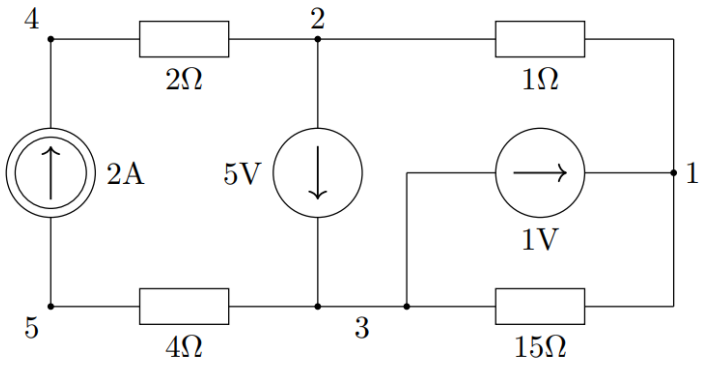
Spis treści

1 ALOPS

Napisać program do **a**nalizy **l**iniowych **o**bwodów **p**rądu **s**tałego. Elementami, które mogą wystąpić w obwodzie, to: siła elektromotoryczna, siła prądowa, rezystor.



Plik wejściowy opisuje obwód w następujący sposób. W każdej linii jest zapisywany tylko jeden element. Kodowanie elementu:

<typ> <węzeł początkowy> <węzeł końcowy> <wartość>

<typ> przyjmuje wartości: **E** – dla siły elektromotorycznej, **I** – dla siły prądowej i **R** – dla rezystora. Węzły kodowane są kolejnymi liczbami naturalnymi. Wartości elementów mogą być wartościami niecałkowitymi. W układzie może być dowolna liczba elementów.

Obwód z powyższego rysunku może być zapisany jako (kolejność elementów jest dowolna):

```
R 4 2 2
I 5 4 2
R 5 3 4
E 3 2 −5
E 3 1 1
R 1 2 1
R 3 1 15
```

W pliku wynikowym zostają wypisane elementy obwodu, natężenie prądu, spadek napięcia na elemencie i moc wydzielona na elemencie. Sporządzany jest także bilans mocy układu.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i plik wejściowy z obwodem

-o plik wyjściowy z wyliczonymi wartościami

-h wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

2 B-drzewo

Napisać program sortujący liczby rzeczywiste w pewnym zbiorze. Do przechowywania liczb należy wykorzystać B-drzewo, którego rząd podany jest w pierwszym wierszu pliku. Liczby podawane są w ściśle określony sposób. Liczba może być dodana lub usunięta ze zbioru.

Dodanie liczb jest realizowane przez komendę **add**, po której może wystąpić jedna lub więcej liczb (rozdzielonych białymi znakami). Komenda **remove** usuwa podaną po niej liczbę (lub liczby rozdzielone białymi znakami) ze zbioru. Komenda **print** powoduje wypisane liczb zawartych w zbiorze w porządku rosnącym. Po komendzie tej można podać nazwę pliku, wtedy wartości zostaną zapisane do tegoż pliku zamiast na standardowe wyjście.

Komenda **graph** wypisuje drzewo w postaci graficznej – głębsze poziomy drzewa są wypisywane z coraz większym wcięciem. Dodatkowo pierwszy element każdego węzła poprzedzony jest symbolem **[**, a po ostatnim umieszczony jest symbol **]**. Podobnie jak w przypadku komendy **print**, po komendzie **graph** można podać nazwę pliku do zapisu. Jeżeli komendy **print** i **graph** są użyte do zapisu do pliku, możliwe jest poprzedzenie nazwy pliku znakiem **+**. Wtedy plik nie zostanie nadpisany, ale nowa treść zostanie dopisana na końcu pliku, nie niszcząc dotychczasowej jego zawartości

Znak **%** rozpoczyna komentarz do końca linii. Każda komenda jest zapisana w osobnej linii. Niepoprawne komendy są ignorowane.

Przykładowy plik wejściowy:

```
% przykladowy plik wejsciowy
1 % rzad B–drzewa
add 1 2 3 4 5 6 7
graph % wypisane B–drzewa
remove 4
print   % wypisane liczb na ekran
add 3.45 −0.32

print test−1.txt % wypisanie liczb do pliku
remove 1
print + test−1.txt % dopisanie do pliku
```

Po komendzie **graph** zostanie wypisane drzewo:

```
        [7]
6]
        5]
        [4
[3
        2]
        [1
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i plik wejściowy

-h wyświetlenie wszystkich możliwych przełączników

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

Uwaga: Węzeł B-drzewa ma:

- k kluczy (liczb) pamiętanych w nim,
- $k + 1$ wskaźników do węzłów potomnych.

Rząd drzewa, czyli minimalny stopień drzewa, $t \geq 2$, co oznacza, że

- korzeń niepustego drzewa musi mieć co najmniej jeden klucz,
- każdy węzeł niebędący korzeniem musi mieć co najmniej $t - 1$ kluczy (zatem ma co najmniej t potomków),
- węzeł może mieć co najwyżej $2t - 1$ kluczy (zatem ma co najwyżej $2t$ potomków).

3 Cykl

Napisać program do wyznaczania cykli w grafie skierowanym. W pliku wejściowym podane są krawędzie pewnego grafu. Są one zapisane w następujący sposób:

<wierzchołek początkowy> -> <wierzchołek końcowy>

Poszczególne łuki są rozdzielone przecinkami. Przykładowy plik wejściowy:

```
2 -> 3, 1 -> 3, 3

->

3, 3
-> 3
```

W pliku wynikowym zostają zapisane znalezione cykle (każdy cykl w osobnej linii) lub informacja, że w grafie nie występują cykle

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

`-g`	plik wejściowe z grafem
`-c`	plik wyjściowy ze znalezionymi cyklami
`-h`	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

4 Czerwono-czarni

Proszę napisać program implementujący drzewo czerwono-czarne i kilka operacji z jego wykorzystaniem. Liczba może być dodana do drzewa lub z niego usunięta. Dodanie liczb jest realizowane przez komendę **add**, po której może wystąpić dowolna nieujemna liczba liczb (rozdzielonych spacjami). Komenda **remove** usuwa podane po niej liczby z drzewa. Komenda **print** powoduje wypisane liczb zawartych w drzewie w porządku rosnącym. Komenda **graph** wypisuje drzewo w postaci graficznej – głębsze poziomy drzewa są wypisywane z coraz większym wcięciem, ponadto wartości w węzłach czarnych są wypisywane w nawiasach kwadratowych, np. [13], w węzłach czerwonych – w nawiasach okrągłych, np. (13). Po komendzie **print** podawana jest nazwa pliku, do którego zostaną wypisane liczby. Jeżeli zostanie podana nazwa pliku liczby zostaną wypisane na ekran. Znak **%** oznacza komentarz do końca linii.

Przykładowy plik wejściowy:

```
% przykladowy plik wejsciowy
add -3.14 43.9 4
graph % wypisane z wcieciami i z oznaczeniami kolorow wezlow
remove 4
print   % wypisane na ekran

add 3.45 -0.32
print test-1.txt % wypisanie do pliku
add 490 32.3

print % na ekran
```

Po komendzie **graph** zostanie wypisane drzewo:

```
        (-3.14)
[4]
        (43.9)
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z populacją
-o	plik wyjściowy z populacją
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

Do przechowywania liczb należy wykorzystać drzewa czerwono-czarne.

5 Darwin

Napisać program symulujący ewolucję populacji osobników. Populacja może liczyć dowolną liczbę osobników. Każdy osobnik zawiera chromosom, który jest ciągiem liczb naturalnych. Chromosomy mogą być różnej długości. W każdym pokoleniu wylosowywanych jest k par osobników, które się następnie krzyżują. Krzyżowanie polega na tym, że u każdego osobnika dochodzi do pęknięcia chromosomu w dowolnym miejscu. Część początkowa chromosomu jednego osobnika łączy się z częścią końcową drugiego. Inaczej mówiąc: osobniki wymieniają się fragmentami swoich chromosomów. Jeden osobnik może być wylosowany do kilku krzyżowań. Po dokonaniu wszystkich krzyżowań w pokoleniu sprawdzane jest przystosowanie osobników do warunków środowiska. W tym celu dla każdego osobnika wyznaczana jest wartość $f \in [0,1]$ funkcji dopasowania. Osobniki, dla których wartość $f < w$ (gdzie w jest progiem wymierania), są usuwane z populacji. Osobniki, dla których $f > r$ (gdzie r jest progiem rozmnażania) są klonowane. A osobniki, dla których $w \leq f \leq r$ pozostają w populacji, ale się nie rozmnażają.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z populacją
-o	plik wyjściowy z populacją
-w	współczynnik wymierania $w \in [0,1]$
-r	współczynnik rozmnażania $r \in [0,1]$
-p	liczba pokoleń p
-k	liczba k par osobników losowanych do krzyżowania
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

Plik wejściowy ma następującą postać: Każda linia zawiera jednego osobnika. Osobnik charakteryzowany jest chromosomem, który jest przedstawiony jako ciąg liczb naturalnych rozdzielonych białymi znakami. Przykładowy plik wejściowy zawierający populację złożoną z czterech osobników:

```
2 9 84 9 5 6 25 12
2 98 56 2 54
5 2
8 5 22 5 48 6 1 9 8 7 554 25 235 32
```

Plik wyjściowy ma identyczny format.

6 Dijkstra

Napisać program, do znajdowania najkrótszych ścieżek między zadanym wierzchołkiem grafu a wszystkimi pozostałymi wierzchołkami tego grafu. Program wykorzystuje algorytm Dijkstry.

Plik z grafem ma następującą postać:

- każda krawędź jest podana w osobnej linii,
- przykład krawędzi skierowanej: $4->5:54.4$ oznacza krawędź skierowaną od wierzchołka 4 do 5 waga (długość) krawędzi wynosi 54.4,
- przykład krawędzi nieskierowanej: $3-6:12.5$ oznacza krawędź nieskierowaną między wierzchołkami 3 i 6 o wadze (długości) 12.5,
- w pliku mogą wystąpić puste linie,
- w linii mogą wystąpić dodatkowe (nadmiarowe) znaki białe.

Przykładowy plik z grafem:

```
3 -> 2 : 54.5
12 -> 3 : 4.5

2 ->    5 : 34.65
5       -> 3 : 2.4
3 -> 12 : 1.00
```

Drugim plikiem wejściowym programu, jest plik z numerami wierzchołków, dla których chcemy wyznaczyć najkrótsze odległości do pozostałych wierzchołków. Przykładowy plik:

```
2
6

12
```

W pliku wynikowym zostaną zapisane trasy o minimalnej długości dla zadanych wierzchołków, np.

```
wierzcholek startowy: 2
2 -> 5 -> 3 : 37.05
2 -> 5 : 34.65
2 -> 5 -> 3 -> 12 : 38.05

wierzcholek startowy: 6
brak wierzcholka 6 w grafie

wierzcholek startowy: 12
```

```
12 -> 3 : 4.5
12 -> 3 -> 2 : 59.0
12 -> 3 -> 2 -> 5 : 93.65
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

`-g`	plik wejściowy z grafem
`-w`	plik wyjściowy z wierzchołkami
`-o`	plik wyjściowy z wynikami
`-h`	wyświetlenie wszystkich możliwych przełączników.

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

7 Drzewo rozpinające

Napisać program wyszukujący w grafie minimalne drzewo rozpinające grafu ważonego. Graf jest zapisany w pliku wejściowym w postaci rozdzielonych przecinkami trójek rozdzielonych przecinkami liczb. Trójki liczb otoczone są nawiasami. Liczby oznaczają po kolei: wierzchołki definiujące krawędź i wagę krawędzi. Znalezione minimalne drzewo rozpinające zostaje zapisane w takim samym formacie do pliku wynikowego.

Przykładowy plik wejściowy:

```
(1, 2, 4.5), (4, 3, 4.5), (4, 2, 0.4)
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z grafem
-o	plik wyjściowy ze znalezionym minimalnym drzewem rozpinającym
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

8 Decyzja

Napisać program dokonujący analizy zbioru danych z wykorzystaniem drzewa decyzyjnego. Program wykorzystuje dwa pliki wejściowe: jeden zawierający zbiór danych (przykładów opisanych pewnymi atrybutami), drugi opisujący strukturę drzewa. Działanie programu polega na przyporządkowaniu każdemu przykładowi ze zbioru pewnej etykiety na podstawie drzewa, a następnie zapisaniu przyporządkowań w pliku wyjściowym. W pierwszym wierszu pliku z danymi podane są nazwy atrybutów. Kolejne wiersze pliku to tabela z danymi, której wiersze odpowiadają przykładom, a kolumny atrybutom. Wartości atrybutów to liczby rzeczywiste. Elementy w ramach pojedynczego wiersza rozdzielone są białymi znakami, a symbol **%** oznacza komentarz do końca linii. Przykładowy plik danych ma postać:

```
wzrost wyskok % dwa atrybuty
195.9 40.8
187.4 45.5
197.0 51.6
176.2 55.0
178.6 49.4
```

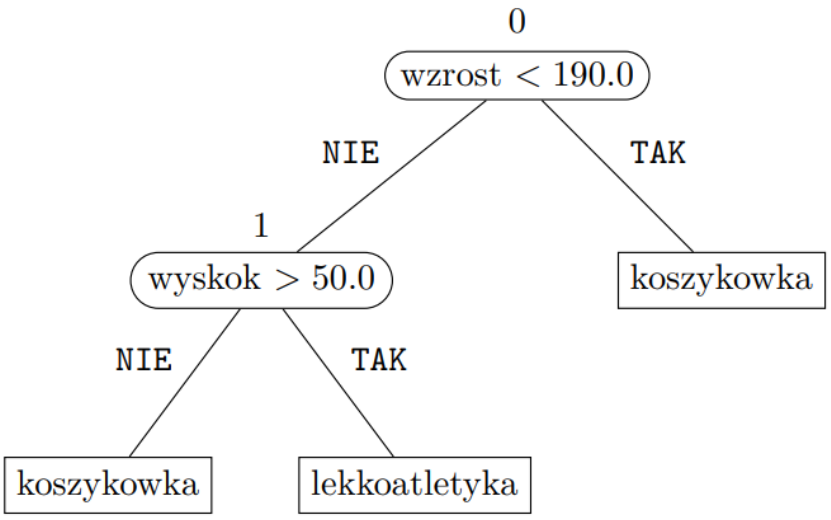
Drzewo decyzyjne to drzewo binarne, którego węzły zawierają warunki. Każdy wiersz w pliku zawiera opis węzła zgodnie z formatem:

<indeks wejścia> <warunek> <indeks wyjścia NIE> <indeks wyjścia TAK>

Indeks wejściowy 0 oznacza korzeń drzewa. Warunek ma postać **<atrybut> <operator> <wartość>**, gdzie operator to symbol **<** lub **>**. Wyjścia **NIE** i **TAK** odpowiadają sytuacji, odpowiednio, niespełnienia i spełnienia warunku. Jeżeli węzeł nie posiada któregoś z potomków, to zamiast odpowiedniego indeksu umieszczona jest tekstowa etykieta, którą należy przyporządkować przykładowi.

Przykładowy plik z opisem drzewa ma postać:

```
0 wzrost < 190.0 1 koszykowka % korzen
1 wyskok > 50.0 koszykowka lekkoatletyka
```



W pliku wyjściowym mają znaleźć się przykłady pogrupowane ze względu na etykietę. Porządek grup oraz przykładów w ramach grupy jest dowolny.

```
koszykowka
176.2  55.0
195.9  40.8
197.0  51.6

lekkoatletyka
187.4  45.5
178.6  49.4
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy
-t	plik drzewa
-o	plik wyjściowy
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

9 Dwudzielny

Napisać program, do sprawdzania, czy graf nieskierowany jest dwudzielny. Plik z grafem ma następującą postać:

- Każda krawędź jest podana w osobnej linii; podane są dwa wierzchołki, które łączy krawędź
- W pliku mogą wystąpić puste linie.
- W linii mogą wystąpić dodatkowe (nadmiarowe) znaki białe.

Przykładowy plik z grafem:

```
3  2
12  3
2  5
          5  3

3  12
```

Program wypisuje do pliku wyjściowego zadany graf i komunikat, czy jest to graf dwudzielny, czy nie.Jeżeli zadany graf jest dwudzielny, program wypisuje wierzchołki obu grup grafu.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z krawędziami grafu
-o	plik wyjściowy z wynikami
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

10 Huffman

Napisać program do kompresji plików metodą **Huffmanna**.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy,
-o	plik wyjściowy,
-k	tryb kompresji,
-d	tryb dekompresji,
-s	plik ze słownikiem (tworzonym w czasie kompresji, używanym w czasie dekompresji),
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję.

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

11 Jarvis-Patrick

Napisać program grupujący dane algorytmem Jarvisa-Patricka. Dane zapisane sa w pliku wejściowym w następującym formacie (znak % rozpoczyna komentarz do końca linii):

```
25              % liczba punktow danych N
3               % liczba wymiarow D
% ponizej znajduje sie lista N punktow danych,
% kazdy opisany przez D wspolrzednych
1.5  7.6  5.3   % punkt 1
4  45.3  4.6    % punkt 2
0.2  -3.7  22.9 % punkt 3
...
-6  5.7  -0.8   % punkt 25
```

Program przyporządkowuje każdemu punktowi indeks grupy $g \in 1, 2, ..., k$, gdzie k to parametr algorytmu oznaczający liczbę grup. Indeksy zapisywane są w pliku wyjściowym zgodnym z formatem:

```
2       % indeks grupy punktu 1
3       % indeks grupy punktu 2
2       % indeks grupy punktu 3
...
1       % indeks grupy punktu 25
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-k	parametr K algorytmu
-i	plik wejściowy
-o	plik wyjściowy
-Kmin	parametr K_{min} algorytmu
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

12 Komiwojażer

Napisać program rozwiązujący problem komiwojażera algorytmem genetycznym. Plik wejściowy zawiera odległości pomiędzy miastami zapisane w postaci macierzy, np.:

```
A  B  C  D
0  10  15  5
11  0  9  7
6  -1  0  6
6  14  8  0
```

Uwaga: macierz nie musi być symetryczna, a wartość -1 oznacza brak połączenia. Problem komiwojażera polega na znalezieniu najkrótszej trasy prowadzącej przez wszystkie miasta i powracającej do miasta źródłowego. Przykładowa trasa ma postać:

```
D B C A D
```

W ramach projektu wymagane jest zaimplementowanie algorytmu genetycznego z operatorami selekcji i krzyżowania (mutację można pominąć). Należy pamiętać, że algorytm genetyczny nie daje gwarancji znalezienia optymalnego rozwiązania (ale często jest ono wystarczająco dobre do praktycznego zastosowania). Plik wyjściowy ma zawierać najlepsze rozwiązania problemu w kolejnych generacjach algorytmu, np.:

```
generacja 1, dlugosc 34
D B C A D

generacja 2, dlugosc 33
D A B C D

...
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z macierzą odległości
-o	plik wyjściowy z najlepszymi rozwiązaniami w kolejnych pokoleniach
-g	liczba generacji
-n	liczba osobników w ramach generacji

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

13 K-średnich

Napisać program grupujący dane algorytmem *k*-średnich. Dane zapisane sa w pliku wejściowym w następującym formacie (znak % rozpoczyna komentarz do końca linii):

```
25              % liczba punktow danych N
3               % liczba wymiarow D
% ponizej znajduje sie lista N punktow danych,
% kazdy opisany przez D wspolrzednych
1.5  7.6  5.3   % punkt 1
4  45.3  4.6    % punkt 2
0.2  -3.7  22.9 % punkt 3
...
-6  5.7  -0.8   % punkt 25
```

Program przyporządkowuje każdemu punktowi indeks grupy $g \in 1, 2, ..., k$, gdzie k to parametr algorytmu oznaczający liczbę grup. Indeksy zapisywane są w pliku wyjściowym zgodnym z formatem:

```
2       % indeks grupy punktu 1
3       % indeks grupy punktu 2
2       % indeks grupy punktu 3
...
1       % indeks grupy punktu 25
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-k	liczba grup docelowych k
-i	plik wejściowy
-o	plik wyjściowy
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

14 Kurier

Kurier ma za zadanie zawieść towar do klientów w różnych lokalizacjach i powrócić do miejsca, z którego wyjechał. Kurier musi odwiedzić każdego klienta raz i tylko raz. Należy znaleźć zamkniętą najkrótszą drogę, która umożliwia odwiedzenie wszystkich klientów. W pliku wejściowym zapisane są długości dróg pomiędzy miastami. Drogi zapisane są w następujący sposób:

<klient A> - <klient B>: <odległość>

Niektóre drogi nie są symetryczne, tzn. jest pewna różnica między drogą tam a z powrotem. Zapis:

<klient C> - <klient D>: <odległość CD>

oznacza długość drogi jednokierunkowej od klienta **C** do klienta **D**. Poszczególne drogi są rozdzielone przecinkami. Nie jest podana liczba dróg. Jeżeli nie jest możliwe wyznaczenie drogi, program zgłasza odpowiedni komunikat.

Przykładowy plik wejściowy:

```
(1 – 2 : 4.5), (4 –> 3:
4.5),
(4 – 2: 0.4)
```

W pliku wynikowym należy zapisać drogę kuriera (kolejność odwiedzania klientów i długość drogi).

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowe z drogami między klientami
-d	plik wyjściowy ze znalezioną drogą kuriera
-n	liczba osobników w ramach generacji

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

15 LZ77

Napisać program do kompresji plików algorytmem LZ77.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy
-o	plik wyjściowy
-k	tryb kompresji
-d	tryb dekompresji
-n	rozmiar bufora wejściowego (maks. długość dopasowania)
-k	rozmiar bufora historii
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

16 LZ77

Napisać program do kompresji plików algorytmem LZSS.

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy
-o	plik wyjściowy
-k	tryb kompresji
-d	tryb dekompresji
-n	rozmiar bufora wejściowego (maks. długość dopasowania)
-k	rozmiar bufora historii
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

17 Mapa

Napisać program, który umożliwia znalezienie najkrótszej trasy między dwoma miastami. Miasta połączone są drogami o pewnej długości. Drogi są jednokierunkowe. Plik mapy dróg ma następującą postać:

<miasto początkowe> <miasto końcowe> <odległość>

Przykładowy plik dróg (liczba dróg nie jest ograniczona):

```
Katowice Krakow 70
Krakow Tarnow 70
Tarnow Jaslo 50
Katowice Gliwice 22
Lodz Poznan 205
Gliwice Katowice 22
Katowice Czestochowa 70
Czestochowa Lodz 120
Lodz Torun 165
Krakow Katowice 70
Gliwice Wroclaw 180
```

Drugim plikiem wejściowym jest plik z trasami do wyznaczenia. Każda linia pliku zawiera jedną trasę w postaci:

<miasto początkowe> <miasto końcowe>

Przykładowy plik tras do wyznaczenia (liczba tras nie jest ograniczona):

```
Katowice Torun
Krakow Poznan
Tarnow Wroclaw
```

Wynikiem działania programu jest plik wyjściowy z wyznaczonymi trasami, tzn. podana jest nazwa trasy, całkowita długość, a potem poszczególne odcinki z długościami, np.

```
trasa: Katowice --> Torun (355 km):
Katowice --> Czestochowa 70
Czestochowa --> Lodz 120
Lodz --> Torun 165
trasa: Krakow --> Poznan (465 km):
Krakow --> Katowice 70
Katowice --> Czestochowa 70
Czestochowa --> Lodz 120
Lodz --> Poznan 205
trasa: Tarnow --> Wroclaw
TRASA NIEMOZLIWA DO WYZNACZENIA
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-d plik wejściowy z drogami

-t plik wejściowy z trasami do wyznaczenia

-o plik wynikowy z wyznaczony trasami

-h wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

18 Plecak

Napisać program rozwiązujący problem plecakowy algorytmem genetycznym. Plik wejściowy opisuje zbiór przedmiotów wraz z ich wagą oraz wartością, np.:

```
telewizor 10.0 3300
kubek 0.5 20
laptop 2.2 4000
monitor 4.5 1500
tablet 0.5 900
pralka 35.0 2200
aparat 1.2 3500
czajnik 1.5 130
```

Celem jest wybór przedmiotów o największej sumarycznej wartości i sumarycznej wadze nieprzekraczającej ładowności plecaka. Przykładowo, dla plecaka o ładowności 2.5kg, optymalne rozwiązanie ma postać:

```
tablet 0.5 900
aparat 1.2 3500
```

W ramach projektu wymagane jest zaimplementowanie algorytmu genetycznego z operatorami selekcji i krzyżowania (mutację można pominąć). Należy pamiętać, że algorytm genetyczny nie daje gwarancji znalezienia optymalnego rozwiązania (ale często jest ono wystarczająco dobre do praktycznego zastosowania). Plik wyjściowy ma zawierać najlepsze rozwiązania problemu w kolejnych generacjach algorytmu, np.:

```
generacja 1, waga 2.0, wartosc 150:
kubek 0.5 20
czajnik 1.5 130

generacja 2, waga 1.7, wartosc 3520:
kubek 0.5 20
aparat 1.2 3500

...
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

`-i`	plik wejściowy opisujący zbiór przedmiotów
`-o`	plik wyjściowy z najlepszymi rozwiązaniami w kolejnych pokoleniach
`-p`	ładowność plecaka
`-g`	liczba generacji
`-n`	liczba osobników w ramach generacji
`-h`	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

19 Sąsiedzi

Napisać program klasyfikujący dane algorytmem *k* najbliższych sąsiadów. Dane treningowe zapisane są w pliku wejściowym w następującym formacie (znak % rozpoczyna komentarz do końca linii):

```
100     % liczba punktow treningowych N
2       % liczba wymiarow D

        % ponizej znajduje sie lista N punktow danych,
        % kazdy opisany przez D wspolrzednych
        % oraz etykiete klasy bedaca lancuchem tekstowym
7.6  5.3  klasa_B       % punkt 1
45.3  4.6  klasa_C      % punkt 2
-3.7  22.9  klasa_A     % punkt 3
...
5.7  -0.8  klasa_C      % punkt 100
```

Dane testowe zapisane są w analogicznym pliku, ale bez etykiet klas:

```
10      % liczba punktow testowych M
3       % liczba wymiarow D
        % ponizej znajduje sie lista M punktow danych,
        % kazdy opisany przez D wspolrzednych
1.5  12.3       % punkt 1
4  7.6          % punkt 2
0.2  -0.9       % punkt 3
...
-6  3.15        % punkt 10
```

Wykorzystując zbiór treningowy, program przyporządkowuje każdemu punktowi ze zbioru testowego etykietę klasy metodą *k* najbliższych sąsiadów (*k* to parametr algorytmu). Plik wyjściowy posiada format analogiczny do pliku testowego, ale z uzupełnionymi etykietami klas:

```
10      % liczba punktow testowych
3       % liczba wymiarow D
1.5  12.3  klasa_C      % punkt 1
4  7.6  klasa_A         % punkt 2
0.2  -0.9  klasa_A       % punkt 3
...
-6  3.15  klasa_C       % punkt 10
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-train plik treningowy

-test plik testowy

`-out` plik wyjściowy

`-k` liczba najbliższych sąsiadów k ze zbioru treningowego

`-h` wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

20 Spedycja

Firma spedycyjna przyjmuje towary w centrali, skąd rozsyła do odbiorców w całym kraju. Trasy między miastami są zapisane w pliku w następujący sposób:

<miasto> <miasto> <odległość>

Przykładowy plik:

```
Szczecin Poznan 220
Szczecin Koszalin 110
Poznan Bytom 300
Poznan Lodz 130
Lodz Katowice 170
Bytom Katowice 15
Bytom Wroclaw 180
```

W wyniki działania programu zostanie tworzony plik z trasami spedycyjnymi w następującej postaci (przykład dla centrali w Poznaniu):

```
Poznan -> Bytom: 300
Poznan -> Lodz -> Katowice: 300
Poznan -> Szczecin -> Koszalin: 330
Poznan -> Lodz: 130
Poznan -> Szczecin: 220
Poznan -> Bytom -> Wroclaw: 480
```

Trasy spedycyjne mają najkrótsze możliwe długości. Należy wykorzystać algorytm Dijkstry.

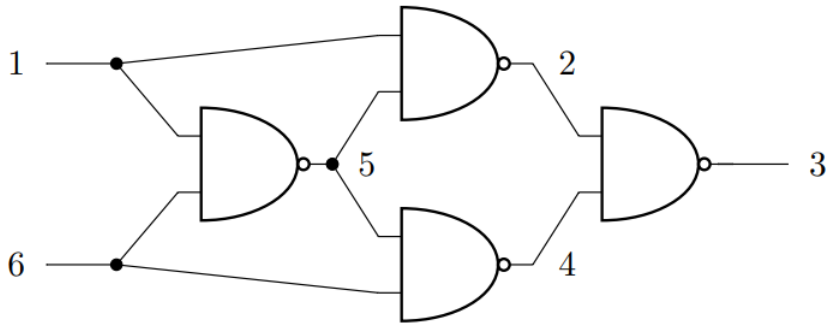
Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-i	plik wejściowy z drogami
-o	plik wyjściowy z trasami spedycyjnymi
-s	nazwa miasta startowego, gdzie znajduje się centrala
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

21 TUC

Napisać program symulujący działanie układu złożonego z bramek logicznych. Dostępne są następujące bramki: **and**, **nand**, **or**, **nor**, **xor**, **xnor**, **neg**. Każda bramka ma jedno wyjście i dwa wejścia. Jedynym wyjątkiem jest bramka **neg**, która ma jedno wejście i jedno wyjście. Połączenie wejść i wyjść bramek jest traktowane jako węzeł. Na poniższym rysunku zaznaczono węzły prostego układu.



Plik wejściowy przedstawiający układ na następujący format: W pierwszej linii podane są numery węzłów będących wejściem układu. W drugiej linii numery węzłów będące wyjściem układu. Każda następna linia zawiera opis jednej bramki w postaci:

<węzeł wejściowy> <węzeł wejściowy> <węzeł wyjściowy>

Plik z układem dla powyższego rysunku ma postać:

```
IN: 1 6
OUT: 3
NAND 1 6 5
NAND 1 5 2
NAND 5 6 4
NAND 2 4 3
```

Drugi plik wejściowy zawiera w każdej linii stany wejść, dla których należy znaleźć stan wyjść:

```
1:0 6:0
1:0 6:1
1:1 6:0
1:1 6:1
```

Plik wynikowy podaje wartości wyjść dla zadanych stanów wejść:

```
IN: 1:0 6:0 OUT: 3:0
IN: 1:0 6:1 OUT: 3:1
IN: 1:1 6:0 OUT: 3:1
IN: 1:1 6:1 OUT: 3:0
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

-u plik wejściowy z układem

-i plik wejściowy ze stanami wejść

-o plik wyjściowy ze stanami wyjść

-h wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.

22 Vigenère

Proszę napisać program, który:

1. szyfruje pliki tekstowe metodą Vigenère'a:

   Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników (kolejność przełączników jest dowolna):

–en	flaga szyfrowania
-i	plik tekstowy wejściowy jawny
-o	plik tekstowy wyjściowy zaszyfrowany
-k	plik tekstowy z kluczem
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję.

   przykładowo: **./main –en -i jawny -k klucz -o zaszyfrowany**

2. deszyfruje pliki tekstowe zaszyfrowane metodą Vigenère'a:

–de	flaga szyfrowania
-i	plik tekstowy wyjściowy zaszyfrowany
-o	plik tekstowy wejściowy jawny
-k	plik tekstowy z kluczem
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję.

   przykładowo: **./main –de -k klucz -o odszyfrowany -i tajny**

3. łamie pliki tekstowe zaszyfrowane metodą Vigenère'a:

–br	flaga łamania szyfru
-i	plik tekstowy wejściowy zaszyfrowany
-o	plik tekstowy wyjściowy jawny
-h	wyświetlenie wszystkich możliwych przełączników oraz krótką instrukcję

   przykładowo: **./main –br -o odszyfrowany -i tajny -k klucz**

Uruchomienie programu bez parametrów powoduje wypisanie tej samej instrukcji, co w przypadku uruchomienia programu z przełącznikiem *-h*.