężczyzn, niech funkcja umieści w komórce wartość 1, jeśli nie — wartość 0.

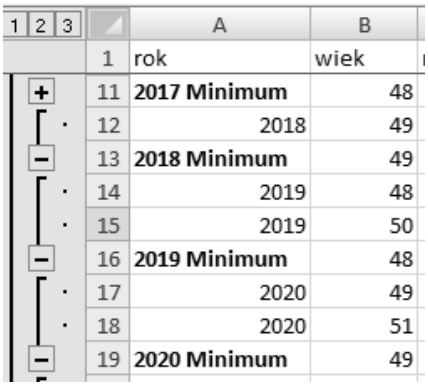
W kolumnie `zmiana` także użyjemy funkcji warunkowej `JEŻELI`, która sprawdzi, czy w bieżącym wierszu jest przewaga kobiet, a jednocześnie — czy w poprzednim wierszu nie ma przewagi kobiet. Jeżeli ma miejsce taka sytuacja, funkcja zwróci wartość 1, w przeciwnym razie zwróci 0.

G	H	I	J
kobiety	mężczyźni	k>m	zmiana
=D51+F51	=C51+E51	=JEŻELI(G51>H51;1;0)	=JEŻELI(ORAZ(I51=1;I50=0);1;0)
=D152+F152	=C152+E152	=JEŻELI(G152>H152;1;0)	=JEŻELI(ORAZ(I152=1;I151=0);1;0)

Wykorzystując **filtr,** wybierzemy teraz te wiersze, w których pole `zmiana` ma wartość 1.

Okazuje się, że dla niektórych lat kalendarzowych liczba kobiet w pewnym wieku zaczyna przewyższać liczbę mężczyzn, lecz ta przewaga nie jest trwała. W grupie o rok starszej znów przeważają mężczyźni, a w następnej grupie wiekowej — znów kobiety. Aby wyznaczyć dla

każdego roku kalendarzowego **najmłodszy** wiek kobiet, w którym ich liczba przekracza liczbę mężczyzn, można powyższe wyniki filtru skopiować do nowego arkusza i użyć narzędzia `sumy częściowe`: dla każdej zmiany w polu `rok` użyć funkcji `Minimum` w polu `wiek`.



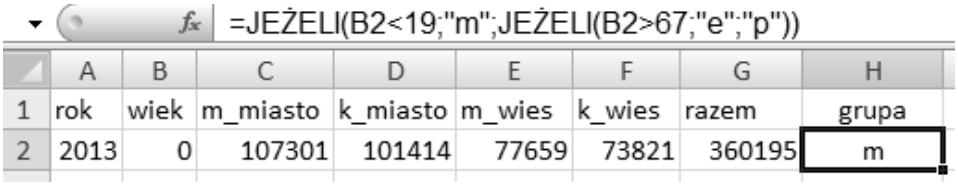
	A	B
1	rok	wiek
11	**2017 Minimum**	48
12	2018	49
13	**2018 Minimum**	49
14	2019	48
15	2019	50
16	**2019 Minimum**	48
17	2020	49
18	2020	51
19	**2020 Minimum**	49

96.5.

Należy wyznaczyć w każdym roku kalendarzowym liczbę osób należących do trzech grup wiekowych: młodzieży (0-18), osób w wieku produkcyjnym (19-67) i emerytów (68-100).

Przede wszystkim zsumujemy liczbę kobiet i mężczyzn z miast oraz wsi w każdym wierszu źródłowej tabeli danych. W ten sposób nasza tabela zyska dodatkową kolumnę, nazwijmy ją `razem`.

Następnie utworzymy nową kolumnę o nazwie `grupa`. Za pomocą funkcji warunkowej `JEŻELI` umieścimy komórkach tej kolumny jedną z liter: *m* (jak młodzież), *p* (produkcyjni) lub *e* (emeryt), zależnie od pola `wiek` w tym wierszu tabeli.



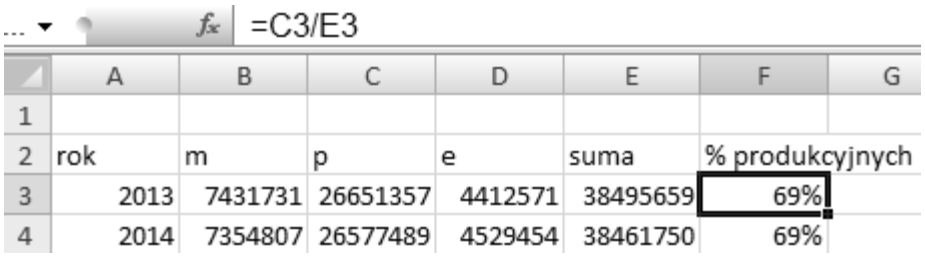
=JEŻELI(B2<19;"m";JEŻELI(B2>67;"e";"p"))

	A	B	C	D	E	F	G	H
1	rok	wiek	m_miasto	k_miasto	m_wies	k_wies	razem	grupa
2	2013	0	107301	101414	77659	73821	360195	m

Następnie wykonamy **tabelę przestawną**: w nagłówkach wierszy umieścimy pole `rok`, w nagłówkach kolumn — pole `grupa`, a jako wartości danych przyjmiemy pole `razem`.W utworzonej już tabeli przestawnej możemy zmienić kolejność kolumn, aby ustawić je rosnąco według wieku: najpierw młodzież, później osoby w wieku produkcyjnym, na końcu emerytów. W tym celu zaznaczamy komórkę nagłówka kolumny emerytów (z literą „e"), ustawiamy kursor na krawędzi komórki i ciągniemy w prawo na nowe miejsce.

Na podstawie zawartości tabeli przestawnej wykonujemy wykres kolumnowy skumulowany.

Aby obliczyć procent osób w wieku produkcyjnym w kolejnych latach, utworzymy nową kolumnę obok tabeli przestawnej. Wyznaczymy w niej dla każdego roku, stosunek liczby osób w wieku produkcyjnym do liczby wszystkich osób w tym roku. Dla całej tej kolumny wybierzemy format % liczb, podając wyniki obliczeń z dokładnością do 0 cyfr po przecinku.



=C3/E3

	A	B	C	D	E	F	G
1							
2	rok	m	p	e	suma	% produkcyjnych	
3	2013	7431731	26651357	4412571	38495659	69%	
4	2014	7354807	26577489	4529454	38461750	69%	

Zadanie 97.

Zadania rozwiążemy z pomocą MS Excel, najpierw wczytując dane do arkusza tak, aby wiersz 1 zawierał nagłówki kolumn, a dane o stopie bezrobocia w kolejnych miesiącach znajdowały się w kolumnach `B-M`.

97.1.

Rozwiązanie pierwszego zadania uzyskamy, wykorzystując funkcję `LICZ.JEŻELI` do wyznaczenia liczby miesięcy, w których stopa bezrobocia jest większa od 10:

```
=LICZ.JEŻELI($B$2:$M$71;">10"),
```

Zakres danych w powyższej formule stanowią wszystkie komórki z miesięcznymi stopami bezrobocia (`$B$2:$M$71`), zaś napis ”>10” oznacza wyrażenie logiczne, określające, które komórki będą uwzględniane przy zliczaniu. Należy pamiętać, że wszelkie kryteria zawierające symbole matematyczne lub logiczne (a także tekstowe) należy ująć w podwójny cudzysłów(”).

97.2.

W tym zadaniu musimy podać najniższą i najwyższą średnią roczną stopę bezrobocia oraz rok, w którym zostałe ony odnotowane. W tym celu dodajemy kolejną kolumnę `Średnia roczna stopa bezrobocia` i w komórce `N2` budujemy formułę `=ŚREDNIA(B2:M2)`, którą przekopiowujemy do pozostałych komórek w kolumnie. Zestawienie sortujemy malejąco ze względu na otrzymane średnie, dzięki czemu na pierwszej pozycji znajdzie się rok z najniższą średnią roczną stopą bezrobocia, a na ostatniej pozycji rok z najwyższą średnią.

97.3.

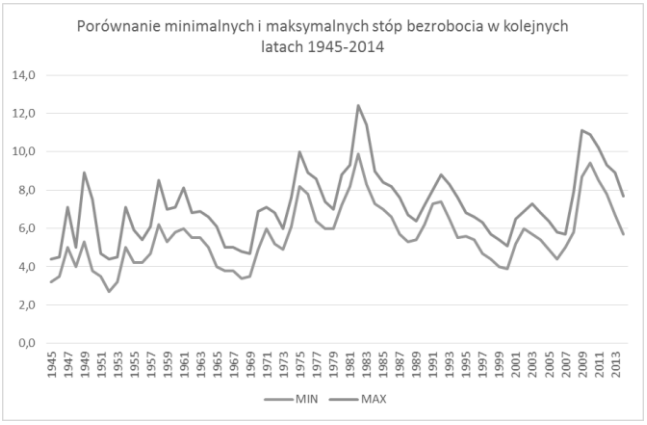
Sporządzenie wykresu rozpoczynamy od przygotowania danych. W tym celu dodajemy dwie kolumny, w których wyznaczamy odpowiednio minimalną (kolumna B) i maksymalną (kolumna C) miesięczną stopę bezrobocia dla każdego roku.

	A	B	C
1	ROK	MIN	MAX
2	1945	=MIN(Stopa_bezrobocia!B2:M2)	=MAX(Stopa_bezrobocia!B2:M2)
3	1946	=MIN(Stopa_bezrobocia!B3:M3)	=MAX(Stopa_bezrobocia!B3:M3)
4	1947	=MIN(Stopa_bezrobocia!B4:M4)	=MAX(Stopa_bezrobocia!B4:M4)
5	1948	=MIN(Stopa_bezrobocia!B5:M5)	=MAX(Stopa_bezrobocia!B5:M5)
6	1949	=MIN(Stopa_bezrobocia!B6:M6)	=MAX(Stopa_bezrobocia!B6:M6)
7	1950	=MIN(Stopa_bezrobocia!B7:M7)	=MAX(Stopa_bezrobocia!B7:M7)

Na podstawie otrzymanego zestawienia:

	A	B	C
1	ROK	MIN	MAX
2	1945	3,2	4,4
3	1946	3,5	4,5
4	1947	5,0	7,1
5	1948	4,0	5,0
6	1949	5,3	8,9
7	1950	3,8	7,5
8	1951	3,5	4,7

wstawiamy wykres liniowy:



97.4.

W zadaniu mamy znaleźć najdłuższy nierosnący ciąg miesięcznych stóp bezrobocia w kolejnych miesiącach. Wykonamy to, budując formułę logiczną, w której dla każdego miesiąca wyznaczamy największą długość ciągu nierosnącego, w którym dany miesiąc jest ostatnim elementem (styczeń 1945 roku ma wartość początkową równą 0). Z uwagi na sposób rozmieszczenia danych formuła ta ma dwa warianty:

- dla miesięcy od lutego do grudnia: `=JEŻELI(B2>=C2;O2+1;1);`
- dla stycznia: `=JEŻELI(M2>=B3;Z2+1;1).`

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	ROK	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII		I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII
2	1945	3,2	4,2	4,1	3,9	3,9	4,0	4,0	4,2	4,4	4,1	4,0	3,8		0	1	2	3	4	1	2	1	1	2	3	4
3	1946	4,5	3,9	3,6	3,6	3,7	3,5	3,5	3,6	3,7	3,9	4,1	4,5		1	2	3	4	1	2	3	1	1	1	1	1
4	1947	5,0	5,9	6,2	6,7	6,9	6,9	6,6	6,8	7,0	7,1	6,7	6,3		1	1	1	1	1	2	3	1	1	1	2	3
5	1948	4,0	4,4	4,8	5,0	4,9	4,5	4,6	4,6	4,9	4,8	4,7	4,8		4	1	1	1	2	3	1	2	1	2	3	1
6	1949	6,6	5,3	5,7	6,0	6,3	7,1	7,2	7,7	7,8	7,6	8,9	7,4		1	2	1	1	1	1	1	1	1	2	1	2

Wielkość najdłuższego nierosnącego ciągu miesięcznych stóp bezrobocia w kolejnych miesiącach jest maksymalna wartość z zakresu (`=MAX(O2:Z71)`). Miesiąc i rok końca znalezionego ciągu odnajdujemy, wyszukując wyznaczoną wcześniej wartość maksimum w tym samym zakresie komórek, i na tej podstawie znajdujemy rozwiązanie.

97.5.

Rozwiązanie rozpoczynamy od dodania kolumny `Czy większe w każdym miesiącu?` i zbudowania w komórce `N3` formuły logicznej, sprawdzającej dla danego roku, czy stopa bezrobocia w każdym jego miesiącu była większa niż w tym samym miesiącu roku poprzedniego:

```
=JEŻELI(ORAZ(B3>B2;C3>C2;D3>D2;E3>E2;F3>F2;G3>G2;H3>H2;I3>I2;
J3>J2;K3>K2;L3>L2;M3>M2);1;0)
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	Czy większe w każdym miesiącu?
2	1945	3,2	4,2	4,1	3,9	3,9	4,0	4,0	4,2	4,4	4,1	4,0	3,8	
3	1946	4,5	3,9	3,6	3,6	3,7	3,5	3,5	3,6	3,7	3,9	4,1	4,5	0
4	1947	5,0	5,9	6,2	6,7	6,9	6,9	6,6	6,8	7,0	7,1	6,7	6,3	1
5	1948	4,0	4,4	4,8	5,0	4,9	4,5	4,6	4,6	4,9	4,8	4,7	4,8	0
6	1949	6,6	5,3	5,7	6,0	6,3	7,1	7,2	7,7	7,8	7,6	8,9	7,4	1
7	1950	4,3	7,5	7,4	7,3	6,8	6,5	6,4	6,0	5,5	5,0	4,5	3,8	0

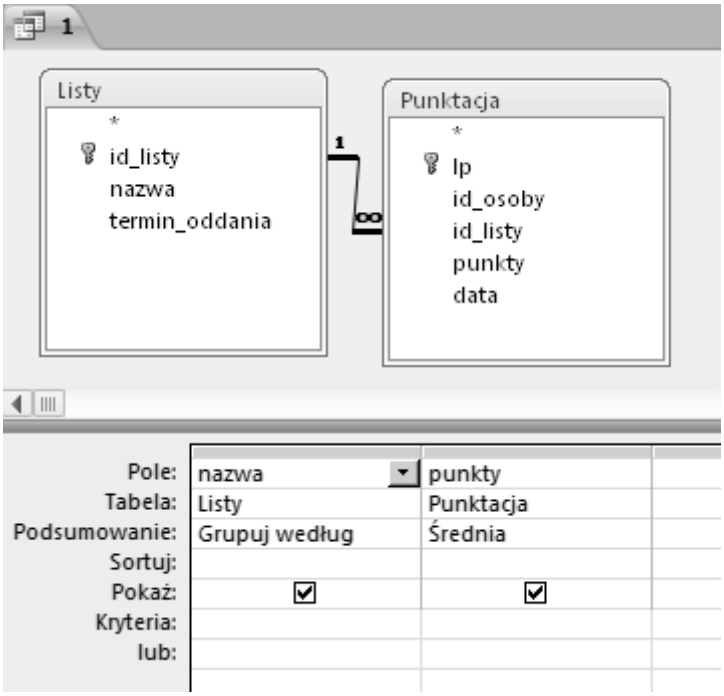
Sumując wartości w kolumnie N (`=SUMA(N3:N71)`), otrzymamy szukaną liczbę lat.

Zadanie 100.

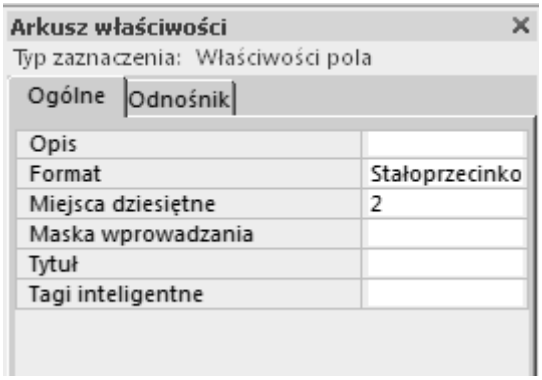
Po wczytaniu danych do tabel oraz ustaleniu związków między tabelami (Relacje) można przystąpić do rozwiązania zadania.

100.1.

W tym zadaniu do utworzenia zapytania potrzebne są 2 tabele: `Listy` oraz `Punktacja`. Z pierwszej tabeli wybieramy nazwę listy zadań, a z drugiej tabeli — liczbę punktów. Ponieważ chcemy policzyć średnią liczbę punktów dla każdej listy, nazwy list grupujemy, a dla pola `punkty` z tabeli `Punktacja` wybieramy `Średnia`.



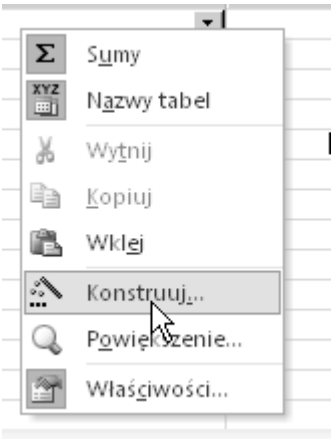
Zauważmy jeszcze, że średnie wartości mają być podane z dokładnością do dwóch miejsc po przecinku. Można to zrobić, np. ustawiając właściwości pola `punkty` w arkuszu właściwości: wybieramy format stałoprzecinkowy z dwoma miejscami po przecinku.

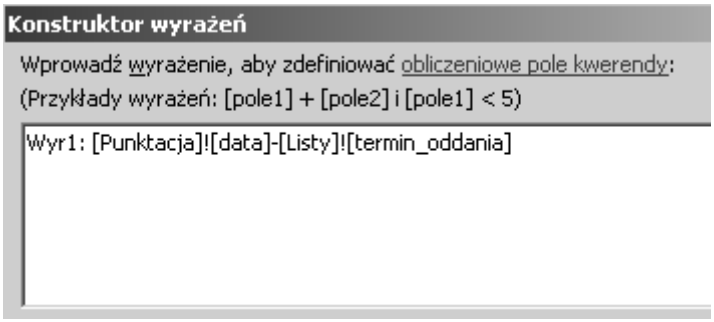


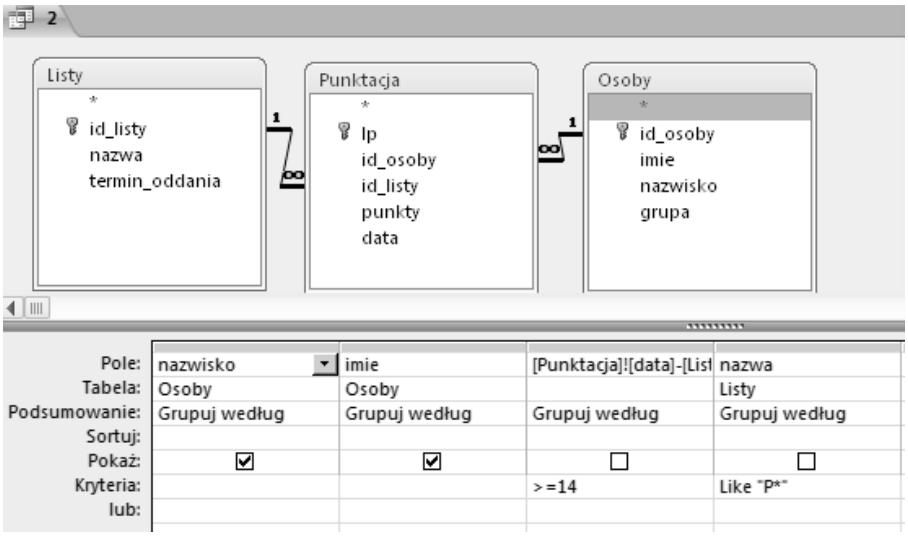
100.2.

Do zbudowania zapytania potrzebujemy wszystkich tabel. Musimy wybrać imiona i nazwiska osób (tabela `Osoby`) oraz nazwę listy (tabela `Listy`). Ponieważ lista ma rozpoczynać się od litery „P”, w wierszu filtrującym (*Kryteria*) wystarczy wpisać: `Like "P*"`. Dodatkowo mamy wybrać osoby, które spóźniły się z oddaniem listy o 14 dni lub więcej. Ponieważ wie-

my, kiedy osoby przesyłały zadania oraz jaki był termin oddania listy zadań, wystarczy zbadać różnicę tych dat. Można to zrobić poprzez zbudowanie odpowiedniego wyrażenia. Następnie w kryteriach ustawiamy, że obliczona różnica ma być większa bądź równa 14.

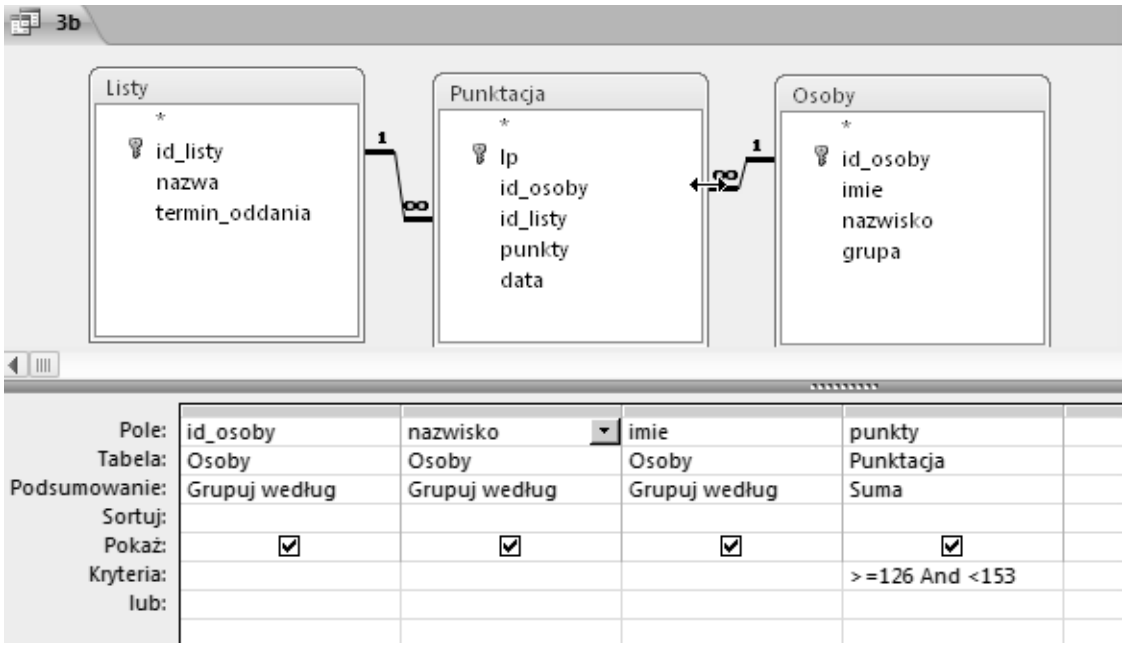






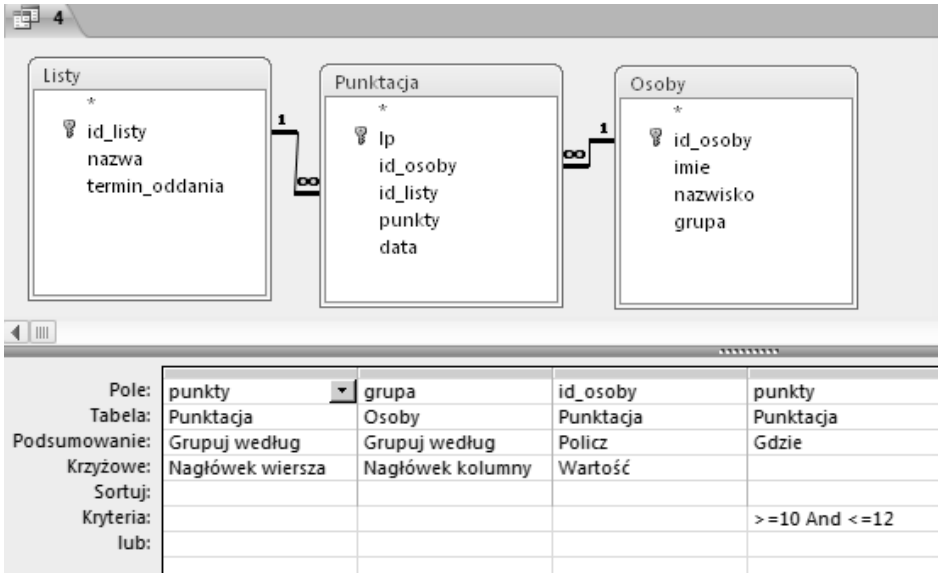
100.3.

Zadanie można wykonać na kilka sposobów. Jeśli wybierzemy narzędzie bazodanowe, to najprościej jest utworzyć jedną kwerendę, wybierającą osoby, których suma punktów mieści się w zadanym przedziale, a następnie odczytać, ile takich osób było (sprawdzając liczbę rekordów pojawiających się w wyniku uruchomienia zapytania). Poniżej prezentujemy kwerendę wyznaczającą liczbę osób z oceną cztery. Analogicznie wyznaczamy pozostałe wartości.



100.4.

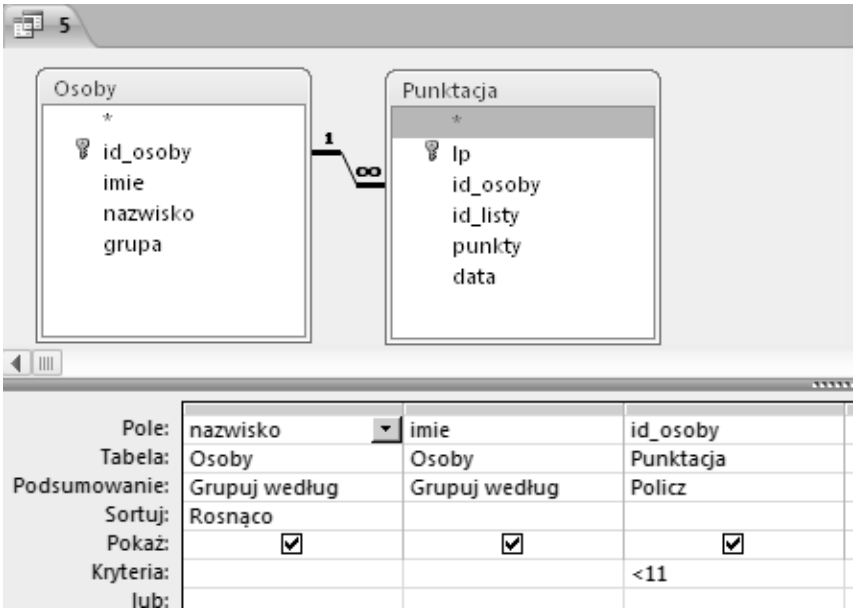
Zadanie można rozwiązać na 2 sposoby. W pierwszym przypadku można dla każdej grupy obliczyć, ile osób otrzymało 10, 11, 12 punktów. W rezultacie trzeba zrobić 15 kwerend, co jest dość czasochłonne. Można też wykorzystać kwerendę krzyżową, która pozwoli wykonać czytelne zestawienie. Przy tworzeniu takiej kwerendy (Tworzenie → Projekt kwerendy) wybieramy z karty *Projektowanie* typ kwerendy *Krzyżowa*. Pole `punkty` z tabeli `Punktacje` ustawiamy jako nagłówek wiersza, pole `grupa` z tabeli `Osoby` — jako nagłówek kolumny. Natomiast identyfikatory osób (`id_osoby`) ustawimy w wierszu `Krzyżowe` jako wartość. Dodatkowo chcemy, aby punktacja uwzględniała tylko 10, 11 oraz 12 punktów, dlatego dodajemy odpowiednie kryterium.



100.5.

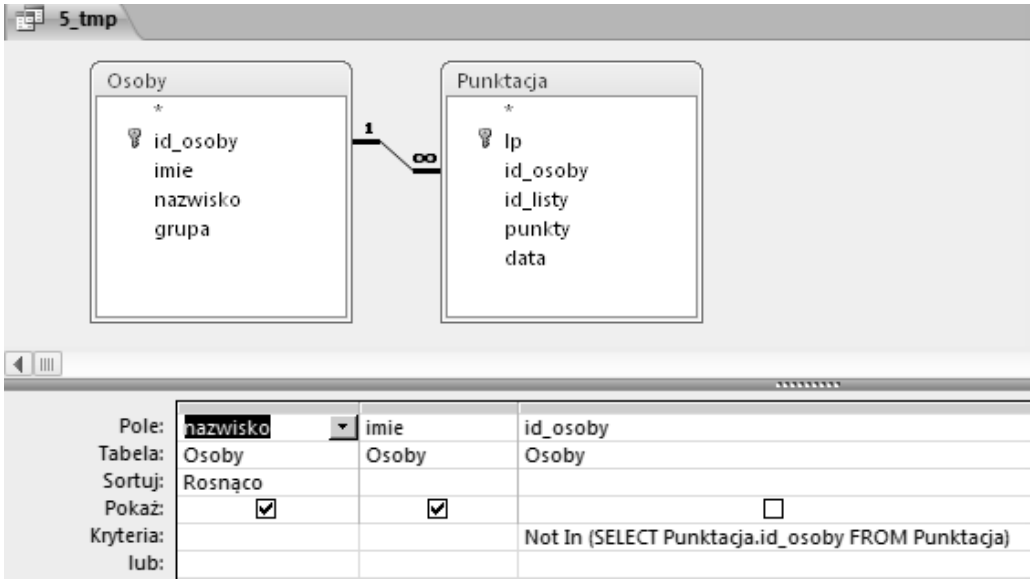
W tym zadaniu trzeba zwrócić uwagę na sformułowanie polecenia: nie chodzi o to, żeby podać osoby, które nie wysłały żadnej listy zadań, ale o osoby, które nie wysłały co najmniej

jednej listy zadań. Ponieważ wszystkich list zadań jest 11, wystarczy sprawdzić, kto oddał mniej niż 11 list zadań.



Trzeba jednak pamiętać o tym, że takie podejście nie gwarantuje, że znajdziemy osobę, która nie oddała żadnej listy zadań. Żeby sprawdzić, czy jest taka osoba, można np. napisać zapytanie w języku SQL, w którym wybierzemy te imiona i nazwiska osób, których identyfikatory nie występują w tabeli `Punktacja`:

```
SELECT Osoby.nazwisko, Osoby.imie
FROM Osoby INNER JOIN Punktacja
ON Osoby.id_osoby = Punktacja.id_osoby
WHERE Osoby.id_osoby NOT IN
SELECT Punktacja.id_osoby FROM Punktacja)
ORDER BY Osoby.nazwisko;
```



Zadanie 101.

Zadanie rozwiążemy za pomocą MS Access. Zawartości plików `osoby.txt`, `zajecia.txt` i `wejścia.txt` wczytamy do tabel `Osoby`, `Zajecia` i `Wejscia`, które połączymy odpowiednimi relacjami (szczegóły pozostawiamy czytelnikowi).

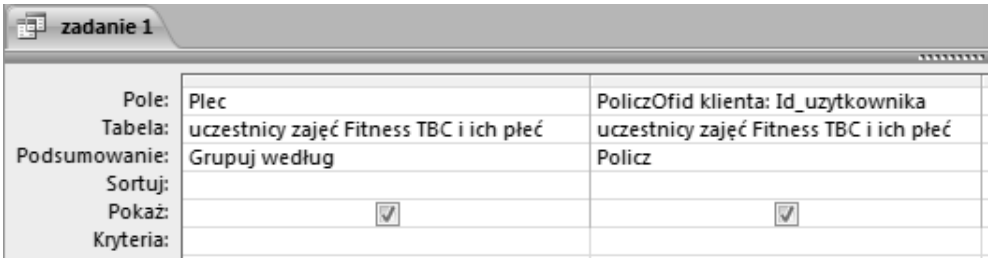
101.1.

W zadaniu należy obliczyć liczbę kobiet i mężczyzn uczestniczących w zajęciach `Fitness TBC`. Do uzyskania odpowiedzi potrzebne nam są dane z trzech tabel:

- `Id_uzytkownika` z tabeli `Wejscia` — do zliczenia liczby wejść,
- rodzaj zajęć z tabeli `Zajecia`,
- płeć uczestnika zajęć z tabeli `Osoby`.

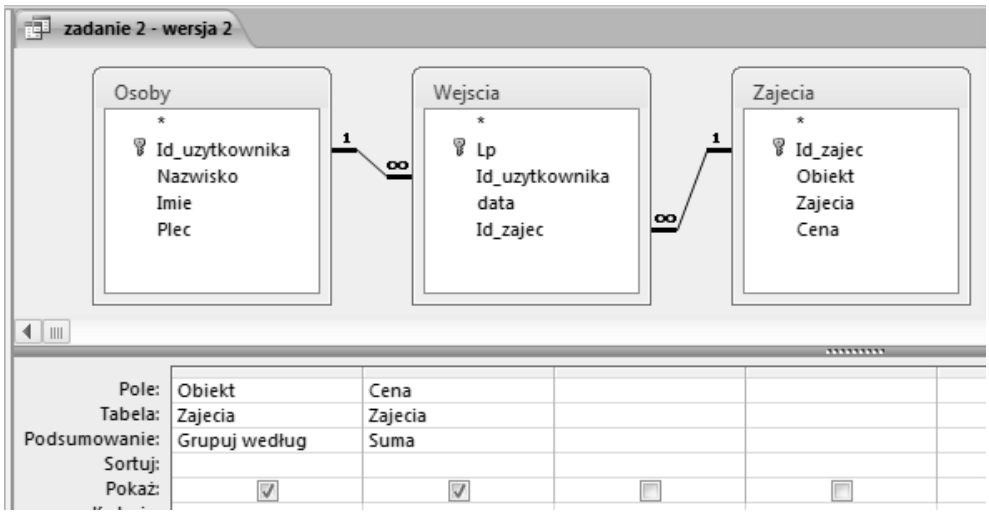
Zadanie rozwiążemy w dwóch krokach. W kroku pierwszym utworzymy pomocniczą kwerendę grupującą użytkowników według pola `Id_uzytkownika`, aby być pewnym, że policzymy każdą osobę jednokrotnie. W kwerendzie uwzględnimy tylko uczestników zajęć `Fitness TBC`. Widok projektu tej kwerendy prezentujemy poniżej:

W kroku drugim tworzymy kwerendę, dla której jako źródło danych wybieramy utworzoną wcześniej kwerendę pomocniczą `uczestnicy zajęć Fitness TBC i ich płeć`. Następnie grupujemy użytkowników według pola płeć i używamy funkcji `Policz` do zliczenia liczby wejść.



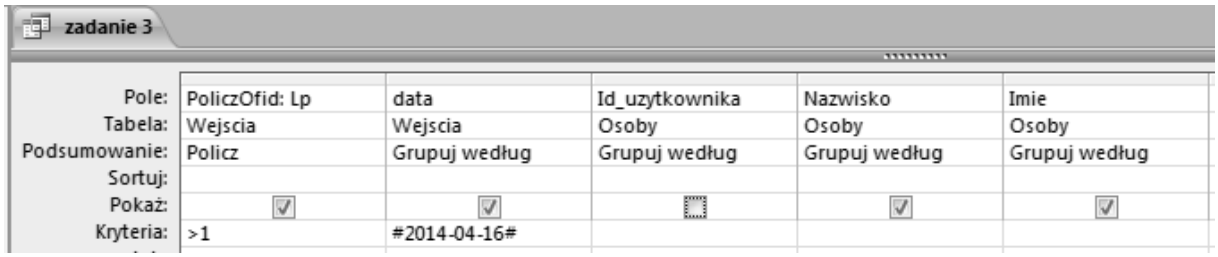
101.2.

Do rozwiązania zadania wykorzystamy złączenie wszystkich trzech tabel: `Osoby`, `Wejscia` i `Zajecia`, które gwarantuje nam, że każde wejście na zajęcia jest uwzględnione jednokrotnie. Jeśli pogrupujemy te wejścia według nazw obiektu i zsumujemy ceny wejść, uzyskamy poprawnie wyliczony koszt zajęć.



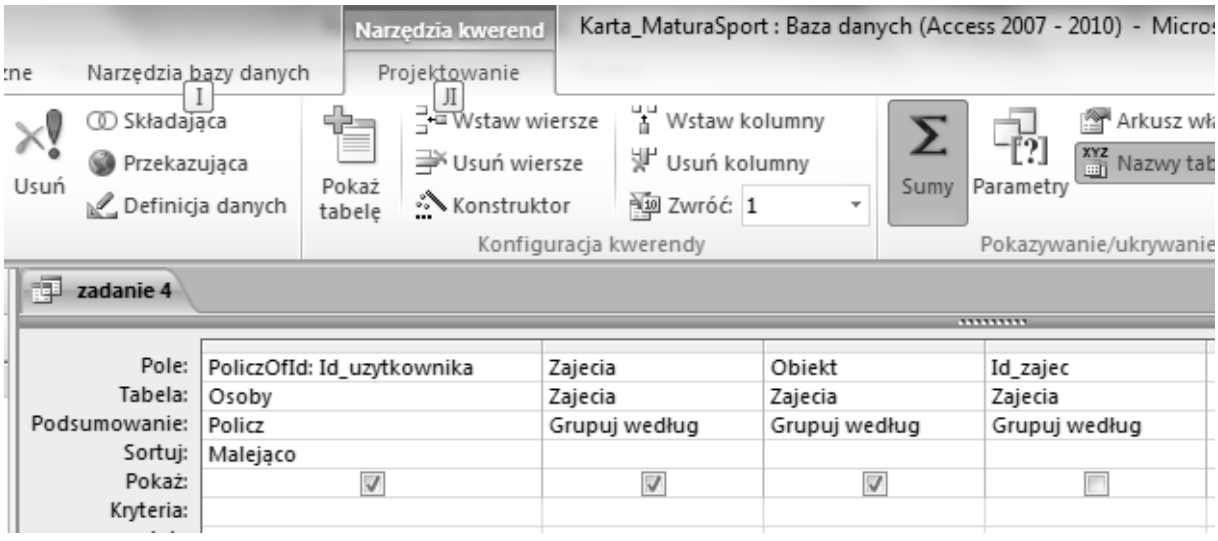
101.3.

Aby rozwiązać zadanie, zliczymy liczbę wejść poszczególnych osób na dowolne zajęcia w dniu 16 kwietnia oraz podamy nazwiska i imiona osób, dla których liczba wejść jest większa od 1. W tym celu tworzymy poniższą kwerendę:



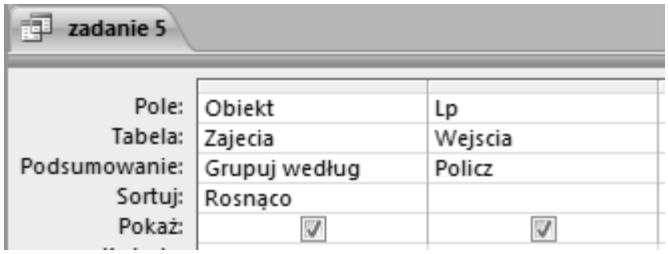
101.4.

W kwerendzie zliczamy liczbę osób uczestniczących w poszczególnych zajęciach i prezentujemy rodzaj zajęć oraz nazwę obiektu, w którym były one prowadzone. Uzyskane wyniki sortujemy malejąco i ustalamy, że kwerenda ma zwracać tylko pierwszy wiersz, zawierający największą liczbę uczestników. W tym celu na karcie `Projektowanie` ustawiamy wartość parametru `Zwróć` na 1.



101.5.

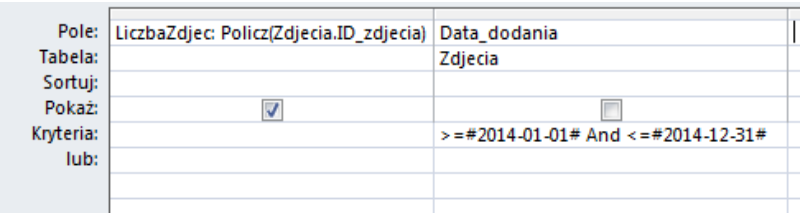
W kwerendzie zliczamy liczbę wejść na zajęcia oraz grupujemy i sortujemy rosnąco dane według kolumny `Obiekt` z tabeli `Zajecia`.



Zadanie 102.

102.1.

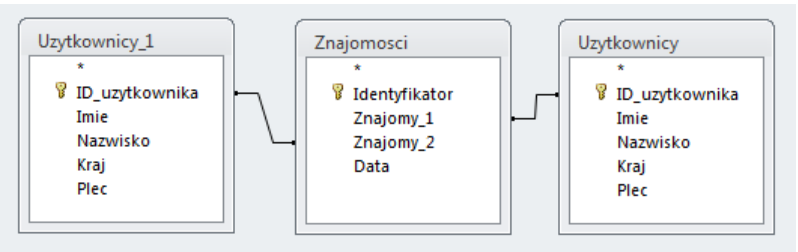
W tym zadaniu wystarczy wyfiltrować zdjęcia dodane między 1 stycznia a 31 grudnia 2014 (klauzula WHERE), a następnie podać ich liczbę (klauzula COUNT). Odpowiednia kwerenda SQL wygląda następująco:



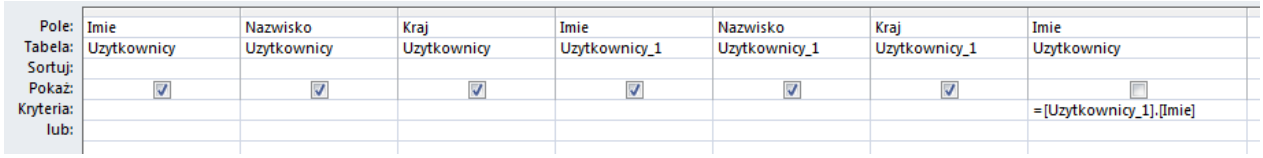
SELECT COUNT(Zdjecia.ID_zdjecia) **AS** LiczbaZdjec
FROM Zdjecia
WHERE (((Zdjecia.Data_dodania)>=#1/1/2014# **AND** (Zdjecia.Data_dodania)<=#12/31/2014#));

102.2.

Tutaj konieczne jest połączenie tabeli *znajomosci* z tabelą *uzytkownicy*. Dla każdej znajomości — wiersza bazy przechowującego dwa identyfikatory użytkowników — chcemy znaleźć imiona użytkowników ukrywających się pod tymi identyfikatorami. Połączymy zatem tabelę *znajomosci* z dwiema kopiami tabeli *uzytkownicy* dwukrotnie, raz według pola Znajomy_1, drugi raz według pola Znajomy_2:



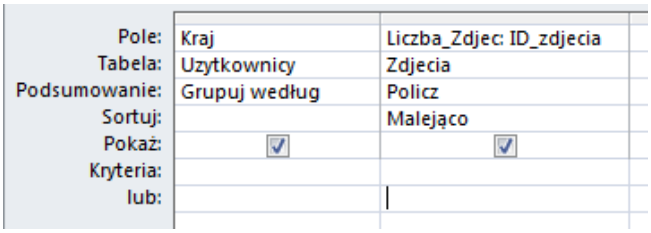
Teraz już wystarczy wyfiltrować wiersze, w których obaj przyłączeni użytkownicy mają tak samo na imię.



SELECT Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Uzytkownicy.Kraj, Uzytkownicy_1.Imie, Uzytkownicy_1.Nazwisko, Uzytkownicy_1.Kraj
FROM (Znajomosci **INNER JOIN** Uzytkownicy **ON** Znajomosci.Znajomy_1 = Uzytkownicy.ID_uzytkownika) **INNER JOIN** Uzytkownicy **AS** Uzytkownicy_1 **ON** Znajomosci.Znajomy_2 = Uzytkownicy_1.ID_uzytkownika
WHERE (((Uzytkownicy.Imie)=[Uzytkownicy_1].[Imie]));

102.3.

Oczywiste jest, że należy połączyć tabele *uzytkownicy* oraz *zdjecia* po wspólnym polu przechowującym ID użytkownika. Powstałą złączoną tabelę trzeba teraz zgrupować według nazwy kraju, licząc dla każdej nazwy liczbę wierszy:

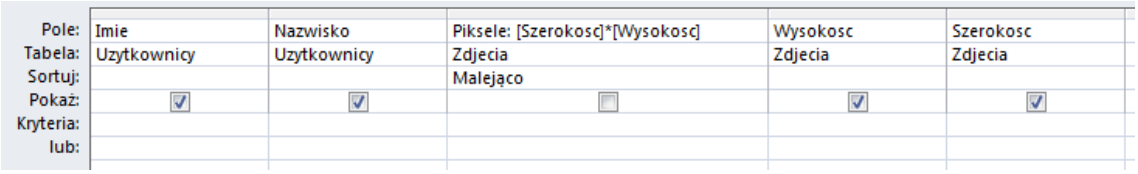


SELECT TOP 10 Uzytkownicy.Kraj, Count(Zdjecia.ID_zdjecia) **AS** Liczba_Zdjec
FROM Uzytkownicy **INNER JOIN** Zdjecia
ON Uzytkownicy.ID_uzytkownika = Zdjecia.Uzytkownik
GROUP BY Uzytkownicy.Kraj
ORDER BY Count(Zdjecia.ID_zdjecia) **DESC**;

Na końcu sortujemy tabelę według liczby zdjęć i ograniczamy wyniki do pierwszych dziesięciu (klauzula TOP lub LIMIT albo odpowiednia opcja w MS Access), zgodnie z poleceniem zadania.

102.4.

Znowu łączymy tabele *uzytkownicy* oraz *zdjecia*. Tym razem jednak chcemy dla każdego zdjęcia obliczyć liczbę pikseli, czyli jego szerokość pomnożoną przez wysokość:



SELECT TOP 1 Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Zdjecia.Wysokosc, Zdjecia.Szerokosc
FROM Uzytkownicy **INNER JOIN** Zdjecia
ON Uzytkownicy.ID_uzytkownika = Zdjecia.Uzytkownik
ORDER BY Zdjecia.[Szerokosc]*[Wysokosc] **DESC**;

Zwracamy tylko jeden, najlepszy wynik.

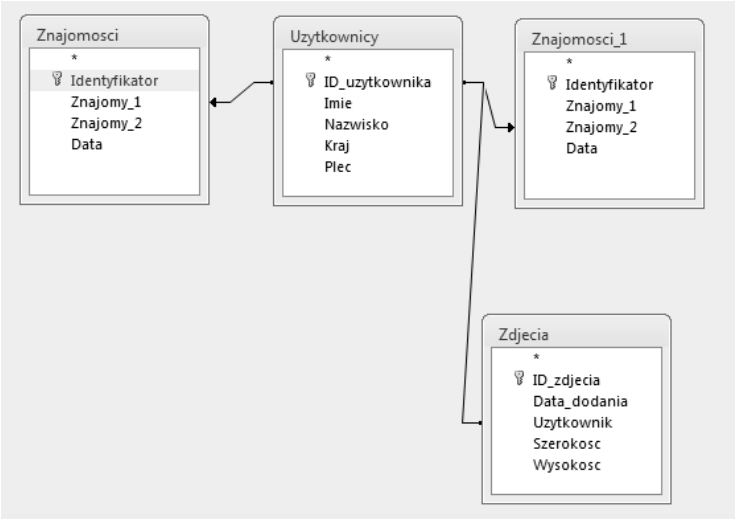
102.5.

Zadanie to wymaga nieco więcej pracy niż poprzednie. Jednym z możliwych sposobów rozwiązania go jest znalezienie dla każdego użytkownika, najwcześniej dodanego zdjęcia oraz najwcześniej nawiązanej znajomości. Będą nas interesowali ci użytkownicy, dla których data

pierwszego zdjęcia jest wcześniejsza (pierwsze zdjęcie pojawiło się wcześniej niż pierwsza znajomość).

Najwcześniejszą datę zdjęcia wyznaczamy, łącząc tabelę *uzytkownicy* z tabelą *zdjecia*, grupując po *ID_uzytkownika* i biorąc minimum każdej grupy.

Najwcześniejszą zawartą znajomość można zrobić podobnie, napotykamy tutaj na dodatkową trudność — czy połączyć pole *ID_uzytkownika* z polem *Znajomy_1* czy *Znajomy_2*. Przy każdym z tych połączeń dostaniemy inną najwcześniejszą datę — jedna będzie odpowiadała pierwszej znajomości, w której użytkownik występuje jako *Znajomy_1*, druga zaś występowaniu jako *Znajomy_2*. Potrzebujemy więc obu tych dat, a szukać będziemy tych użytkowników, którzy dodali zdjęcie wcześniej niż wypada którakolwiek z nich. Połączmy użytkowników ze znajomymi i zdjęciami w jednej kwerendzie:



Pomocnicza kwerenda, która liczy trzy najwcześniejsze daty (datę dodania zdjęcia jako *Data_pierwszego_zdjecia*, zawarcia znajomości jako *Data_zawarcia_1* oraz *Data_zawarcia_2*), wyglądać może tak:

Pole:	Nazwisko	Kraj	Data_pierwszego_zdjecia:	Data_zawarcia_1: Data	Data_zawarcia_2: Data	ID_uzytkownika	Plec
Tabela:	Uzytkownicy	Uzytkownicy	Zdjecia	Znajomosci	Znajomosci_1	Uzytkownicy	Uzytkownicy
Podsumowanie:	Grupuj według	Grupuj według	Minimum	Minimum	Minimum	Grupuj według	Grupuj według
Sortuj:							
Pokaż:	☑	☑	☑	☑	☑	☐	☐
Kryteria:							='M'
lub:							

```
SELECT Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Uzytkownicy.Kraj,
Min(Zdjecia.Data_dodania) AS Data_pierwszego_zdjecia,
Min(Znajomosci.Data_znajomosci) AS Data_zawarcia_1,
Min(Znajomosci_1.Data_znajomosci) AS Data_zawarcia_2
FROM ((Znajomosci INNER JOIN Uzytkownicy ON Znajomosci.Znajomy_1 = Uzytkownicy.ID_uzytkownika) INNER JOIN Znajomosci AS Znajomosci_1 ON Uzytkownicy.ID_uzytkownika = Znajomosci_1.Znajomy_2) INNER JOIN Zdjecia ON Uzytkownicy.ID_uzytkownika = Zdjecia.Uzytkownik
GROUP BY Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Uzytkownicy.Kraj, Uzytkownicy.ID_uzytkownika;
```

Przy okazji od razu filtrujemy wyniki zapytania, aby znaleźć tylko mężczyzn, a dla każdego z nich podajemy imię, nazwisko oraz kraj. Tę pomocniczą kwerendę nazwiemy *e_pom* i będziemy jej używać do dalszych operacji.

Teraz wystarczy wyfiltrować tych użytkowników, dla których *Data_pierwszego_zdjecia* jest wcześniejsza zarówno od *Data_zawarcia_1*, jak i *Data_zawarcia_2*:

Pole:	Imie	Nazwisko	Kraj	Data_zawarcia_1	Data_zawarcia_2
Tabela:	e_pom	e_pom	e_pom	e_pom	e_pom
Sortuj:		Rosnąco			
Pokaż:	☑	☑	☑	☑	☑
Kryteria:				> [Data_pierwszego_zdjecia]	> [Data_pierwszego_zdjecia]
lub:					

SELECT "ID_uzytkownika", "Imie", "Nazwisko", "Kraj" **FROM** "e_pom" **WHERE** "Data_zawarcia_1" > "Data_pierwszego_zdjecia"**AND** "Data_zawarcia_2" > "Data_pierwszego_zdjecia" **ORDER BY** "Nazwisko" **ASC**

Na tym w zasadzie moglibyśmy zakończyć omówienie, ale zwróćmy uwagę na pewną subtelność: mogłoby się jednak zdarzyć, że ten sposób nie dałby dobrej odpowiedzi! W podanych plikach wejściowych taka sytuacja wprawdzie nie występuje, ale wyobraźmy sobie, że któryś użytkownik nigdy nie występuje w tabeli *znajomosci* jako *Znajomy_2*, a zawsze — jako *Znajomy_1*. Przy domyślnym sposobie połączenia (wewnętrznym) nie pojawiłby się on w ogóle w kwerendzie *e_pom*.

Żeby zadanie zostało rozwiązane prawidłowo, połączenia w pierwszej kwerendzie musiałyby być połączeniami **zewnętrznymi**, przy których każdy użytkownik pojawia się jako rekord w kwerendzie, być może z pustą wartością w niektórych kolumnach (w tym wypadku na przykład *Data_zawarcia_2*).

Said	Koudri	Algieria	2012-07-05	2013-05-03	2014-06-09
Samuel	Ramirez	Stany Zjednoc	2012-06-25	2012-09-13	2012-08-08
Sandor	Takacs	Wegry	2012-09-19	2012-05-31	2012-08-16
Seamus	Breathnach	Irlandia	2013-02-10	2012-12-10	2012-06-13
Sebastian	Quispe	Peru	2012-11-27	2012-12-26	
Serhij	Kovalenko	Ukraina	2012-05-23	2013-01-18	2012-08-07
Theo	Peeters	Belgia	2012-06-15	2012-09-01	2013-05-31

(wynik kwerendy dla nieco innych danych wejściowych, w których Sebastian Quispe nie występuje jako Znajomy_2)

SELECT Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Uzytkownicy.Kraj,
Min(Zdjecia.Data_dodania) **AS** Data_pierwszego_zdjecia,
Min(Znajomosci.Data_znajomosci) **AS** Data_zawarcia_1,
Min(Znajomosci_1.Data_znajomosci) **AS** Data_zawarcia_2
FROM ((Znajomosci **RIGHT JOIN** Uzytkownicy ON Znajomosci.Znajomy_1 = Uzytkownicy.ID_uzytkownika) **LEFT JOIN** Znajomosci AS Znajomosci_1 **ON** Uzytkownicy.ID_uzytkownika = Znajomosci_1.Znajomy_2) **INNER JOIN** Zdjecia **ON** Uzytkownicy.ID_uzytkownika = Zdjecia.Uzytkownik
GROUP BY Uzytkownicy.Imie, Uzytkownicy.Nazwisko, Uzytkownicy.Kraj, Uzytkownicy.ID_uzytkownika, Uzytkownicy.Plec
HAVING (((Uzytkownicy.Plec)='M'));

Trzeba tę możliwość uwzględnić też w drugiej kwerendzie:

Pole:	Imie	Nazwisko	Kraj	Data_zawarcia_1	Data_zawarcia_2
Tabela:	e_pom	e_pom	e_pom	e_pom	e_pom
Sortuj:		Rosnąco			
Pokaż:	☑	☑	☑	☑	☑
Kryteria:				> [Data_pierwszego_zdjecia] Or Is Null	> [Data_pierwszego_zdjecia] Or Is Null
lub:					

```
SELECT e_pom.Imie, e_pom.Nazwisko, e_pom.Kraj, e_pom.Data_zawarcia_1,
e_pom.Data_zawarcia_2
FROM e_pom
WHERE
(e_pom.Data_zawarcia_1>[Data_pierwszego_zdjecia] OR e_pom.Data_zawarcia_1 IS
NULL) AND
(e_pom.Data_zawarcia_2>[Data_pierwszego_zdjecia]  OR e_pom.Data_zawarcia_2 IS
NULL)
ORDER BY e_pom.Nazwisko;
```

Zadanie 103.

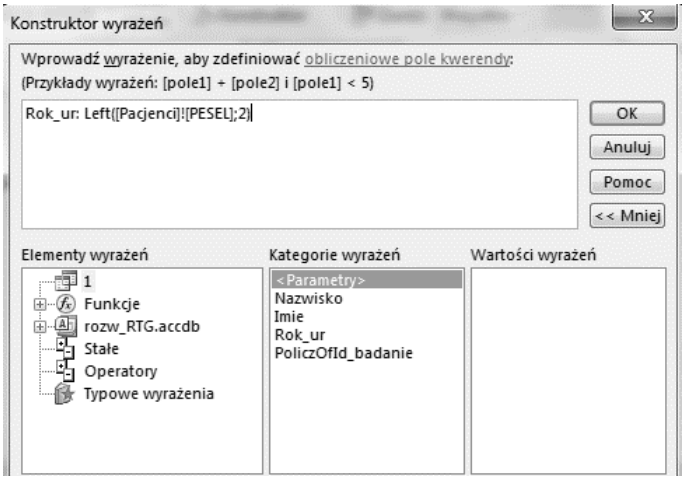
Po wczytaniu danych do tabel oraz ustaleniu związków między tabelami (Relacje) możemy przystąpić do rozwiązania zadania.

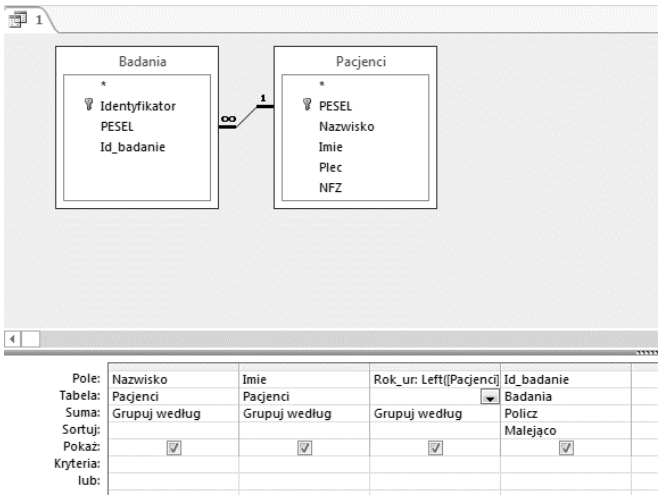
103.1.

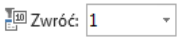
W zadaniu tym należy przygotować odpowiedź na pytanie, jakie jest nazwisko, imię oraz rok urodzenia pacjenta, któremu wykonano najwięcej zdjęć RTG. Należy podać też liczbę zdjęć tego pacjenta. Aby przygotować stosowną kwerendę, potrzebujemy pól `Nazwisko` i `Imie` z tabeli `pacjenci` (grupujemy według tych pól) `oraz` pola `Id_badanie` z tabeli `badania` (stosujemy funkcję `Policz`).

Niezbędne w kwerendzie jest także pole `Rok_ur`, niestety nie ma takiego pola w żadnej z tabel. Możemy jednak to pole stworzyć, a jego zawartość wyliczyć, korzystając z zawartej w treści zadania wskazówki, że dwie pierwsze cyfry numeru PESEL to rok urodzenia.

W tym celu konstruujemy zawartość pola, korzystając z funkcji `Left()`. Argumentem funkcji jest PESEL, z którego pobieramy 2 pierwsze znaki.

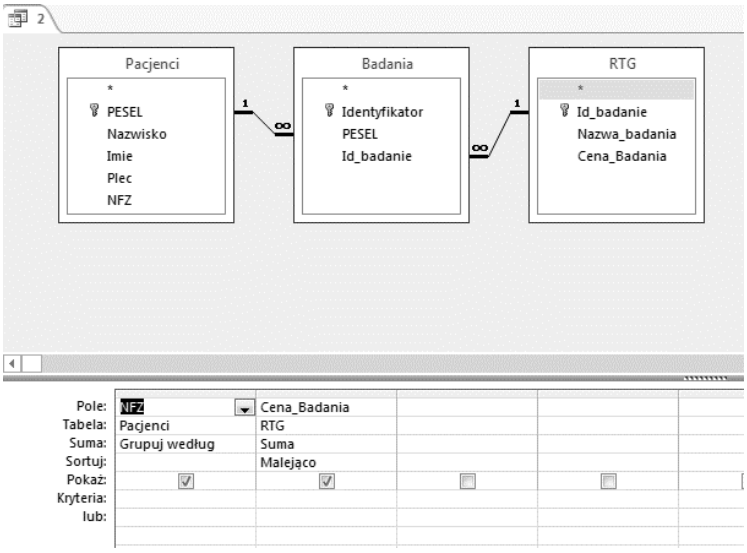




Musimy także pamiętać, aby pole `Id_badanie` posortować malejąco. Wówczas na górze tabeli znajdzie się pacjent, któremu wykonano najwięcej badań. Jako wynik możemy zwrócić jeden rekord, zaznaczając w konfiguracji kwerendy (w widoku projektu kwerendy, karta Projektowanie): 

103.2

W zadaniu tym tworzymy zestawienie, zawierające informacje o sumie kosztów zdjęć RTG poniesionych przez poszczególne oddziały NFZ. Przygotowujemy kwerendę podsumowującą i wybieramy do niej pola `NFZ` z tabeli `pacjenci` (grupujemy według tego pola) oraz pole `Cena_Badania` z tabeli `RTG` (wybierając dla tego pola funkcję `Suma`).



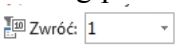
Zestawienie sortujemy malejąco ze względu na sumę kosztów.

103.3.

Zadanie to podzielono na 2 części.

Podpunkt a)

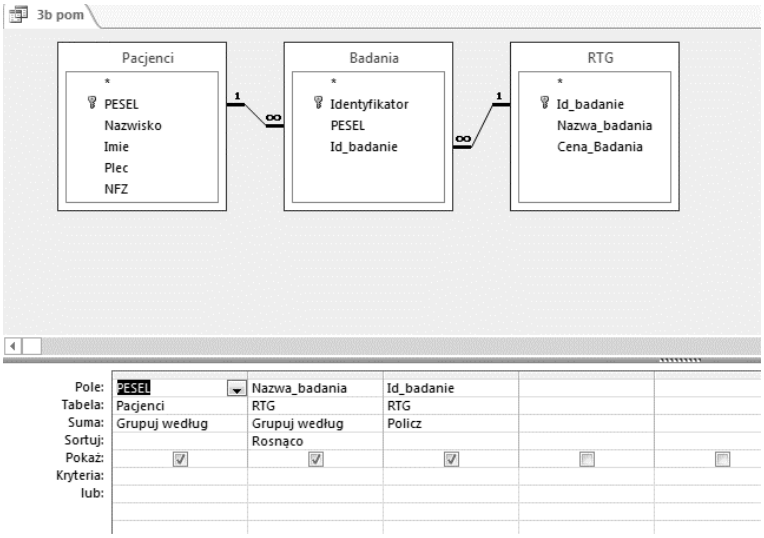
W tej części należy podać nazwę zdjęcia RTG, które wykonano najwięcej razy, i odpowiedzieć też, ile razy wykonano dane badanie. Przygotowujemy kwerendę podsumowującą, wybierając do niej pola `Nazwa_Badania` z tabeli `Badania` (grupujemy według

tego pola) oraz pole `Id_badanie` z tabeli `Badania`, dla którego wybieramy funkcję `Policz`. Aby kwerenda zwracała odpowiedni wynik (tj. nazwę zdjęcia RTG, które wykonano najwięcej razy), trzeba wyniki posortować malejąco według pola `Cena_Badania` i zwrócić jeden rekord, zaznaczając w konfiguracji kwerendy: 

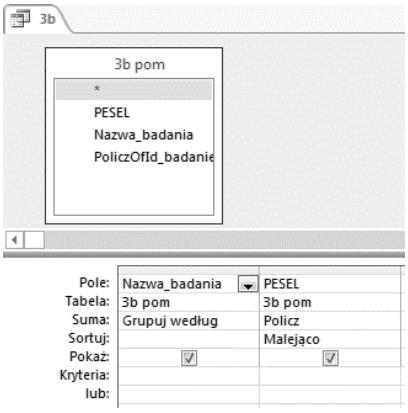
Podpunkt b)

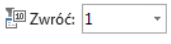
W tej części trzeba podać nazwę zdjęcia RTG, które wykonano u największej liczby pacjentów, oraz liczbę pacjentów, u których wykonano dane badanie. Zadanie rozwiązujemy dwuetapowo.

Najpierw przygotowujemy kwerendę określającą, które badanie wykonywano najczęściej. To kwerenda podsumowująca. Wybieramy do niej pola: `PESEL` z tabeli `Pacjenci`, `Nazwa_Badania` z tabeli `RTG` (grupujemy według tych pól) oraz `Id_Badanie` z tabeli `RTG` (wartości w tym polu zliczamy, stosując funkcję `Policz`).



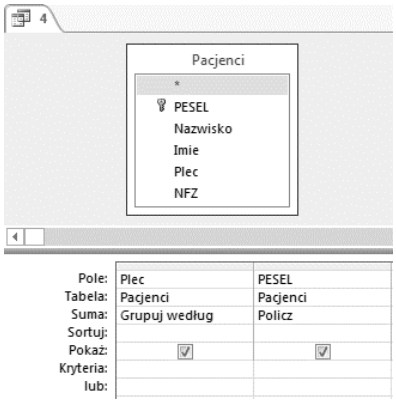
Na bazie tak przygotowanej kwerendy możemy już ustalić, które badanie wykonywano najczęściej. Wystarczy teraz przygotować kwerendę podsumowującą na bazie kwerendy pomocniczej. Wybieramy z niej pola `Nazwa_badania` oraz `PESEL`, grupując według pierwszego pola i zliczając wystąpienia w drugim polu.



Pamiętamy o sortowaniu malejąco pola zliczającego PESEL-e i wyświetleniu 1 rekordu, zaznaczając w konfiguracji kwerendy: 

103.4.

W zadaniu mamy podać liczbę kobiet oraz liczbę mężczyzn, którym wykonano zdjęcia RTG. Wystarczy przygotować kwerendę podsumowującą opartą na tabeli `Pacjenci`.



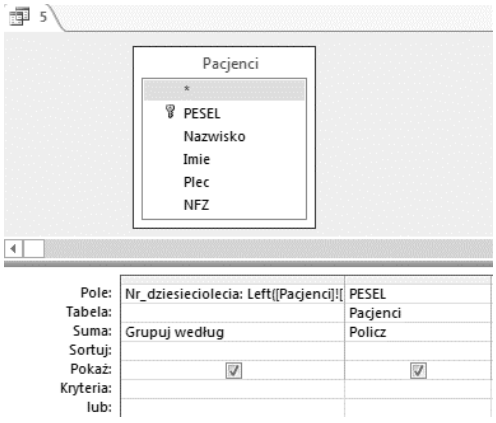
Z tabeli wybieramy pola `Plec` (grupujemy według tego pola) i zliczamy numery PESEL wybierając funkcję `Policz` dla pola `PESEL`.

103.5.

W zadaniu tym należy utworzyć zestawienie zawierające numer dziesięciolecia oraz liczbę pacjentów urodzonych w tym dziesięcioleciu dla okresu od roku 1900 do 1999. Pamiętając, że pierwsze dziesięciolecie to okres 1900–1909, a drugie dziesięciolecie to okres 1910–1919 wystarczy wziąć pod uwagę pierwszą liczbę z numeru PESEL pacjenta. Tworzymy pole Nr_dziesięciolecia i korzystamy z funkcji

```
Nr_dziesieciolecia: Left([Pacjenci]![PESEL];1)+1
```

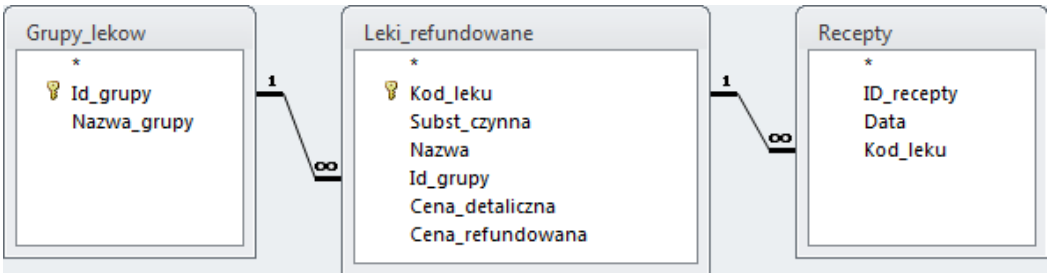
W drugim poluzliczamy dla danego dziesięciolecia numery PESEL pacjentów.



Zadanie 104.

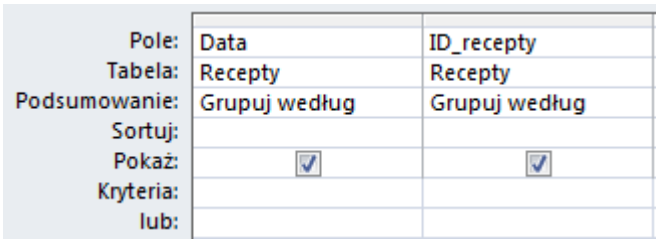
W tabeli `Leki_refundowane` zapisano zestawienie leków, które są częściowo refundowane przez fundusz zdrowia; nie wszystkie leki z tego zestawienia były wypisywane przez lekarzy przychodni w kontrolowanym okresie. Kluczem głównym tej tabeli jest pole `Kod_leku`. Leki należą do różnych grup, grupy leków zostały wymienione w tabeli `Grupy_lekow`, kluczem głównym tabeli jest `Id_grupy`. Jedna recepta może zawierać maksymalnie pięć leków. Jeśli na jednej recepcie wypisano kilka leków, to identyfikatory recept w tabeli `Recepty` powtarzają się, dlatego pole `Id_recepty` nie może być kluczem w tej tabeli.

Tabele połączone są relacjami jeden do wielu:



104.1.

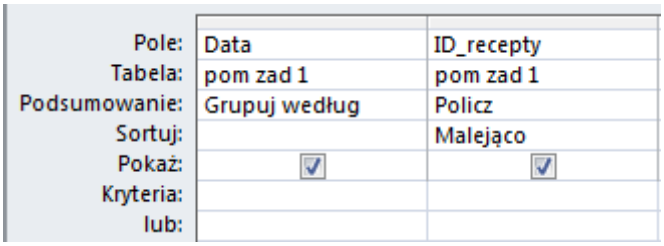
W pierwszym zadaniu należy dla każdego dnia policzyć liczbę recept o niepowtarzających się identyfikatorach. Aby wyeliminować powtarzające się identyfikatory, można skonstruować kwerendę pomocniczą:



Pole:	Data	ID_recepty
Tabela:	Recepty	Recepty
Podsumowanie:	Grupuj według	Grupuj według
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

```
SELECT Recepty.Data, Recepty.ID_recepty
FROM Recepty
GROUP BY Recepty.Data, Recepty.ID_recepty;
```

w której usuniemy je poprzez grupowanie. Dalej, korzystając z tej kwerendy, policzymy dla każdego dnia liczbę niepowtarzających się identyfikatorów:



Pole:	Data	ID_recepty
Tabela:	pom zad 1	pom zad 1
Podsumowanie:	Grupuj według	Policz
Sortuj:		Malejąco
Pokaż:	☑	☑
Kryteria:		
lub:		

```
SELECT pomzad 1.Data, Count(pomzad 1.ID_recepty) AS PoliczOfID_recepty
FROM pomzad 1
GROUP BY pomzad 1.Data
ORDER BY Count(pomzad 1.ID_recepty) DESC;
```

Inny sposób postępowania to skorzystanie z klauzuli SELECT DISTINCT (wybierz niepowtarzające się identyfikatory recept).

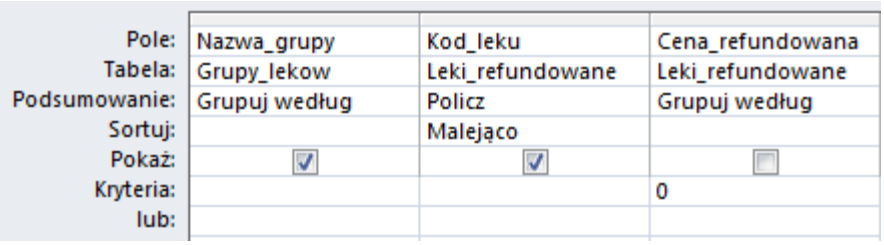
Zapytanie w języku SQL:

```
SELECT r.Data, count(*)
FROM (SELECT  DISTINCT recepty.ID, recepty.Data FROM recepty) AS r
GROUP BY r.Data
ORDER BY count(*) DESC
LIMIT 1
```

104.2.

Konstruując odpowiedź do zadania, wykorzystamy dane z dwóch tabel: `Grupy_leków` i `Leki_refundowane`. Dla każdej grupy leków policzymy wszystkie leki z tej grupy przy

założeniu, że cena refundowana wynosi 0 zł. Ponadto należy wybrać grupę, w której znajduje się najwięcej leków refundowanych w 100%, dlatego zliczone kody leków należy posortować malejąco. Pierwszy wiersz w zestawieniu stanowi odpowiedź do zadania.



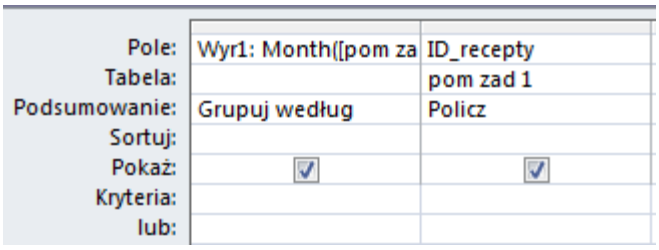
Pole:	Nazwa_grupy	Kod_leku	Cena_refundowana
Tabela:	Grupy_lekow	Leki_refundowane	Leki_refundowane
Podsumowanie:	Grupuj według	Policz	Grupuj według
Sortuj:		Malejąco	
Pokaż:	☑	☑	☐
Kryteria:			0
lub:			

Zapytanie w języku SQL:

SELECT grupy_lekow.nazwa_grupy
FROM leki_refundowane
JOIN grupy_lekow **ON** grupy_lekowId_grupy = leki_refundowane.Id_grupy
WHERE leki_refundowane.Cena_refundowana = 0
GROUP BY grupy_lekow.Nazwa_grupy
ORDER BY COUNT(*)
LIMIT 1

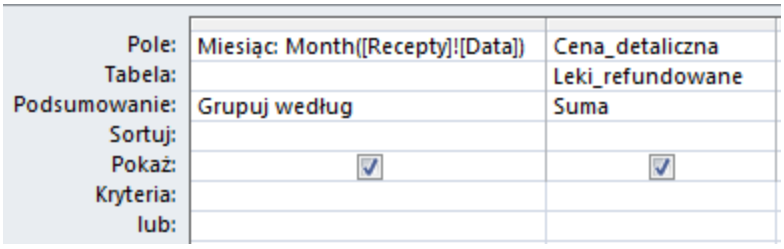
104.3.

Aby podać dla każdego miesiąca liczbę wydanych recept, można skorzystać z kwerendy pomocniczej do zadania pierwszego (która usuwa powtarzające się identyfikatory recept). Za pomocą funkcji `MONTH` wyodrębniamy z daty miesiąc. Następnie dla każdego miesiąca zliczamy liczbę niepowtarzających się identyfikatorów recept.



Pole:	Wyr1: Month([pom za	ID_recepty
Tabela:		pom zad 1
Podsumowanie:	Grupuj według	Policz
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

W drugiej kwerendzie obliczymy sumaryczną wartość wszystkich leków z recept wypisanych w kolejnych miesiącach. Dokonujemy grupowania według miesięcy, zaś cenę detaliczną wypisanych leków sumujemy.

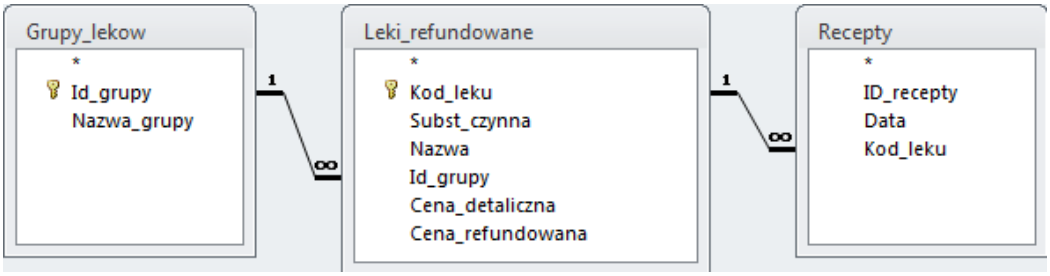


Pole:	Miesiąc: Month([Recepty]![Data])	Cena_detaliczna
Tabela:		Leki_refundowane
Podsumowanie:	Grupuj według	Suma
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

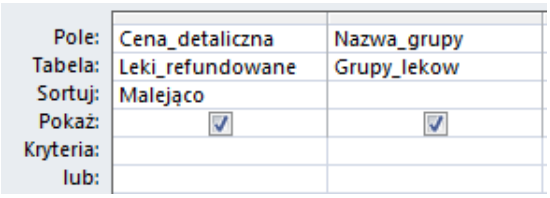
Zapytanie w języku SQL:
SELECT w.mc, **COUNT**(*), **SUM**(w.sumw)
FROM (SELECT recepty.ID_recepty, MONTH(recepty.Data) AS mc,
SUM(leki_refundowana.Cena_detaliczna) **AS** sumw **FROM** recepty join leki_refundowane
ON leki_refundowane.Kod_leku = recepty.Kod_leku **GROUP BY** recepta.id_recepty,
MONTH(recepty.Data)) **AS** w
GROUP BY w.mc

104.4.

Aby podać cenę detaliczną najdroższego leku, na który wypisano receptę, należy wziąć pod uwagę wszystkie trzy tabele połączone ze sobą relacjami.



Połączenie z tabelą Recepty zapewnia nam wybranie leku wypisanego (nie dowolnego). Z tabeli `Leki_refundowane` wybieramy cenę detaliczną (największą), a z tabeli `Grupy_leków` wybieramy nazwę grupy, do jakiej należy najdroższy lek.

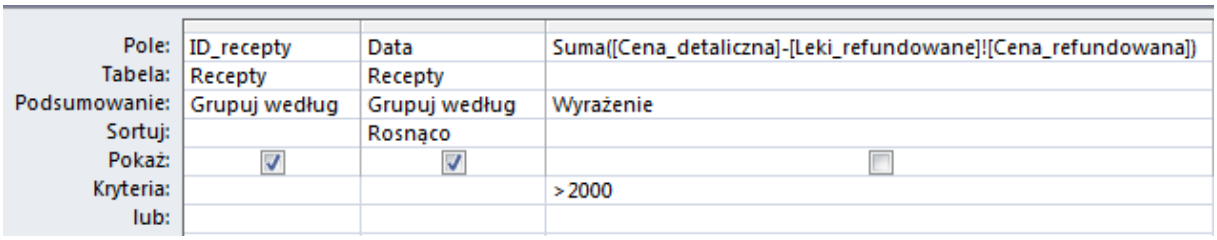


Zapytanie w języku SQL:

```
SELECT leki_refundowane.Cena_detaliczna, grupy_lekow.Nazwa_grupy
FROM leki_refundowane
JOIN grupy_lekow ON grupy_lekow.Id_grupy = leki_refundowane.Id_grupy
JOIN recepty ON recepty.Kod_leku = leki_refundowane.Kod_leku
ORDER BY leki_refundowane.Cena_detaliczna DESC
LIMIT 1
```

104.5.

Dla każdej recepty (rozróżniane są one po identyfikatorach) wybieramy datę jej wystawienia i obliczamy sumę z wyrażenia `Cena_detaliczna–Cena_refundowana`. Następnie zakładamy warunek, który wybiera sumy wyższe niż 2000. Na koniec zestawienie należy posortować rosnąco według dat.



Zapytanie w języku SQL:

```
SELECT recepty.Id_recepty, recepty.Data, SUM(leki_refundowane.Cena_detaliczna –
leki_refundowane.Cena_refundowana)
FROM recepty
JOIN leki_refundowane ON leki_refundowane.Kod_leku = recepty.Kod_leku
GROUP BY recepty.Id_recepty, recepty.Data
HAVING SUM (leki_refundowane.Cena_detaliczna –
leki_refundowane.Cena_refundowana) > 2000
ORDER BY recepty.Data
```

Zadanie 105.

Zadanie rozwiążemy z pomocą MS Access. Zawartości plików `osoby.txt`, `zamowienia.txt` i `rosliny.txt` wczytamy do tabel `Osoby`, `Zamowienia` i `Rosliny`, które połączymy odpowiednimi relacjami (szczegóły pozostawiamy czytelnikowi).

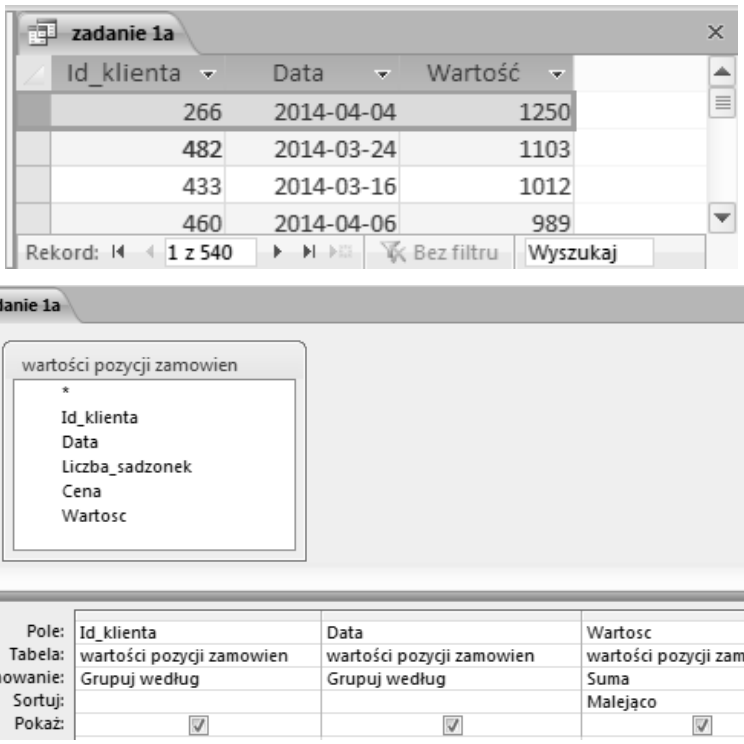
105.1.

Do rozwiązania zadania wykorzystamy dane ze wszystkich trzech tabel:

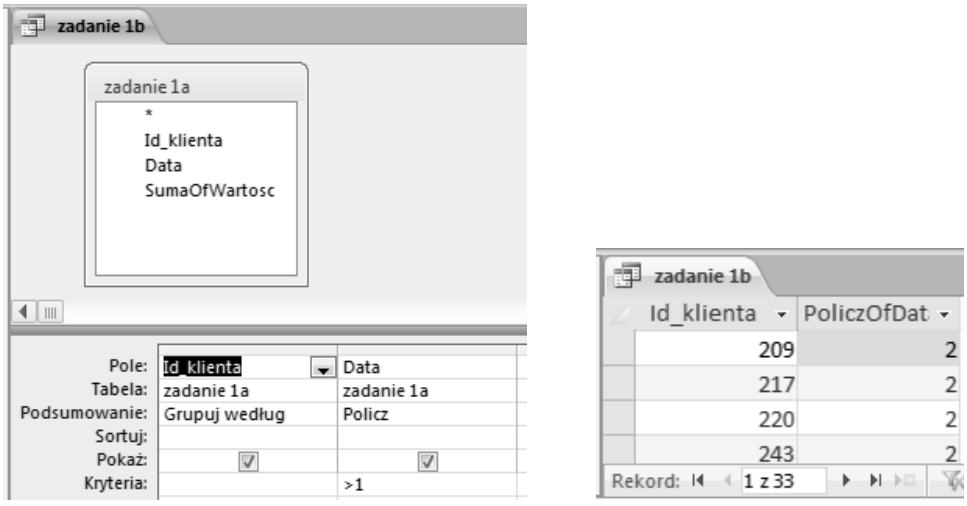
- `Id_klienta` z tabeli `Osoby`, niezbędne dla zliczenia, ile faktur otrzymali poszczególni klienci,
- datę zamówienia z tabeli `Zamowienia`, aby pogrupować zamówienia według dat i uzyskać informację o liczbie wszystkich wystawionych faktur,
- liczbę sadzonek z tabeli `Zamowienia` i cenę sadzonki z tabeli `Rosliny`, aby obliczyć wartość poszczególnych zamówień.

Zadanie rozwiążemy w trzech krokach. W pierwszym kroku utworzymy pomocniczą kwerendę o nazwie `wartości pozycji zamowien`, przy pomocy której obliczymy wartości poszczególnych zamówień. W kwerendzie tej pogrupujemy wiersze według identyfikatorów klientów, dat, liczb sadzonek, cen oraz dodamy nową kolumnę z danymi dotyczącymi wartości każdej pozycji zamówienia. Wartość pozycji zamówienia obliczymy jako iloczyn liczby zamówionych sadzonek i ich ceny.

W kroku drugim utworzymy kwerendę `zadanie 1a`, w której zsumujemy wartości pozycji zamówienia dla każdego klienta i każdego dnia. W ten sposób dowiemy się, ile wystawiono faktur i jakie były ich wartości. Jako źródło danych wskażemy kwerendę pomocniczą `wartości pozycji zamowien`. Wyniki posortujemy malejąco według kolumny `Wartosc`, aby łatwo można było odczytać, jaka jest największa wartość faktury. Liczba wierszy w tabeli z wynikami kwerendy `zadanie 1a` stanowi odpowiedź na pytanie o liczbę wystawionych faktur.



W kroku trzecim skorzystamy z wyników kwerendy `zadanie 1a`. Utworzymy kwerendę `zadanie 1b`, przy pomocy której zliczymy, ile faktur otrzymał każdy z klientów, i wybierzemy tych, którzy otrzymali więcej niż jedną fakturę. Liczba wierszy zestawienia utworzonego przez kwerendę `zadanie 1b` jest odpowiedzią na pytanie, ilu klientów otrzymało więcej niż jedną fakturę (faktura dla każdego klienta wystawiana jest na zakończenie dnia za wszystkie dokonane przez niego zamówienia, zatem różne daty oznaczają różne faktury). Na poniższym rysunku widzimy, że na dole zestawienia wyświetlana jest informacja o liczbie wierszy i numerze wiersza bieżącego: „Rekord: 1 z 33”.



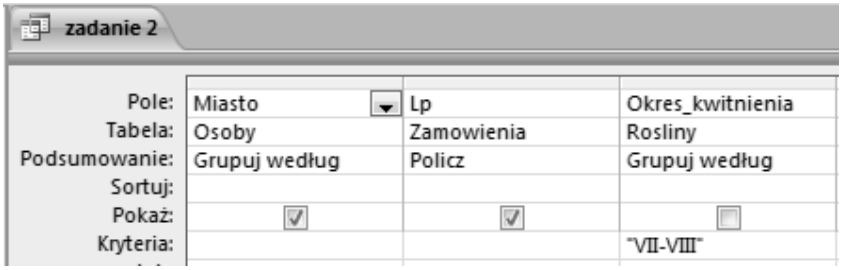
105.2.

W rozwiązaniu wykorzystamy dane:

- nazwę miasta z tabeli `Osoby`,
- liczbę porządkową z tabeli `Zamowienia`, aby zliczyć liczbę pozycji,

- okres kwitnienia z tabeli `Rosliny`, aby wskazać rośliny, których dotyczą zamówienia.

Utworzymy kwerendę, w której pogrupujemy wiersze według nazwy miasta, zliczymy liczbę pozycji i wybierzemy rośliny kwitnące tylko w okresie VII-VIII.

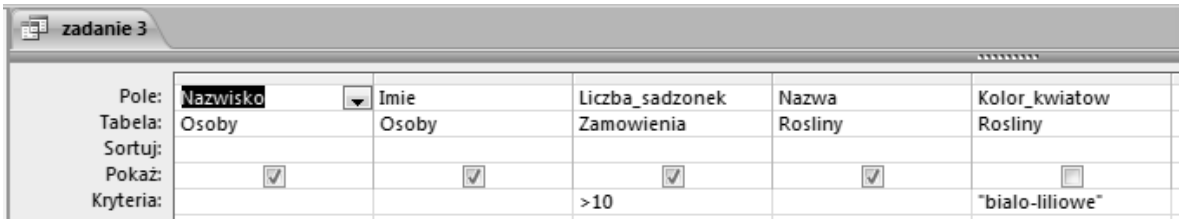


105.3.

Do utworzenia wymaganego w zadaniu zestawienia potrzebujemy następujących danych:

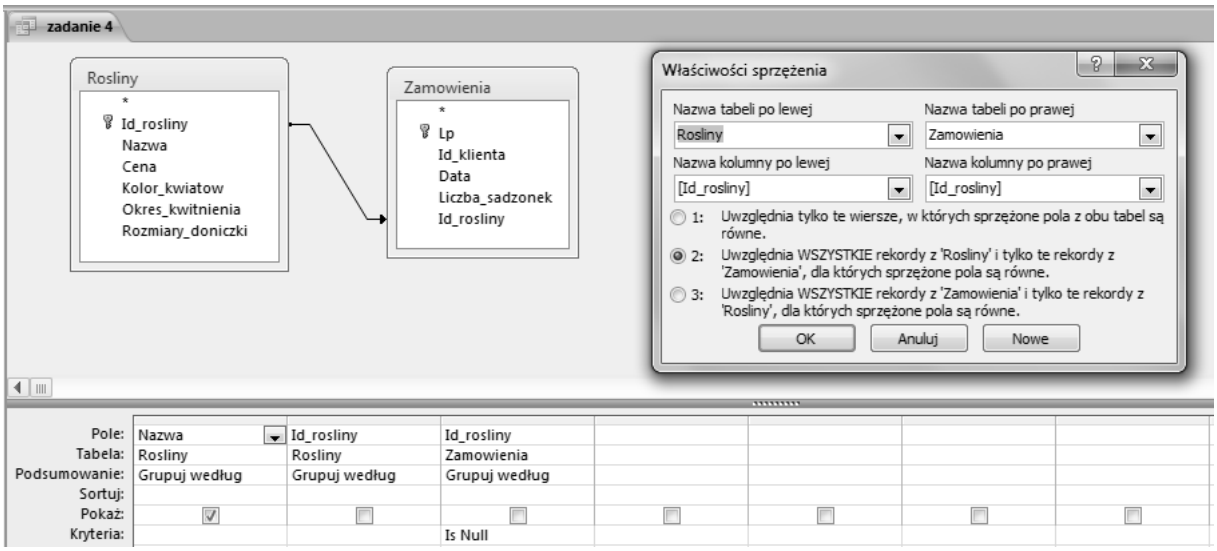
- nazwiska i imienia z tabeli `Osoby`,
- liczby sadzonek z tabeli `Zamowienia`,
- nazwy rośliny oraz koloru kwiatów z tabeli `Rosliny`.

Utworzymy kwerendę według poniższego projektu, zaznaczając w kryteriach, że interesuje nas liczba sadzonek większa niż 10 i kolor kwiatów `bialo-liliowe`.



105.4.

Aby odpowiedzieć na pytanie postawione w zadaniu, należy ustalić odpowiedni rodzaj złączenia dla tabel `Rosliny` i `Zamowienia` (patrz poniższy rysunek). Szukamy roślin, które nie zostały uwzględnione w żadnym zamówieniu, czyli takich wierszy, dla których identyfikator rośliny w tabeli `Rosliny` nie ma odpowiednika w tabeli `Zamowienia`. Zmieniamy własności sprzężenia tabel tak, jak pokazano na rysunku, i w kwerendzie wskazujemy, że interesują nas wiersze tabeli `Rosliny`, dla których w tabeli `Zamowienia` wartości identyfikatora są nieokreślone (Null).



105.5.

W rozwiązaniu zadania pogrupujemy wiersze tabeli `Rosliny` (złączonej przez pole `Id_rosliny` z tabelą `Zamowienia`) według rozmiaru doniczek i w tabeli `Zamowienia` zliczymy pozycje odpowiadające zamówieniom roślin w poszczególnych rozmiarach doniczek.

Zadanie 106.

Importujemy dane z plików tekstowych do tabel.

- z pliku `obserwacje.txt` tworzymy tabelę `Obserwacje`, w której pola `poczatek` i `koniec` są typu Data/Godzina, pola `ID_gatunku`, `ID_lokalizacji` i `liczebnosc` są liczbami całkowitymi, a pole `zachowanie` jest tekstowe,
- z pliku `gatunki.txt` tworzymy tabelę `Gatunki`, w której pole `ID_gatunku` jest liczbą całkowitą, a pozostałe pola są tekstowe.
- z pliku `lokalizacje.txt` tworzymy tabelę `Lokalizacje`, w której pole `ID_lokalizacji` jest liczbą całkowitą, a pozostałe pola są tekstowe.

Jako pola kluczowe w tabelach wybieramy:

- w tabeli `Gatunki`: pole `ID_gatunku`,
- w tabeli `Lokalizacje`: pole `ID_lokalizacji`.
- W tabeli `Obserwacje` zgadzamy się na utworzenie dodatkowego pola `Identyfikator`, które będzie jednocześnie przechowywać numer rekordu w tabeli.

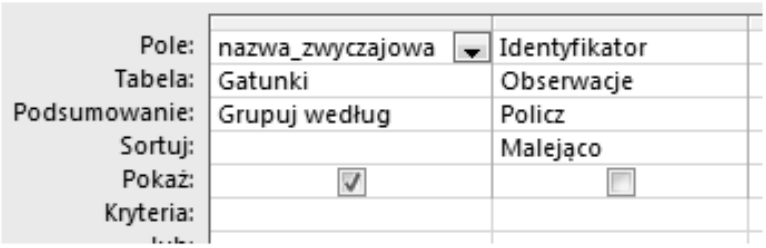
W programie Microsoft Access możemy od razu ustawić i zapisać relacje tabel:

- pole ID_lokalizacji połączy tabele: Lokalizacje i Obserwcje;

- pole ID_gatunku połączy tabele: Gatunki i Obserwcje.

106.1

Należy podać nazwy zwyczajowe trzech gatunków najczęściej obserwowanych ptaków.



Pole:	nazwa_zwyczajowa	Identyfikator
Tabela:	Gatunki	Obserwacje
Podsumowanie:	Grupuj według	Policz
Sortuj:		Malejąco
Pokaż:	☑	☐
Kryteria:		

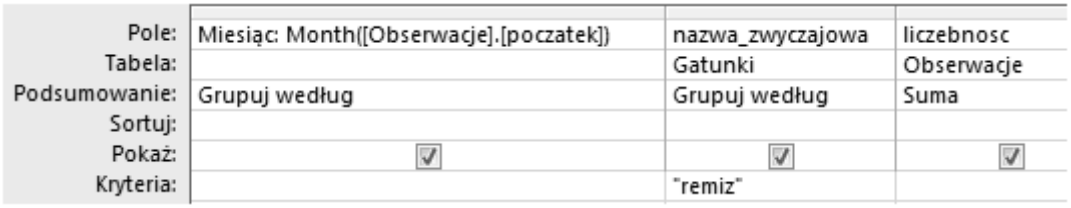
Dla każdego gatunku będziemy zliczać identyfikatory obserwacji, które dotyczyły tego gatunku. Niech kwerenda wybierze nam pola: `Identyfikator` z tabeli `Obserwacje` i `nazwa_zwyczajowa` z tabeli: `Gatunki`. Wybrane rekordy zgrupujemy według nazwy zwyczajowej, a do pola `Identyfikator` zastosujemy funkcję `Policz`.

Posortujemy wyniki malejąco według licznika identyfikatorów, aby najczęściej obserwowane gatunki pojawiły się na początku zestawienia.

106.2.

Dla każdego miesiąca należy podać łączną liczbę wszystkich zaobserwowanych osobników remiza. Do wyłuskania numeru miesiąca z pełnej daty, z pola `poczatek` lub `koniec` z tabeli `Obserwacje`, będzie nam potrzebna funkcja `MONTH`.

Wykorzystamy tabele Obserwacje i Gatunki. W kwerendzie umieścimy pola: `nazwa_zwyczajowa` i `liczebnosc`. Dodamy pole obliczane `Miesiąc`. Dla nazwy zwyczajowej ustawimy kryterium „remiz”. Wyniki zgrupujemy według pola `Miesiąc`, a do pola `liczebnosc` zastosujemy funkcję `Suma`.



Pole:	Miesiąc: Month([Obserwacje].[poczatek])	nazwa_zwyczajowa	liczebnosc
Tabela:		Gatunki	Obserwacje
Podsumowanie:	Grupuj według	Grupuj według	Suma
Sortuj:			
Pokaż:	☑	☑	☑
Kryteria:		"remiz"	

106.3.

Należy wybrać gatunki krukowate (czyli takie, których łacińska nazwa zawiera słowo „*Corvus*”), zaobserwowane na terenach miejskich. Dla każdego z tych gatunków należy podać nazwę zwyczajową i liczbę przeprowadzonych obserwacji.

Utworzymy kwerendę wybierającą pola `nazwa_zwyczajowa` i `nazwa_lacinska` z tabeli `Gatunki` oraz `lokalizacja` i `opis` z tabeli `Obserwacje`.

Dla pola `nazwa_lacinska` ustawimy kryterium: zawiera ”Corvus”, a dla pola `opis` — zawiera ”miasto”.

Pole:	nazwa_zwyczajowa	nazwa_lacinska	lokalizacja	opis
Tabela:	Gatunki	Gatunki	Lokalizacje	Lokalizacje
Sortuj:				
Pokaż:	☑	☑	☑	☑
Kryteria:		Like "*Corvus*"		Like "*miasto*"

Na wybranych rekordach wykonamy operację grupowania według nazwy zwyczajowej, a do pola `lokalizacja` zastosujemy funkcję `Policz`.

Pole:	nazwa_zwyczajowa	lokalizacja
Tabela:	3a	3a
Podsumowanie:	Grupuj według	Policz
Sortuj:		
Pokaż:	☑	☑

106.4.

Dla każdej grupy obserwacji należy wyznaczyć czas jej trwania w minutach oraz łączną liczebność zaobserwowanych osobników, a następnie podać:

a) lokalizację dla grupy obserwacji trwającej najdłużej, datę, czas trwania obserwacji w minutach i łączną liczebność zaobserwowanych osobników;

b) datę, lokalizację i sprawność grupy obserwacji o największej sprawności, gdzie sprawność jest stosunkiem liczebności obserwacji w grupie do czasu trwania obserwacji w minutach.

Do obliczenia czasu trwania każdej obserwacji wykorzystamy funkcję

```
DATEDIFF(interwał; początek; koniec)
```

Interwał w minutach w programie Microsoft Access oznacza się przez „n”.

a) Niech kwerenda wybiera pola: `lokalizacja`, `poczatek` i `liczebność`. Dodamy pole obliczane, które wyznaczy czas obserwacji jako różnicę czasów: początku i końca obserwacji.

   Zgrupujemy lokalizacje, daty i obliczony czas, zaś do pola liczebność zastosujemy funkcję Suma. Zestawienie posortujemy malejąco według czasu trwania obserwacji, aby rekord z najdłuższym czasem pojawił się jako pierwszy.

Pole:	lokalizacja	data: poczatek	czas: DateDiff("n";[Obserwacje]![poczatek];[Obserwacje]![koniec])	osobniki: liczebnosc
Tabela:	Lokalizacje	Obserwacje		Obserwacje
Podsumowanie:	Grupuj według	Grupuj według	Grupuj według	Suma
Sortuj:			Malejąco	
Pokaż:	☑	☑	☑	☑

b) Wykorzystamy wyniki poprzedniej kwerendy. Wybierzemy z niej pola: lokalizacja i data. Dodamy nowe pole obliczane `sprawnosc`, które zwróci sprawność obserwacji jako iloraz: liczba obserwowanych osobników podzielona przez czas obserwacji. Posortujemy zestawienie malejąco według pola `sprawnosc`.

Pole:	lokalizacja	data	sprawność: [osobniki]/[czas]
Tabela:	4a	4a	
Sortuj:			Malejąco
Pokaż:	☑	☑	☑

W ustawieniach właściwości pól kwerendy wybierzemy odpowiedni format:

- pole `data` sformatujemy jako *datę krótką*, aby pokazywało jedynie datę obserwacji, bez godziny i minuty;
- dla pola `sprawność` ustawimy format stałoprzecinkowy z liczbą cyfr dziesiętnych 3.

106.5.

Należy podać liczbę wszystkich zaobserwowanych osobników żurawia, a następnie wykonać zestawienie, w którym dla poszczególnych powiatów podamy liczbę osobników żurawia zaobserwowanych na ich terenie, z podziałem na różne zachowania obserwowanego ptaka.

a) Aby zliczyć wszystkie zaobserwowane osobniki żurawia, wybierzemy te obserwacje, w których występuje nazwa zwyczajowa ”zuraw”, a następnie zsumujemy wartości w polu `liczebnosc` wybranych obserwacji.

Pole:	nazwa_zwyczajowa	liczebnosc
Tabela:	Gatunki	Obserwacje
Podsumowanie:	Grupuj według	Suma
Sortuj:		
Pokaż:	☑	☑
Kryteria:	"zuraw"	

b) Druga część zadania wymaga ponadto uwzględnienia informacji o nazwie powiatu i zachowaniu obserwowanego ptaka. Rozszerzymy poprzednią kwerendę o dodatkowe pola: `powiat` i `zachowanie`.

Pole:	nazwa_zwyczajowa	powiat	liczebnosc	zachowanie
Tabela:	Gatunki	Lokalizacje	Obserwacje	Obserwacje
Podsumowanie:	Grupuj według	Grupuj według	Suma	Grupuj według
Sortuj:				
Pokaż:	☑	☑	☑	☑
Kryteria:	"zuraw"			

Następnie przedstawimy powyższe dane w formie 2-wymiarowej tabeli.

Wykonamy kwerendę krzyżową, umieszczając nazwę powiatu w **nagłówkach wierszy**, `zachowanie` w **nagłówkach kolumn**, a sumę liczebności jako **wartość** pola danych.

Pole:	powiat	zachowanie	SumaOfliczebnosc
Tabela:	5b_1	5b_1	5b_1
Podsumowanie:	Grupuj według	Grupuj według	Suma
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość

Zamiast tworzenia kwerendy krzyżowej można wyeksportować wyniki do arkusza kalkulacyjnego i tam wstawić tabelę przestawną o takim samym układzie kolumn i wierszy.

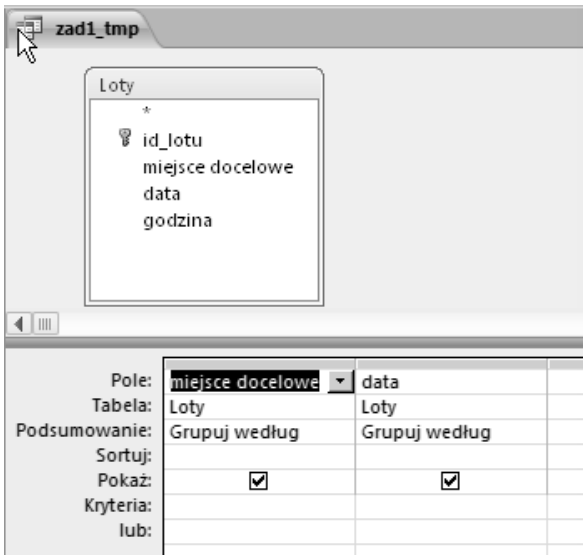
Zadanie 107.

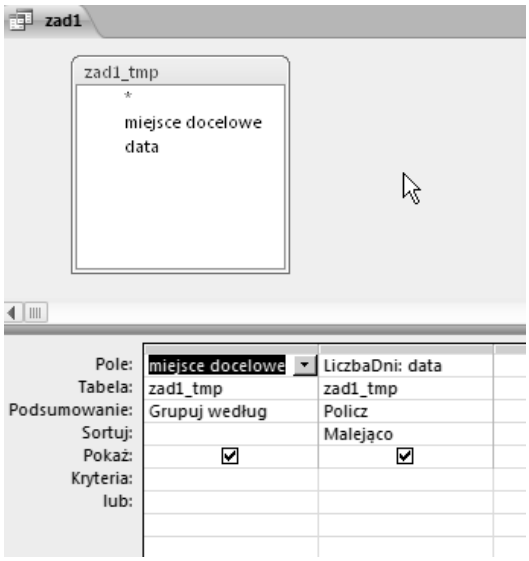
Po wczytaniu danych do tabel oraz ustaleniu związków między tabelami (Relacje) można przystąpić do rozwiązania zadania.

107.1.

W zadaniu należy zauważyć, że interesuje nas liczba dni, a nie liczba wylotów do miejsc docelowych. Zatem jeśli np. w danym dniu odbyły się 3 wyloty przykładowo do Barcelony, to taki dzień powinien być zliczony jeden raz, a nie 3 razy. W języku SQL można skorzystać

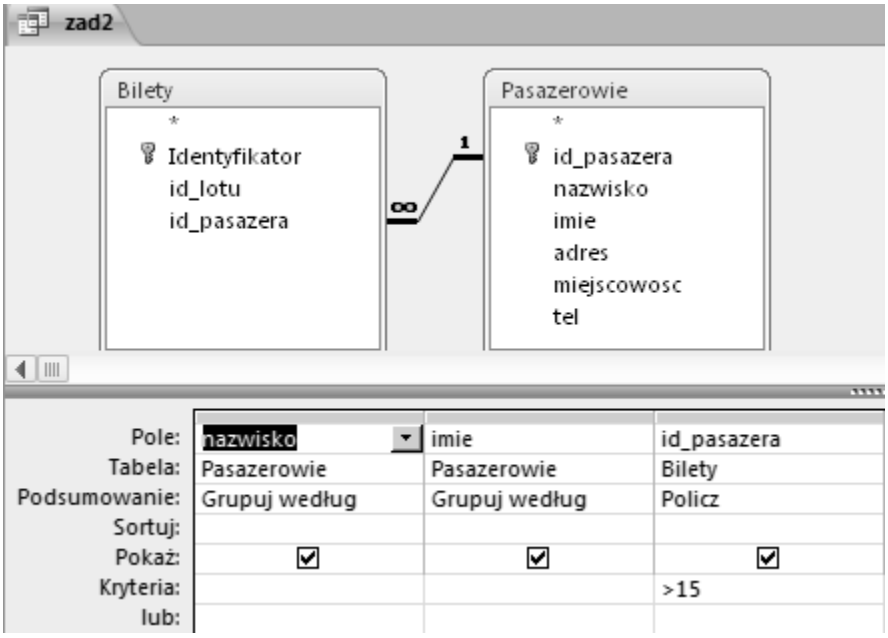
z funkcji `COUNT DISTINCT()`, jednak MS Access nie daje takiej możliwości. Zadanie można rozwiązać, projektując kwerendę pomocniczą, w której wybieramy miejsca docelowe oraz daty, każde z pól grupując. Dzięki temu dla każdego miejsca docelowego mamy ustaloną datę wylotu (bez powtórzeń). Następnie tworzymy właściwą kwerendę, dla której źródłem danych jest kwerenda pomocnicza. Z niej wybieramy miejsca docelowe. Dla każdego miejsca zliczamy liczbę dni. Poprawny wynik otrzymamy, wykonując sortowanie liczby dni i wybierając trzy pierwsze rekordy z uzyskanej listy.





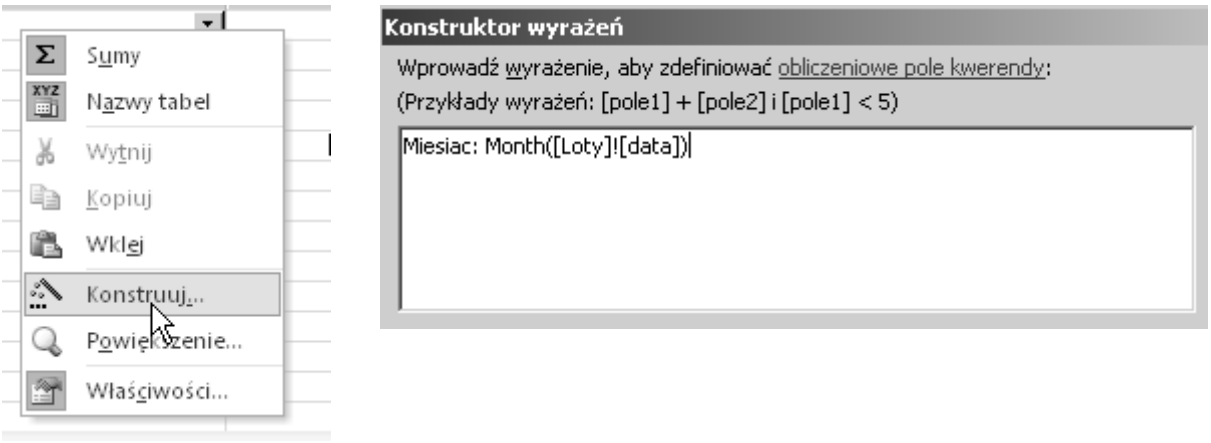
107.2.

W zadaniu zliczamy dla każdego pasażera (imię, nazwisko), ile kupił on biletów. Następnie w kryteriach pola `id_pasazera` dla tabeli `Bilety` ustawiamy, że liczba biletów ma być większa od 15.

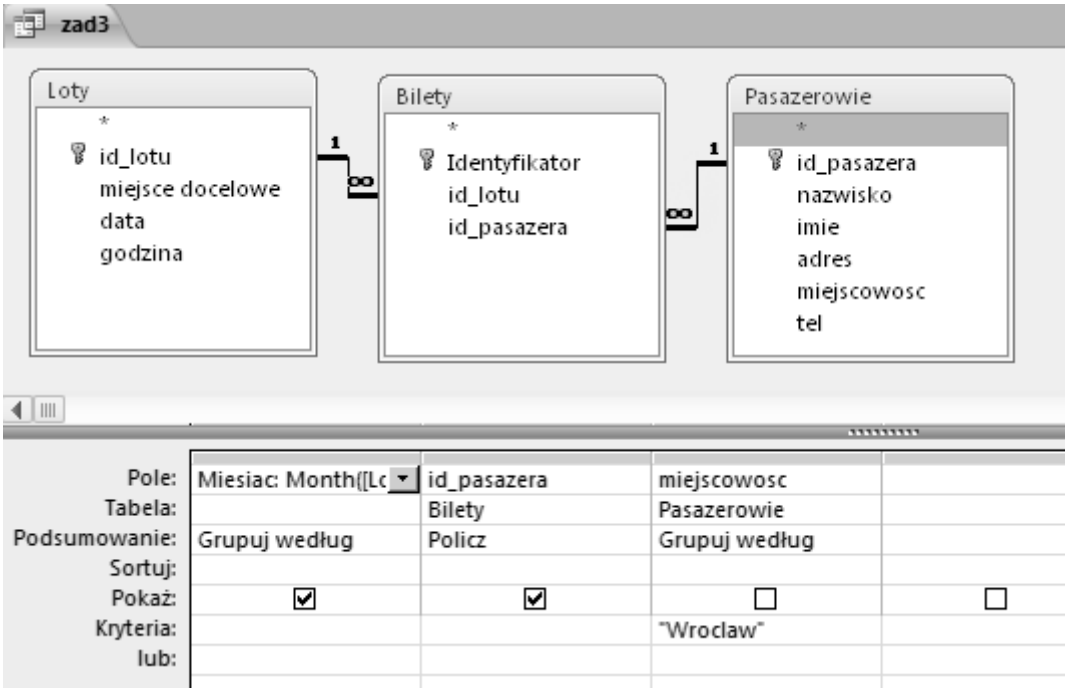


107.3.

Do zbudowania zapytania potrzebujemy wszystkich tabel. Do zestawienia potrzebujemy numeru miesiąca, dlatego skonstruujemy wyrażenie, które z daty wylotu uzyska miesiąc.

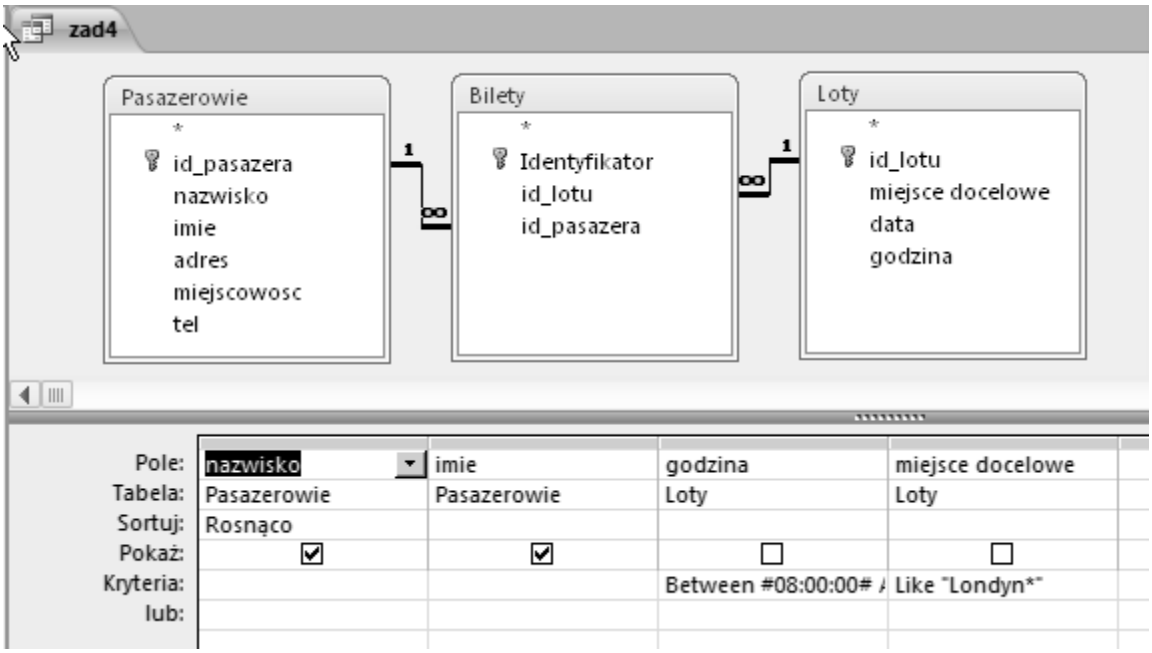


Pole to grupujemy. Następnie z tabeli `Bilety` wybieramy pole `id_pasazera`, w którym stosujemy funkcję `Policz`. Z tabeli `Pasazerowie` wybieramy pole `miejscowosc`, dla którego w kryteriach ustawiamy `Wroclaw`.



107.4.

W tym zadaniu wystarczy zwykła kwerenda wybierająca. Ważne są kryteria: dla pola `godzina` ustawiamy, że jest to godzina między 8 a 10 (`Between 8:00 And 10:00`), a dla pola miejsce docelowe ustawiamy, że jest to dowolne lotnisko w Londynie, (w nazwie występuje słowo `Londyn` i dalej dowolny ciąg znaków — `Like "Londyn*"`).

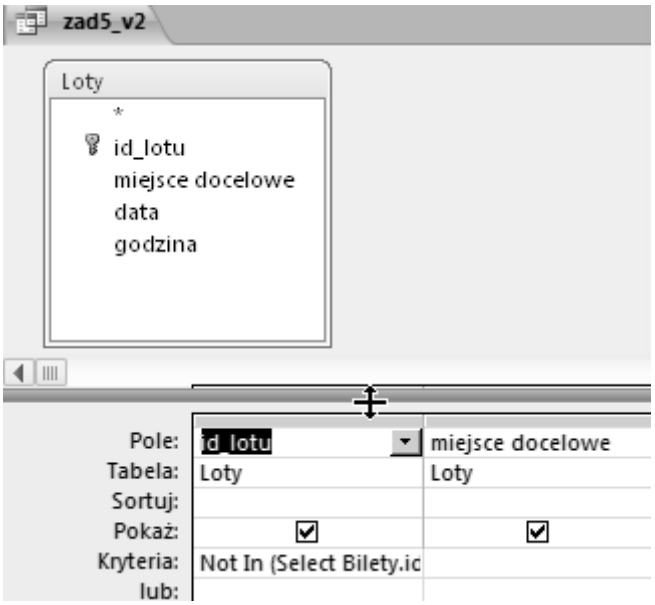


107.5.

Zadanie wymaga odnalezienia numerów lotów, które występują w tabeli `Loty`, ale nie występują w tabeli `Bilety`. Można to zrobić na 2 sposoby. Pierwszym z nich jest napisanie prostego zapytania w SQL:

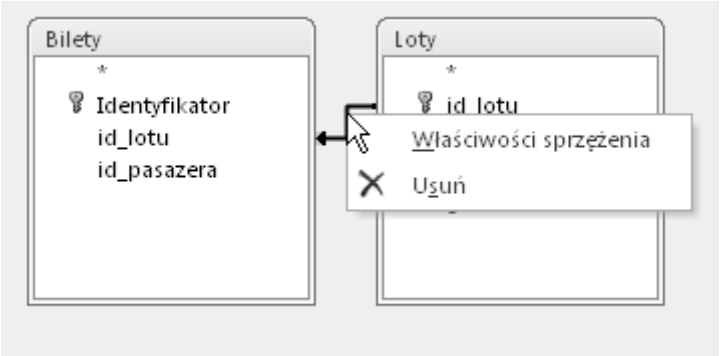
```
SELECT Loty.id_lotu, Loty.[miejsce docelowe]
FROM Loty
WHERE (((Loty.id_lotu)
NOT IN (SELECT Bilety.id_lotu FROM Bilety)));
```

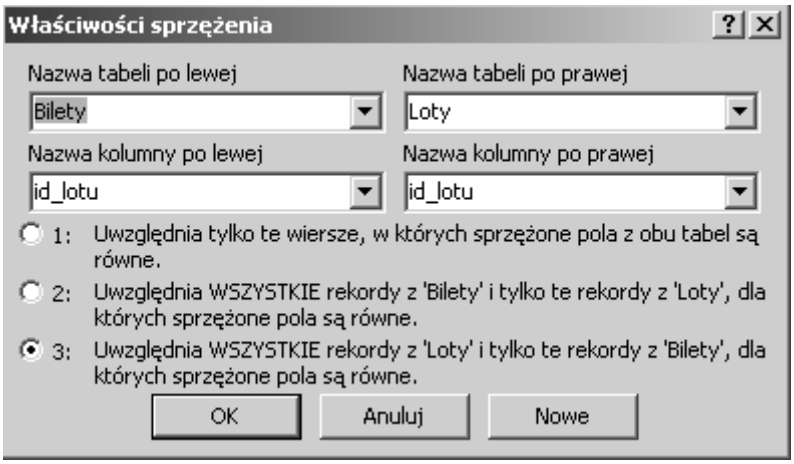
Podobny rezultat można osiągnąć, budując kwerendę, w której w kryteriach pola `id_lotu` dla tabeli `Loty` wpiszemy `NOT IN (SELECT Bilety.id_lotu FROM Bilety))`.

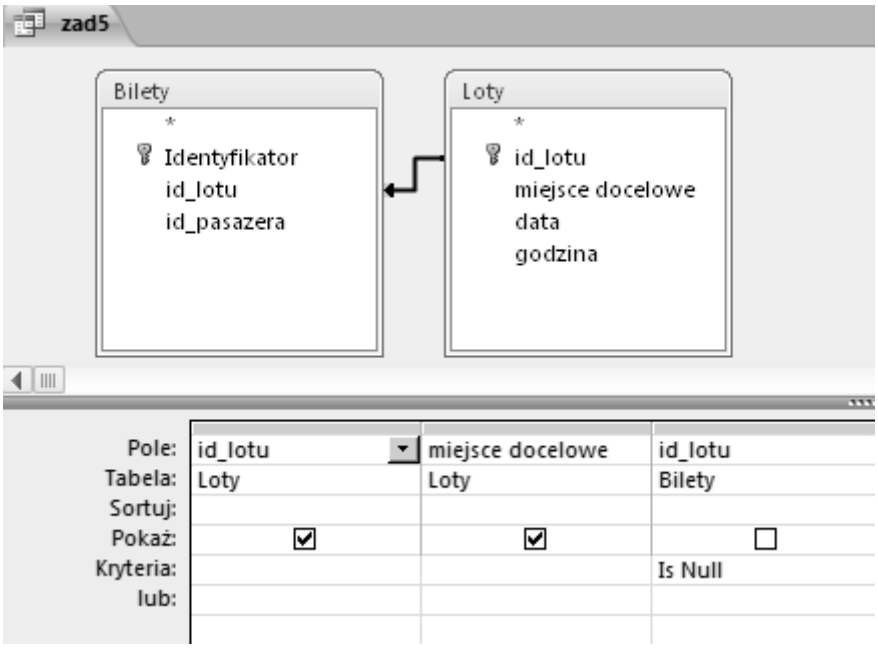


Drugi sposób to zastosowanie złączenia tabel `RIGHT JOIN()`. We właściwościach sprzężenia uwzględniamy wszystkie rekordy z tabeli `Loty` i tylko te rekordy z tabeli `Bilety`, dla

których pola sprzężone są równe. Żeby uzyskać rozwiązanie zadania, wystarczy do kwerendy wybrać pole `id_lotu` z tabeli `Bilety`, a w kryteriach wpisać `Is Null`.







108.1.

Aby obliczyć sumaryczną długość dróg wszystkich kategorii tworzymy kwerendę, która korzystając z informacji z tabeli `drogi` będzie sumować wartości pola `długość` wszystkich wierszy tabeli.

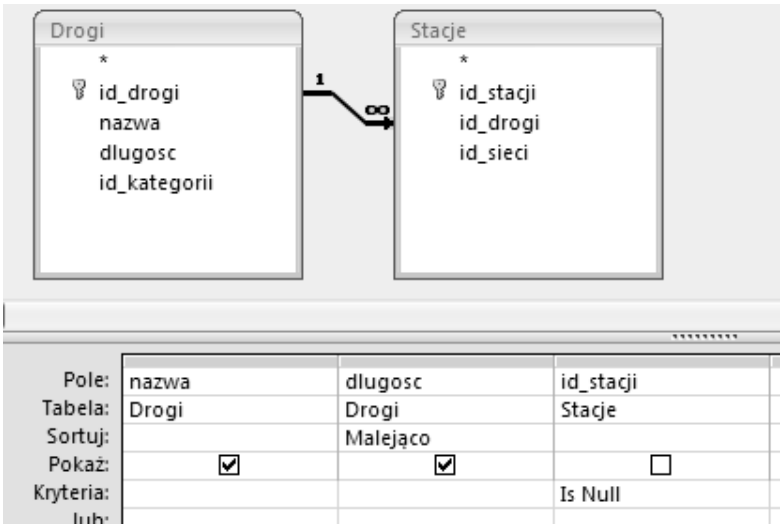
Odpowiednie zapytanie w języku SQL ma postać:

```
SELECT Sum(Drogi.dlugosc) AS [długość całkowita] FROM Drogi;
```

108.2.

Aby znaleźć nazwę i długość najdłuższej drogi, przy której nie leży żadna stacja benzynowa, wykonujemy zestawienie, korzystając z tabel `drogi` i `stacje`. W związku z tym, że szukamy tych rekordów, w których po połączeniu tabel będzie brakowało informacji z tabeli `stacje`, to zmieniamy właściwości sprzężenia tych tabel na niesymetryczne (patrz rysunek). Uwzględniamy zatem wszystkie elementy z tabeli `drogi` i do nich dopasowujemy elementy z tabeli stacje.

Następnie w kwerendzie wybieramy interesujące nas pola `nazwa` i `dlugosc` z tabeli `drogi` oraz identyfikatory stacji (pole `id_stacji`) z tabeli stacje. Ponieważ szukamy tych dróg, przy których nie ma żadnej stacji, wybieramy takie wpisy, dla których brakuje `id_stacji` (pole ma wartość `NULL`). Aby wybrać najdłuższą z tak znalezionych dróg, sortujemy całość malejąco według wartości pola `dlugosc`.

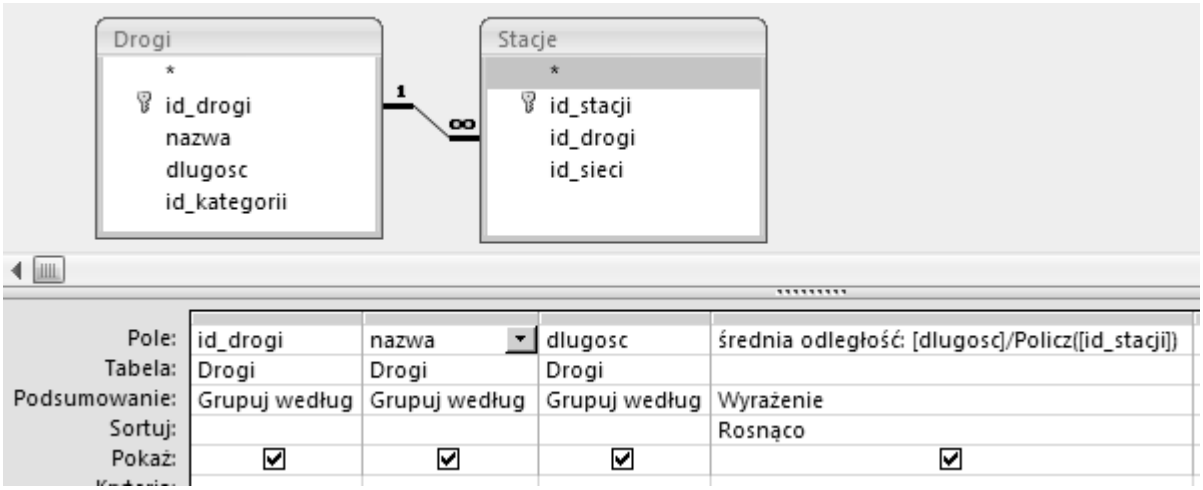


W języku SQL zapytanie wygląda następująco:

```
SELECT Drogi.nazwa, Drogi.dlugosc
FROM Drogi LEFT JOIN Stacje
ON Drogi.id_drogi=Stacje.id_drogi
WHERE ((Stacje.id_stacji) IS NULL)
ORDER BY Drogi.dlugosc DESC;
```

108.3.

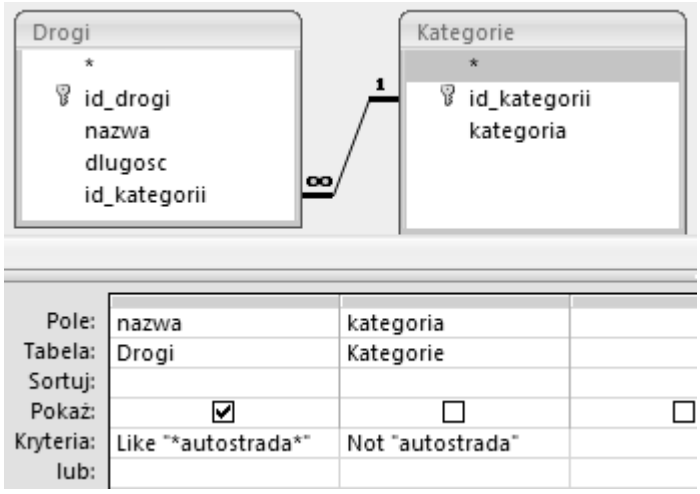
Aby znaleźć drogę, przy której odległość między stacjami benzynowymi jest najmniejsza, wykonujemy zestawienie podobne jak w poprzednim zadaniu. Tym razem powiązanie pomiędzy tabelami `drogi` i `stacje` pozostaje symetryczne (uwzględniamy równe wartości z obu tabel). W zestawieniu uwzględniamy pola: `id_drogi`, `nazwa` i `dlugosc` z tabeli `drogi` (i grupujemy po tych kolumnach) oraz obliczamy średnią odległość pomiędzy stacjami, używając formuły: `dlugosc/COUNT(id_stacji)`.Wyniki sortujemy rosnąco według obliczonej średniej odległości między stacjami.



Odpowiednie zapytanie w języku SQL wygląda następująco:

```
SELECT Drogi.id_drogi, Drogi.nazwa, Drogi.dlugosc,
dlugosc/Count(id_stacji) AS [średnia odległość]
FROM Drogi INNER JOIN Stacje
ON Drogi.id_drogi = Stacje.id_drogi
GROUP BY Drogi.id_drogi, Drogi.nazwa, Drogi.dlugosc
ORDER BY dlugosc/Count(id_stacji);
```

108.4.



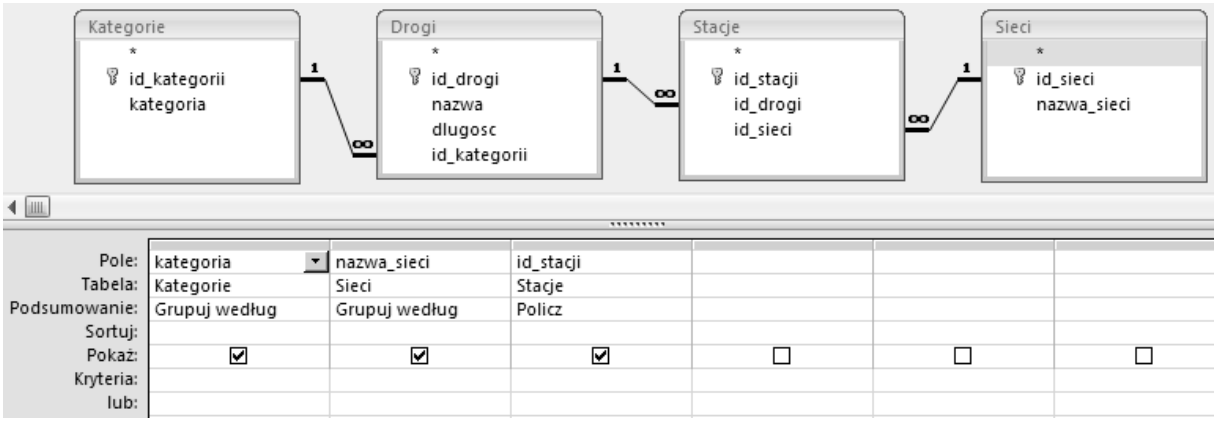
Aby znaleźć drogi nazywane autostradami, ale w rzeczywistości nimi niebędące, wykonujemy zestawienie, wykorzystując tabele `drogi` i `kategorie`. Wybieramy z niego wiersze odnoszące się do dróg, w których nazwie występuje słowo autostrada (`nazwa LIKE "*autostrada*"`), ale które mają inną kategorię (`NOT kategoria = "autostrada"`).

Odpowiednie zapytanie w języku SQL ma postać:

```
SELECT Drogi.nazwa
FROM Kategorie INNER JOIN Drogi
ON Kategorie.id_kategorii = Drogi.id_kategorii
WHERE ((Drogi.nazwa LIKE "*autostrada*") AND (NOT
Kategorie.kategoria="autostrada"));
```

108.5.

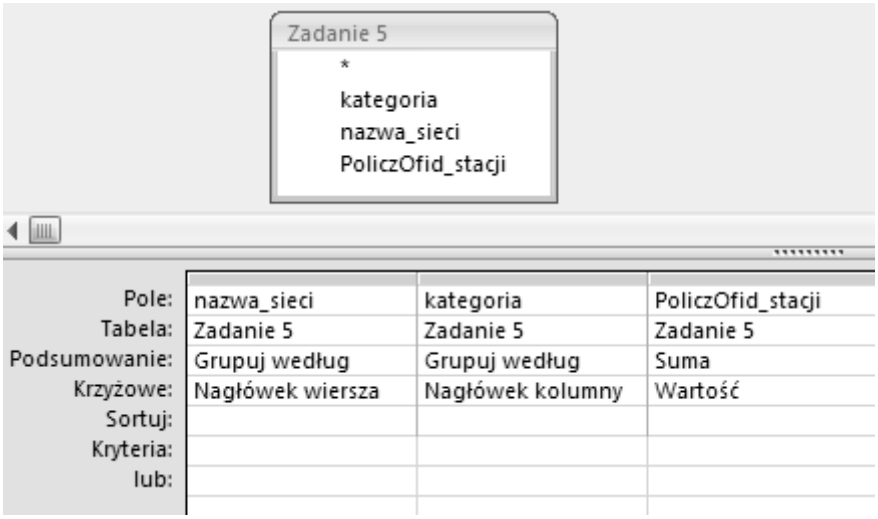
Do wykonania zestawienia liczby stacji każdej z sieci z podziałem na kategorie dróg, przy których te stacje są położone, używamy odpowiednio powiązanych wszystkich czterech tabel. Do zestawienia wybieramy pola: `kategoria` z tabeli `kategorie`, `nazwa_sieci` z tabeli `sieci` oraz `id_stacje` z tabeli `stacje`. Tabela drogi jest tutaj potrzeba do zachowania relacji pomiędzy tabelami `kategorie` i `stacje`. Zestawienie grupujemy po polach `kategoria` i `nazwa_sieci`, zaś pola `id_stacji` zliczamy.



Odpowiednie zapytanie w języku SQL ma postać:

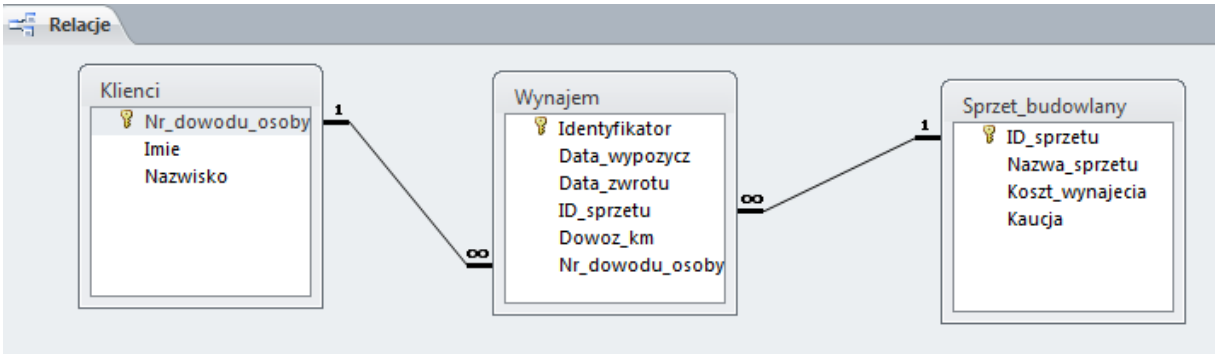
```
SELECT Kategorie.kategoria, Sieci.nazwa_sieci,
Count(Stacje.id_stacji) AS [liczba stacji]
FROM Sieci INNER JOIN (Kategorie INNER JOIN (Drogi INNER JOIN
Stacje ON Drogi.id_drogi = Stacje.id_drogi) ON
Kategorie.id_kategorii = Drogi.id_kategorii) ON Sieci.id_sieci
= Stacje.id_sieci
GROUP BY Kategorie.kategoria, Sieci.nazwa_sieci;
```

Jeżeli używamy bazy danych z graficznym interfejsem użytkownika (np. MS Acccess), dla lepszej prezentacji wyniku tworzymy na bazie tej kwerendy kwerendę krzyżową.



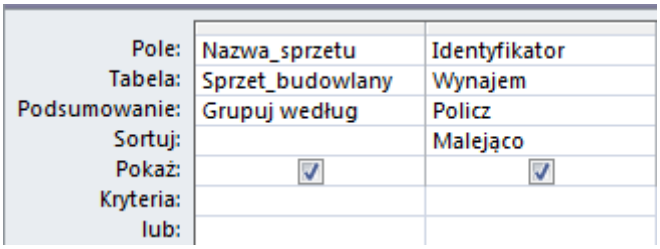
Zadanie 109.

W tabeli `Sprzet_budowlany` kluczem głównym będzie identyfikator sprzętu (`ID_sprzetu`), ponieważ to on jednoznacznie identyfikuje urządzenie, nawet w przypadku kilku urządzeń o identycznych nazwach. Podobna sytuacja dotyczy tabeli `Klienci`, której kluczem głównym będzie numer dowodu osobistego osoby wynajmującej sprzęt (`Nr_dowodu_osoby`), ponieważ każdy dowód ma inny numer, będący kombinacją trzech liter i sześciu cyfr, zaś imiona i nazwiska mogą się powtarzać. W przypadku tabeli `Wynajem` nie ma jednej kolumny, która mogłaby stanowić klucz główny, można więc utworzyć klucz złożony z kilku kolumn, ale w tym wypadku jest to zbędne, ponieważ można pozostawić tabelę bez klucza głównego albo dodać kolumnę z identyfikatorem kolejnych wypożyczeń. Ostatecznie otrzymujemy następujące relacje typu jeden do wielu:



109.1.

Aby wyszukać nazwę urządzenia wypożyczanego najczęściej oraz podać liczbę wypożyczeń tego urządzenia, skorzystamy z informacji zawartych w dwóch tabelach: `Sprzet_budowlany` i `Wynajem`. Dla każdego identyfikatora i nazwy sprzętu policzymy liczbę identyfikatorów wypożyczeń sprzętu (COUNT), następnie posortujemy malejąco według lizcby wypożyczeń (ORDER BY….. DESC) i wybierzemy pierwszą wartość (klauzula LIMIT 1).



Zapytanie w jezyku SQL:

SELECT Sprzet_budowlany.Nazwa_sprzetu, Count(*) **AS** liczba_urz
FROM Sprzet_budowlany **INNER JOIN** Wynajem **ON** Sprzet_budowlany.ID_sprzetu = Wynajem.ID_sprzetu
GROUP BY Sprzet_budowlany.Nazwa_sprzetu
ORDER BY Count(*) **DESC LIMIT** 1;

109.2.

Konstruując zapytanie dotyczące kwoty kaucji za wszystkie urządzenia wypożyczone przez Andrzeja Rydawskiego, skorzystamy się funkcji grupującej (sumowanie wartości w wierszach) oraz filtra dla pola tekstowego. Ważne jest, abyśmy podczas grupowania posłużyli się numerem dowodu (kluczem głównym), gdyż okazuje się, że mamy w bazie dwie osoby o nazwisku Andrzej Rydawski.

Klauzula GROUP BY umożliwia podział wierszy na kategorie na podstawie wartości w kolumnie (np. nr dowodu) i skorzystanie z funkcji grupujących (sumowanie) dla różnych numerów dowodu. Założenie warunku może odbyć się na dwa sposoby: albo za pomocą klauzuli HAVING, albo klauzuli WHERE.

Zapytanie z klauzulą HAVING:

SELECT Klienci.Imie, Klienci.Nazwisko, Sum(Sprzet_budowlany.Kaucja) **AS** SumaKaucji
FROM Sprzet_budowlany **INNER JOIN** Klienci **INNER JOIN** Wynajem **ON** Klienci.Nr_dowodu_osoby = Wynajem.Nr_dowodu_osoby **ON** Sprzet_budowlany.ID_sprzetu = Wynajem.ID_sprzetu
GROUP BY Klienci.Nr_dowodu_osoby, Klienci.Imie, Klienci.Nazwisko
HAVING Klienci.Nr_dowodu_osoby="JCK343973";

Widok projektu:

Pole:	Imie	Nazwisko	Kaucja	Nr_dowodu_osoby
Tabela:	Klienci	Klienci	Sprzet_budowlany	Klienci
Podsumowanie:	Grupuj według	Grupuj według	Suma	Grupuj według
Sortuj:				
Pokaż:	☑	☑	☑	☐
Kryteria:				"JCK343973"
lub:				

109.3

Zadanie stawia problem wyeliminowania tych samych urządzeń (poprzez grupowanie), a następnie wymaga zliczenia liczby różnych urządzeń wypożyczanych przez te same osoby (COUNT) i wybrania tych nazwisk, dla których ta liczba jest większa niż 3 (HAVING). W dalszym ciągu pamiętamy, że kluczem głównym w tabeli `Klienci` jest numer dowodu.

Kwerenda pomocnicza (`pom`) eliminująca powtarzalność urządzeń:

SELECT Wynajem.Nr_dowodu_osoby, Sprzet_budowlany.ID_sprzetu
FROM Sprzet_budowlany **INNER JOIN** Wynajem **ON** Sprzet_budowlany. ID_sprzetu = Wynajem.ID_sprzetu
GROUP BY Wynajem.Nr_dowodu_osoby, Sprzet_budowlany.ID_sprzetu;

Widok projektu:

Pole:	Nr_dowodu_osob	ID_sprzetu
Tabela:	Wynajem	Sprzet_budowlany
Podsumowanie:	Grupuj według	Grupuj według
Sortuj:		
Pokaż:	☑	☑
Kryteria:		

Druga kwerenda korzysta z wyników pierwszej:

SELECT Klienci.Imie, Klienci.Nazwisko
FROM Klienci **INNER JOIN** pom **ON** Klienci.Nr_dowodu_osoby = pom.Nr_dowodu_osoby
GROUP BY Klienci.Imie, Klienci.Nazwisko, Klienci.Nr_dowodu_osoby
HAVING Count(pom.ID_sprzetu)>3;

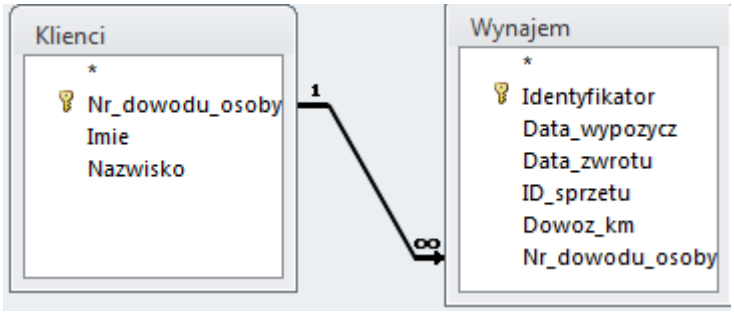
Widok projektu:

Pole:	Imie	Nazwisko	ID_sprzetu	Nr_dowodu_osoby
Tabela:	Klienci	Klienci	pom	Klienci
Podsumowanie:	Grupuj według	Grupuj według	Policz	Grupuj według
Sortuj:				
Pokaż:	☑	☑	☐	☐
Kryteria:			>3	
lub:				

109.4.

Aby podać liczbę zarejestrowanych w bazie firmy klientów, którzy w 2014 roku nie wypożyczyli żadnego urządzenia, zastosujemy pomiędzy tabelami złączenie typu LEFT JOIN. Pozwala ono na uwzględnienie w wyniku tych danych z pierwszej tabeli, które nie posiadają swoich odpowiedników w drugiej. Dane te zostaną wzięte pod uwagę podczas złączenia, ale puste miejsca zostaną wypełnione wartościami NULL. Kolejnym krokiem jest wybranie rekordów z wartością NULL.

Widok projektu:



Pole:	Nr_dowodu_osob	Identyfikator
Tabela:	Klienci	Wynajem
Sortuj:		
Pokaż:	☑	☑
Kryteria:		Is Null
lub:		

Zapytanie z łączeniem LEFT JOIN:

SELECT Klienci.Nr_dowodu_osoby, Wynajem.Identyfikator
FROM Klienci LEFT JOIN Wynajem **ON** Klienci.Nr_dowodu_osoby = Wynajem.Nr_dowodu_osoby
WHERE Wynajem.IdentyfikatorIsNull;

109.5

Na początek obliczmy przychód firmy z tytułu wypożyczeń urządzeń budowlanych w kolejnych miesiącach 2014 r. Do wyodrębnienia numeru miesiąca z daty wypożyczenia posłuży funkcja `MONTH`. Aby obliczyć kwotę uzyskaną za konkretne wypożyczenie, należy w pierwszej kolejności wyznaczyć liczbę dób, na które urządzenie było wynajęte, a następnie pomnożyć ją przez koszt wynajęcia na jedną dobę. Takie wyniki grupujemy po miesiącach, zaś kwoty sumujemy.

Zapytanie przyjmuje formę:

SELECT Month(Wynajem.Data_wypozycz) **AS** Miesiac, Sum(Wynajem.Data_zwrotu-Wynajem.Data_wypozycz)*Sprzet_budowlany.Koszt_wynajecia AS Kwota
FROM Sprzet_budowlany **INNER JOIN** Wynajem **ON** Sprzet_budowlany.ID_sprzetu = Wynajem.ID_sprzetu
GROUP BY Month(Wynajem.Data_wypozycz);

Widok projektu:

Pole:	Miesiac: Month([Wynajem]![Data_wypozyc	Kwota: ([Wynajem]![Data_zwrotu]-[Wynajem]![Data_wypozycz])*[Sprzet_budowlany]![Koszt_wynajecia]
Tabela:		
Podsumowanie:	Grupuj według	Suma
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

Aby ustalić przychody uzyskane z transportu, policzymy liczbę transportów do 10 km włącznie i pomnożymy ją przez 50 zł oraz obliczymy liczbę transportów powyżej 10 km i pomnożymy ją przez 100 zł. Następnie wszystkie wartości w danym miesiącu zsumujemy.

Kwerenda pomocnicza: obliczanie liczby transportów do 10 km:

SELECT Month(Wynajem.Data_wypozycz) **AS** Miesiac, Count(Wynajem.Dowoz_km) **AS** do_10
FROM Wynajem
WHERE ((Wynajem.Dowoz_km)>0 **And** (Wynajem.Dowoz_km)<11)
GROUP BY Month(Wynajem.Data_wypozycz);

Widok projektu:

Pole:	Wyr1: Month([Wynaje	Dowoz_km	Dowoz_km
Tabela:		Wynajem	Wynajem
Podsumowanie:	Grupuj według	Policz	Gdzie
Sortuj:			
Pokaż:	☑	☑	☐
Kryteria:			>0 And <11
lub:			

Kwerenda pomocnicza: obliczanie liczby transportów powyżej 10 km:

SELECT Month(Wynajem.Data_wypozycz) **AS** Miesiac, Count(Wynajem.Dowoz_km) **AS** powyzej_10
FROM Wynajem
WHERE Wynajem.Dowoz_km>10
GROUP BY Month(Wynajem.Data_wypozycz);

Widok projektu:

Pole:	Miesiac: Month([Wyn	Dowoz_km	Dowoz_km
Tabela:		Wynajem	Wynajem
Podsumowanie:	Grupuj według	Policz	Gdzie
Sortuj:			
Pokaż:	☑	☑	☐
Kryteria:			>10
lub:			

Kwerenda obliczająca przychód z transportu (korzysta z powyższych kwerend pomocniczych):

SELECT pom2.Miesiac,Sum(pom1.do_10km*50+pom2.powyżej_10km*100) **AS** Dochod_transport
FROM pom1 **INNER JOIN** pom2 **ON** pom1.Miesiac = pom2.Miesiac
GROUP BY pom2.Miesiac;

Widok projektu:

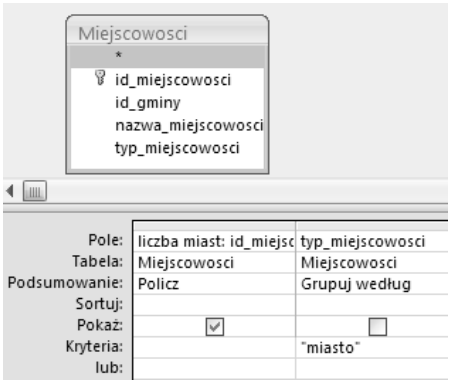
Pole:	Miesiac	Dochod_transport: [pom1]![PoliczOfDowoz_km]*50+[pom2]![PoliczOfDowoz_km]*100
Tabela:	pom2	
Podsumowanie:	Grupuj według	Suma
Sortuj:		
Pokaż:	☑	☑
Kryteria:		
lub:		

Zadanie 110.

Po poprawnym wczytaniu zawartości plików tekstowych do tabel przystępujemy do rozwiązywania zadań.

110.1.

Aby obliczyć liczbę miast na terenie Polski, tworzymy kwerendę, która na podstawie tabeli `miejscowosci` policzy liczbę miejscowości o typie „`miasto`”.

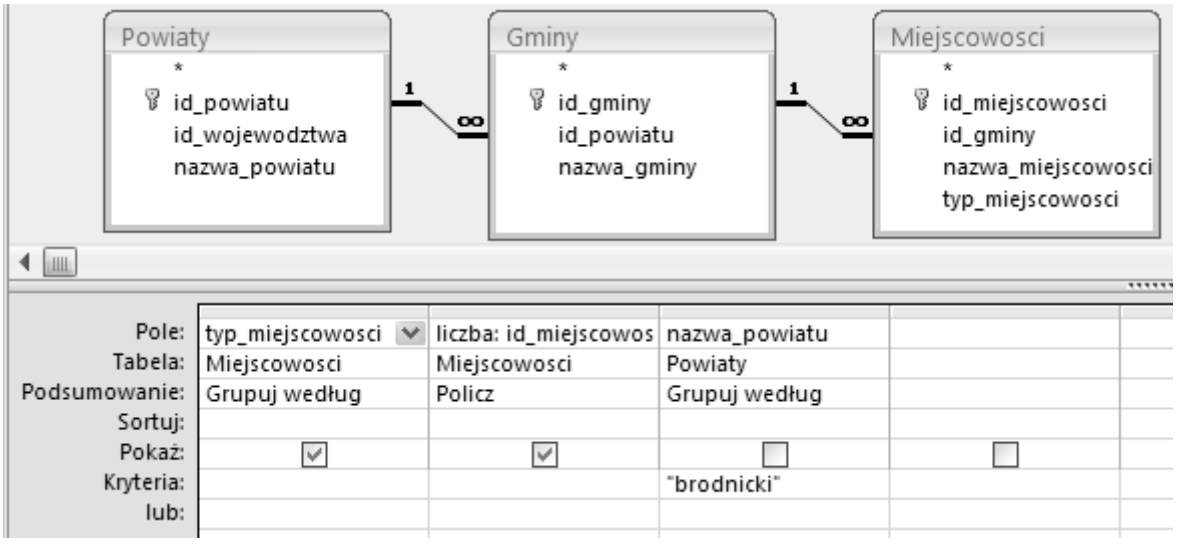


Odpowiednie zapytanie w języku SQL ma posać:

```
SELECT
Count(Miejscowosci.id_miejscowosci) AS
[liczba miast]
FROM Miejscowosci
GROUP BY Miejscowosci.typ_miejscowosci
HAVING ((Miejscowosci.typ_miejscowosci)="miasto");
```

110.2.

Aby obliczyć, ile w powiecie brodnickim jest miejscowości każdego typu (wieś, miasto, osada itd.), wykorzystujemy w zapytaniu tabele: `powiaty`, `gminy` oraz `miejscowosci`. Ustalamy, że interesują nas miejscowości z powiatu o nazwie „`brodnicki`”, grupujemy wyniki po polu `typ_miejscowosci` i zliczamy `id_miejscowosci`.

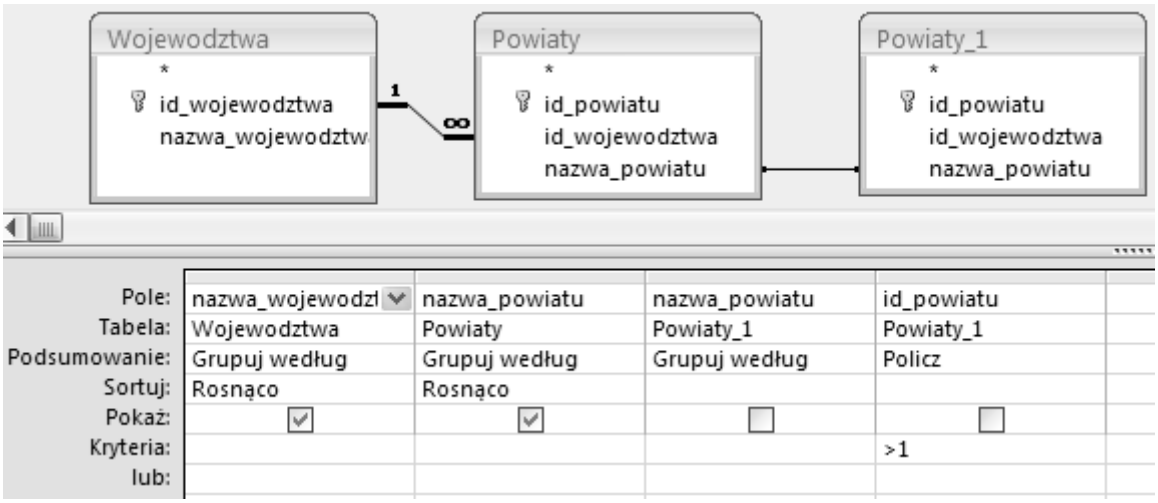


Odpowiednie zapytanie w języku SQL wygląda następująco:

```
SELECT Miejscowosci.typ_miejscowosci,
Count(Miejscowosci.id_miejscowosci) AS liczba
FROM Powiaty INNER JOIN (Gminy INNER JOIN Miejscowosci ON
Gminy.id_gminy = Miejscowosci.id_gminy) ON Powiaty.id_powiatu
= Gminy.id_powiatu
GROUP BY Miejscowosci.typ_miejscowosci, Powiaty.nazwa_powiatu
HAVING ((Powiaty.nazwa_powiatu)="brodnicki");
```

110.3.

Aby znaleźć nazwy powiatów powtarzające się w skali kraju, wykorzystujemy w zapytaniu tabele `wojewodztwa` oraz `powiaty`. Tabeli `powiaty` używamy dwukrotnie. Pierwsze użycie zachowuje właściwą relację z tabelą `wojewodztwa`. Drugie (w przykładzie i rozwiązaniu SQL nazwane `powiaty_1`) jest powiązane z pierwszym przez pole `nazwa`. Umożliwi to połączenie dwóch różnych powiatów o tej samej nazwie. Wyniki grupujemy według nazw województw i powiatów (w tabeli `powiaty` będącej w odpowiedniej relacji z tabelą `wojewodztwa`), zaś `id_powiatu zliczamy` z tabeli `powiaty_1`. Wynikiem w tym zadaniu jest lista powiatów, w których naliczyliśmy więcej niż jeden powiat z tabeli `powiaty_1`. Wynik odpowiednio sortujemy według nazwy powiatu i nazwy województwa.

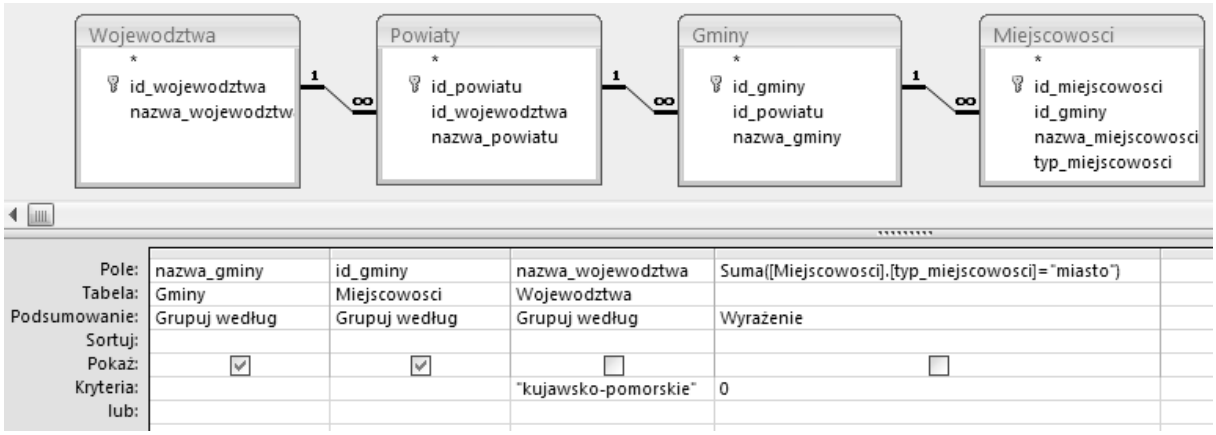


Odpowiednie zapytanie w języku SQL ma postać:

```
SELECT Wojewodztwa.nazwa_wojewodztwa, Powiaty.nazwa_powiatu
FROM Wojewodztwa INNER JOIN (Powiaty INNER JOIN Powiaty AS
Powiaty_1 ON Powiaty.nazwa_powiatu = Powiaty_1.nazwa_powiatu)
ON Wojewodztwa.id_wojewodztwa = Powiaty.id_wojewodztwa
GROUP BY Wojewodztwa.nazwa_wojewodztwa, Powiaty.nazwa_powiatu,
Powiaty_1.nazwa_powiatu
HAVING (Count(Powiaty_1.id_powiatu)>1)
ORDER BY Powiaty.nazwa_powiatu, Wojewodztwa.nazwa_wojewodztwa;
```

110.4.

Aby policzyć gminy wiejskie w województwie kujawsko-pomorskim, wykorzystujemy wszystkie cztery tabele dostępne w zadaniu. Dla wszystkich wierszy zapytania przyrównujemy typ miejscowości do `"miasto"`. Grupujemy po nazwie gminy i wybieramy z nich jedynie te, które leżą w odpowiednim województwie. W wyniku wybieramy jedynie te z nich, w których suma porównań typów miejscowości do `"miasto"` jest róc)wna 0. W wyniku takiego zapytania otrzymujemy tabelę wszystkich gmin wiejskich w województwie. Odpowiedzią do zadania jest liczba wierszy w tej tabeli.



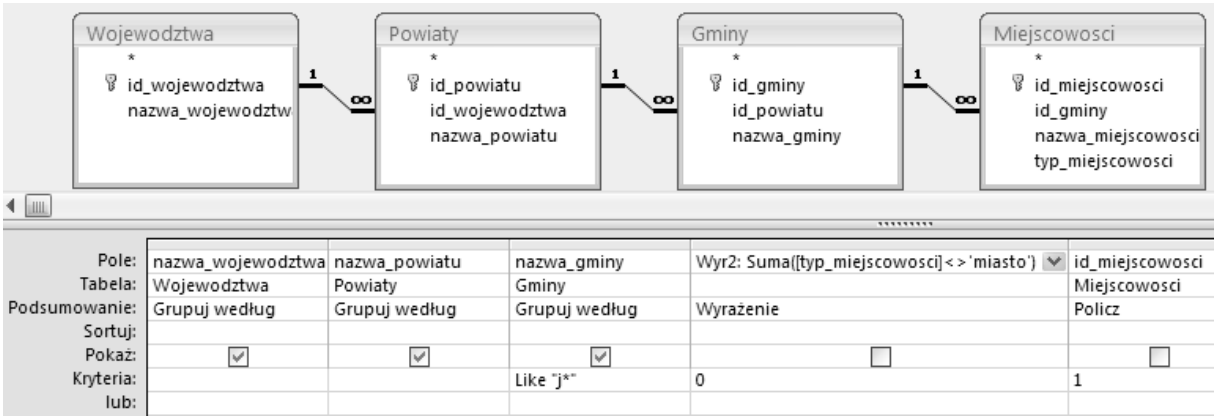
Następnie odczytujemy liczbę wierszy odpowiedzi albo tworzymy kolejne zapytanie, zliczające liczbę wierszy z zaprezentowanego powyżej zapytania.

Odpowiednie zapytanie w języku SQL, od razu zliczające liczbę gmin, wygląda następująco:

```
SELECT Count(*) AS [liczba gmin]
FROM (SELECT Gminy.nazwa_gminy, Miejscowosci.id_gminy
      FROM Wojewodztwa INNER JOIN (Powiaty INNER JOIN (Gminy
      INNER JOIN Miejscowosci ON Gminy.id_gminy =
      Miejscowosci.id_gminy) ON Powiaty.id_powiatu =
      Gminy.id_powiatu) ON Wojewodztwa.id_wojewodztwa =
      Powiaty.id_wojewodztwa
      GROUP BY Gminy.nazwa_gminy, Miejscowosci.id_gminy,
      Wojewodztwa.nazwa_wojewodztwa
      HAVING (((Wojewodztwa.nazwa_wojewodztwa)="kujawsko-
      pomorskie") AND
      (Sum([Miejscowosci].[typ_miejscowosci]="miasto")=0))
);
```

110.5.

Aby wykonać zestawienie gmin miejskich o nazwach zaczynających się na „J”, wykorzystujemy dane zawarte we wszystkich czterech tabelach. Tabel `wojewodztwa` i `powiaty` używamy jedynie po to, by dla każdej gminy określić nazwy jednostek administracyjnych wyższego rzędu. Wynik grupujemy po nazwie gminy i wybieramy te gminy, których nazwy zaczynają się na literę J (`nazwa_gminy LIKE "j*"`).Wyboru gmin jedynie miejskich dokonujemy przez nałożenie dwóch dodatkowych warunków: policzenie miejscowości na terenie gminy musi dać w wyniku 1, zaś zsumowanie wynikód)w warunku logicznego (`typ_miejscowosci<>'miasto'`) musi być równe 0. Uwzględnienie jedynie pierwszego z tych dwóch warunków w tym pytaniu da błędny wynik z powodu gminy *Jajkowice*, na terenie której leży dokładnie jedna miejscowość, która jednak nie jest miastem.



Pole:	nazwa_wojewodztwa	nazwa_powiatu	nazwa_gminy	Wyr2: Suma([typ_miejscowosci]<>'miasto')	id_miejscowosci
Tabela:	Wojewodztwa	Powiaty	Gminy		Miejscowosci
Podsumowanie:	Grupuj według	Grupuj według	Grupuj według	Wyrażenie	Policz
Sortuj:					
Pokaż:	☑	☑	☑	☐	☐
Kryteria:			Like "j*"	0	1
lub:					

Odpowiednie zapytanie w języku SQL ma posać:

```
SELECT Wojewodztwa.nazwa_wojewodztwa, Powiaty.nazwa_powiatu,
Gminy.nazwa_gminy
FROM Wojewodztwa INNER JOIN (Powiaty INNER JOIN (Gminy INNER
JOIN Miejscowosci ON Gminy.id_gminy = Miejscowosci.id_gminy)
ON Powiaty.id_powiatu = Gminy.id_powiatu) ON
Wojewodztwa.id_wojewodztwa = Powiaty.id_wojewodztwa
GROUP BY Wojewodztwa.nazwa_wojewodztwa, Powiaty.nazwa_powiatu,
Gminy.nazwa_gminy
HAVING (((Gminy.nazwa_gminy) LIKE "j*") AND
(Sum([typ_miejscowosci]<>'miasto')=0) AND
(Count(Miejscowosci.id_miejscowosci)=1));
```

Zadanie 111.

Importujemy dane z plików tekstowych:

- z pliku `malware.txt` tworzymy tabelę `Malware`, w której pole `data` jest typu Data/Godzina, pole `ASN` jest liczbą całkowitą, a pozostałe pola są tekstowe;
- z pliku `asn.txt` tworzymy tabelę `ASN`, w której pole `ASN` jest liczbą całkowitą, a pozostałe pola są tekstowe;
- z pliku `kraje.txt` tworzymy tabelę `Kraje`, w której oba pola są tekstowe.

Jako pola kluczowe w tabelach wybieramy

- w tabeli `Asn` pole `ASN`;
- w tabeli `Kraje`: pole `ID_kraju` .
- W tabeli `Malware` zgadzamy się na utworzenie dodatkowego pola `Identyfikator`, które będzie jednocześnie przechowywać w niej numer rekordu.

W programie Microsoft Access możemy od razu ustawić i zapisać relacje tabel

- pole `ASN` połączy tabele `Malware` i `Asn`;
- pole `ID_kraju` połączy tabele: `Asn` i `Kraje` .

111.1.

Należy wyszukać te pozycje *malware*, które w opisie zawierają słowo „phishing” pisane małą lub wielką literą, oraz podać: `URL`, `opis` i nazwę kraju, skąd pochodzi zagrożenie.

Tworzymy kwerendę wybierającą pola: `kraj` z tabeli `Kraje`, `opis` i `URL` z `Malware`. Dla pola `opis` ustawiamy kryterium: **zawiera** „phish”. Program *Microsoft Access* domyślnie nie rozróżnia wielkości liter, ale rozwiązując zadanie w innym programie, trzeba dodać drugie kryterium: **lub zawiera** „Phish”.

Pole:	kraj	opis	URL
Tabela:	Kraje	Malware	Malware
Sortuj:			
Pokaż:	☑	☑	☑
Kryteria:		Like "*phish*"	

111.2.

Należy znaleźć pięć sieci, które udostępniły najwięcej pozycji złośliwego oprogramowania. Dla każdej z tych sieci podamy jej nazwę, ponadto: nazwę kraju, w którym ta sieć jest zarejestrowana, liczbę pozycji *malware* pochodzących tej sieci i liczbę różnych adresów IP, udostępniających malware w tych sieciach.

Problem rozwiązujemy w dwóch krokach. Najpierw zliczamy pozycje malware udostępnione z różnych adresów IP. Wybieramy pola `Kraje.kraj`, `Asn.siec`, `Malware.IP` i `Malware.Identyfikator`. Dodajemy wiersz podsumowania. Grupujemy pola: `IP`, `kraj` i `siec`, dla pola `Identyfikator` wybieramy funkcję `Policz`.

Pole:	kraj	siec	IP	Identyfikator
Tabela:	Kraje	Asn	Malware	Malware
Podsumowanie:	Grupuj według	Grupuj według	Grupuj według	Policz
Sortuj:				

W drugim kroku wykorzystujemy wyniki pierwszej kwerendy. Dla każdej sieci zliczamy różne adresy IP oraz sumujemy liczby identyfikatorów *malware* ze wszystkich adresów IP w tej sieci. Rekordy sortujemy malejąco według licznika identyfikatorów i zwracamy pięć pierwszych rekordów z wyniku kwerendy.

Pole:	kraj	siec	IP	PoliczOfIdentyfikator
Tabela:	2a	2a	2a	2a
Podsumowanie:	Grupuj według	Grupuj według	Policz	Suma
Sortuj:				Malejąco

111.3.

Należy zliczyć różne domeny, z których pochodzi szkodliwe oprogramowanie, oraz znaleźć te domeny, z którymi skojarzony jest więcej niż jeden adres IP. Do rozwiązania problemu potrzebne są tylko dane z tabeli `Malware`.

Aby wyłuskać domenę z pola `URL`, wykorzystamy funkcje `InStr` i `Left` języka SQL. W kwerendzie umieszczamy adres IP z tabeli `Malware` oraz dwa pola obliczeniowe:

- `slash: InStr(1;[Malware]![URL];"/")` –wyszuka pozycję pierwszego wystąpienia znaku ‘/’ w polu `URL`

- `domena: Left([Malware]![adres];[slash]-1)` – wyznaczy adres internetowy domeny zawarty w polu `URL`, biorąc z lewej strony (*slash*-1) znaków.

Uwaga: w programie *Base* funkcja wyszukująca pozycję znaku '/' w `URL` miałaby postać:

```
slash: LOCATE('/', "Malware"."URL", 1)
```

Pole:	slash: InStr(1;[Malware]![URL];"/")	domena: Left([Malware]![URL];[slash]-1)	IP
Tabela:			Malware
Sortuj:			

W kolejnym kroku wykorzystamy wyniki pierwszej kwerendy, jej pola: `domena` i `IP`. Wykorzystując operację grupowania, dla każdej domeny sporządzimy wykaz różnych należących do niej adresów IP.

Pole:	domena	IP
Tabela:	3a	3a
Podsumowanie:	Grupuj według	Grupuj według

Wreszcie w trzecim kroku, wykorzystując wyniki drugiej kwerendy, zliczymy różne adresy IP w każdej domenie. Niech nowa kwerenda grupuje domeny, zaś do pola `IP` zastosuje funkcję `Policz`. Kwerenda zwróci nam 687 rekordów, tyle ile jest różnych domen, podając dla każdej domeny liczbę adresów IP z nią skojarzonych.

Pole:	domena	IP
Tabela:	3b	3b
Podsumowanie:	Grupuj według	Policz
Sortuj:		Malejąco

Sortujemy rekordy malejąco według licznika IP, aby znaleźć domeny skojarzone z największą liczbą adresów IP. Możemy też ustawić dla licznika IP kryterium: "> 1", wówczas kwerenda zwróci tylko te rekordy, w których z domeną skojarzony jest więcej niż jeden adres IP.

111.4.

Należy sporządzić zestawienie, w którym dla każdego miesiąca w roku 2014 podamy liczbę pozycji *malware* z podziałem na regiony.

Do rozwiązania potrzebne są funkcje przetwarzania daty. Funkcja `YEAR` wyłuska rok, a funkcja `MONTH` wyłuska miesiąc z pełnej daty, zwracając jego numer z zakresu 1..12.

W projekcie kwerendy umieszczamy dwa pola danych potrzebne do zestawienia: `Asn.region` i `Malware.Identyfikator`, którego wartości będziemy zliczać. Tworzymy dwa pola obliczane: `Miesiąc`, które będzie zawierać miesiąc wyłuskany z daty, oraz `Rok` — rok wyłuskany z daty.

Dla pola obliczanego `Rok` ustawiamy kryterium, które wybierze nam tylko dane z roku 2014.

Pole:	Rok_2014: Year([Malware]![data])	Miesiac: Month([Malware]![data])	region	Identyfikator
Tabela:			Asn	Malware
Sortuj:				
Pokaż:	☑	☑	☑	☑
Kryteria:	2014			

Na podstawie wyników tej kwerendy zbudujemy drugą kwerendę, krzyżową, która zwróci nam zestawienie w postaci 2-wymiarowej tablicy wyników. W nagłówku wiersza ustawimy

`Miesiąc`, w nagłówku kolumny `region`, a wartością niech będzie licznik identyfikatorów (w wierszu `Podsumowania` dla pola `Identyfikator` zastosuj funkcję `Policz`).

Pole:	Miesiac	region	Identyfikator
Tabela:	4a	4a	4a
Podsumowanie:	Grupuj według	Grupuj według	Policz
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość
Sortuj:			
Kryteria:			

Zamiast tworzyć drugą kwerendę mogliśmy wyeksportować wyniki pierwszej kwerendy do arkusza kalkulacyjnego i tam wstawić tabelę przestawną, w której numer miesiąca byłby nagłówkiem wiersza, nazwa regionu — nagłówkiem kolumny, a licznik identyfikatorów — wartością.

111.5.

Dla każdego kraju, z którego udostępniono malware ukryte w plikach graficznych, należy podać liczbę pozycji *malware* z podziałem na format jpg i png.

Najpierw wybierzemy wszystkie pozycje *malware*, których URL kończy się znakami „.jpg” lub „.png”, przy każdym z nich umieśćmy informację o jego kraju pochodzenia.

Pole:	kraj	URL
Tabela:	Kraje	Malware
Sortuj:		
Pokaż:	☑	☑
Kryteria:		Like "*.jpg" Or Like "*.png"

Teraz należałoby zgrupować rekordy według kraju i w każdej grupie zliczyć wybrane URL.

Pole:	kraj	PoliczOfadres: URL
Tabela:	5_0	5_0
Podsumowanie:	Grupuj według	Policz

W ten sposób dla każdego kraju dostaniemy łączną liczbę *malware* w plikach graficznych, jednak bez podziału na formaty plików.

Aby uzyskać podział na różne formaty plików, trzeba dodać w pierwszej kwerendzie pole obliczane `format`. Umieścimy w nim 4 ostatnie znaki pola URL, które wytniemy za pomocą funkcji `RIGHT(”tekst”,liczba znaków)`. W tym polu ustawimy kryterium: wartość równa ”.jpg” lub ”.png”

Pole:	kraj	URL	format: Right([Malware]![URL];4)
Tabela:	Kraje	Malware	
Sortuj:			
Pokaż:	☑	☑	☑
Kryteria:			".jpg" Or ".png"

Następnie utworzymy kwerendę krzyżową, która zliczy wybrane pola URL dla każdego kraju, osobno w każdym z formatów:

Pole:	kraj	Format	URL
Tabela:	5_1_0	5_1_0	5_1_0
Podsumowanie:	Grupuj według	Grupuj według	Policz
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość

Zadanie 112.

Najpierw importujemy dane z plików tekstowych do tabel. W tabelach dobieramy typy pól odpowiednie do treści danych i ustawiamy klucz główny.

Dane z pliku `towary.txt` importujemy do tabeli `Towary`: pola `nazwa` i `rodzaj` jako tekst, pola `masa` i `VAT` jako liczby rzeczywiste (np. o podwójnej precyzji), pole `EAN` jako tekst (choć wygląda jak liczba, ma wartości zbyt duże, aby przechować je w polu o typie liczby całkowitej). Kluczem głównym może być pole EAN, ponieważ jego wartości w pliku danych są unikatowe, jednoznacznie identyfikujące towar.

Dane z pliku `producenci.txt` importujemy do tabeli `Producenci`, wszystkie cztery pola jako tekstowe. Kody krajów i oddziałów mogłyby być liczbami całkowitymi, lecz jeśli będą tekstami, wygodniej będzie później tworzyć wiązania z tabelą `Towary`. Jako klucz główny wybieramy pole `kod_oddzialu`.

Dane z pliku `kraje.txt` importujemy do tabeli `Kraje`, oba pola jako tekstowe. Klucz główny ustawiamy w polu `kod_kraju`.

112.1.

Dla każdego rodzaju towaru należy wyznaczyć liczbę krajów, w których zarejestrowani są producenci. Pierwsze trzy cyfry kodu EAN każdego towaru są kodem kraju producenta. Wystarczy więc wyciąć kod kraju z kodu EAN i zliczyć różne kody krajów w ramach każdego rodzaju towarów.

Wystarczą nam dane tylko z jednej tabeli: `Towary`. Tworzymy kwerendę wybierającą, umieszczamy w niej pole `rodzaj` i pole obliczane: kod kraju wycięty z pola `EAN` za pomocą funkcji LEFT.

W projekcie kwerendy dodajemy wiersz `Podsumowanie` i grupujemy w nim rodzaje towarów oraz uzyskane kody krajów. Kwerenda tworzy zestawienie, w którym dla każdego rodzaju towaru mamy listę kodów różnych krajów. Dzięki grupowaniu kodów, każdy kod kraju wystąpi tylko jeden raz w grupie wybranego rodzaju towaru.

Pole:	rodzaj	kod_kraju: Left([Towary]![EAN];3)
Tabela:	Towary	
Podsumowanie:	Grupuj według	Grupuj według
Sortuj:	Rosnąco	

Utworzymy następnie drugą kwerendę na podstawie wyników pierwszej. W wierszu Podsumowanie wybierzemy grupowanie w polu rodzaj, zaś dla kodów krajów wybierzemy funkcję Policz, która zwróci liczbę kodów w każdej grupie rodzaju towarów.

Pole:	rodzaj	ile_krajow: kod_kraju
Tabela:	1a	1a
Podsumowanie:	Grupuj według	Policz
Sortuj:		

112.2.

Dla wszystkich producentów obecnych na naszym rynku, a zarejestrowanych w Czechach, należy podać **nazwy** ich **firm** oraz **lokalizacje** ich oddziałów.

Potrzebne są nam dane z dwóch tabel: `Producenci` i `Kraje`. Tworzymy złączenie tych tabel (relację), łącząc pola `kod_kraju`, które występują w obu tabelach.

Do projektu kwerendy wybieramy pola: `kraj` z tabeli `Kraje` oraz `nazwa_firmy` i `lokalizacja` z tabeli `Producenci`. Dla pola `kraj` ustawiamy kryterium „Czechy”. Możemy usunąć zaznaczenie pola `kraj` w wierszu `Pokaż`, wówczas pole nie będzie widoczne w wynikach kwerendy, ale mimo to ustawione dlań kryterium będzie obowiązywać.

Pole:	kraj	nazwa_firmy	lokalizacja
Tabela:	Kraje	Producenci	Producenci
Sortuj:		Rosnąco	
Pokaż:	☐	☑	☑
Kryteria:	"Czechy"		

112.3.

Do znalezienia **minimalnej masy** towaru dla każdego rodzaju towarów wystarczy kwerenda wybierająca z wierszem `Podsumowanie`. Weźmiemy pola: `rodzaj` i `masa` z tabeli `Towary`. Pogrupujemy dane według rodzaju, a dla pola `masa` wybierzemy funkcję `Minimum`.

Pole:	rodzaj	masa
Tabela:	Towary	Towary
Podsumowanie:	Grupuj według	Minimum
Sortuj:		

Wypisanie nazwy towaru o minimalnej masie oraz nazwy firmy producenta jest trudniejsze.

Pierwszym krokiem do rozwiązania zadania jest utworzenie kwerendy, która posortuje towary według dwóch kluczy: ze względu na rodzaj, a w każdym rodzaju według masy — tu koniecznie **rosnąco**.

Pole:	rodzaj	masa	nazwa
Tabela:	Towary	Towary	Towary
Sortuj:	Rosnąco	Rosnąco	
Pokaż:	☑	☑	☑

w

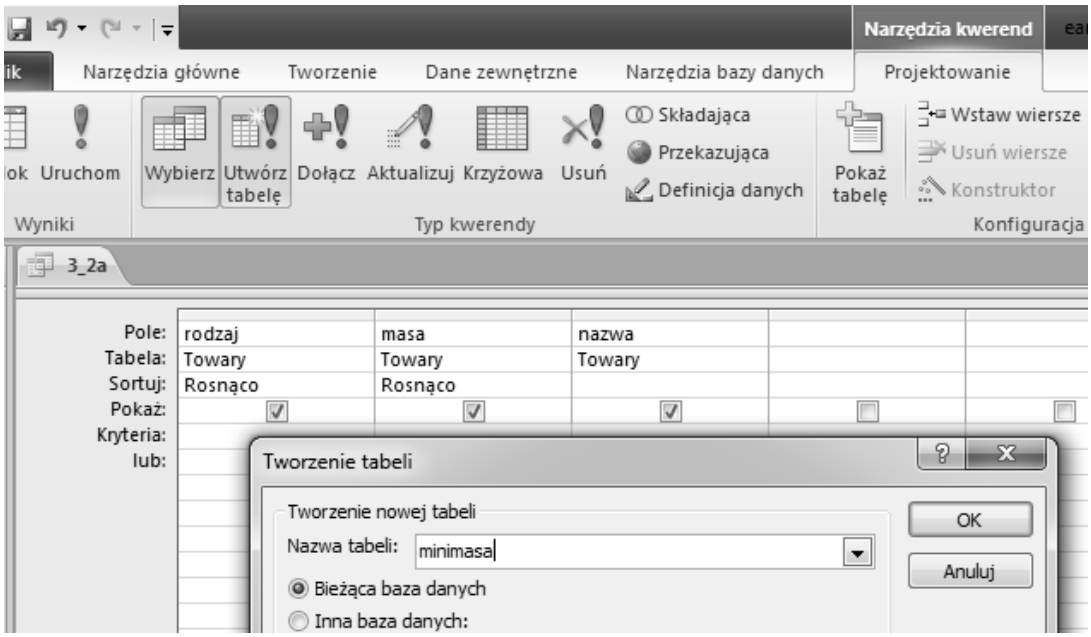
Najprościej będzie, gdy wyeksportujmy wyniki tej kwerendy do arkusza kalkulacyjnego.

W arkuszu utworzymy nową kolumnę o nazwie `duplikat`. W jej komórkach umieścimy wartość 0 jeśli rodzaj towaru w bieżącym wierszu jest inny niż w wierszu poprzednim, lub wartość 1, jeśli ten rodzaj jest taki sam. Następnie przy pomocy filtra wybierzemy tylko te wiersze, które w polu `duplikat` mają wartość 0.

	A	B	C	
1	rodzaj	duplikat	masa	
2	bakalie	0	50	MIGDAŁY BLANSZOWANE
3	bakalie	=JEŻELI(A3=A2;1;0)	100	RODZYNKI
4	bakalie	=JEŻELI(A4=A3;1;0)	100	SKORKA POMARAńCZY

W programie *Microsoft Access* możemy rozwiązać problem bez użycia arkusza.

Wyniki kwerendy (sortującej według dwóch kluczy) zapiszemy w nowej tabeli. Aby taką kwerendę uruchomić, trzeba było wcześniej zgodzić się na włączenie zawartości aktywnej przy otwieraniu pliku bazy.



Następnie tworzymy nową kwerendę, wybierającą dane z tej nowej tabeli. Grupujemy rekordy w polu `rodzaj`, w polu *masa* wybieramy, jak poprzednio, funkcję `Minimum`. W polu: `nazwa` wybieramy funkcję `Pierwszy`. Wówczas po uruchomieniu kwerendy ujrzymy dane z pierwszych rekordów poszczególnych grup (wyodrębnionych ze względu na rodzaj), a już wcześniej zadbaliśmy o to, aby w tym pierwszym rekordzie znalazł się produkt o najmniejszej masie.

Pole:	rodzaj	masa	nazwa
Tabela:	minimasa	minimasa	minimasa
Podsumowanie:	Grupuj według	Minimum	Pierwszy
Sortuj:	Rosnąco		

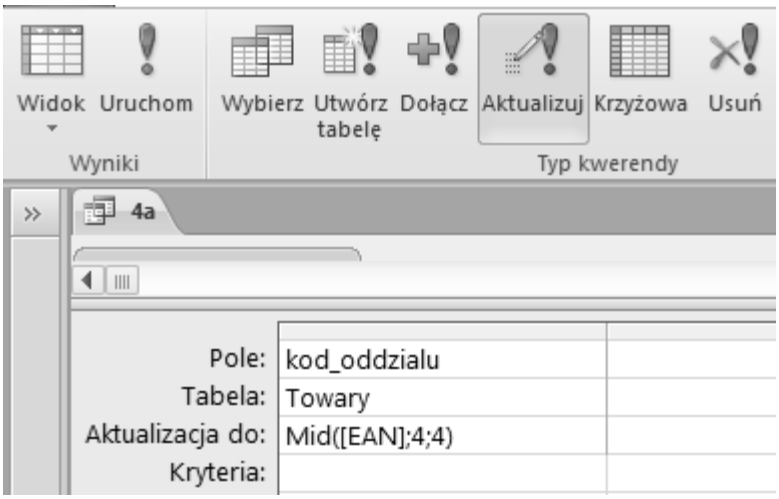
112.4.

Wyszukamy batony w opakowaniach o masie przekraczającej 300g. Dla każdego z nich podamy: nazwę towaru, masę oraz nazwę firmy i nazwę kraju producenta.

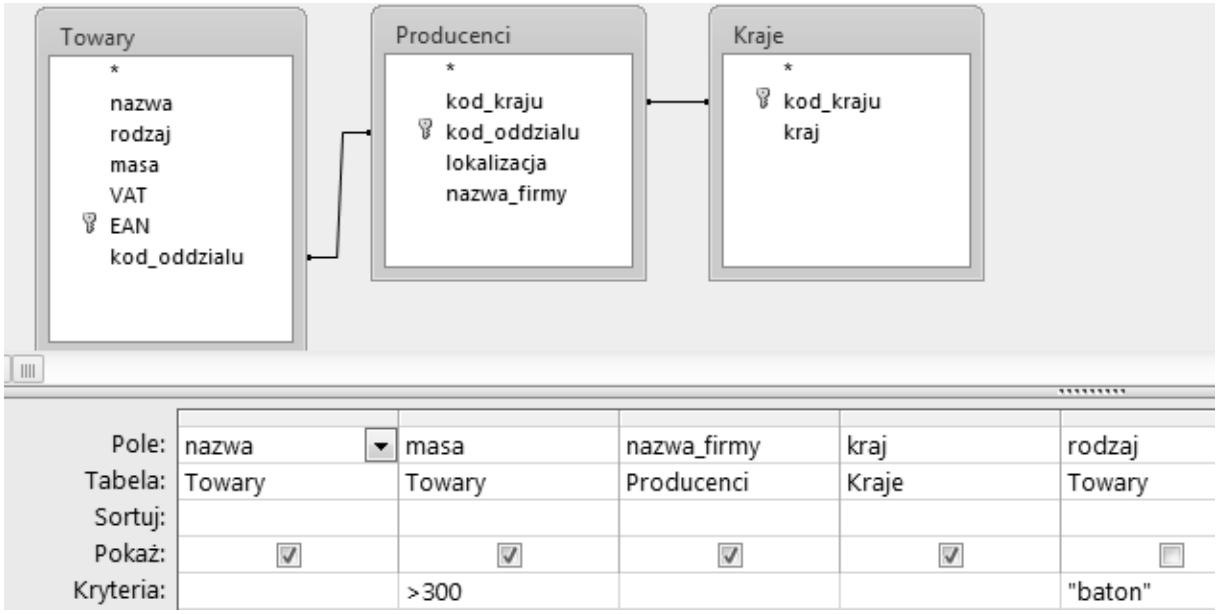
W tym celu potrzebujemy danych z trzech połączonych tabel. Tabele `Kraje` i `Producenci` złączymy za pomocą pola `kod_kraju` i dowiemy się, jaki jest kraj wybranego producenta.

Brakuje nam powiązania tabeli `Towary` z tabelą `Producenci`, aby móc poznać nazwy firm, które wyprodukowały wybrane towary. W tym celu trzeba wyłuskać kod oddziału producenta z kodu EAN towaru.

W tabeli `Towary` dodajemy nowe pole o nazwie `kod_oddzialu`, typu tekstowego. Następnie tworzymy kwerendę typu: `Aktualizująca`, która umieści w tym polu, w każdym wierszu tabeli, 4-cyfrowy łańcuch znaków wycięty z kodu EAN, począwszy od znaku na czwartej pozycji. Do operacji na tekście wykorzystamy funkcję `MID`.



Teraz już nie ma przeszkód, aby utworzyć relacje wiążące wszystkie trzy tabele, wybrać potrzebne pola i ustawić żądane kryteria dla rodzaju towaru („baton") oraz jego masy (>300).



112.5.

Dla każdego kraju należy podać liczbę towarów pochodzących od producentów z tego kraju, z podziałem na poszczególne stawki podatku VAT.

Do zliczania potrzebne jest nam pole `kraj` z tabeli `Kraje` oraz pole VAT z tabeli `Towary` i pole identyfikujące towary (`EAN`).

Jeśli rozwiązaliśmy zadanie 4, to dla każdego towaru mamy określone pole `kod_oddzialu` i możemy, podobnie jak w zadaniu 4, związać tabele `Towary` i `Kraje` za pośrednictwem tabeli `Producenci`.

Jeśli nie mamy w tabeli `Towary` pola z kodem oddziału, możemy z kodu EAN każdego towaru wyłuskać kod kraju, zapisać jego wartość w nowym polu tabeli `Towary` (w sposób podobny jak w zadaniu 4, w którym tworzyliśmy pole `kod_oddzialu`) i wykorzystać to pole do połączenia tabeli `Towary` wprost z tabelą `Kraje`.

Ponieważ rozwiązaliśmy zadanie 4, więc możemy postąpić według sposobu, opisanego wyżej jako pierwszy. Utworzymy kwerendę opartą na trzech tabelach i zapewnimy ich powiązanie

za pomocą pól `kod_oddzialu` i `kod_kraju`. Weźmiemy do projektu pola `Kraje.kraj`, `Towary.VAT` i `Towary.EAN`.

W wierszu Podsumowania będziemy grupować dane w polach `kraj` i `VAT`, dla pola `EAN` wybierzemy funkcję `Policz`.

Zmienimy typ kwerendy na **kwerendę krzyżową**. Jako nagłówek wiersza wybierzemy `kraj`, jako nagłówek kolumny — `VAT`, jako wartość — licznik kodów EAN.

Pole:	kraj	VAT	EAN
Tabela:	Kraje	Towary	Towary
Podsumowanie:	Grupuj według	Grupuj według	Policz
Krzyżowe:	Nagłówek wiersza	Nagłówek kolumny	Wartość

4. Odpowiedzi

Zadanie 2.

2.1.

Kolejne wykonanie	**Wartość zmiennej** *y*
1	0,6
2	0,2
3	0,4
4	0,8
5	0,6

Liczba wypisana przez algorytm: 0,10011.

2.2.

Musi to być liczba, która ma dokładnie cztery cyfry po przecinku w zapisie dwójkowym. Możliwe odpowiedzi to zatem: 1/16 (0,0625), 3/16 (0,1875), 5/16 (0,3125),..., 15/16 (0,9375).

2.3.

```
funkcja trójkowy(x, k)
    wypisz „0,”
    y ← x
    dla i=1, 2, ..., k wykonuj
        jeżeli y ≥ 2/3
            wypisz „2”
        jeżeli y ≥ 1/3 oraz y < 2/3
            wypisz „1”
        jeżeli y < 1/3
            wypisz „0”
        y ← y * 3
        jeżeli y≥2
            y ← y – 2          // prawidłową odpowiedzią jest również y ← y-1
        jeżeli y≥1
            y ← y – 1
```

Zadanie 3.

3.1.

wywołanie	**wynik**
F(2, 10)	1024
F(2; 9)	512
F(2, 3)	8
F(2, 1)	2

3.2.

x	*N*	wynik **F(*x*, *n*)**
2	2	4
2	3	8
3	4	81
2	5	32
2	8	256
2	10	1024

3.3.

X	*N*	Liczba operacji mnożenia
2	2	1
2	3	2
3	4	3
4	7	4
4	8	5
4	9	4

3.4.

$lmnozen(n) = 2 \cdot \log_3 n$

Zadanie 4.

4.1.

liczba w systemie silniowym	**liczba w systemie dziesiętnym**
$(310)_!$	20
$(2011)_!$	51
$(54211)_!$	711

4.2.

Największa liczba 5-cyfrowa zapisana w zapisie silniowym to $(54321)_!$.

4.3.

x	*k*	*x* div *k*!	*x* mod *k*!
5489	7	1	449
449	6	0	449
449	5	3	89
89	4	3	17
17	3	2	5
5	2	2	1
1	1	1	0

Liczba dziesiętna 5489 w zapisie silniowym: $(1033221)_!$

4.4.

Przykładowa poprawna odpowiedź (luki oznaczono szarym kolorem):

silnia ← 1
k ←1
dopóki (*silnia* < *x*) **wykonuj**
    k ←*k* + 1
    silnia← *silnia** k*
jeżeli *silnia* > *x*
silnia ← *silnia* div *k*
k ← *k* – 1
s ←” ”
dopóki (*k*>0) **wykonuj**
    cyfra ← *x* div *silnia*
    s ← *s* ◦ tekst (*cyfra*)
    x ← *x* – *cyfra* **silnia* lub *x* ← *x* mod *silnia*
    silnia ← *silnia* div *k*
k ←*k* – 1

Zadanie 5.

5.1.

Poprawna odpowiedź:

Wartość *j*	Liczba wykonań	
	k ← *i*	*p* ← *i*
5	1	0
4	1	1
3	2	0
2	1	1
1	0	3

5.2.

Uzupełnione fragmenty algorytmu:

dla *j* = *n* - 1, *n* - 2, … , 1 **wykonuj**
    x ← *A*[*j*]
    i ← *j* + 1
    dopóki (*i* ≤ *n*) **i** (*x* > A[i]) **wykonuj**
        A[*i* - 1] ← *A*[*i*]
        i ← *i* + 1
    A[*i* - 1] ← *x*

5.3.

FPPP

Zadanie 6.

6.1.

k	Początkowa zawartość tablicy $A[1...2^k]$	Końcowa zawartość tablicy $A[1...2^k]$
2	[4, 3, 1, 2]	[1, 4, 3, 2]
2	[2, 3, 4, 1]	[1, 3, 2, 4]
3	[1, 2, 3, 4, 5, 6, 7, 8]	[1, 2, 3, 4, 5, 6, 7, 8]
3	[8, 7, 6, 5, 4, 3, 2, 1]	[1, 8, 7, 6, 5, 4, 3, 2]
3	[4, 5, 6, 1, 8, 3, 2, 4]	[1, 5, 4, 6, 2, 8, 3, 4]

6.2.

FFP

6.3.

PFPF

6.4.

Poprawnie uzupełnione luki:

Po zakończeniu działania algorytmu 2 element $A[i]$ jest *nie większy (mniejszy bądź równy)*

niż element $A[i+1]$ dla każdego i większego od *0*

oraz mniejszego od *n*

Wiersz (*) **algorytmu 2** wykonywany będzie w przebiegu algorytmu

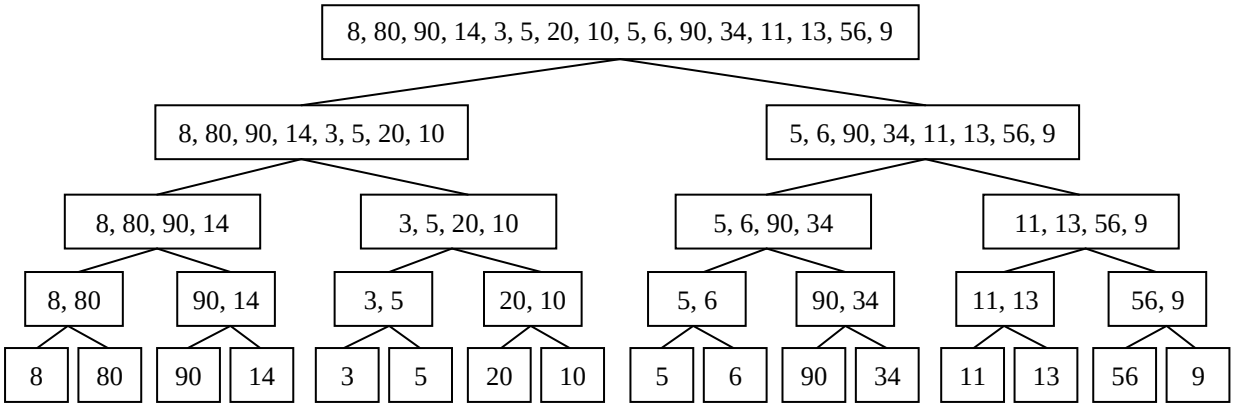
- *więcej* niż n razy
- *mniej* niż n^2 razy

Zadanie 7.

7.1.

- T = [15, 11], wynik algorytmu to [11, 15];
- T = [8, 4, 2, 1], wynik algorytmu to [1, 2, 4, 8];
- T = [10, 15, 1 , 6, 9, 2, 5, 90], wynik algorytmu to [1, 2, 5, 6, 9, 10, 15, 90].

7.2.



7.3.

49

Zadanie 8.

8.1.

Tablica *A*	Tablica *B*	*x*	Wynik działania algorytmu	Liczba porównań w kroku (*)
3, 5, 12, 17	8, 10, 13, 14	21	**FAŁSZ**	**7**
4, 6, 8, 10	5, 7, 9, 11	13	**PRAWDA**	**2**

8.2.

Przykład poprawnej odpowiedzi:

Wynik: *PRAWDA*, gdy istnieje co najmniej jeden element a należący do tablicy *A* i co najmniej jeden element b należący do tablicy *B* takie że $x = a + b$;
FAŁSZ, gdy dla wszystkich elementów a należących do tablicy *A* i elementów b należących do tablicy *B* zachodzi $x \neq a + b$.

8.3.

Przykład poprawnej odpowiedzi:

A=[1, 3, 7, 9, 11] *B*=[1, 13, 14, 16, 18]

Zadanie 9.

9.1.

n	X	y	z
2	A	C	B
1	A	B	C
1	B	C	A
2	C	B	A
1	C	A	B
1	A	B	C

9.2.

A⇒B; A⇒C; B⇒C; A⇒B; C⇒A; C⇒B; A⇒B

9.3.

n	H(n)
1	1
2	3
3	7
4	15
5	31
7	127
10	1023

$H(n) = 2^n - 1$

9.4.

A⇒C, A⇒B, C⇒B, A⇒C, B⇒A, B⇒C, A⇒C, A⇒B, C⇒B, C⇒A, B⇒A, C⇒B, A⇒C, A⇒B, C⇒B.

Zadanie 10.

10.1.

$Z[0] = 2,\ \ Z[1] = 4,\ \ Z[2] = 16,\ Z[3] = 256$

10.2.

n	**Liczba operacji mnożenia**
1	0
2	1
4	3
8	7
16	15
1024	1023

oraz

$$f(n) = 2f(n/2) + 1.$$

10.3.

Prawidłowa odpowiedź:

- *F*([9,1,0,-6,0,10,0,2,0,4,-1,0,0,0,-3,5])
  - *F*([9,1,0,-6,0,10,0,2])
    - *F*([9,1,0,-6])
      - *F*([9,1])
        - *F*([9])
        - *F*([1])
      - *F*([0,-6])
        - *F*([0])
        - *F*([-6])
    - *F*([0,10,0,2])
      - *F*([0,10])
        - *F*([0])
        - *F*([10])
      - *F*([0,2])
        - *F*([0])
        - *F*([2])
  - *F*([0,4,-1,0,0,0,-3,5])
    - *F*([0,4,-1,0])
      - *F*([0,4])
        - *F*([0])
        - *F*([4])
      - *F*([-1,0])
        - *F*([-1])
        - *F*([0])
    - *F*([0,0,-3,5])
      - *F*([0,0])
        - *F*([0])
        - *F*([0])
      - *F*([-3,5])
        - *F*([-3])
        - *F*([5])

Prawidłowy wzór na liczbę wywołań rekurencyjnych:

$$g(n) = 2n - 1.$$

10.4.

n	**Liczba operacji mnożenia**
3	2
9	8
27	
81	
243	

Zadanie 11.

11.1.

$W = W_1$ op W_2	W_1	W_2	op	ONP(W)
4 + 3	4	3	+	4 3 +
(4 + 3) ∗ 2	(4+3)	2	∗	4 3 + 2 ∗
5 ∗ (7 – 6)	5	(7 – 6)	∗	5 7 6 – ∗
((4 + 3) ∗ 2) – (5 ∗ (7 – 6))	((4 + 3) ∗ 2)	(5 ∗ (7 – 6))	–	4 3 + 2 ∗ 5 7 6 – ∗ –

Uwaga

Za poprawne należy też uznać odpowiedzi podające wyrażenia w kolumnach *W*1 i *W*2 bez nawiasów zewnętrznych, np. 4 + 3 zamiast (4 + 3).

11.2.

i	Wartość zmiennej *k* po *i*-tym przebiegu pętli	Zawartość tablicy *T*[1..*k*-1] po *i*-tym przebiegu pętli
1	2	9
2	3	9, 7
4	3	16, 3
5	2	48
6	3	48,5
10	3	48,2
11	2	46

11.3.

Napis	Wartość zmiennej *licznik* po zakończeniu	Czy poprawne wyrażenie ONP?
1 2 + ∗	1	NIE
1 2 + 3 4 – 5 ∗ 7 8 + 9	4	NIE
1 2 3 4 5 + + + +	1	TAK
1 2 3 4 5 + + + + + +	1	NIE
1 2 3 4 5 + + + + +	1	NIE
1 2 + 2 3 – 3 4 ∗ 4 5 + – – –	1	TAK
1 2 + 2 3 – 3 4 ∗ 4 5 + – – – – –	1	NIE
1 2 + 3 4 – 5 ∗ 7 8 + 9 + + +	1	TAK

11.4.

ONP(X): 1 2 op_1 3 op_2 4 op_3 5 op_4 6 op_5 7 op_6 8 op_7 9 op_8 10 op_9

ONP(Y): 1 2 3 4 5 6 7 8 9 10 op_9 op_8 op_7 op_6 op_5 op_4 op_3 op_2 op_1

Zadanie 12.

12.1.

Lp	*i*	*k*
1	0	20
2	**1**	**21**
3	**2**	**23**
4	**3**	**26**
5	**4**	**30**
6	**5**	**35**

12.2.

Możliwych wartości *k*, dla których algorytm daje różne szyfrogramy, jest 26.

Krótkie uzasadnienie:

Podczas szyfrowania do różnicy liczby 90 i kodu kolejnego znaku dodawana jest wartość *k*. Dla tak obliczonej wartości obliczana jest reszta z dzielenia jej przez 26. Zatem zastąpienie *k* wartością *k*+26 da identyczny wynik, czyli tylko wartości *0*, *1*, ..., 25 dadzą istotnie różne szyfrogramy.

12.3.

K	*Tekst*[]	*Szyfrogram*[]
1	MAPA	**BOBT**
14	**KOD**	DAN

12.4.

Przykład poprawnej odpowiedzi:

```
i ← 0
dopóki (i<n) wykonuj
     Tekst[i] ← (90-Szyfrogram[i]+k) mod 26 + 65
     i ← i+1
     k ← k+i
```

Zadanie 13.

13.1.

TAK, NIE, TAK.

13.2.

P_6 (3,-2)

A (2,1)

Zadanie 14.

14.1.

i	x_i	y_i	$x_{i+1} - x_{i-1}$	$(x_{i+1} - x_{i-1}) \cdot y_i$
1	2	2	-7	-14
2	1	7	7	49
3	9	8	9	72
4	10	4	-1	-4
5	8	2	-8	-16

Pole działki jest równe 87/2.

14.2.

Prawidłowy wzór:

$$\frac{(x_2 - x_3)y_1 + (x_3 - x_1)y_2 + (x_1 - x_2)y_3}{2}.$$

14.3.

Działanie	**Liczba operacji**
dodawanie lub odejmowanie	$2n$
mnożenie	n

Dopuszczalna jest również odpowiedź, w której liczba operacji mnożenia jest równa $n + 1$, tj. taka, w której uwzględniono ostatnie dzielenie przez 2 jako mnożenie przez 1/2.

Zadanie 15.

15.1.

n	Liczba odcinków w zbiorze Cantora rzędu *n*
0	1
1	2
2	4
3	8
5	**32**
6	**64**
9	**512**
10	**1024**

Ogólny wzór określający liczbę odcinków w zbiorze Cantora rzędu n: $\mathbf{2^n}$

15.2.

n	Długość jednego odcinka w zbiorze Cantora rzędu *n*
0	1
1	1/3
2	1/9
3	1/27
4	**1/81**
5	**1/243**
6	**1/729**
7	**1/2187**

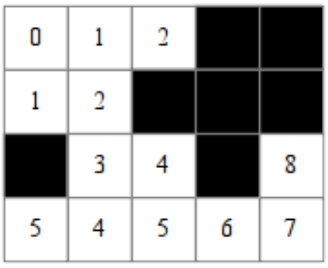
Ogólny wzór określający długość jednego odcinka w zbiorze Cantora rzędu n: $\frac{1}{3^n}$

15.3.

Zbiór Cantora rzędu 3 z końcami odcinków zapisanymi jako ułamki nieskracalne:
[0; 1/27], [2/27; 1/9], [2/9;7/27], [8/27;1/3],
[2/3; 19/27], [20/27,7/9], [8/9;25/27], [26/27; 1].
Zbiór Cantora rzędu 3 z końcami odcinków zapisanymi w systemie trójkowym:
[0; 0,001],[0,002;0,01], [0,02; 0,021], [0,022; 0,1],
[0,2; 0,201],[0,202;0,21], [0,22;0,221], [0,222; 1].

Zadanie 16.

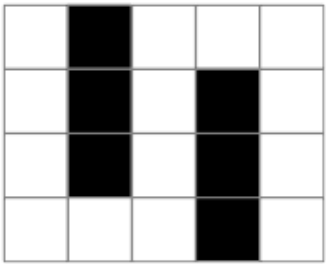
16.1.

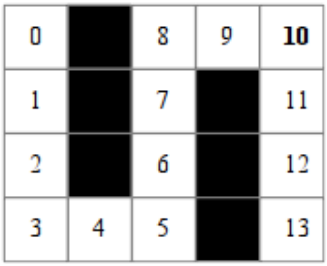


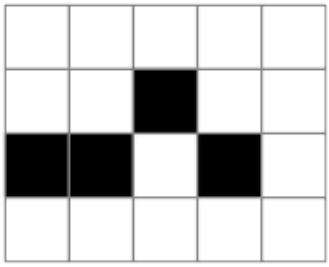
16.2.

Algorytm liczy dla każdego pola jego *odległość* od lewego górnego rogu: minimalną liczbę kroków, którą trzeba zrobić, aby przechodząc po polach sąsiednich, dostać się od lewego górnego rogu do tego pola.

Prawidłowym rozwiązaniem jest dowolna plansza, na której trzeba zrobić przynajmniej 10 kroków. Dwa możliwe rozwiązania przedstawione są na rysunkach:







0	1	2	3	4
1	2	■	4	5
■	■	**10**	■	6
11	**10**	9	8	7

16.3.

Prawidłowym rozwiązaniem jest dowolna plansza, na której niektóre białe pola nie są osiągalne (są „odcięte") z lewego górnego rogu planszy. Na przykład:

0	■	**-1**	**-1**
1	■	**-1**	**-1**

Zadanie 17.

17.1.

Błędny zapis w pseudokodzie	**Poprawny zapis w pseudokodzie**
jeżeli *prędkość samochodu* <30 **lub** *odległość między samochodami* <15	**jeżeli** *prędkość samochodu* <30 **i** *odległość między samochodami* <15
powtarzaj dopóki *prędkość samochodu* = 100:	**powtarzaj dopóki** *prędkość samochodu* > 0:

17.2.

Dane testowe	**Typ danych testowych**
***prędkość samochodu* — 29 km/h,** ***odległość między samochodami* — 1 m**	**brzegowe**
prędkość samochodu — 400 km/h, *odległość od najbliższego pojazdu — 0 m*	niezgodne
prędkość samochodu — 13 km/h, *odległość od najbliższego pojazdu — 8 m*	standardowe
prędkość samochodu — 45 km/h, *odległość od najbliższego pojazdu — 17 m*	standardowe
prędkość samochodu — 0 km/h, *odległość od najbliższego pojazdu — 17 m*	brzegowe

17.3.

PFFP

17.4.

Prędkość samochodu	**Odległość między samochodami**	**Stan automatycznego hamowania**
poniżej 30 km/h	poniżej 15m	włączony

poniżej 30 km/h	równa15m	nieustalony
równa 30 km/h	poniżej 15m	wyłączony
równa 30 km/h	równa 15m	wyłączony
powyżej 30 km/h	dowolna	wyłączony

Zadanie 19.

19.1.

Towar	**Cena towaru zapisana w systemie liczbowym planety**			
	Liczbowo$_2$	**Liczbowo$_4$**	**Liczbowo$_8$**	**Liczbowo$_{10}$**
Kozaki	10111011	2323	273	187
Płaszcz	111010100	13110	724	468
Skuter	10110110010	112302	2662	1458

19.2.

Liczbowo$_i$	Relacja	Liczbowo$_i$
110000100_2	>	556_8
3123_4	≤	1747_8
110_{10}	≤	11010_3
266_9	≤	110100_3
110111101_2	≥	674_8

19.3.

Suma rachunku pana Dwójkowskiego: 1 0 1 1 0 0 1 0

Suma rachunku pana Czwórkowskiego: 2 3 3 3

Różnica rachunków w systemie obowiązującym na planecie Liczbowo$_{10}$ wynosi: 13

19.4.

Przykładowe rozwiązanie:

$i \leftarrow 1$
$r \leftarrow 0$
dopóki $i < n+1$ **wykonuj**
    $C[i] \leftarrow A[i] + B[i] + r$
    jeżeli $C[i] \geq p$
        $C[i] \leftarrow C[i] - p$
        $r \leftarrow 1$
    w przeciwnym razie
        $r \leftarrow 0$
    $i \leftarrow i + 1$
$C[n+1] \leftarrow r$

Zadanie 20.

20.1.

371, 407, 54748.

20.2.

x	B	prawda/fałsz
3433	6	prawda
4890	5	prawda
8956	3	fałsz
15345	2	fałsz

20.3.

Przykładowe prawidłowe rozwiązanie:

```
m ← n;
d ← 0;
dopóki m>0 wykonuj
    m ← m div B;
    d ← d+1;
m ← n;
suma ← 0;
dopóki m>0 wykonuj
        suma ← suma + potega( m mod B, d );
        m ← m div B;
jeżeli suma=n
        zwróć TAK i zakończ
w przeciwnym razie
        zwróć NIE i zakończ
```

gdzie *potega*(x, n) oblicza *x* do potęgi *n* np. w następujący sposób:

```
potega(x,n)
    jeżeli n = 0
            zwróć 1 i zakończ
    w przeciwnym razie
            zwróć x*potega(x, n-1) i zakończ
```

Zadanie 21.

21.1.

$x, x^2, x^4, x^8, x^9, x^{18}, x^{19}, x^{38}$

21.2.

k	reprezentacja binarna k	liczba mnożeń
4	100	2
5	101	3

6	110	3
7	111	4
8	1000	3
15	1111	6
16	10000	4
22	10110	6
32	100000	5

21.3.

Przykładowa odpowiedź:

$p \leftarrow x$
$i \leftarrow n - 1$
dopóki $i \geq 0$ **wykonuj**
    $p \leftarrow p * p$
    jeżeli $k_i = 1$
        $p \leftarrow p * x$
    $i \leftarrow i - 1$

Zadanie 22.

22.1.

Dane	**Wartość**
liczba naturalna $n \geq 0$	5
liczba rzeczywista x	6
liczby rzeczywiste $a_0, a_1, \ldots, a_n$	7, -8, 2, 1, -13, 10

22.2.

Działanie	**Liczba operacji**
dodawanie	n
mnożenie	n

22.3.

	Wartość w
$k = 5$	5
$k = 4$	10
$k = 3$	22
$k = 2$	39
$k = 1$	85
$k = 0$	179

Algorytm zwróci wynik 179.

22.4.

Przykładowy algorytm:

Dane:
n — liczba całkowita, $n \geq 0$,
x — liczba rzeczywista,
$a_0, a_1, \ldots, a_n$ — liczby rzeczywiste.
Wynik: wartość $R(x)$
Algorytm:
y $\leftarrow x \cdot x$
$w \leftarrow a_n$
dla $k = n-1, n-2, \ldots, 0$ **wykonuj**
$w \leftarrow y \cdot w + a_k$
zwróć w **i zakończ**

Zadanie 23.

23. 1.

Poprawna odpowiedź:

Wartość i	Wartość a	Wartość r
0	0	1
1	1	3
2	4	5
3	9	7
4	16	9
5	25	11

23.2.

PFPF

23.3.

Przykładowe poprawne rozwiązanie:
xi ← *x* / 2
kontynuacja ← prawda
dopóki *kontynuacja* **wykonuj**
    xi ← (*xi* + *x* / *xi*) / 2
    p ← część_całkowita(*xi*)
    jeżeli $p \cdot p \leq x$ **oraz** $(p+1) \cdot (p+1) > x$
        kontynuacja ← fałsz
    jeżeli $(p-1) \cdot (p-1) \leq x$ **oraz** $p \cdot p > x$
        p ← *p* – 1
        kontynuacja ← fałsz
zwróć *p* **i zakończ**

Przykładowe poprawne rozwiązanie języka C:

```
int pierw(double x){
    int p;
    double xi = x / 2;
    bool kontynuacja = true;

    while (kontynuacja){
        xi = (xi + x / xi) / 2;
        p = floor(xi);
        if (p*p<=x && (p+1)*(p+1)>x)
            kontynuacja=false;
        if ((p-1)*(p-1)<=x && p*p>x){
            p=p-1;
            kontynuacja=false;
        }
    }
    return p;
}
```

Zadanie 24.

24.1.

T	F (*p*, *k*, *e*)
[3, 4, 6, 8, 9]	6
[15, 16, 18, 22, 24]	1
[2, 10, 16, 24, 26]	3
[1, 3, 10, 10, 18]	5

24.2.

Funkcja F wykorzystuje

metodę zachłanną.	
strategię „dziel i zwyciężaj”.	X
programowanie dynamiczne.	

24.3.

Liczba wywołań rekurencyjnych funkcji F dla początkowych wartości $p = 1$ oraz $k = n$, będącej potęgą dwójki, jest w najgorszym przypadku równa

n div 2	
$\sqrt{n}$	
$\log_2 n$	X

24.4.

Przykładowe poprawne odpowiedzi:

Rozwiązanie o złożoności logarytmicznej:

prawy ← F(1, *n*, *b*)
lewy ← F(1, *n*, *a* – 1)
w ← *prawy* – *lewy*

Rozwiązanie o złożoności liniowej:

prawy ← F(1, *n*, *b*)
lewy ← F(1, *n*, *a*)
i ← *lewy* – 1
dopóki (*i*>0 **oraz** *T* [*i*] = *T* [*lewy*])
    i ← *i* – 1
lewy ← *i*+1
w ← *prawy* – *lewy*

Zadanie 25.

25.1.

Poprawną odpowiedzią jest słowo, w którym pierwsza i ostatnia litera są różne, a bez pierwszej i ostatniej litery słowo to jest palindromem, np. `tamtotmar`.

25.2.

Przykładowe poprawne rozwiązanie:

```
i ← 1
j ← długość(Zdanie)
dopóki i <j wykonuj
    dopóki Zdanie[i] =' ' wykonuj
      i ← i +1
    dopóki Zdanie[j] =' ' wykonuj
      j ← j – 1
    jeżeli Zdanie[i] ≠ Zdanie [j]
      zwróć NIE i zakończ
    i ← i + 1
    j ← j – 1
zwróć TAK i zakończ
```

Zadanie 26.

26.1.

X	*Y*	Podrzędność
HHGGFFEEDDCCBBAA	*ABCDEFGH*	0
DCBADCBA	*FGHABCJD*	4
ABCDE	*ABCCBAE*	NIE
AAAAA	*AA*	0
ABA	*ACA*	NIE
ACEGJ	*ABCDEFGHJ*	4

26.2.

X	*Y*	wynik algorytmu A
HHGGFFEEDDCCBBAA	*ABCDEFGH*	1
DCBADCBA	*FGHABCJD*	1
ABCDE	*ABCCBA*	0
AAAAA	*AA*	1
AA	*AAAAA*	1
ACEGJ	*ABCDEFGH*	0

Dane: X, Y — słowa, w których występują tylko litery ze zbioru {*A, B, C, D, E, F, G, H, I, J*}

Wynik: **1** — gdy *X* jest słowem podrzędnym względem *Y*, 0 — w przeciwnym przypadku.

26.3.

Poprawnie uzupełnione fragmenty algorytmu:

```
jeżeli Czy_y[i]=prawda oraz Czy_x[i]=fałsz
                k← k+1
jeżeli Czy_y[i]=fałsz oraz Czy_x[i]=prawda
```

26.4.

Przykładowe rozwiązanie 1:

```
dla i=1,2,…,10 wykonuj
        Ile_x[i] ← 0,
        Ile_y[i] ← 0
dx ← dlugosc(X)
dla i=1,2,…,dx wykonuj
        lit ← X[i]
        Ile_x[kod(lit)] ← Ile_x[kod(lit)] + 1
dy ← dlugosc(Y)
dla i=1,2,…,dy wykonuj
        lit ← Y[i]
        Ile_y[kod(lit)] = Ile_y[kod(lit)] + 1
dla i=1,2,…,10 wykonuj
        jeżeli Ile_x[i] ≠ Ile_y[i]
                zwróć 0 i zakończ
zwróć 1
```

Alternatywne rozwiązanie polegać może na przykład na tym, że dla kolejnych liter (A, B, …,I, J) algorytm wyznacza liczbę ich wystąpień w *X* i liczbę ich wystąpień w *Y*, a następnie zwraca –1, gdy są one różne. W rozwiązaniu takim nie potrzebujemy dodatkowych tablic.

Można też to zrobić w jednej tablicy. W pętli dla słowa *X* zwiększamy liczniki wystąpienia liter, zaś w pętli dla słowa *Y* je zmniejszamy. Jeśli w tablicy są same zera, to słowa są równoważne:

Przykładowe rozwiązanie 2:

```
dla i=1,2,…,10 wykonuj
        Ile[i] ← 0,
dx ← dlugosc(X)
dla i=1,2,…,dx wykonuj
        lit ← X[i]
        Ile[kod(lit)] ← Ile[kod(lit)] + 1
dy ← dlugosc(Y)
dla i=1,2,…,dy wykonuj
        lit ← Y[i]
        Ile[kod(lit)] = Ile[kod(lit)] – 1
dla i=1,2,…,10 wykonuj
        jeżeli Ile_x[i] ≠ 0
                zwróć 0 i zakończ
zwróć 1
```

Zadanie 27.

27.1.

Wzorzec	**Tekst**	**W jaki sposób wzorzec występuje w tekście?**
para	*opera*	z błędem
para	*aparat*	dokładnie
kran	*karawana*	nie występuje
sport	*bezspornie*	z błędem
ryt	*zakryty*	dokładnie
sofa	*solanka*	z błędem

27.2.

Prawidłowe odpowiedzi to na przykład *abab* i *abababab* albo *aaaa* i *aaaaaaaa*. Jest jednak dużo więcej możliwości.

27.3.

Przykładowy algorytm:

```
dla i = 1, 2, ..., n–m+1 wykonuj
        bledy ← 0
        dla j = 1, 2, ..., m wykonuj
                jeżeli wzorzec[j] ≠ tekst[i+j-1]
                        błędy ← błędy + 1
        jeżeli błędy ≤ 1
                wypisz „TAK”
                zakończ wykonywanie algorytmu
wypisz „NIE”
```

Zadanie 28.

28.1.

Słowo	***Czy jest spełniony warunek***		***Czy słowo jest słabym A-palindromem?***	***Uzasadnienie***
	2a?	***2b?***		
AABA ABAA	tak	tak	tak	AABA
AAAB BAAA	tak	nie	nie	
AAAB BAAB	nie	nie	nie	
AAAA BBBB	**nie**	**tak**	**nie**	
ABBB ABAA	**tak**	**tak**	**tak**	**ABAA**
ABAA AABA	**tak**	**tak**	**tak**	**ABAA lub AABA**
AAAAA AAAAA	**tak**	**nie**	**nie**	

28.2.

Poprawne wypełnienie luk w pierwszej tabeli:

m	***najmniejsza waga słabego A-palindromu o długości m***	***słaby A-palindromu o długości m i najmniejszej liczbie liter A***
2	2	AA
4	3	**AABA albo ABAA**
8	4	**AABA BBBA albo ABAA BBBA albo ABBB AABA albo ABBB ABAA**

Poprawne wypełnienie luk w drugiej tabeli:

m	***najmniejsza liczba liter A słabego A-palindromu o długości m***
16	**5**
32	**6**
2^{10}	**11**
2^{20}	**21**

28.3.

Przykład poprawnej odpowiedzi:

```
CzySlabe (W)
    jeżeli W jest słowem pustym
        zwróć „Nie” i zakończ
    jeżeli W=A
        zwróć ”Tak” i zakończ
    m ← długość(W)
    jeżeli m mod 2=1
        zwróć ”Nie” i zakończ
    w przeciwnym razie
            jeżeli W[1]≠A lub W[m] ≠A
                zwróć „Nie” i zakończ
            r1 ← CzySlabe(W[1, m/2] )
            jeżeli r1 = „Nie”, to zwróć CzySlabe(W[m/2+1,m] )
            w przeciwnym razie zwróć „Tak”
```

Zadanie 29.

29.1.

MATURA

29.2.

Przykład poprawnej odpowiedzi:

```
pusty(litera)
poprzednia ← pobierz(wiadomosc)
kulka ← pobierz(wiadomosc)
dołącz(litera, poprzednia)
dołącz(litera, kulka)
```

```
dopóki (poprzednia ≠ '●' LUB kulka ≠ '●') wykonuj
        poprzednia ← kulka
        kulka ← pobierz(wiadomosc)
        dołącz (litera, kulka)
```

29.3.

Przykład poprawnej odpowiedzi:

```
pusty(wiadomosc)
dopóki (czy_sa_kulki(ciag)) wykonuj
        dopisz(wiadomosc, pobierz_litere(ciag))
```

Zadanie 30.

30.1.

Przykład poprawnej odpowiedzi

```
funkcja szyfruj(zn, k)
        k←k mod 26
        kod_zn←kod(zn)+k
        jeżeli kod_zn> 90
                kod_zn←kod_zn -26
        wynik znak(kod_zn)
```

30.2.

Słowo jawne	Słowo zaszyfrowane
INFORMATYKA	**JPISWSHBHUL**
KOMPUTER	LQPTZZLZ

30.3.

Przykład poprawnej odpowiedzi:

```
dla i=1,2,… n wykonuj
        J[i]←deszyfruj(S[i], i)

funkcja deszyfruj(zn, k)
        k←k mod 26
        kod_zn←kod(zn) – k
        jeżeli kod_zn< 65
                kod_zn←kod_zn +26
        wynik znak(kod_zn)
```

Zadanie 31.

31.1.

- *k*=3, *W*=ZADANIE1JESTŁATWE
  Tekst wypisany przez algorytm: ZEŁA1ADJTAEWNSEIT
- *k*=4, *W*=ZADANIE1JESTPROSTE
  Tekst wypisany przez algorytm: ZISSAETTD1PEAJRNEO

31.2.

- *k*=3, *W*= UMIEMDEKODOWAĆ
  Tekst wypisany przez algorytm: UDOMEWIKAEOĆMD
- *k*=4, *W*= DOBRZEJEST
  Tekst wypisany przez algorytm: DRJTOZEBES

31.3.

Przykładowe rozwiązanie:

Algorytm:
n ← *dlugosc*(*X*)
dla *i*=1,2,…,*k* **wykonuj**
    j←*i*
    dopóki *j* ≤ *n* **wykonuj**
        wypisz *X*[*j*]
        j ← *j*+*k*

31.4.

Przykładowe rozwiązania:

Algorytm 1:
n ← *dlugosc*(*W*)
m ← *n* div *k*
jeżeli *n* mod *k*≠0
    m←*m*+1
dla *i*=1,2,…,*m* **wykonuj**
    jeżeli (*i* mod 2=1)
        j ← *i*
        dopóki *j* ≤ *n* **wykonuj**
            wypisz *W*[*j*]
            j ← *j*+*m*
    w przeciwnym razie
        j ← *i*+(*k* – 1) · *m*
        jeżeli *j*>*n*
            j ← *j* – *m*
        dopóki *j* > 0 **wykonuj**
            wypisz *W*[*j*]
            j ← *j* – *m*

Algorytm 2:

```
n ← dlugosc(W)
m ← n div k
jeżeli n mod k≠0
        m←m+1
dla i=1,2,…,m wykonuj
        j←i
        jeżeli (i mod 2=1)
                dopóki j ≤ n wykonuj
                        wypisz W[j]
                        j ← j+m
        w przeciwnym razie
                dopóki j ≤ n wykonuj
                        j ← j+m
                j ← j – m
                dopóki j > 0 wykonuj
                        wypisz W[j]
                        j ← j – m
```

Zadanie 32.

32.1.

Tekst skompresowany	**Tekst oryginalny**
a(cd)a	acdcda
(pur)owy	purpurowy
(z)(zz)	zzzzzz
(ab)a(abcd)	abaabcdabcd

32.2.

Przykładowy algorytm:

```
mozliwe ← fałsz
jeśli n mod 2 = 0 wykonaj
        mozliwe ← prawda
        dla i = 1, 2, ..., n/2
                jeśli napis[i] ≠ napis[i+n/2]
                        mozliwe ← fałsz
jeśli mozliwe wykonaj
        wypisz '('
        dla i = 1, 2, ..., n/2
                wypisz napis[i]
        wypisz ')'
```

32.3.

Przykładowy algorytm:

```
p ← 1
dopóki p ≤ n wykonuj
        jeśli napis[p] jest literą
                wypisz napis[p]
                p ← p+1
        jeśli napis[p] = '('
                k ← p+1
                dopóki napis[k] ≠ ')'
                        k ← k+1
                powtórz 2 razy:
                        dla j = p+1, ..., k-1
                                wypisz napis[j]
                p ← k+1
```

Zadanie 33.

33.1.

Tablica B:

	1	2	3	4	5
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	1	0	0
3	1	0	0	1	0
4	0	0	0	0	1
5	0	0	0	0	0
6	0	0	0	0	0

Wartość zwracana przez algorytm: **1**

33.2.

Specyfikacja

Dane: n — liczba naturalna większa od 1
A — plansza rozmiaru $n \times n$ wypełniona liczbami całkowitymi.

Wynik:

- **1** — jeśli istnieje trasa wędrowca typu **chodzącego** i prowadząca tylko przez pola o wartościach **dodatnich**
- **0** — w przeciwnym przypadku

33.3.

Przykładowe rozwiązanie

Algorytm:

```
suma ← 0
dla j=1,2, …,n wykonuj
        mx ← A[1,j]
        dla i=2,3…,n wykonuj
                jeżeli A[i,j] >mx
                        mx ← A[i,j]
        suma ← suma + mx
zwróć suma
```

33.4.

Przykładowe rozwiązanie

Algorytm:

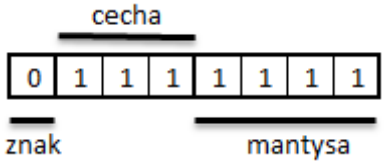
```
w←1
dla j=1,2, …,n wykonuj
        dopóki w<=n oraz A[w,j]<0
                w←w+1
        jeżeli w>n
                zwróć 0
zwróć 1
```

Zadanie 35.

35a

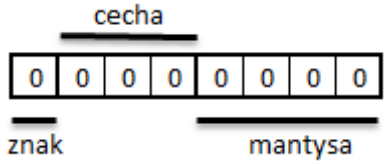
FPFP

35b



$C = 111_2 - 4 = 7 - 4 = 3$
$M = 1.1111_2 = 1 + 0.5 + 0.25 + 0.125 + 0.0625 = 1.9375$
$x = 1.9375*2^3 =$ **15.5**

35c



$C = 0_2 - 4 = -4$
$M = 1.0$
$x = 1.0*2^{-4} = 1/16 = 0.0625$

Zadanie 36.

NIE TAK TAK NIE NIE NIE

Zadanie 37.

NIE TAK NIE TAK NIE

Zadanie 38.

NIE TAK NIE NIE

Zadanie 39.

39.1.

Przykładowa odpowiedź:

Powinien zostać dodany obraz `aquapark2.jpg`, ponieważ rozmiar tego jest mniejszy. Dzięki temu obraz szybciej się załaduje.

39.2.

Przykładowa odpowiedź:

Najlepszą jakość druku zapewni format `TIFF`. Obrazy są przechowywane bez utraty jakości. Format ten używany jest standardowo przez firmy DTP przygotowujące materiały do druku.

39.3.

Poprawna odpowiedź: 2304 kB

Odpowiedź wynikająca z obliczenia jedynie liczby bitów: 18874368 bitów lub bajtów: 2359296 bajtów

Zadanie 40.

FPFP

Zadanie 41.

PFFF

Zadanie 42.

42.1.

Tablica T	**Liczba operacji porównania wykonanych w wierszu oznaczonym (*)**	**Liczba operacji porównania wykonanych w wierszu oznaczonym (**)**
4, 2, 5, 8, 1, 9, 7, 6, 3	7	4
5, 4, 3, 2, 1, 6, 7, 8, 9, 10	7	5
1, 2, 3, ... , 100	100	1
100, 99, 98, ... , 1	2	100

42.2.

Tablica T	Liczba operacji zamiany wykonanych w wierszu oznaczonym (***)
4, 2, 5, 8, 1, 9, 7, 6, 3	2
5, 4, 3, 2, 1, 6, 7, 8, 9, 10	1
1, 2, 3, ... , 100	0
100, 99, 98, ... , 1	1

Zadanie 43.

43.1.

Przykładowa odpowiedź:

- Wyszukiwanie optymalnej trasy samochodowej

  Uzasadnienie: Dzięki lokalizacji znane jest położenie użytkownika co daje możliwość podawania wskazówek odnośnie wyboru drogi, dopasowanych do jego położenia oraz zmiany trasy, w przypadku informacji o korkach. Dane o lokalizacji użytkowników pozwalają oszacować natężenie ruchu na różnych odcinkach dróg

- Serwis pogodowy

  Uzasadnienie: Dzięki usłudze lokalizacji użytkownicy mogą uzyskiwać dane pogodowe i prognozę pogody dla ich aktualnego położenia bez konieczności podawania lokalizacji samemu

- Serwis informacji turystycznej Wrocławia

  Uzasadnienie: Wykorzystując dane o lokalizacji użytkownika, serwis może podać, jakie obiekty znajdują się najbliżej jego aktualnego położenia, a także wskazać trasę od miejsca, w którym użytkowniksię aktualnie znajduje, do wybranego obiektu

43.2.

Przykładowa odpowiedź:

- System komputerowego składu tekstu
  Uzasadnienie: Edycja wielu elementów (tekst + grafika) wymaga wprowadzania wielu danych (wpisywanie tekstu, wstawianie i tworzenie obiektów graficznych), a do tego celu przydatna jest ergonomiczna klawiatura, mysz itp. Wizualizacja w skali 1:1 wymaga odpowiednio dużego ekranu.
- Edytor tekstu
  Uzasadnienie: Dużą ilość tekstu dużo wygodniej wprowadza się, korzystając z dużej, ergonomicznej klawiatury. Ponadto duży ekran umożliwia wizualizację wynikowego dokumentu w rzeczywistych wymiarach.
- Arkusz kalkulacyjny
  Uzasadnienie: Dane (zestawienia liczbowe, teksty) i formuły wygodniej wprowadza się, korzystając z dużej, ergonomicznej klawiatury. Dotyczy to również tworzenia i przeglądania wykresów.

43.3.

FPFP

43.4.

FPPFP

Zadanie 45.

PFPF

Zadanie 46.

PFFP

Zadanie 47.

FPFP

Zadanie 48.

PFFF

Zadanie 49.

PPFP

Zadanie 50.

FPFP

Zadanie 51.

Prawidłowa odpowiedź:

	atak *brute force*	**metoda słownikowa/psychologiczna**
Użytkownik A	x	x
Użytkownik B		x
Użytkownik C		
Użytkownik D		x

Zadanie 52.

PPFF

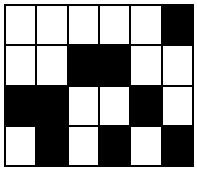
Zadanie 53.

53.1.

Stan zegara na zdjęciu nr 1: 9:20:15 (09:20:15).

Stan zegara na zdjęciu nr 4: 23:10:36.

53.2.



Zadanie 54.

PFFP

Zadanie 55.

55.1.

FPFF

55.2.

FPPF

Zadanie 56.

PFPF

Zadanie 57.

PPFP

Zadanie 59.

59.1.

114

59.2.

181

59.3.

moc	ile liczb
1	381
2	344
3	171
4	76
5	22
6	4
7	0
8	2

Minimalna liczba o mocy 1 jest równa 11.

Maksymalna liczba o mocy 1 jest równa 999342708.

Zadanie 60.

60.1.

Przykładowe rozwiązanie znajduje się w pliku `rozw1.cpp`**.** Prawidłowa odpowiedź:

```
Mniejszych od 1000: 12
Dwie ostatnie: 540 633
```

60.2.

Przykładowe rozwiązanie znajduje się w pliku `rozw2.cpp`. Prawidłowa odpowiedź*:*

```
989532
1 2 3 4 6 9 12 18 36 27487 54974 82461 109948 164922 247383
329844 494766 989532

131072
1 2 4 8 16 32 64 128 256 512 1024 2048 4096 8192 16384 32768
65536 131072

702027
1 3 9 27 81 107 243 321 729 963 2187 2889 6561 8667 26001
78003 234009 702027

461817
1 3 9 23 69 97 207 291 529 873 1587 2231 4761 6693 20079 51313
153939 461817

17298
1 2 3 6 9 18 31 62 93 186 279 558 961 1922 2883 5766 8649
17298
```

60.3.

Przykładowe rozwiązanie znajduje się w pliku `rozw3.cpp`. Prawidłowa odpowiedź*:*

```
Najwieksza wzglednie pierwsza: 988027
```

Zadanie 61.

61.1.

Przykładowe rozwiązanie znajduje się w pliku `rozw1.cpp`. Prawidłowa odpowiedź:

```
Ciagow arytmetycznych: 44
Najwieksza roznica: 246849
```

61.2.

Przykładowe rozwiązanie znajduje się w pliku `rozw2.cpp`. Prawidłowa odpowiedź:

```
1
27
551368
1000000
250047
1000
35937
531441
```

61.3.

Przykładowe rozwiązanie znajduje się w pliku `rozw3.cpp`. Prawidłowa odpowiedź*:*

```
6
36
421503
362326
303276
534127
40828
949
90367
177326
390126
842436
37889
11106
149243
80882
529815
988401
29559
953023
```

Zadanie 62.

62.1.

Najmniejsza liczba to **1002**, największa — **777044**.

62.2.

Pierwszym elementem tego ciągu jest **639** i ciąg składa się z **6** elementów.

62.3.

- liczb o tych samych wartościach jest **160**;
- liczba z pierwszego pliku jest większa od liczby z drugiego pliku w **357** wierszach.

62.4.

625 razy w zapisie dziesiętnym, **411** razy w zapisie ósemkowym.

Zadanie 63.

63.1.

Poprawna odpowiedź (18 ciągów):

11
101000101000
10110001011000
1010
110110110110
100100
11001100
111111
10111011
101101
100100100100
110110
1100110011001100
1111
10011001
1001110011
1100111001
110110110110110110

63.2.

93

63.3.

259 ciągów jest binarną reprezentacją liczb półpierwszych.

Najmniejszą liczbą półpierwszą jest 6.

Największa z nich jest równa 248667.

Odpowiedź błędnie podająca liczbę liczb półpierwszych, wynikająca z zastosowania algorytmu sita Eratostenesa, który nie uwzględnia tego, że 0 oraz 1 nie są liczbami pierwszymi: 436

Odpowiedź błędnie podająca liczbę liczb półpierwszych, która wynika z uwzględnienia dwóch różnych czynników pierwszych podczas sprawdzania półpierwszości: 252

Odpowiedź błędnie podająca liczbę liczb półpierwszych, która wynika ze zliczenia również liczb pierwszych, które mają tylko jeden czynnik pierwszy: 418

Zadanie 64.

64.1.

Poprawna odpowiedź:

Liczba rewersów: 13

Największa liczba pikseli czarnych: 381

Niepoprawna odpowiedź wynikająca z przyjęcia, że rewersem jest również obrazek z równą liczbą pikseli czarnych i białych:

Liczba rewersów: 37

Największa liczba pikseli czarnych: 381

64.2.

Poprawna odpowiedź:

Liczba obrazków rekurencyjnych: 60

Pierwszy obrazek rekurencyjny:

```
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
11111111111111111111
00000000000000000000
```

Niepoprawna odpowiedź wynikająca ze sprawdzania sąsiednich podobrazków parami tylko poziomo (inny obrazek będzie też wówczas pierwszym spełniającym sprawdzane warunki): 95.

Niepoprawna odpowiedź wynikająca ze sprawdzania sąsiednich podobrazków parami tylko pionowo (inny obrazek będzie też wówczas pierwszym spełniającym sprawdzane warunki): 101.

64.3.

Poprawna odpowiedź:

Liczba obrazków poprawnych: 171

Liczba obrazków naprawialnych: 14

Liczba obrazków nienaprawialnych: 15

Największa liczba błędnych parzystości: 7

Niepoprawna odpowiedź wynikająca z błędnie przyjętego założenia, że obrazek z dwoma błędami jest naprawialny, nawet gdy błędy występują w tym samym wierszu lub tej samej kolumnie:

Liczba obrazków poprawnych: 171

Liczba obrazków naprawialnych: 18

Liczba obrazków nienaprawialnych: 11

Największa liczba błędnych bitów parzystości: 7

64.4.

Poprawna odpowiedź:

Obrazki naprawialne (numer obrazka, wiersz, kolumna):

```
(14,1,15)
(19,4,20)
(26,21,13)
(29,2,8)
(33,15,21)
(115,21,14)
(116,21,13)
(129,21,14)
(131,10,7)
(143,7,15)
(154,7,7)
(161,21,17)
(162,21,16)
(187,21,18)
```

Niepoprawna odpowiedź wynikająca z błędnie przyjętego założenia, że obrazek z dwoma błędami jest naprawialny, nawet gdy błędy występują w tym samym wierszu lub tej samej kolumnie

```
(11,5,21) lub (11,9,21)
(14,1,15)
(19,4,20)
(26,21,13)
(29,2,8)
(33,15,21)
```

```
(53,6,21) lub (53,15,21)
(115,21,14)
(116,21,13)
(129,21,14)
(131,10,7)
(143,7,15)
(146,21,19) lub (146,21,4)
(154,7,7)
(161,21,17)
(162,21,16)
(170,21,19) lub (170,21,12)
(187,21,18)
```

Uwaga: W podanych powyżej wierszach z dwoma odpowiedziami przyjmuje się, że w odpowiedzi maturzysty została podana jedna z nich — pozwalająca skorygować jeden z dwóch błędów parzystości.

Zadanie 65.

65.1.

Poprawna odpowiedź: 1 225

Odpowiedzi błędne: 2 450, 7 1575. Podają one ułamek o najmniejszej wartości, ale nie spełniają drugiego warunku zadania, tzn. w pliku występuje inny ułamek o tej samej wartości i **mniejszym** mianowniku.

65.2.

Poprawna odpowiedź: 410.

Odpowiedzi błędne:

- 409 lub 411 — odpowiedzi wynikające z błędnego zainicjowania licznika.
- 459 — odpowiedź wynikająca z policzenia jako nieskracalnych wszystkich ułamków, w których licznik jest dzielnikiem mianownika (lub mianownik jest dzielnikiem licznika).

65.3.

Poprawna odpowiedź: 128446.

Odpowiedź z błędem polegającym na pominięciu pierwszego lub ostatniego ułamka: 128445 lub 128399.

65.4.

Poprawna odpowiedź: 578219135.

Odpowiedź błędna: 564486930. Błąd w niej wynika z zastosowania typu danych o zbyt małym zakresie (gdy w kompilatorze gcc zastosuje się typ long zamiast long long do reprezentacji wyników pośrednich).

Zadanie 66.

66.1.

Poprawna odpowiedź:

25320 29269 40
24810 10353 27
15276 20113 28
13867 24491 45
19677 6037 46
19688 2020 36

66.2.

Poprawna odpowiedź:

6131 20807 127567717
26297 6329 166433713
24767 809 20036503
28477 19289 549292853
24799 6359 157696841
691 11003 7603073
9631 28351 273048481
6661 26393 175803773
5881 28429 167190949
18587 21739 404062793

Błędna odpowiedź (podająca 12 wierszy) wynikająca z zastosowania ostrej nierówności w warunku zakończenia pętli przy sprawdzaniu, czy liczba jest pierwsza:

6131 20807 127567717
26297 6329 166433713
24767 809 20036503
28477 19289 549292853
24799 6359 157696841
691 11003 7603073
9631 28351 273048481
23 529 12167
89 7921 704969
6661 26393 175803773
5881 28429 167190949
18587 21739 404062793

Błędna odpowiedź (podająca 14 wierszy) wynikająca ze sprawdzenia tylko tego, czy trzecia liczba jest iloczynem dwóch pierwszych liczb:

6131 20807 127567717
26297 6329 166433713
24767 809 20036503
2572 29539 75974308
28477 19289 549292853
24799 6359 157696841
691 11003 7603073
9631 28351 273048481

6705 1606 10768230
23 529 12167
89 7921 704969
6661 26393 175803773
5881 28429 167190949
18587 21739 404062793

66.3.

Poprawna odpowiedź:
58140 58141 341
343 58824 58825
681 231880 231881
683 233244 233245
1021 521220 521221
1023 523264 523265
926160 1361 926161
928885 1363 928884
1701 1446700 1446701
1703 1450104 1450105

Błędna odpowiedź wynikająca z wypisania wszystkich (nie tylko sąsiadujących) wierszy, w których liczby reprezentują długości boków trójkąta prostokątnego, nie tylko sąsiadujące pary:

3 4 5
58140 58141 341
343 58824 58825
681 231880 231881
683 233244 233245
1021 521220 521221
1023 523264 523265
926160 1361 926161
928885 1363 928884
1701 1446700 1446701
1703 1450104 1450105

Błędna odpowiedź wynikającą z przyjęcia założenia, że liczby w wierszu są uporządkowane rosnąco:

681 231880 231881
683 233244 233245

1021 521220 521221
1023 523264 523265
1701 1446700 1446701
1703 1450104 1450105

66.4.

Liczba wierszy, w których liczby reprezentują długości boków trójkąta: 604

Długość najdłuższego ciągu trójkątnego: 11

Zadanie 67.

67.1.

10. 55
20. 6765
30. 832040
40. 102334155

67.2.

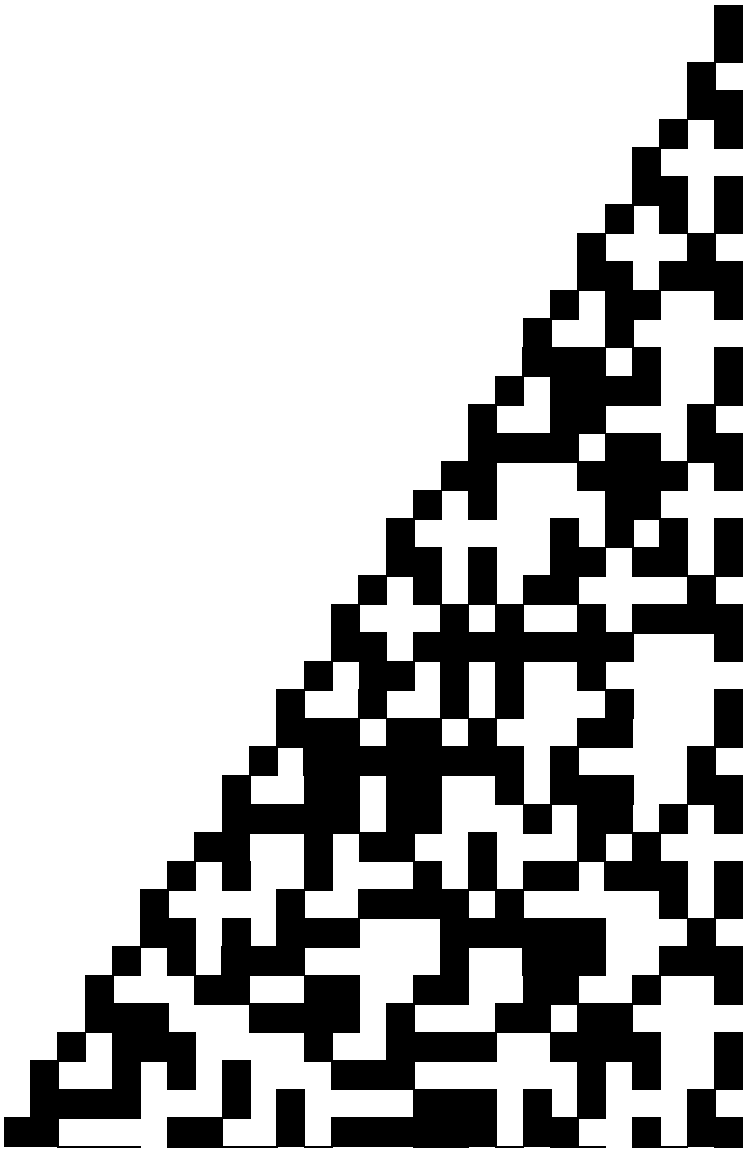
3. 2
4. 3
5. 5
7. 13
11. 89
13. 233
17. 1597
23. 28657
29. 514229

67.3.

1
1
10
11
101
1000
1101
10101
100010
110111
1011001
10010000
11101001
101111001
1001100010
1111011011
11000111101
101000011000
1000001010101
1101001101101
10101011000010
100010100101111
110111111110001
1011010100100000
10010010100010001
11101101000110001
101111111101000010
1001101100101110011
1111101100010110101

1100101100100010100 0
1010010000101011011101
1000010011110100000101
1101011100011111100010
1010111000001001110011 1
1000110011001100110010 01
1110001111010001101100 00
1011100001001111001111001
1001010100011100000010 1001
1111000101000011101010 0010
1100001100101111111011 001011

fraktal binarny Fibonacciego:



67.4.

101111001
10101011000010
1011010100100000
10010010100010001

Zadanie 68.

68.1.

Poprawna odpowiedź: 9.

Odpowiedź błędna: 14. Uwzględnia ona wiersze zawierające napisy różnej długości, ale składające się z tych samych liter.

68.2.

Poprawna odpowiedź: 93

Odpowiedź błędna: 101. Uwzględnia ona wiersze zawierające napisy złożone z tych samych liter, ale o różnej długości.

68.3.

Poprawna odpowiedź: 17

Odpowiedź błędna: 59. Wynika ona z policzenia napisów, które zawierają te same litery, ale o różnej liczbie wystąpień w poszczególnych napisach.

Zadanie 69.

69.1

Liczba gatunków:206

Rozmiar największego gatunku: 20

69.2

8

69.3

Największa liczba genów w genotypie: 11

Najdłuższy gen: 189

69.4

Liczba genotypów odpornych: 249

Liczba genotypów silnie odpornych: 187

Zadanie 70.

70.1.

Pozostała powierzchnia materiału: 117.705

70.2.

Pani Binarna musi kupić 126 m taśmy.

Odpowiedź błędna, wynikająca z błędnego zaokrąglenia: 125.

Odpowiedź błędna, wynikająca z braku dodania odcinków poziomych i pionowych do długości krzywej: 57.

70.3.

Suma długości pasów jest równa 429 m.

Odpowiedź błędna: 445 m. Wynika ona ze złego obliczenia całkowitej długości każdego pasa (niezaokrąglenia w dół).

Odpowiedź błędna: 447 m. Wynika ona ze złego obliczenia długości pasa — podzielenia jednego pasa na 2 części (odpowiednio dla funkcji $f(x)$ oraz $g(x)$) oraz błędnego obliczenia całkowitej długości każdego z dwóch pasów (brak zaokrąglania w dół).

Zadanie 71.

71.1.

2.09032

71.2.

Dla dwu wartości: $x = 4.092$, $x = 4.093$ zachodzi przybliżona równość $f(x) \approx 3.06495$.

71.3.

Miejsce zerowe : 0.53656

Miejsce zerowe : 2.52073

Miejsce zerowe : 3.31769

Miejsce zerowe : 4.84971

Zadanie 72.

72.1.

Przykładowe rozwiązanie znajduje się w pliku `rozw1.cpp`. Prawidłowa odpowiedź:

```
Pierwsza para:
feycznvjtboxfx gthzgbpglrwjqxqlculgwzdhuevrmvssozrwdjcqcb
Par znalezionych:
70
```

72.2.

Przykładowe rozwiązanie znajduje się w pliku `rozw2.cpp`. Prawidłowa odpowiedź:

```
kxazlp kxazlpe e
ifvjhuqvh ifvjhuqvhcupzcpw cupzcpw
aznqxr aznqxryrbgshtceaylwak yrbgshtceaylwak
```

72.3.

Przykładowe rozwiązanie znajduje się w pliku `rozw3.cpp`. Prawidłowa odpowiedź:

```
Maksymalna koncowka: 15
zccvywdcmjrdokqzcnayixplhkrf sarkqzcnayixplhkrf
iokvlepqzeyvycfjkliiutmzqawwjxgf dfcfyddwodduznkmivqxnr-
dliiutmzqawwjxgf
psmwjyystgwchofokzvmkmgusfakroambngky mabgusfakroambngky
```

Zadanie 73.

73.1.

Przykładowe rozwiązanie znajduje się w pliku `rozw1.cpp`. Prawidłowa odpowiedź:

Slow z dwoma kolejnymi literami: 204

73.2.

Przykładowe rozwiązanie znajduje się w pliku `rozw2.cpp`. Prawidłowa odpowiedź:

```
A: 632 (7.56%)
B: 196 (2.34%)
C: 162 (1.94%)
D: 422 (5.05%)
E: 1092 (13.06%)
F: 213 (2.55%)
G: 151 (1.81%)
H: 565 (6.76%)
I: 521 (6.23%)
J: 2 (0.02%)
K: 64 (0.77%)
L: 402 (4.81%)
M: 193 (2.31%)
N: 557 (6.66%)
O: 641 (7.66%)
P: 93 (1.11%)
Q: 6 (0.07%)
R: 523 (6.25%)
S: 484 (5.79%)
T: 792 (9.47%)
U: 185 (2.21%)
V: 84 (1.00%)
W: 196 (2.34%)
X: 3 (0.04%)
Y: 185 (2.21%)
Z: 0 (0.00%)
```

73.3.

Przykładowe rozwiązanie znajduje się w pliku `rozw3.cpp`. Prawidłowa odpowiedź:

```
Najdluzszy ciag spolglosek: 4
Znalezionych slow: 6
```

```
Pierwsze z nich: FRIENDSHIP
```

Zadanie 74.

74.1.

Prawidłowa odpowiedź:

16

Błędne odpowiedzi:

84 — liczba haseł złożonych tylko z liter (zapomniano zanegować sprawdzany warunek)

9 — rozwiązanie z warunkiem '0' <= c && c < '9'

3 — rozwiązanie z warunkiem '0' < c && c <= '9'

1 — rozwiązanie z warunkiem '0' < c && c < '9'

74.2.

```
8Y7JGYXXR5
Ehz018657
PAsCMQaervw
cefdi
cek
ikfLDegQXj
jir
yvm249t83o04
```

74.3.

Prawidłowa odpowiedź:

39

Błędne odpowiedzi:

30 — rozwiązanie, które rozważa tylko hasła co najmniej 5-literowe

2 — rozwiązanie, które tylko sprawdza, czy cztery kolejne znaki ASCII występują na kolejnych pozycjach, np. ABCD

47 lub 38 — rozwiązanie, które szuka 4 kolejnych znaków ASCII w całym haśle, a nie we fragmencie 4-literowym.

74.4.

Prawidłowa odpowiedź:

40

Błędne odpowiedzi:

73 — rozwiązanie, które nie sprawdza, czy w haśle występuje cyfra

72 — rozwiązanie, które nie sprawdza, czy w haśle występuje mała litera

68 — rozwiązanie, które nie sprawdza, czy w haśle występuje duża litera

Zadanie 75.

75.1.

Przykładowe rozwiązanie znajduje się w pliku *rozw1.cpp*. Prawidłowa odpowiedź:

```
did
desired
destroyed
devised
```

75.2.

Przykładowe rozwiązanie znajduje się w pliku `rozw2.cpp`. Prawidłowa odpowiedź:

```
wkqoocjqwo
iujolqzzwr
udwjtljuip
tlcpgujurjqk
yplcjdwotwr
kqrrfwwcjtl
mupmwjpqpg
kqrrfwwcjtl
wotchfqolwr
```

75.3.

Przykładowe rozwiązanie znajduje się w pliku `rozw3.cpp`. Prawidłowa odpowiedź:

```
Klucz szyfrujacy numer 1: (3,7)
Klucz deszyfrujacy numer 1: (9,15)
Klucz szyfrujacy numer 2: (1,3)
Klucz deszyfrujacy numer 2: (1,23)
Klucz szyfrujacy numer 3: (17,14)
Klucz deszyfrujacy numer 3: (23,16)
Klucz szyfrujacy numer 4: (7,0)
Klucz deszyfrujacy numer 4: (15,0)
Klucz szyfrujacy numer 5: (25,13)
Klucz deszyfrujacy numer 5: (25,13)
```

Zadanie 76.

76.1.

```
ramnfrayooyauymtkgsrrnyyzianmaooldlctatsocthurtzuk
slsWdrnazkryjiAeaseeznktiAnoejrdraikacFwNmKtkeeoyK
jltDereaaonaLaAiiknBimeurAalaAliKrzrasgDaaaicaoaot
zawSKndGoOLaozWrorlcaalnnwnpkorocaWcPzwdieazszaesk
btsdogdbeKoulrSanPotkMenmlaiwasjzMnyryLuywrtmirRBe
ktjwaoSkKoctcoSrasioarlrluewblTnfzoGetiaSkeruDKapi
```

76.2.

```
cosnyzcahnzdoolynturmoifaraatmkaalgorytmystruktury
```

76.3.

csonozolzhcynadyrutkurtsymtyroglaakytamrofniarutam

Zadanie 77.

77.1.

19 powtórzeń klucza

EI XILLG AWLMA HCFLLGGKG LKCO JSHKAJNBSIOJ, JPVUCYZX CEUWC OQQ UKDKR SMLX QZLWEGMWZKXS, UW ZGG VGXMA RXAQQ HGRRXSOJ. U KMEJK MGX WKPGZ OPXKD HXSV-PMNG, FM PHC PIGXS EWWYF YKBETKUNG. M HGRJB KVZM OQQ UKD BHKCO TNQQPBZ, RH PTLQEWBMFNZGIG KGJMDXC V XEYO VCFZGLMT. RXRTJ JSIFAJDRM.

77.2.

W RZECZYWISTOSCI PRZESZLOSC NIE ISTNIEJE, JEDYNIE WSPOMNIENIE PRZESZLOSCI. MRUGNIJCIE OCZAMI, A SWIAT KTORY ZOBACZYCIE NIE ISTNIAL KIEDY ZAMYKALISCIE POWIEKI. JEDYNYM WLASCIWYM STANEM UMYSLU JEST ZASKOCZENIE. JEDYNYM WLASCI-WYM STANEM SERCA JEST RADOSC. NIEBO KTORE WIDZICIE NIE BYLO JESZCZE NIGDY PRZEZ WAS OGLADANE. MOMENT ABSOLUTNEJ PERFEKCJI JEST TERAZ. CIESZCIE SIE NIM. TERRY PRATCHETT.

77.3.

Liczniki wystąpień liter:

A - 8	H - 21	O - 10	V - 11
B - 19	I - 12	P - 10	W - 21
C - 13	J - 13	Q - 8	X - 12
D - 21	K - 6	R - 11	Y - 10
E - 19	L - 14	S - 12	Z - 7
F - 4	M - 16	T - 17	
G - 8	N - 15	U - 13	

12.52 — szacowana dlugosc klucza

13 — dokladna dlugosc klucza

Zadanie 78.

78.1.

Dlugosc pierwszej wiadomosci = 232

Bajty pierwszej wiadomosci po przetworzeniu: 12 94 94 57 96 115 51 90

Skrot pierwszej wiadomosci = MQQFSLZM

78.2.

```
1     MQQFSLZM
2     WWXRFKEC
3     UAQFAKQN
4     WVTFWMLI
5     WMXQBVTU
6     IRCEIIDN
7     DLGULFST
8     UIZVEUSE
9     DMWHUXTH
10    UFHRHSXM
11    TFUKGFXW
```

78.3.

```
1 3 4 8 9 11
```

Zadanie 79.

79.1.

Ćwiartka I: 1339,

Ćwiartka II: 274,

Ćwiartka III: 166,

Ćwiartka IV: 218,

Pozostałe okręgi: 3.

79.2.

Liczba lustrzanych par: 158.

79.3.

Liczba par prostopadłych : 231.

79.4.

Długości kolejnych łańcuchów: 6, 3, 11, 7, 4, 5, 12, 36, 18, 53, 87, 21, 73, 39, 31, 12, 41, 10, 64, 22, 57, 31, 41, 50, 65, 73, 45, 18, 37, 28

Najdłuższy łańcuch ma długość 87.

Zadanie 80.

80.1.

Poprawna odpowiedź: 4, 3, 5; 13, 85, 84; 33, 65, 56; 28, 45, 53.

Odpowiedź z błędem, który wynika z pominięcia jednego z trójkątów albo z podania tylko trójkątów, których boki podane są w takiej kolejności, że na końcu występuje bok najdłuższy: 4, 3, 5; 28, 45, 53.

80.2.

Poprawna odpowiedź: 29694.

Odpowiedź z błędem, który wynika z pominięcia jednego z trójkątów albo z podania wyniku dla błędnej nierówności trójkąta, z błędem polegającym na zamianie nierówności ostrej na nieostrą (czyli za długości boków trójkąta uznawane są liczby a, b, c, takie że a+b=c). Odpowiedź: 285412.

80.3.

Poprawna odpowiedź: 2343790

Odpowiedź z błędem, który wynika z liczenia każdego trójkąta sześciokrotnie, z uwzględnieniem różnej kolejności długości krawędzi (np. (1, 3, 5), (1, 5, 3), (3, 1, 5), (3, 5, 1), (5, 1, 3) i (5, 3, 1)). Odpowiedź: 6·2343790=14062740

Odpowiedź z błędem, w którym podany jest wynik dla błędnej nierówności trójkąta, z błędem polegającym na zamianie nierówności ostrej na nieostrą (czyli za długości boków trójkąta uznane są liczby a, b, c, takie że a+b=c). Odpowiedź: 2354017.

Zadanie 81.

81.1.

2

81.2.

24

81.3.

A(-30;23), B(75, -62), C(25, 48), obwód: 316,34

81.4.

18

81.5.

Są cztery takie czworokąty:

-18,-23 15,1 -49,-38 -48,-48

17,-19 -36,6 -37,-28 1,1

-28,-35 -16,39 -36,-10 -38,-38

-14,49 27,39 48,-3 7,7

Zadanie 82.

82.1.

Prawidłowa odpowiedź:

populacja wilków: 43,46, populacja zajęcy: 118,06.

Błędna odpowiedź (wynik wzięty o jeden miesiąc za wcześnie/późno):

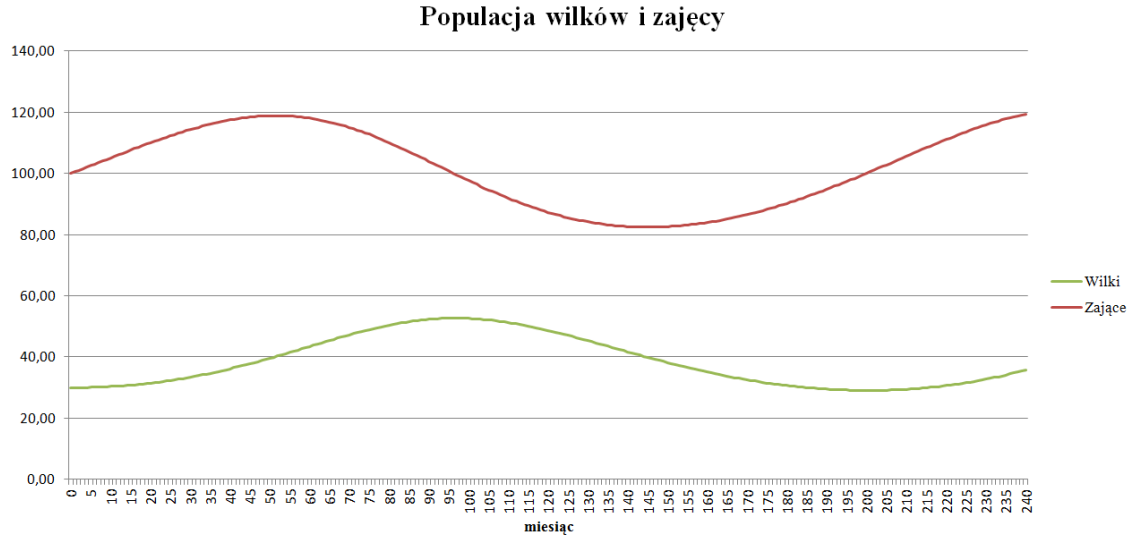
populacja wilków: 43,07, populacja zajęcy: 118,24

populacja wilków: 43,85, populacja zajęcy: 117,86

82.2.

n=53, m=98.

82.3.



82.4.

Prawidłowa odpowiedź

	Zające	Wilki
Najmniejsza wartość populacji	80,71	28,05
Największa wartość populacji	123,32	54,15

Błędna odpowiedź (rozwiązanie, w którym zaokrągla się wyniki pośrednie):

	Zające	Wilki
Najmniejsza wartość populacji	80,65	28,00
Największa wartość populacji	123,39	54,21

Zadanie 84.

84.1.

43 razy tankował Pb95.

78 razy tankował LPG

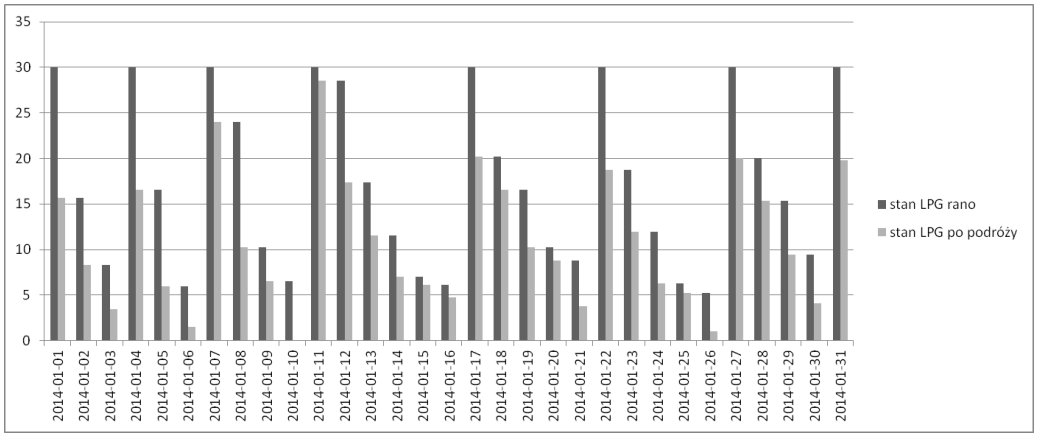
200 dni jechał, korzystając tylko z LPG.

84.2.

26.01.2014 r.

84.3.

Przykładowy poprawny wykres:



84.4.

Wydatki na Pb95 i LPG (wraz z kosztem instalacji): 8373,06

Wydatki na Pb95: 8965,23

Zadanie 85.

85.1.

41 406 l

85.2.

8 360 sztuk

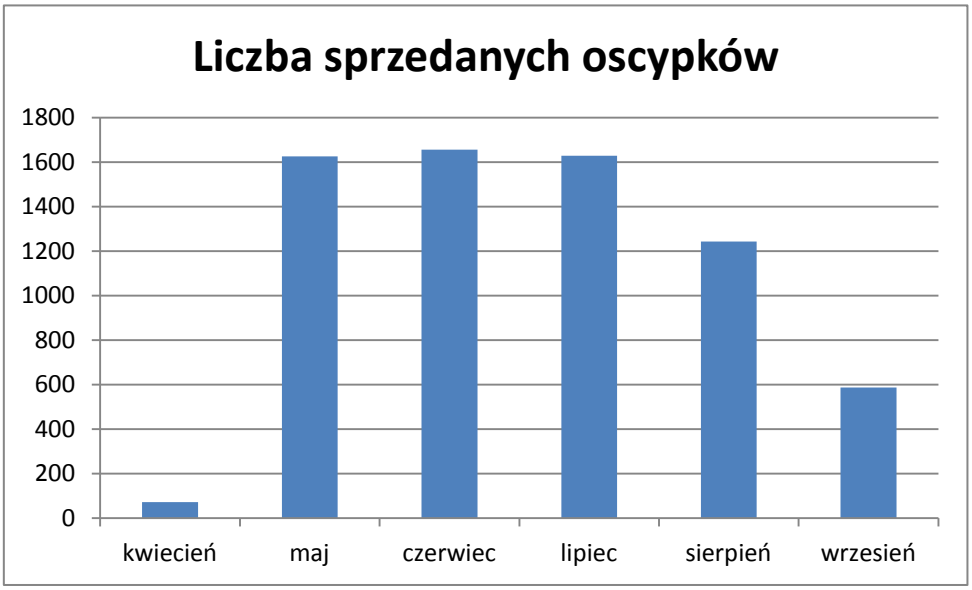
85.3.

2014-05-04

50 dni

85.4.

miesiąc	liczba sprzedanych oscypków
kwiecień	72
maj	1626
czerwiec	1656
lipiec	1628
sierpień	1243
wrzesień	586



85.5.

744

85.6.

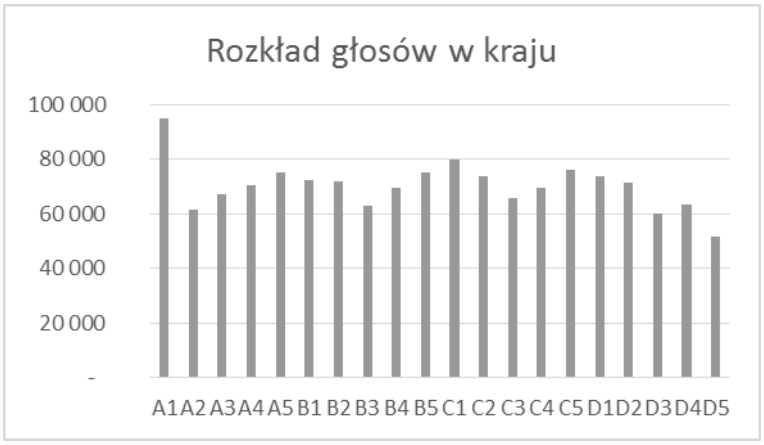
skład mleka	liczba oscypków
tylko mleko owcze	6811
20% mleka krowiego	8470
40% mleka krowiego	11298

Zadanie 86.

86.1.

A1: 94989	B1: 72187	C1: 79735	D1: 73580
A2: 61487	B2: 71950	C2: 73675	D2: 71402
A3: 67178	B3: 62913	C3: 65751	D3: 60146
A4: 70318	B4: 69326	C4: 69332	D4: 63234
A5: 74985	B5: 75045	C5: 75876	D5: 51362

Przykładowy wykres:



86.2.

K1: A2, K2: D5, K3: A5, K4: B4, K5: D5.

86.3.

K1: 8, K2: 4, K3: 7, K4: 8, K5: 3.

86.4.

Poprawna odpowiedź:

Wariant standardowy: K1: 94, K2: 57, K3: 82, K4: 126, K5: 41

Przykładowe błędne odpowiedzi wynikające z zaokrąglania współczynnika w_K do liczby całkowitej: K1: 95, K2: 57, K3: 82, K4: 125, K5: 41

Wariant regionalny: K1: 97, K2: 56, K3: 82, K4: 127, K5: 38

Przykładowe błędne odpowiedzi wynikające z zaokrąglania współczynnika w_K do liczby całkowitej: K1: 98, K2: 56, K3: 82, K4: 126, K5: 38

86.5.

Dla m=10: 47500

Dla m=20: 48750

Dla m=50: 49500

Zadanie 87.

87.1.

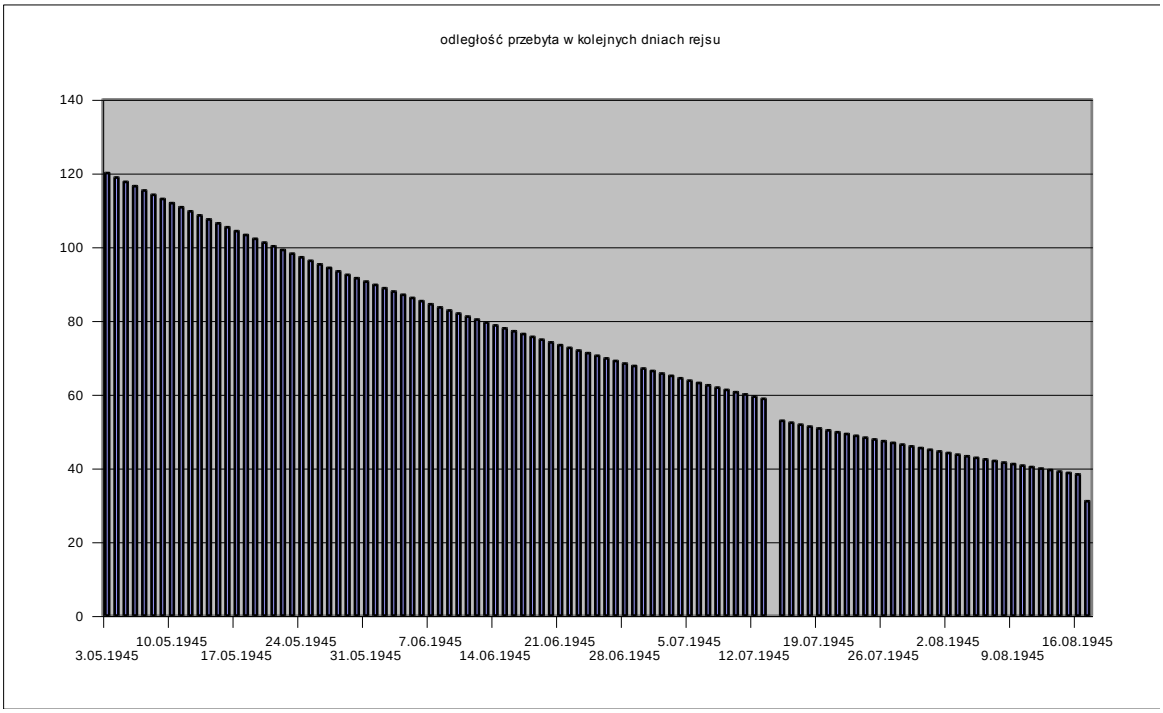
1 czerwca 1945 / 30. dnia podróży.

87.2.

7701,9.

87.3.

Przykład poprawnej odpowiedzi:



87.4.

14 sierpnia 1945 / 104. dnia podróży.

87.5.

9811,7.

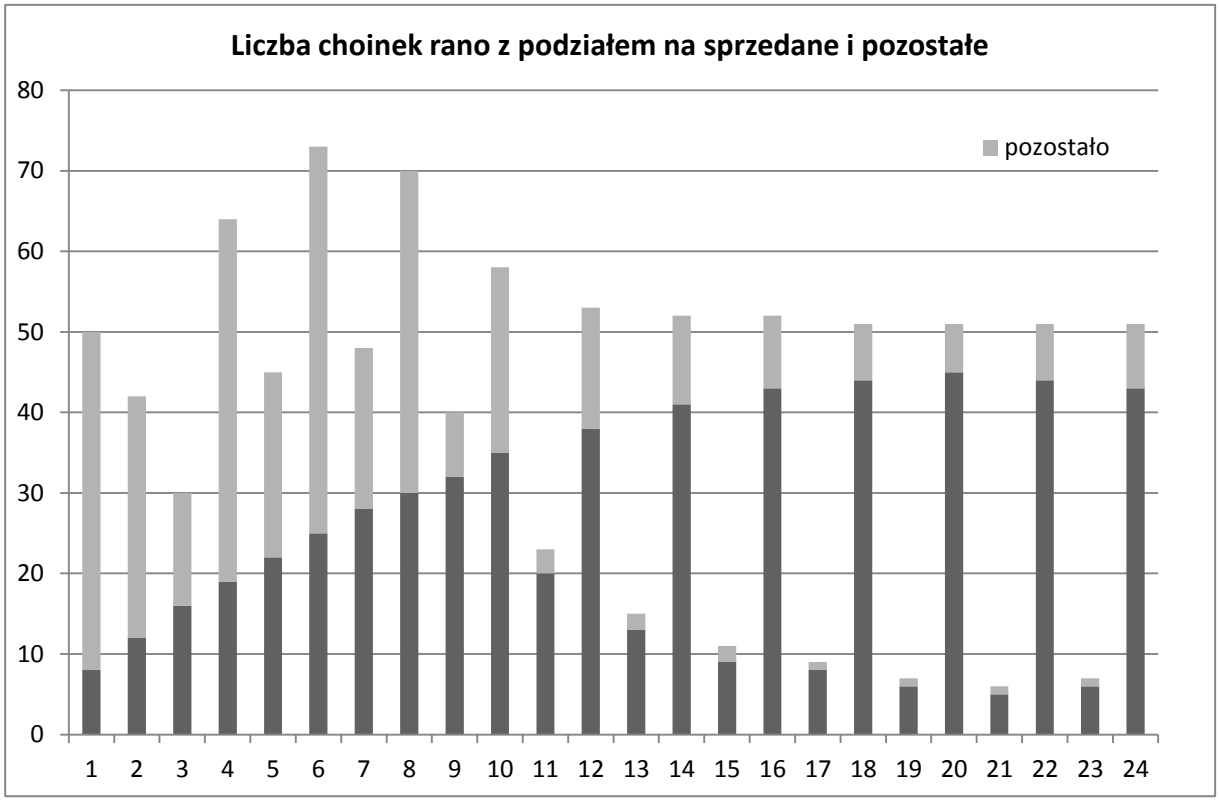
Zadanie 88.

88.1.

Firma sprzeda 592 choinki, a 8 zostanie na placu.

88.2.

Przykład poprawnej odpowiedzi:



88.3.

Poprawna odpowiedź: 11. dnia sprzedaży / 11 grudnia.

88.4.

Poprawna odpowiedź: 17 dostaw; poprawny jest dowolny ich harmonogram, pod warunkiem, że druga będzie nie później niż 4. dnia, trzecia nie później niż 6. dnia, czwarta nie później niż 8. dnia, siódma nie później niż 12. dnia, a czternasta nie później niż 20. dnia.

Przy harmonogramie odpowiadającym tym warunkom firma sprzeda 819 choinek.

88.5.

Poprawna odpowiedź: Po 24 dostawach, każda po 35 choinek, na placu pozostaną 21 drzewka.

Zadanie 90.

90.1.

Poprawna odpowiedź:

Katowice, 57 studentów,

najpopularniejsza uczelnia to Politechnika Informatyczno-Elektroniczna.

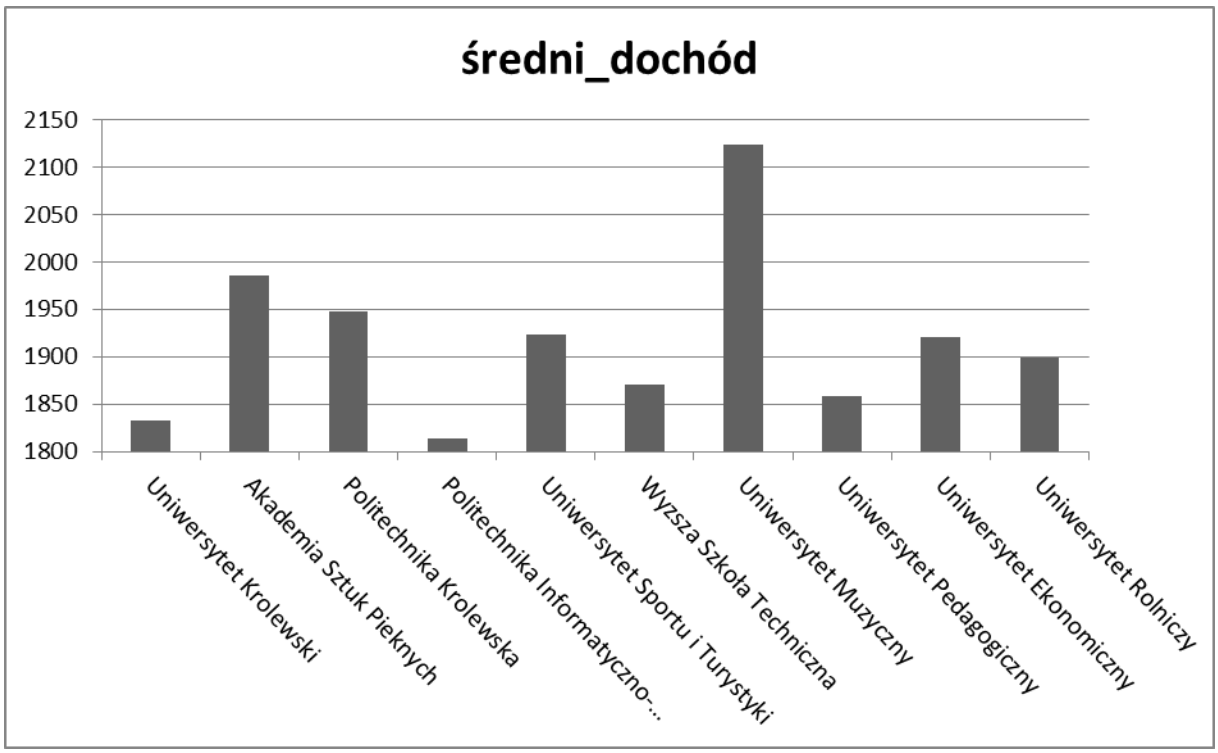
90.2.

143 miejsca

<=2256 lub <2257denarów

90.3.

Uczelnia	średni_dochód
Uniwersytet Krolewski	1832,22
Akademia Sztuk Pieknych	1985,98
Politechnika Krolewska	1947,34
Politechnika Informatyczno-Elektroniczna	1814,20
Uniwersytet Sportu i Turystyki	1923,81
Wyzsza Szkoła Techniczna	1871,07
Uniwersytet Muzyczny	2124,00
Uniwersytet Pedagogiczny	1858,47
Uniwersytet Ekonomiczny	1921,06
Uniwersytet Rolniczy	1898,49



90.4.

Przykład poprawnej odpowiedzi:

Uczelnie	I	II	III	IV	V
Akademia Sztuk Pieknych	32	10	6	2	5
Politechnika Informatyczno-Elektroniczna	299	108	48	54	54
Politechnika Krolewska	168	59	28	31	25
Uniwersytet Sportu i Turystyki	24	5	6	3	5
Uniwersytet Ekonomiczny	32	9	3	2	7
Uniwersytet Krolewski	233	77	53	32	61
Uniwersytet Muzyczny	5	2	0	1	0
Uniwersytet Pedagogiczny	31	11	4	2	5
Uniwersytet Rolniczy	27	14	7	5	6
Wyzsza Szkoła Techniczna	8	1	1	0	5

90.5.

Przykład poprawnej odpowiedzi:

Imie	Nazwisko	Miejsce_zam	Dochod_na_osobe
Iwona	Andrzejewska	Ruda Slaska	2241
Jacek	Andrzejewski	Ruda Slaska	2241
Anna	Barska	Ogrodzieniec	1047
Barbara	Barska	Ogrodzieniec	1047
Krzysztof	Barski	Ogrodzieniec	1047
Joanna	Bilska	Kedzierzyn-Kozle	2807
Michal	Bilski	Kedzierzyn-Kozle	2807
Waclawa	Borkowska	Szczyrk	1683
Sylwester	Borkowski	Szczyrk	1683
Piotr	Jablonski	Sanok	1185
Zygmunt	Jablonski	Sanok	1185
Tomasz	Kaczmarek	WodzislawSlaski	1140
Weronika	Kaczmarek	WodzislawSlaski	1140
Przemyslaw	Kaczmarek	WodzislawSlaski	1140
Klaudia	Kotowicz	Katowice	1950
Tadeusz	Kotowicz	Katowice	1950
Felicja	Ostrowska	Tarnowskie Gory	2466
Boguslawa	Ostrowska	Tarnowskie Gory	2466
Robert	Tomaszewski	Wisla	1511
Bartosz	Tomaszewski	Wisla	1511

Zadanie 91.

91.1.

Beatrycze

Doris

Ines

91.2.

08321103754	Wizniewski	Andrzej
09321501177	Wizniewski	Andrzej
08322201772	Michalak	Krzysztof
09311310792	Michalak	Krzysztof
66100294134	Kowalczyk	Mateusz
59031152059	Kowalczyk	Mateusz
09211700664	Kozlowska	Malgorzata
09313003607	Kozlowska	Malgorzata
67112966668	Kozlowska	Malgorzata

91.3.

Największa liczba porządkowa: Piotr Dzierzak

Najmniejsza liczba porządkowa: Amelia Wendt

91.4.

Liczba osób urodzonych w poszczególnych miesiącach:

styczeń	68
luty	33
marzec	9
kwiecień	16
maj	13
czerwiec	15
lipiec	19
sierpień	22
wrzesień	32
październik	67
listopad	99
grudzień	101

91.5.

AWie3
AWit4
AWoj0
AWoj2
AWoj8
BWas9
JPod4
KMic2
LMar4
MKoc9
MKor0
MKow4
MLub7
NJak2
NJan3
NJan6
SCie9
SDab7
ZAda1

Zadanie 92.

92.1.

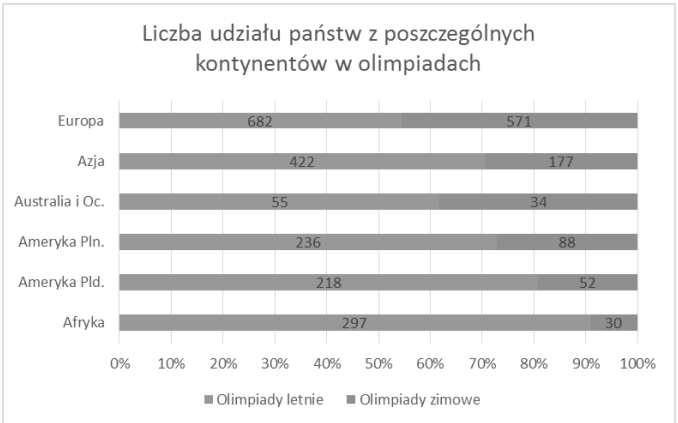
Liczba państw: 54.

Liczba medali zdobytych na letniej olimpiadzie: 1218.

92.2.

Kontynent	Olimpiady letnie	Olimpiady zimowe
Afryka	297	30
Ameryka Pld.	218	52
Ameryka Pln.	236	88
Australia i Oc.	55	34
Azja	422	177
Europa	682	571

Przykładowy wykres przedstawiający poprawne zestawienie:



92.3.

Kamerun

Burundi

Zjednoczone Emiraty Arabskie

92.4.

Afryka	Kenia	86
Ameryka Pld.	Brazylia	108
Ameryka Pln.	StanyZjednoczone	2681
Australia i Oc.	Australia	480
Azja	Chiny	526
Europa	ZSRR	1204

92.5.

Liczba krajów letnich: 35.

Liczba krajów zimowych: 3.

Zadanie 93.

93.1.

Ewelina Adamska

93.2.

Poprawna odpowiedź:

Budzianowska	41
Kolarski	41
Laska	41
Maskor	41
Rzepka	42
Rzymski	42

Odpowiedź błędna, wynikająca z podania także nazwisk osób, które były w delegacji dokładnie 40 dni (zastosowanie nierówności nieostrej):

Andrycz	40
Elawa	40
Budzianowska	41
Kolarski	41
Laska	41
Maskor	41
Rzepka	42
Rzymski	42

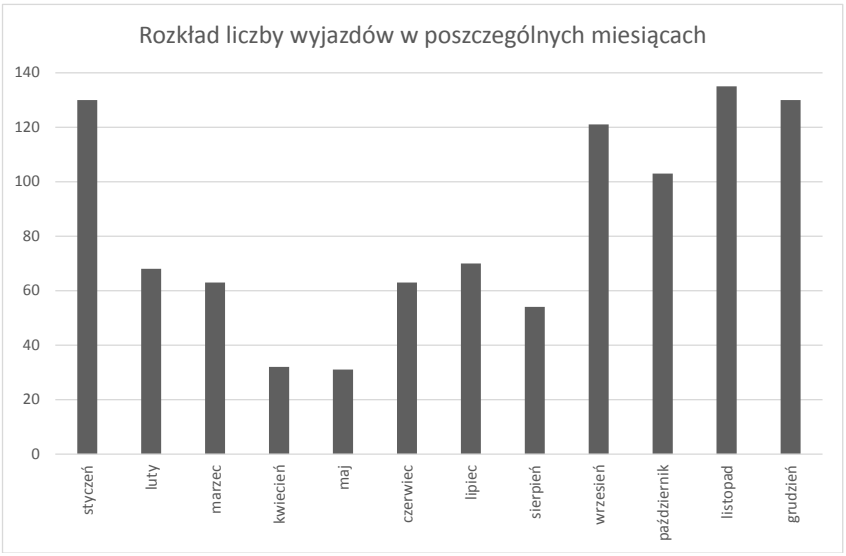
93.3.

Radom	27599,50
Siedlce	34215,40
Zgierz	38263,50
Kutno	43372,80
Kielce	48014,70
Olsztyn	50606,60
Kalisz	51099,00
Lublin	53209,50
Mielec	54144,00
Bydgoszcz	63932,00
Krakow	80344,50
Katowice	81486,30
Malbork	84280,00

93.4.

styczeń	130
luty	68
marzec	63
kwiecień	32
maj	31
czerwiec	63
lipiec	70
sierpień	54
wrzesień	121
październik	103
listopad	135
grudzień	130

Przykładowy wykres przedstawiający poprawne zestawienie:



93.5.

Liczba noclegów pracowników we wszystkich delegacjach — 1,64 ±0,01

Liczba noclegów pracowników w delegacjach co najmniej 2-dniowych — 2,25 ±0,01

Zadanie 94.

94.1.

Przyrost procentowy: 270%

Identyfikator dziecka: ID_dz: 718

94.2.

od 10 lat do 13 lat

94.3.

17 dzieci

94.4.

dla dziewcząt: w ósmym roku życia

dla chłopców: w jedenastym roku życia

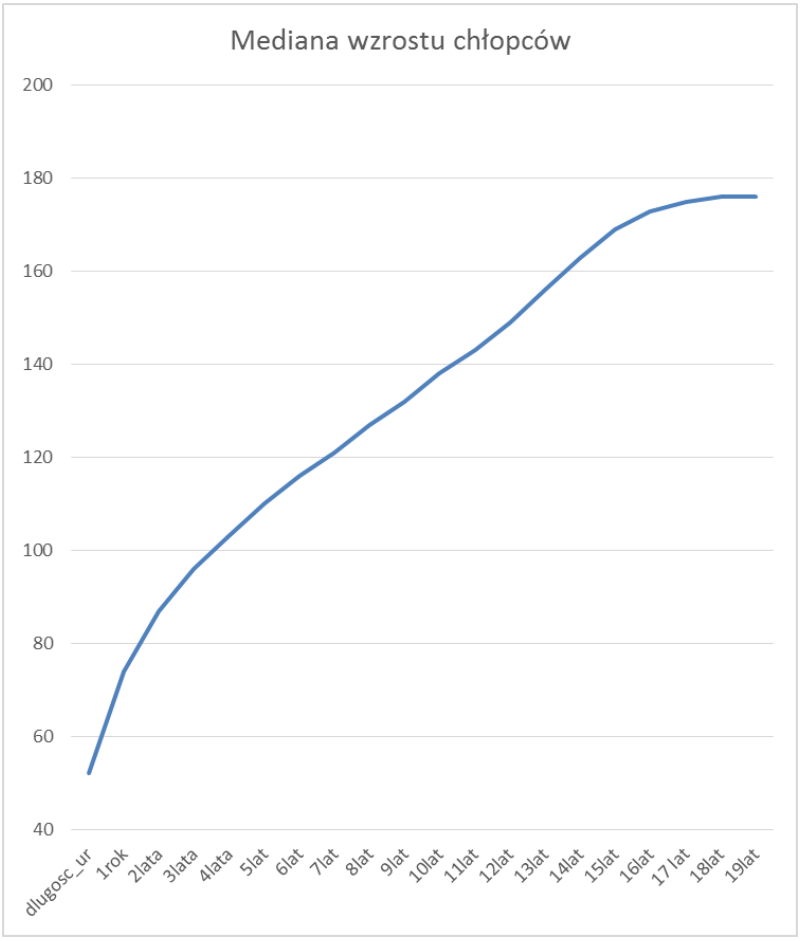
94.5.

	1rok	10lat	19lat
centyl 5.	68	128	166
centyl 95.	80	147	187

94.6.

	dlugosc_ur	1rok	2lata	3lata	4lata	5lat	6lat	7lat	8lat	9lat
mediana	52	74	87	96	103	110	116	121	127	132

	10lat	11lat	12lat	13lat	14lat	15lat	16lat	17lat	18lat	19lat
mediana	138	143	149	156	163	169	173	175	176	176



Zadanie 95.

95.1.

```
LPP        7485,84
WAWEL           960,06
PZU        481,56
```

95.2.

```
MILKILAND   27,85%
```

95.3.

```
rodzaj          licznik     obrot_zl
krajowa         418         2427046811
zagraniczna      52           29736190
```

Procentowy udział obrotów spółek krajowych = 98,79%

95.4.

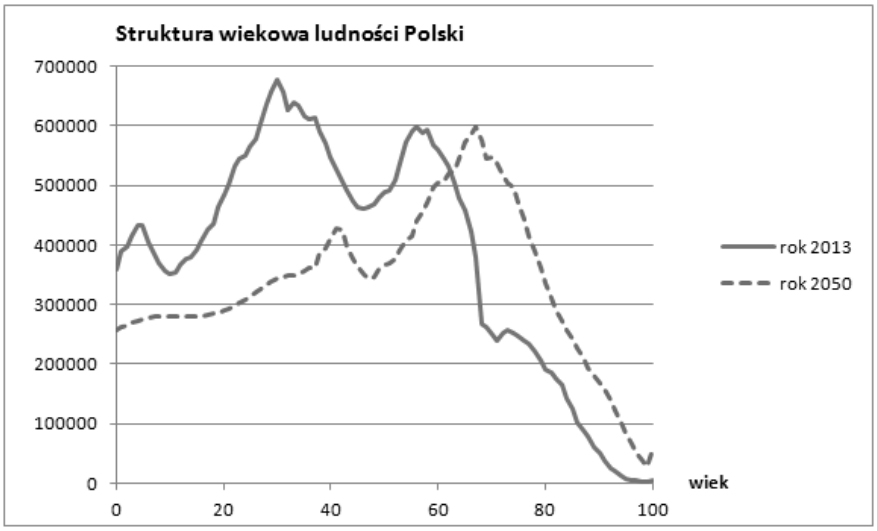
```
data                        M                WIG
2015-01-21          281 091 498 490,00         50 987,13
2015-01-22          284 219 170 040,00         51 554,46
2015-01-23          285 297 764 160,00         51 750,07
```

95.5.

```
kup       53
sprzedaj  20
obserwuj  397
```

Zadanie 96.

96.1.



96.2.

stosunek liczby ludności w miastach do liczby ludności wsi:
w roku 2013 : `1,53`
w roku 2050 : `1,24`

96.3.

średni wiek mężczyzny zamieszkałego w mieście:
w roku 2013 : `39`
w roku 2050 : `48`

96.4.

rok	wiek	2022	51	2032	52	2042	57
2013	49	2023	50	2033	53	2043	57
2014	49	2024	51	2034	53	2044	58
2015	49	2025	52	2035	54	2045	59
2016	49	2026	51	2036	55	2046	60
2017	48	2027	51	2037	54	2047	59
2018	49	2028	52	2038	55	2048	60
2019	48	2029	53	2039	56	2049	61
2020	49	2030	53	2040	57	2050	61
2021	50	2031	53	2041	56		

W niektórych latach kalendarzowych (np. 2019) będzie mieć miejsce taka sytuacja, że kobiety w pewnym wieku (np. 48 lat) uzyskają przewagę liczebną nad mężczyznami, ale w kolejnej grupie wiekowej (49 lat) znów będzie ich mniej niż mężczyzn, dopiero w następnej grupie wiekowej (50 lat) rozpocznie się trwała przewaga liczebna kobiet.

Poniższe zestawienie podaje także te drugie progi wiekowe w danych latach kalendarzowych, np. w roku 2019 kobiety po raz pierwszy przeważą nad mężczyznami w grupie wiekowej 48 lat, a po raz drugi, już trwale, w grupie wiekowej 50 lat.

Ponieważ w treści zadania było pytanie o **najniższy** wiek, w którym kobiety przeważają liczebnie nad mężczyznami, taka odpowiedź uzyskałaby niepełny wymiar punktów.

rok	wiek
2013	49
2014	49
2015	49
2016	49
2017	48
2018	49
2019	48
2019	50
2020	49
2020	51
2021	50
2022	51
2023	50
2024	51
2025	52
2026	51
2027	51
2028	52
2029	53
2030	53
2031	53
2032	52
2032	54
2033	53
2034	53
2035	54
2036	55
2037	54
2037	56
2038	55
2039	56
2040	57
2041	56
2042	57
2043	57
2044	58
2045	59
2046	60
2047	59
2048	60
2049	61
2050	61

96.5.

Poprawne zestawienie procentowe ludności grupy w wieku produkcyjnym w kolejnych latachrok % prod

2013	69%	2021	66%	2029	63%	2037	63%	2045	60%
2014	69%	2022	66%	2030	63%	2038	62%	2046	60%
2015	69%	2023	65%	2031	63%	2039	62%	2047	59%
2016	69%	2024	64%	2032	63%	2040	62%	2048	58%
2017	68%	2025	64%	2033	63%	2041	62%	2049	58%
2018	68%	2026	64%	2034	63%	2042	61%	2050	57%
2019	67%	2027	63%	2035	63%	2043	61%		
2020	67%	2028	63%	2036	63%	2044	61%		

Zadanie 97.

97.1.

Liczba miesięcy wysokiego zagrożenia: 42

97.2.

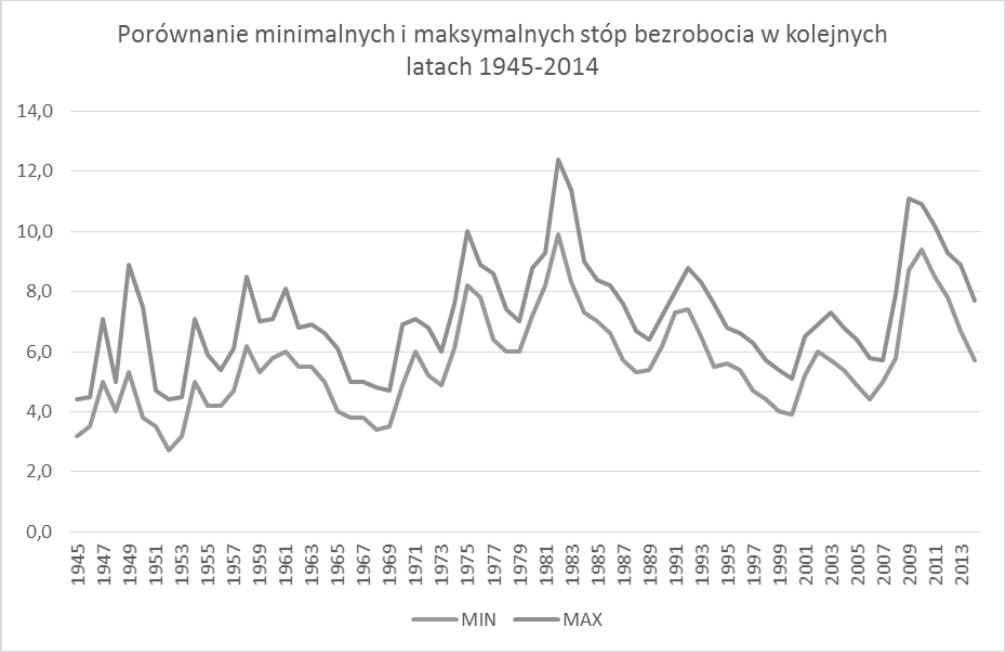
Rok 1953 3,64

Rok 1982 10,85

97.3.

ROK	MIN	MAX	ROK	MIN	MAX
1945	3,2	4,4	1980	7,2	8,8
1946	3,5	4,5	1981	8,2	9,3
1947	5,0	7,1	1982	9,9	12,4
1948	4,0	5,0	1983	8,3	11,4
1949	5,3	8,9	1984	7,3	9,0
1950	3,8	7,5	1985	7,0	8,4
1951	3,5	4,7	1986	6,6	8,2
1952	2,7	4,4	1987	5,7	7,6
1953	3,2	4,5	1988	5,3	6,7
1954	5,0	7,1	1989	5,4	6,4
1955	4,2	5,9	1990	6,2	7,2
1956	4,2	5,4	1991	7,3	8,0
1957	4,7	6,1	1992	7,4	8,8
1958	6,2	8,5	1993	6,5	8,3
1959	5,3	7,0	1994	5,5	7,6
1960	5,8	7,1	1995	5,6	6,8
1961	6,0	8,1	1996	5,4	6,6
1962	5,5	6,8	1997	4,7	6,3
1963	5,5	6,9	1998	4,4	5,7
1964	5,0	6,6	1999	4,0	5,4
1965	4,0	6,1	2000	3,9	5,1
1966	3,8	5,0	2001	5,2	6,5
1967	3,8	5,0	2002	6,0	6,9
1968	3,4	4,8	2003	5,7	7,3
1969	3,5	4,7	2004	5,4	6,8
1970	4,9	6,9	2005	4,9	6,4
1971	6,0	7,1	2006	4,4	5,8
1972	5,2	6,8	2007	5,0	5,7
1973	4,9	6,0	2008	5,8	7,9
1974	6,1	7,6	2009	8,7	11,1
1975	8,2	10,0	2010	9,4	10,9
1976	7,8	8,9	2011	8,5	10,2
1977	6,4	8,6	2012	7,8	9,3
1978	6,0	7,4	2013	6,7	8,9
1979	6,0	7,0	2014	5,7	7,7

Przykładowy wykres przedstawiający poprawne zestawienie:



97.4.

Najdłuższy ciąg: 13.

Początek ciągu: II 1994.

Koniec ciągu: II 1995.

97.5.

Liczba lat: 15.

Zadanie 98.

98.1.

Liczba dziewcząt	2779
Liczba chłopców	2781

98.2.

Rodzaj szkoły	pyt1	pyt2	pyt3	pyt4	pyt5	pyt6
G	2,93	2,33	2,51	2,69	2,99	2,96
LO	3,02	2,27	2,59	2,60	3,01	2,97
SP	3,52	1,99	2,26	2,65	3,03	3,35
SZ	3,01	2,21	2,45	2,63	3,09	3,04
T	3,02	2,01	2,53	2,65	3,07	2,94

98.3.

GM07	3,63	GM12	3,13	GM13	3,08	GM04	3,02
GM08	3,37	GM11	3,12	GM16	3,07	GM05	2,98
GM17	3,28	GM09	3,11	GM19	3,05	GM20	2,96
GM01	3,26	GM02	3,09	GM03	3,05	GM18	2,96
GM15	3,18	GM06	3,09	GM10	3,03	GM14	2,92

98.4.

G	80
LO	94
SP	70
SZ	12
T	35

98.5.

Lipcowa Rosa 390

98.6.

Rodzaj szkoły	liczba dziewczyąt	liczba chłopców
G	130	119
LO	116	108
SP	276	307
SZ	19	35
T	77	65

Zadanie 100.

100.1.

Lista	średnia liczba punktów
C1	11,80
C2	11,07
C3	9,67
C4	10,62
C5	10,02
C6	10,55
C7	11,13
P1	17,95
P2	19,65
P3	19,98
P4	20,83

100.2.

Albert Mikos

Lukasz Rdzanek

Dawid Zajaczkowski

100.3.

ocena	liczba uczniów
1	0
2	1
3	5
4	18
5	36

100.4.

punkty	G1	G2	G3	G4	G5
10	12	9	20	21	14
11	11	17	39	15	14
12	25	33	43	50	31

100.5.

Robert Czaja
Bartlomiej Gosk
Albert Mikos
Krzysztof Plata
Igor Rybarczyk
Karol Wojciul

Zadanie 101.

101.1.

90 kobiet i 45 mężczyzn

101.2.

Obiekt	Wartość
Active	3334
Aqua Park	2218
Bodyfit	2180
Lady Fitness Club	924
Platinium Center	1419
Pure Jatomi	1771
Redeco	3613
Spartan	3173
Top-Gym	1804

101.3.

Nazwisko	Imię
Kowalska	Maria
Olech	Klaudia
Pozarzycka	Justyna

101.4.

Silownia, 127 osób, Spartan

101.5.

Obiekt	Liczba wejść
Active	339
Aqua Park	226
Bodyfit	223
Lady Fitness Club	84
Platinium Center	120
Pure Jatomi	161
Redeco	383
Spartan	300
Top-Gym	164

Zadanie 102.

Realizacja komputerowa rozwiązania (wszystkich zadań z wiązki) znajduje się w pliku *portal.accdb*.

102.1.

231

102.2.

Joshua King Stany Zjednoczone
Joshua Nelson Stany Zjednoczone

Miguel Barbosa Portugalia
Miguel Rocha Brazylia

102.3.

Stany Zjednoczone 61
Kanada 41
Polska 33
Wielka Brytania 32
Niemcy 29
Brazylia 27
Hiszpania 26
Francja 24
Rosja 24
Holandia 22

102.4.

Antonio Reyes (1896 x1926)

102.5.

Prawidłowa odpowiedź (24 wiersze):

Fabian	Aigner	Austria
Matthias	Becker	Niemcy
Leonardo	De Luca	Wlochy
Mehmet	Demir	Turcja
Keiji	Fukuda	Japonia
Lovro	Gradic	Chorwacja
Jayden	Harper	Kanada
Anton	Ivanov	Rosja
Davit	Khachatryan	Armenia
Joshua	King	Stany Zjednoczone
Artjoms	Klavins	Lotwa
Said	Koudri	Algieria
Serhij	Kovalenko	Ukraina
Nathan	Mercier	Francja
Cristian	Munteanu	Rumunia
Jiri	Novotny	Czechy
Theo	Peeters	Belgia
Ioannis	Petridis	Grecja
Sebastian	Quispe	Peru
Samuel	Ramirez	Stany Zjednoczone
Allan	Smith	Wielka Brytania
Hikari	Takeuchi	Japonia
Hakan	Yilmaz	Turcja
Aiden	Young	Stany Zjednoczone

Zadanie 103.

103.1.

Grabowska Waclawa 58 17

103.2.

Poprawna odpowiedź:

Dolnoslaski	14414
Slaski	8540
Opolski	8490
Lubuski	7552
Pomorski	7351
Wielkopolski	7080
Swietokrzyski	5532
Mazowiecki	5345
Kujawsko-Pomorski	5317
Zachodniopomorski	5271
Lubelski	5097
Lodzki	4829
Podlaski	3700
Podkarpacki	3406
Małopolski	2399
Warminsko-Mazurski	358

Odpowiedź błędna, wynikająca z policzenia zamiast kosztów badań ich liczby:

Dolnoslaski	360
Opolski	210
Slaski	209
Lubuski	189
Pomorski	183
Wielkopolski	175
Swietokrzyski	137
Kujawsko-Pomorski	135
Mazowiecki	134
Zachodniopomorski	133
Lubelski	128
Lodzki	120
Podlaski	96
Podkarpacki	84
Małopolski	59
Warminsko-Mazurski	9

103.3.

Poprawna odpowiedź dla punktu a):

RTG przedramienia	352

Poprawna odpowiedź dla punktu b):

RTG przedramienia	337

103.4.

Liczba kobiet — 339.
Liczba mężczyzn — 477.

103.5.

Poprawna odpowiedź:

4	2
5	4
6	19
7	95
8	192
9	289
10	215

Za poprawną odpowiedź uznaje się także zestawienie, do którego włączone są 3 pierwsze dziesięciolecia z liczbą pacjentów równą zero.

Odpowiedź błędna, wynikająca z podania kolejnych dziesiątek jako pierwszej cyfry z roku urodzenia:

3	2
4	4
5	19
6	95
7	192
8	289
9	215

Zadanie 104.

104.1.

30.03.2015, 56 recept

104.2.

Leki przeciwnowotworowe i immunomodulujace — inhibitory enzymow — doustne inhibitory aromatazy

104.3.

Miesiąc	Liczba recept	Wartość wszystkich lekarstw
1	800	71 698,64 zł
2	742	61 522,39 zł
3	936	75 405,95 zł

104.4.

Cena_detaliczna	Nazwa_grupy
4770,46	Hormony przysadki i podwzgorza - inhibitory hormonu wzrostu

104.5.

ID_recepty	Data
28/2015	2015-01-02
64/2015	2015-01-05
655/2015	2015-01-27
673/2015	2015-01-27
710/2015	2015-01-28
1133/2015	2015-02-12
1274/2015	2015-02-18
1289/2015	2015-02-18
1517/2015	2015-02-27
2014/2015	2015-03-17
2209/2015	2015-03-24

Zadanie 105.

105.1.

Sklep wystawił 540 faktur. Największa wartość faktury wynosi 1250. Więcej niż jedną fakturę dostało 33 klientów.

105.2.

Miasto	Liczba zamówień
Bielawa	5
Jelenia Gora	5
Olawa	1
Olesnica	1
Sobotka	8
Trzebnica	4
Walbrzych	5
Wolow	13
Wroclaw	33
Zlotoryja	3

105.3.

Nazwisko	Imie	Liczba sadzonek	Nazwa
Gryczka	Nina	12	Symphytum grandiflorum Goldsmith
Gorska	Oliwia	14	Symphytum grandiflorum Goldsmith
Gondek	Oliwia	12	Symphytum grandiflorum Goldsmith
Szubarczyk	Dawid	11	Symphytum grandiflorum Goldsmith
Mezynska	Lena	14	Symphytum x uplandicum Axmister s Gold
Boleski	Tymon	12	Symphytum x uplandicum Axmister s Gold
Jezierska	Nadia	12	Symphytum x uplandicum Axmister s Gold
Bzówka	Szymon	13	Symphytum x uplandicum Axmister s Gold

105.4.

Primula japonica
Carex Comans
Echinacea Tomato Soup
Echinacea Black Beauty

105.5.

Rozmiary doniczki	Liczba zamówień
10x10x20	85
11x11x11	355
13x13x13	96
9x9x12	429
9x9x9	1051

Zadanie 106.

106.1.

```
bogatka, sroka, myszolow
```

106.2.

```
Miesiąc    liczba remizów
5          2
6          9
7          3
8          12
9          2
```

106.3.

```
nazwa_zwyczajowa  liczba_lokalizacji_miejskich
gawron            7
kawka             16
kruk              7
wrona siwa        20
```

106.4.

a)

```
lokalizacja     data         czas     Łączna_liczebnosc
Tadzino         2014-07-18   960      13
```

b)

```
lokalizacja            data          sprawność
Jezioro Zarnowieckie   2014-12-03    143,875
```

106.5.

a) 1624,

b) tabela poniżej

powiat	Zachowanie obserwowanego ptaka				
	brak informacji	gniazduje	leci	odpoczywa	zeruje
chojnicki					2
gdanski	7		35	3	8
kartuski			1	6	
koscierski			120		
malborski	5				
pucki	111		580		90
starogardzki	307				
wejherowski		22	243	17	67

Zadanie 107.

107.1.

Warszawa 83
Dusseldorf 78
Warszawa Modlin 68

107.2.

Krankowska Joanna 16
Lipinski Artur 16
Wilkonska Marta 18
Wirowski Marcin 16
Wronikowska Magdalena 19

107.3.

4 69
5 86
6 91

107.4.

Gumowska Anna
Hanke Amelia
Kowal Waldemar
Krupa Jan
Krupa Mariola
Kupinski Jakub
Lipinski Artur
Lis Krystyna
Pawlikowska Beata
Sibilska Monika
Tobera Jowita

Waruszewska Agnieszka
Wilczek Agata
Wirowski Marcin
Wojtas Sylwester
Zawadzka Monika

107.5.

21 Bolonia
99 Barcelona
316 Warszawa
522 Liverpool
987 Oslo Torp

Zadanie 108.

108.1.

45 171

108.2.

DW990, 13 km

108.3.

22, Berlinka, 8,2 km

108.4.

- Autostrada Toruńska
- Gliwicka Autostrada

108.5.

Prawidłowa odpowiedź z wykorzystaniem kwerendy krzyżowej (MS Access):

	kategoria drogi			
	autostrada	droga ekspresowa	droga krajowa	droga wojewódzka
Blisko od szosy	23	50	136	233
Czysta Nafta	25	44	105	235
Dobre Paliwo	22	56	112	223
Dyskont paliwowy	21	46	117	230
Green Fuel	19	40	120	261
Najpierw Ropa	23	45	128	217
Nitrogaz	31	52	120	205
Petrochemia Centralna	21	42	120	213
Standard Oil	21	50	127	241

Dopuszczamy też odpowiedź w tabeli składającej się z kolumn: nazwa sieci, kategoria drogi, liczba stacji.

Zadanie 109.

109.1.

Rusztowanie aluminiowe o wysokości 5m, 28 wypożyczeń.

109.2.

1 750 zł

109.3.

Imię	Nazwisko
Marcin	Budzisz
Wojciech	Dudek
Izabela	Kusz

109.4.

68

109.5.

Miesiąc	Przychód z wynajmu
1	43 307,00 zł
2	35 441,00 zł
3	46 181,00 zł
4	54 162,00 zł
5	81 019,00 zł
6	95 315,00 zł
7	111 173,00 zł
8	96 436,00 zł
9	101 491,00 zł
10	66 041,00 zł
11	47 754,00 zł
12	36 435,00 zł

Miesiąc	Przychód z transportu
1	850,00 zł
2	900,00 zł
3	1 200,00 zł
4	1 300,00 zł
5	1 950,00 zł
6	1 600,00 zł
7	2 450,00 zł
8	2 600,00 zł
9	1 800,00 zł
10	850,00 zł
11	700,00 zł
12	900,00 zł

Zadanie 110.

110.1.

913

110.2.

Typ miejscowosci	*Liczba miejscowosci*
wies	148
osada	47
osada lesna	12
kolonia	10
miasto	3
przysiolek	1

110.3.

Nazwa powiatu	*Nazwa wojewodztwa*
bielski	podlaskie
bielski	slaskie
brzeski	małopolskie
brzeski	opolskie
grodziski	mazowieckie
grodziski	wielkopolskie
krosnienski	podkarpackie
krosnienski	lubuskie
nowodworski	pomorskie
nowodworski	mazowieckie
opolski	lubelskie
opolski	opolskie
ostrowski	mazowieckie
ostrowski	wielkopolskie
sredzki	dolnośląskie
sredzki	wielkopolskie
swidnicki	lubelskie
swidnicki	dolnośląskie
tomaszowski	lubelskie
tomaszowski	lodzkie

110.4.

92 gminy

110.5.

Nazwa gminy	*Nazwa powiatu*	*Nazwa wojewodztwa*
Jaroslaw	jaroslawski	podkarpackie
Jaslo	jasielski	podkarpackie
Jastarnia	pucki	pomorskie
Jastrzebie-Zdroj	Jastrzebie-Zdroj	slaskie
Jawor	jaworski	dolnoslaskie
Jaworzno	Jaworzno	slaskie
Jedlina-Zdroj	walbrzyski	dolnoslaskie
Jelenia Gora	Jelenia Gora	dolnoslaskie
Jordanow	suski	malopolskie
Jozefow	otwocki	mazowieckie

Zadanie 111.

111.1.

Lp	kraj	opis	adres
1	Russia	redirects to AppleId **phish**ing	andreyzakharov.com/wp-content/plugins/wp-no-category-base/generic/
2	Spain	AppleId **phish**ing	www.matecocinas.com/productos/mesas/mesa-brenda/4rfv/
3	Luxembourg	AusPost **Phish**, leads to trojan - js	austr-post.net/open/scripts.js
4	Luxembourg	AusPost **Phish**, leads to trojan - php	austr-post.net/open/index.php
5	USA	Destination of banking **phish**ing	whitehorsetechnologies.net/images/clients/x/mail.php
6	Russia	Amazon **phish**ing	www.amazonsicherheitonline.com/
7	Germany	Amazon **phish**ing	cc8.joseppe.ru/de/amazon/amazon_check.php
8	France	HTML.ScrRedir.**Phish**	rec-danse.fr/tmp/photos6.php
9	France	PHP.**Phish**	rec-danse.fr/tmp/4640731669/
10	France	**Phish** Postbank - stat.php	rec-danse.fr/tmp/stat.php
11	France	**Phish** Postbank - very.php	rec-danse.fr/tmp/very.php
12	France	**Phish** Postbank - p.html	rec-danse.fr/tmp/p.html
13	France	JS.exploit, BoA **phish**	gulsproductions.com/css/bank-of-america-credit-card-payment-mailing-address
14	France	Google **phish** - adw.html	www.lajourneeducommercedeproximite.fr/jncp/images/documents/mouy/adw.html
15	France	Google **phish**	www.lajourneeducommercedeproximite.fr/jncp/images/documents/mouy/
16	France	Google **phish**	tremplin84.fr/components/BIG/2013gdocs/

111.2.

siec	kraj	liczbaIP	licznikPozycji
OVH OVH SAS	France	28	375
AMAZON-02 - Amazon.com, Inc.	USA	11	190
AS-COLOCROSSING - ColoCrossing	USA	4	116
HWNG Eweka Internet Services B.V.	Netherlands	3	99
INFOBOX-AS Infobox.ru Autonomous System	Russia	1	52

111.3.

687 różnych domen
trzy domeny mają przypisane więcej niż jeden IP:

```
f.cl.ly              9
directxex.com        5
dl.downf468.com      3
```

111.4.

Miesiac	afrinic	apnic	arin	lacnic	ripencc
1		4	20		118
2	1	21	81	2	202
3		12	52	3	110
4		1	31	3	27
5		56	53	4	128
6		1	3		68
7		8	119	6	117
8			1		5
9		1	12	4	51
10			8		7
11	2	1	14		128
12		190	7	1	24

111.5.

kraj	jpg	png
France	2	1
Germany	0	1
Russia	0	5
USA	38	1

Zadanie 112.

112.1.

rodzaj	ile_krajow
bakalie	2
baton	5
bombonierka	4
ciastka	3
ciasto	1
cukierki	4
czekolada	9
deser	2
dodatki	1
miod	1
napoje	2
przekaska	2
przetwory zbozowe	3
uzywki	5

112.2.

nazwa_firmy	lokalizacja
Hoop	Brno
Kraft-Foods	Praha, Rohansky Ostrov
Kraft-Foods	Praha, Karolinska
Mokate	Votice
Pacific	Antwerpia

112.3.

rodzaj	najmniejsza masa	nazwa towaru
bakalie	50	MIGDALY BLANSZOWANE
baton	28	MARCEPAN
bombonierka	37,5	ROCHER T3X16X6 FLO B POL 11
ciastka	17	PRINCE POLO ORZECHOWE
ciasto	40	FIX DO KARPATEK
cukierki	6	FIGURKA
czekolada	15	MLECZNA
deser	25	KREM O`LADA SCOOBY DOO
dodatki	3	BARWNIKI DO JAJEK
miod	15	MIOD PORCJOWANY
napoje	300	JUPIK BANAN
przekaska	37	NIC NAC'S
przetwory zbo-zowe	20	FITNESS Z CZEKOLADA I POMARANCZA 6-PAK PLATKI I MLEKO
uzywki	12	LA MATTINA CAPPUCCINO Z MAGNEZEM PALUCH

112.4.

Poprawna odpowiedź:nazwa	masa	nazwa_firmy	kraj
KINDER DUPLO T	364	Ferrero	Niemcy
NUTELLA G670X6 VT PR POL 11	670	Ferrero	Polska
BANG BANG	1200	Millano Baron	Polska
MILKA ALPEJSKIE MLECZKO WANI-LIOWE	350	Kraft-Foods	Szwajcaria
MILKA ALPEJSKIE MLECZKO CZE-KOLADOWE	350	Kraft-Foods	Szwajcaria
KINDER DELICE KAKAO	420	Ferrero	Wlochy

112.5.

kraj	0,05	0,08	0,23
Austria	3	4	
Belgia			7
Czechy	20		9
Francja	1		3
Hiszpania			1
Niemcy	3	16	28
Polska	146	220	496
Slowacja			4
Szwajcaria			57
Wegry			3
Wlochy			24

5. Wykaz umiejętności ogólnych i szczegółowych sprawdzanych zadaniami

Zadanie 1.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach;

Zadanie 2.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych

Zadanie 3.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania 11) opisuje podstawowe algorytmy i stosuje: a) algorytmy na liczbach całkowitych

Zadanie 4.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.

Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5. 4 dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5.7 opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 5.11 opisuje podstawowe algorytmy i stosuje a) algorytmy na liczbach całkowitych, np.: reprezentacja liczb w dowolnym systemie pozycyjnym, w tym, w dwójkowym i szesnastkowym,

Zadanie 5.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.6. ocenia własności rozwiązania algorytmicznego, np., efektywność działania 5.11. opisuje podstawowe algorytmy i stosuje b) algorytmy wyszukiwania i porządkowania (sortowania), np. (…) algorytmy sortowania ciągu liczb (…) przez wstawianie 5.16 opisuje własności algorytmów na podstawie ich analizy 5.18 oblicza liczbę operacji wykonywanych przez algorytm

Zadanie 6.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy wyszukiwania i porządkowania (sortowania); 18) oblicza liczbę operacji wykonywanych przez algorytm

Zadanie 7.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 6) ocenia własności rozwiązania algorytmicznego (kompute-

	rowego), np. zgodność ze specyfikacją, efektywność działania; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 8) posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: b) algorytmy wyszukiwania i porządkowania (sortowania) 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 8.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach;

Zadanie 9.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 8) posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów; 9) stosuje rekurencje w prostych sytuacjach problemowych; 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 10.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 8) posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów;

	9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: c) algorytmy numeryczne, np.: — obliczanie wartości wielomianu za pomocą schematu Hornera 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 11.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na tekstach, w tym obliczanie wartości wyrażenia podanego w postaci odwrotnej notacji polskiej;

Zadanie 12.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania 11) opisuje podstawowe algorytmy i stosuje: e) algorytmy kompresji i szyfrowania

Zadanie 13.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 11) opisuje podstawowe algorytmy i stosuje: […] f) algorytmy geometryczne

Zadanie 14.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: f) algorytmy badające własności geometryczne 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 15.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 8) posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów; 11) opisuje podstawowe algorytmy i stosuje: […] a) reprezentacje liczb w dowolnym systemie; f) algorytmy badające właściwości geometryczne […]; 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 16.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 11) opisuje podstawowe algorytmy i stosuje: […] f) algorytmy geometryczne

Zadanie 17.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje […] sytuacje z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązania problemów; 16) opisuje własności algorytmów na podstawie ich analizy;

	17) ocenia zgodność algorytmu ze specyfikacją; 26) ocenia poprawność rozwiązania problemu na podstawie jego testowania;

Zadanie 18.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5. 4 dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5.7 opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 5.11 opisuje podstawowe algorytmy i stosuje a) algorytmy na liczbach całkowitych, np.: — iteracyjna i rekurencyjna realizacja algorytmu Euklidesa, 5.12 projektuje rozwiązanie problemu (realizację algorytmu) i dobiera odpowiednią strukturę danych; 5.16 opisuje własności algorytmów na podstawie ich analizy;

Zadanie 19.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych, w tym reprezentację liczb w dowolnym systemie pozycyjnym.

Zadanie 20.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5) posługuje się podstawowymi technikami algorytmicznymi; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformuło-

	wania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: a) algorytmy na liczbach całkowitych, np.: — reprezentacja liczb w dowolnym systemie pozycyjnym, w tym, w dwójkowym i szesnastkowym,

Zadanie 21.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5. 4 dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5.7 opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 5.11c) opisuje podstawowe algorytmy i stosuje algorytmy numeryczne, np.: zastosowania schematu Hornera: szybkie podnoszenie do potęgi, 5.16 opisuje własności algorytmów na podstawie ich analizy; 5.18 oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 22.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 6) ocenia własności rozwiązania algorytmicznego (komputerowego), np. zgodność ze specyfikacją, efektywność działania; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: c) algorytmy numeryczne, np.: — obliczanie wartości wielomianu za pomocą schematu Hornera 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 23.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.2. stosuje podejście algorytmiczne do rozwiązania problemu 5.4. dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji 5.5. posługuje się podstawowymi technikami algorytmicznymi 5.11. opisuje podstawowe algorytmy i stosuje c) algorytmy numeryczne, np.: — obliczanie wartości pierwiastka kwadratowego 5.14. dobiera odpowiednie struktury danych do realizacji algorytmu 5.16 opisuje własności algorytmów na podstawie ich analizy

Zadanie 24.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5. 4 dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5.7 opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 5.8 posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów; 5.9 stosuje rekurencję w prostych sytuacjach problemowych; 5.11 opisuje podstawowe algorytmy i stosuje b) algorytmy wyszukiwania i porządkowania (sortowania); 5.16 opisuje własności algorytmów na podstawie ich analizy; 5.18 oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 25.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.2. stosuje podejście algorytmiczne do rozwiązania problemu 5.4. dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji 5.5. posługuje się podstawowymi technikami algorytmicznymi 5.11. opisuje podstawowe algorytmy i stosuje d) algorytmy na tekstach, np.: — sprawdzanie, czy dany ciąg znaków tworzy palindrom

	5.14. dobiera odpowiednie struktury danych do realizacji algorytmu

Zadanie 26.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na tekstach

Zadanie 27.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na tekstach, np. [...] wyszukiwanie wzorca w tekście;

Zadanie 28.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Zdający: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na tekstach;

Zadanie 29.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algoryt-

Wymagania szczegółowe	micznego. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania 11) opisuje podstawowe algorytmy i stosuje e) algorytmy kompresji i szyfrowania, np.: — kody znaków o zmiennej długości

Zadanie 30.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] e) algorytmy kompresji i szyfrowania, np. szyfr Cezara

Zadanie 31.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] e) algorytmy kompresji i szyfrowania, np. szyfr przestawieniowy

Zadanie 32.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń:

	2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] e) algorytmy szyfrowania i kompresji

Zadanie 33.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania sytuacji problemowej i zapisuje go w wybranej notacji; 5) posługuje się podstawowymi technikami algorytmicznymi; 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na liczbach całkowitych, w tym reprezentację liczb w dowolnym systemie pozycyjnym. 12) dobiera odpowiednie struktury danych do realizacji algorytmu, w tym struktury dynamiczne;

Zadanie 34.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
Wymagania szczegółowe	1.1) przedstawia sposoby reprezentowania różnych form informacji w komputerze: liczb, znaków, obrazów, animacji, dźwięków;

Zadanie 35.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
Wymagania szczegółowe	1.1) przedstawia sposoby reprezentowania różnych form informacji w komputerze: liczb, znaków, obrazów, animacji, dźwięków;

Zadanie 36.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera danych liczbowych.
Wymagania szczegółowe	2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);

Zadanie 37.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera danych liczbowych.
Wymagania szczegółowe	2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);

Zadanie 38.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera danych liczbowych.
Wymagania szczegółowe	2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);

Zadanie 39.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych. II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	1.1. Uczeń przedstawia sposoby reprezentowania różnych form informacji w komputerze: liczb, znaków, obrazów, animacji, dźwięków; 4.2 określa własności grafiki rastrowej i wektorowej oraz charakteryzuje podstawowe formaty plików graficznych, tworzy i edytuje obrazy rastrowe i wektorowe z uwzględnieniem warstw i przekształceń; 4.3. przetwarza obrazy i filmy, np.: zmienia rozdzielczość, rozmiar, model barw, stosuje filtry;

Zadanie 40.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów.

Zadanie 41.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów.

Zadanie 42.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: b) algorytmy wyszukiwania i porządkowania (sortowania), 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 43.

Wymagania ogólne	V. Ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki
Wymagania szczegółowe	7.1 Uczeń … opisuje najważniejsze elementy procesu rozwoju informatyki i technologii informacyjno-komunikacyjnych.

Zadanie 44.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych.
Wymagania szczegółowe	1. Posługiwanie się komputerem i jego oprogramowaniem, korzystanie z sieci komputerowej. Uczeń: 1) przedstawia sposoby reprezentowania różnych form informacji w komputerze: liczb, znaków, obrazów, animacji, dźwięków;

Zadanie 45.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
Wymagania szczegółowe	PP 1.1) opisuje podstawowe elementy komputera, jego urządzenia zewnętrzne i towarzyszące i ich działanie w zależności od wartości ich podstawowych parametrów, wyjaśnia współdziałanie tych elementów;

Zadanie 46.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych.
Wymagania szczegółowe	1. Posługiwanie się komputerem i jego oprogramowaniem, korzystanie z sieci komputerowej. Uczeń:

	2) wyjaśnia funkcje systemu operacyjnego i korzysta z nich; opisuje różne systemy operacyjne;

Zadanie 47.

Wymagania ogólne	Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej, komunikowanie się za pomocą komputera i technologii informacyjno komunikacyjnych.
Wymagania szczegółowe	1.2) wyjaśnia funkcje systemu operacyjnego i korzysta z nich; opisuje różne systemy operacyjne;

Zadanie 48.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych.
Wymagania szczegółowe	3) przedstawia warstwowy model sieci komputerowych, określa ustawienia sieciowe danego komputera i jego lokalizacji w sieci, (...)

Zadanie 49.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
Wymagania szczegółowe	1.4) zapoznaje się z możliwościami nowych urządzeń związanych z technologiami informacyjno-komunikacyjnymi, poznaje nowe programy i systemy oprogramowania

Zadanie 50.

Wymagania ogólne	I. Bezpieczne posługiwanie się komputerem i jego oprogramowaniem, wykorzystanie sieci komputerowej; komunikowanie się za pomocą komputera i technologii informacyjno-komunikacyjnych
Wymagania szczegółowe	1.4) zapoznaje się z możliwościami nowych urządzeń związanych z technologiami informacyjno-komunikacyjnymi, poznaje nowe programy i systemy oprogramowania

Zadanie 51.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń:
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń:

	5) opisuje mechanizmy związane z bezpieczeństwem danych: (...)

Zadanie 52.

Wymagania ogólne	II Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	1.1) przedstawia sposoby reprezentowania różnych form informacji w komputerze: liczb, znaków, obrazów, animacji, dźwięków; 4.2) określa własności grafiki rastrowej i wektorowej oraz charakteryzuje podstawowe formaty plików graficznych, tworzy i edytuje obrazy rastrowe i wektorowe z uwzględnieniem warstw i przekształceń;

Zadanie 53.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje […] sytuacje problemowe z różnych dziedzin; 11) […] stosuje: a) algorytmy na liczbach całkowitych, np. reprezentacje liczb w systemie dwójkowym.

Zadanie 54.

Wymagania ogólne	III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5.14 Dobiera odpowiednie struktury danych do realizacji algorytmu, w tym struktury dynamiczne;

Zadanie 55.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 16) opisuje własności algorytmów na podstawie ich analizy; 18) oblicza liczbę operacji wykonywanych przez algorytm;

Zadanie 56.

Wymagania ogólne	V. Ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki

Wymagania szczegółowe	7.4 Uczeń … omawia zagadnienia przestępczości komputerowej, w tym piractwo komputerowe, nielegalne transakcje w sieci

Zadanie 57.

Wymagania ogólne	V. Ocena zagrożeń i ograniczeń, docenianie społecznych aspektów rozwoju i zastosowań informatyki
Wymagania szczegółowe	7.4 Uczeń … omawia zagadnienia przestępczości komputerowej, w tym piractwo komputerowe, nielegalne transakcje w sieci

Zadanie 58.

Wymagania ogólne	II. […] opracowanie za pomocą komputera danych liczbowych. III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np. efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy badające właściwości geometryczne […]; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania.

Zadanie 59.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.11 opisuje podstawowe algorytmy i stosuje a) algorytmy na liczbach całkowitych, np.: — rozkładanie liczby na czynniki pierwsze,

	— i inne 5.12 projektuje rozwiązanie problemu (realizację algorytmu) i dobiera odpowiednią strukturę danych; 5.15 stosuje zasady programowania strukturalnego i modularnego do rozwiązywania problemu; 5.21 przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 5.22 sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu i uruchamianiu programów; 5.23 stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 5.25 dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania;

Zadanie 60.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu;

Zadanie 61.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu;

Zadanie 62.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algoryt-

	micznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania 11) opisuje podstawowe algorytmy i stosuje: a) algorytmy na liczbach całkowitych — reprezentacja liczb w dowolnym systemie pozycyjnym 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu

Zadanie 63.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.11 opisuje podstawowe algorytmy i stosuje a) algorytmy na liczbach całkowitych, np.: — reprezentacja liczb w dowolnym systemie pozycyjnym, w tym w dwójkowym i szesnastkowym, — sprawdzanie, czy liczba jest liczbą pierwszą, — i inne 5.12 projektuje rozwiązanie problemu (realizację algorytmu) i dobiera odpowiednią strukturę danych; 5.15 stosuje zasady programowania strukturalnego i modularnego do rozwiązywania problemu; 5.21 przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 5.22 sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu i uruchamianiu programów; 5.23 stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 5.25 dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania;

Zadanie 64.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe;

	2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np. efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] b) algorytmy wyszukiwania […]; a) algorytmy na liczbach; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania

Zadanie 65.

Wymagania ogólne	II. […] opracowanie za pomocą komputera danych liczbowych. III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np. efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych […]; b) algorytmy wyszukiwania […]; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania.

Zadanie 66.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wy-

	korzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.11 opisuje podstawowe algorytmy i stosuje a) algorytmy na liczbach całkowitych, np.: — sprawdzanie, czy liczba jest liczbą pierwszą, f) algorytmy badające własności geometryczne, np. — sprawdzanie warunku trójkąta 5.12 projektuje rozwiązanie problemu (realizację algorytmu) i dobiera odpowiednią strukturę danych; 5.15 stosuje zasady programowania strukturalnego i modularnego do rozwiązywania problemu; 5.21 przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 5.22 sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu i uruchamianiu programów; 5.23 stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 5.25 dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania;

Zadanie 67.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: […] a) algorytmy na liczbach całkowitych, np.: — reprezentacja liczb w dowolnym systemie pozycyjnym, w tym w dwójkowym… — sprawdzanie, czy liczba jest liczbą pierwszą … f) konstrukcje rekurencyjne: binarny fraktal Fibonacciego

Zadanie 68.

Wymagania ogólne	II. […] opracowanie za pomocą komputera tekstów, danych liczbowych. III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algo-

	rytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np. efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] b) algorytmy wyszukiwania […]; d) algorytmy na tekstach; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania

Zadanie 69.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np. efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] b) algorytmy wyszukiwania […]; d) algorytmy na tekstach; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania

Zadanie 70.

Wymagania ogólne	III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.11 opisuje podstawowe algorytmy i stosuje c) algorytmy numeryczne, np.: — obliczanie pola obszarów zamkniętych, 5.21 przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 5.25 dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania;

Zadanie 71.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 8) posługuje się metodą „dziel i zwyciężaj” w rozwiązywaniu problemów; 9) stosuje rekurencję w prostych sytuacjach problemowych. 11) opisuje podstawowe algorytmy i stosuje: c) algorytmy numeryczne, np.: — wyznaczanie miejsc zerowych funkcji metodą połowienia, 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu;

Zadanie 72.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformuło-

	wania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] d) algorytmy na tekstach 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu;

Zadanie 73.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] e) algorytmy na tekstach 12) projektuje rozwiązanie problemu (realizację algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu;

Zadanie 74.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5.Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi 11) opisuje podstawowe algorytmy i stosuje: d) algorytmy na tekstach, np.: — porządkowanie alfabetyczne, 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 25) dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania

Zadanie 75.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, sto-

	sowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: […] e) algorytmy szyfrowania i kompresji 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu;

Zadanie 76.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: e) algorytmy kompresji i szyfrowania, np.: szyfr przestawieniowy 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu;

Zadanie 77.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 11) opisuje podstawowe algorytmy i stosuje e) algorytmy kompresji i szyfrowania np.: — kody znaków o zmiennej długości, np. alfabet Morse'a, kod Huffmana, — szyfr Cezara, — szyfr przestawieniowy, — szyfr z kluczem jawnym (RSA), — wykorzystanie algorytmów szyfrowania, np. w podpisie elektronicznym,

Zadanie 78.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 11) opisuje podstawowe algorytmy i stosuje e) algorytmy kompresji i szyfrowania np.: kody znaków o zmiennej długości, np. alfabet Morse'a, kod Huffmana, szyfr Cezara, szyfr przestawieniowy, szyfr z kluczem jawnym (RSA), wykorzystanie algorytmów szyfrowania, np. w podpisie elektronicznym,

Zadanie 79.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania; 11) opisuje podstawowe algorytmy i stosuje: f) algorytmy badające własności geometryczne 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu;

Zadanie 80.

Wymagania ogólne	II. […] opracowanie za pomocą komputera danych liczbowych. III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 4) dobiera efektywny algorytm do rozwiązania problemu; 5) posługuje się podstawowymi technikami algorytmicznymi; 6) ocenia własność rozwiązania algorytmicznego […], np.

	efektywność działania. 11) opisuje podstawowe algorytmy i stosuje: […] f) algorytmy badające właściwości geometryczne […]; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania.

Zadanie 81.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania 11) opisuje podstawowe algorytmy i stosuje: f) algorytmy badające własności geometryczne 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu

Zadanie 82.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	4.4. wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 83.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych. III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.3 formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 5.25.dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania

Zadanie 84.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych. III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5.Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 5.1 analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 5.2 stosuje podejście algorytmiczne do rozwiązywania problemu; 5.3 formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 5.25.dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania

Zadanie 85.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 86.

Wymagania ogólne	II. […] opracowanie za pomocą komputera tekstów, danych liczbowych. III. Rozwiązywanie problemów i podejmowanie decyzji […] z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. […] opracowanie za pomocą komputera tekstów, danych liczbowych. Zdający: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5. Rozwiązywanie problemów i […] stosowanie podejścia algorytmicznego. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 12) projektuje rozwiązanie problemu (realizacja algorytmu) i dobiera odpowiednią strukturę danych; 21) przeprowadza komputerową realizację algorytmu i rozwiązania problemu; 23) stosuje podstawowe konstrukcje programistyczne w wybranym języku programowania, instrukcje iteracyjne i warunkowe, rekurencję, funkcje i procedury, instrukcje wejścia i wyjścia, poprawnie tworzy strukturę programu; 26) ocenia poprawność komputerowego rozwiązania problemu na podstawie jego testowania.

Zadanie 87.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: rysunków, tekstów, danych liczbowych, animacji, prezentacji multimedialnych i filmów. 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń:

	2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania

Zadanie 88.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: rysunków, tekstów, danych liczbowych, animacji, prezentacji multimedialnych i filmów. 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5. Rozwiązywanie problemów i podejmowanie decyzji, stosowanie podejścia algorytmicznego. Uczeń: 2) stosuje podejście algorytmiczne do rozwiązywania problemu 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania

Zadanie 89.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5.Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 5.25.dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania

Zadanie 90.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: […] tekstów, danych liczbowych. Uczeń: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 91.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych III. Rozwiązywanie problemów, podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego.
Wymagania szczegółowe	4.4.wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów 5.Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu; 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 5.25.dobiera właściwy program użytkowy lub samodzielnie napisany program do rozwiązywanego zadania

Zadanie 92.

Wymagania ogólne	II. [...] opracowywanie za pomocą komputera danych liczbowych. III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: […] tekstów, danych liczbowych. Uczeń: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 93.

Wymagania ogólne	II. [...] opracowywanie za pomocą komputera danych liczbowych. III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: […] tekstów, danych liczbowych. Uczeń: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 94.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: […] tekstów, danych liczbowych. Uczeń: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 95.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 4) znajduje odpowiednie informacje niezbędne do realizacji projektów z różnych dziedzin;

Zadanie 96.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 4) znajduje odpowiednie informacje niezbędne do realizacji projektów z różnych dziedzin;

Zadanie 97.

Wymagania ogólne	II. [...] opracowywanie za pomocą komputera danych liczbowych. III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	4. Opracowywanie informacji za pomocą komputera, w tym: […] tekstów, danych liczbowych. Uczeń: 4) wykorzystuje arkusz kalkulacyjny do obrazowania zależności funkcyjnych i do zapisywania algorytmów. 5. Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, stosowanie podejścia algorytmicznego. Uczeń: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 2) stosuje podejście algorytmiczne do rozwiązywania problemu 3) formułuje przykłady sytuacji problemowych, których rozwiązanie wymaga podejścia algorytmicznego i użycia komputera; 7) opracowuje i przeprowadza wszystkie etapy prowadzące do otrzymania poprawnego rozwiązania problemu: od sformułowania specyfikacji problemu po testowanie rozwiązania;

Zadanie 98.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1.projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2.2.stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3.tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 99.

Wymagania ogólne	II. […] przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera […] danych liczbowych […]. III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, […].
Wymagania szczegółowe	2. […] selekcjonowanie, przetwarzanie […] informacji, korzystanie z różnych źródeł […] informacji. Zdający: 1) projektuje relacyjną bazę danych […]; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integral-

	ność danych na poziomie pól, tabel, relacji; 5. Rozwiązywanie problemów […]. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 25) dobiera właściwy program użytkowy […] do rozwiązania zadania.

Zadanie 100.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1. projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3. tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 101.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji

Zadanie 102.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1. projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3. tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 103.

Wymagania ogólne	II. […] przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera […] danych liczbowych […]. III Rozwiązywanie problemów i podejmowanie decyzji z wykorzystaniem komputera, […].
Wymagania szczegółowe	2. […] selekcjonowanie, przetwarzanie […] informacji, korzystanie z różnych źródeł […] informacji. Zdający: 1) projektuje relacyjną bazę danych […]; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji; 5. Rozwiązywanie problemów […]. Zdający: 1) analizuje, modeluje i rozwiązuje sytuacje problemowe z różnych dziedzin; 25) dobiera właściwy program użytkowy […] do rozwiązania zadania.

Zadanie 104.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1.projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2.2.stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3.tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 105.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowanie za pomocą komputera danych liczbowych
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji

Zadanie 106.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);

Zadanie 107.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1. projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2.2. stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3. tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 108.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 109.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji.
Wymagania szczegółowe	2.1.projektuje relacyjną bazę danych z zapewnieniem integral-

	ności danych; 2.2.stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych; 2.3.tworzy aplikację bazodanową, w tym sieciową, wykorzystując język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 110.

Wymagania ogólne	II. Wyszukiwanie, gromadzenie i przetwarzanie informacji z różnych źródeł; opracowywanie za pomocą komputera: rysunków, tekstów, danych liczbowych, motywów, animacji, prezentacji multimedialnych.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL); 3) tworzy aplikację bazodanową, w tym sieciową, wykorzystującą język zapytań, kwerendy, raporty; zapewnia integralność danych na poziomie pól, tabel, relacji;

Zadanie 111.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);

Zadanie 112.

Wymagania ogólne	III. Rozwiązywanie problemów i podejmowanie decyzji z zastosowaniem podejścia algorytmicznego.
Wymagania szczegółowe	2. Wyszukiwanie, gromadzenie, selekcjonowanie, przetwarzanie i wykorzystywanie informacji, współtworzenie zasobów w sieci, korzystanie z różnych źródeł i sposobów zdobywania informacji. Uczeń: 1) projektuje relacyjną bazę danych z zapewnieniem integralności danych; 2) stosuje metody wyszukiwania i przetwarzania informacji w relacyjnej bazie danych (język SQL);