# Tematy części praktycznej egzaminu

**Podstawy programowania komputerów**

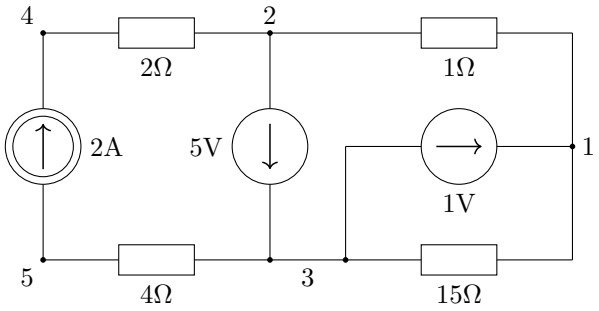
23 listopada 2022

## Spis treści

# 1 ALOPS

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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:

`-i` plik wejściowy z obwodem

`-o` plik wyjściowy z wyliczonymi wartościami

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 2 B-drzewo

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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 `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.

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

Przykładowy plik wejściowy:

```
% przykladowy plik wejsciowy
1 % rząd B-drzewa
add 6 3 7 5 1 4 2
graph % wypisane B-drzewa
print     % wypisane liczb na ekran
add 3.45 -0.32
print test-1.txt  % wypisanie liczb do pliku
```

Po komendzie `graph` zostanie wypisane drzewo:

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

Program uruchamiany jest z linii poleceń z wykorzystaniem jednego przełącznika:

`-i` plik wejściowy

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

**Uwaga:** Węzeł B-drzewa ma:

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

Rząd drzewa, czyli minimalny stopień drzewa, $t \geqslant 2$, 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

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

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

⟨wierzcholek poczatkowy⟩ `->` ⟨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:

`-g` plik wejściowe z grafem

`-c` plik wyjściowy ze znalezionymi cyklami

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 4 Darwin

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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 $w \leqslant f \leqslant r$ pozostają w populacji, ale się nie rozmnażają.

Funkcja oceny osobnika jest ustawiana w funkcji `main` przez wskaźnik lub jako funkcja lambda.

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

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

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.

# 5 Decyzja

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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
```

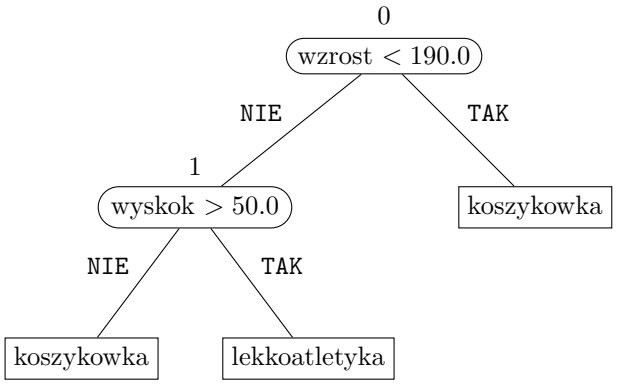
Drzewo decyzyjne to drzewo binarne, którego węzły zawierają warunki. Każdy wiersz w pliku zawiera opis węzła zgodnie z formatem:

```
<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 % korzeń
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:

- `-i` plik wejściowy
- `-t` plik drzewa
- `-o` plik wyjściowy

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 6 Dijkstra

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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 wejściowy z wierzchołkami
- `-o` plik wyjściowy z wynikami

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 7 Dwudzielny

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 8 Jarvis-Patrick

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę napisać napisać program grupujący dane algorytmem, który polega na przypisaniu tej samej grupy (klasy, klastra) punktom, których mają $w$ wspólnych sąsiadów spośród $n$ najbliższych sąsiadów każdego z punktów. Jeżeli dla dwóch punktów $A$ i $B$ wyznaczymy $n$ najbliższych sąsiadów i spośród tych sąsiadów przynajmniej $w$ jest wspólnych, to punkty $A$ i $B$ należą do tej samej grupy (klasy, klastra).

**Przykład 8.1.** Rozpatrzmy dwa punkty $A$ i $B$. Niech $n = 5$ i $w = 2$. $n$ najbliższych sąsiadów punktu $A$: $S_A = \{D, E, F, G, H\}$, punktu $B$: $S_B = \{E, G, K, H, J\}$. Punkty $A$ i $B$ należą do tej samej grupy (klasy, klastra), bo mają przynajmniej $w = 2$ wspólnych najbliższych sąsiadów.

**Przykład 8.2.** Rozpatrzmy dwa punkty $A$ i $C$. Niech $n = 5$ i $w = 2$. $n$ najbliższych sąsiadów punktu $A$: $S_A = \{D, E, F, G, H\}$, punktu $C$: $S_C = \{B, H, J, K, Q\}$. Punkty $A$ i $C$ należą do różnych grup (klas, klastrów), bo mają mniej niż $w = 2$ wspólnych najbliższych sąsiadów.

Dane zapisane sa w pliku wejściowym w następującym formacie (znak `%` rozpoczyna komentarz do końca linii): Każdy punkt zapisany jest w jednej linii, współrzędne punktu rozdzielone są białymi znakami, np.:

```
% poniżej znajduje się lista punktów danych, każdy opisany przez współrzędne
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
```

Liczba punktów w pliku może być dowolna. Liczba współrzędnych jest dowolna, ale taka sama dla wszystkich punktów. Program korzysta z metryki euklidesowej. Program przyporządkowuje każdemu punktowi numer grupy $g \in \{1, 2, \ldots\}$. Indeksy zapisywane są w pliku wyjściowym zgodnym z formatem: numer grupy, a następnie współrzędne punktu. Każdy punkt jest zapisany w osobnej linii.

```
2   1.5 7.6 5.3
3   4 45.3 4.6
2   0.2 -3.7 22.9
...
1   -6 5.7 -0.8
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-i` plik wejściowy
- `-o` plik wyjściowy
- `-n` liczba najbliższych sąsiadów
- `-w` liczba wspólnych najbliższych sąsiadów

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 9 $k$-NN

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

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

```
% poniżej znajduje się lista punktów danych, każdy opisany przez D współrzędnych
% oraz etykietę klasy będącą łańcuchem tekstowym
klasa_B 7.6 5.3        % punkt 1
klasa_C 45.3 4.6       % punkt 2
klasa_A -3.7 22.9      % punkt 3
...
klasa_C 5.7 -0.8       % punkt 100
```

Liczba punktów w pliku może być dowolna. Liczba współrzędnych jest dowolna, ale taka sama dla wszystkich punktów. Dane testowe zapisane są w analogicznym pliku, ale bez etykiet klas:

```
% poniżej znajduje się lista punktów danych, każdy opisany przez D współrzędnych
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 ma format analogiczny do pliku testowego, ale z uzupełnionymi etykietami klas:

```
klasa_C 1.5  12.3
klasa_A 4  7.6
klasa_A 0.2  -0.9
...
klasa_C -6  3.15
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-train` plik treningowy
- `-test` plik testowy
- `-out` plik wyjściowy
- `-k` liczba najbliższych sąsiadów $k$ ze zbioru treningowego

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 10 Geny

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę napisać program rozwiązujący problem komiwojażera algorytmem genetycznym. Problem komiwojażera polega na znalezieniu trasy rozpoczynającej się w jednym punkcie, przechodzącej przez wszystkie punkty dokładnie raz i kończącej się w punkcie początkowym. Ściślej: problem komiwojażera polega na znalezieniu cyklu Hamiltona o minimalnej sumie wag krawędzi w pełnym grafie ważonym.

Plik wejściowy zawiera numery miast i ich współrzędne, np.:

```
0  10.5 15.1
1  56.5 32.7
2  34.2 87.9
3  12.7 24.9
4  45.8 14.6
```

Przykładowa trasa ma postać:

```
0 3 2 4 1 0
```

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

```
pokolenie 1, dlugosc 34
0 3 2 1 4 0

pokolenie 2, dlugosc 33
0 2 1 3 4 0

...
```

Bez straty ogólności można przyjąć, że trasy zaczynają się i kończą w punkcie 0. Możemy tak przyjąć, bo trasy są cyklami i trasa `0 1 2 3 4 0` jest taka sama jak `1 2 3 4 0 1` i inne przesunięcia cykliczne. Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-i` plik wejściowy z punktami
- `-o` plik wyjściowy z najlepszymi rozwiązaniami w kolejnych pokoleniach
- `-g` liczba pokoleń
- `-n` liczba osobników w pokoleniu

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 11 Cząsteczki

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę napisać program wyszukujący optimum zadanej funkcji metodą roju cząsteczek. Funkcja (np. Rastrigina, Rosenbrocka, Himmelblaua, ...) jest podana w funkcji `main` jako funkcja lambda. Powinno być możliwe zdefiniowane funkcji o dowolnej liczbie parametrów.

Plik wyjściowy zawiera najlepsze rozwiązania problemu w kolejnych iteracjach algorytmu, np.:

```
iteracja 1, f(34.5, 45.3, 56.5, 4.1) = 45.5
iteracja 2, f(35.4, 42.1, 50.7, 4.2) = 42.1
...
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-o` plik wyjściowy z najlepszymi rozwiązaniami w kolejnych iteracjach
- `-g` liczba iteracji
- `-n` liczba cząstek
- `-b` bezwładność
- `-k` współczynnik indywidualny (kognitywny)
- `-s` współczynnik globalny (socjalny)
- `-min` minimalizacja
- `-max` maksymalizacja

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 12 Mrówki

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę napisać program rozwiązujący problem komiwojażera algorytmem mrówkowym. Problem komiwojażera polega na znalezieniu trasy rozpoczynającej się w jednym punkcie, przechodzącej przez wszystkie punkty dokładnie raz i kończącej się w punkcie początkowym. Ściślej: problem komiwojażera polega na znalezieniu cyklu Hamiltona o minimalnej sumie wag krawędzi w pełnym grafie ważonym.

Plik wejściowy zawiera numery miast i ich współrzędne, np.:

```
0  10.5 15.1
1  56.5 32.7
2  34.2 87.9
3  12.7 24.9
4  45.8 14.6
```

Przykładowa trasa ma postać:

```
0 3 2 4 1 0
```

W ramach projektu wymagane jest zaimplementowanie algorytmu mrówkowego. Należy pamiętać, że algorytm mrówkowy nie daje gwarancji znalezienia optymalnego rozwiązania (ale często jest ono wystarczająco dobre do praktycznego zastosowania). Plik wyjściowy zawiera długości tras kolejnych mrówek, np.:

```
mrówka 1, dlugosc 34
0 3 2 1 4 0

mrówka 2, dlugosc 33
0 2 1 3 4 0

...
```

Bez straty ogólności można przyjąć, że trasy zaczynają się i kończą w punkcie 0. Możemy tak przyjąć, bo trasy są cyklami i trasa `0 1 2 3 4 0` jest taka sama jak `1 2 3 4 0 1` i inne przesunięcia cykliczne.
Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

- `-i` plik wejściowy ze współrzędnymi punktów
- `-o` plik wyjściowy z trasami kolejnych mrówek
- `-g` liczba mrówek
- `-f` współczynnik $f \in [0, 1]$ parowania feromonu

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 13 Plecak

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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 2.5 kg, 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 pokoleniach algorytmu, np.:

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

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

...
```

Program uruchamiany jest z linii poleceń z wykorzystaniem następujących przełączników:

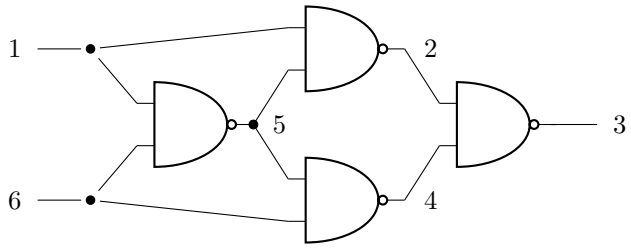
- `-i` plik wejściowy opisujący zbiór przedmiotów
- `-o` plik wyjściowy z najlepszymi rozwiązaniami w kolejnych pokoleniach
- `-p` ładowność plecaka
- `-g` liczba pokoleń
- `-n` liczba osobników w pokoleniu

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 14 TUC

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę 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⟩ ⟨wezeł 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

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.

# 15 Vigenère

1. **Przed implementacją konieczna jest akceptacja struktury danych przez prowadzącego zajęcia.**
2. **Program powinien być podzielony na pliki z deklaracjami (`*.h`) i definicjami (`*.cpp`).**
3. **Wszystkie funkcje muszą być skomentowane w doxygenie.**
4. **Program musi obsługiwać różne permutacje parametrów linii poleceń.**
5. **Uruchomienie programu bez parametrów powoduje wyświetlenie krótkiej pomocy.**
6. **Program musi być odporny na pewne niedoskonałości formatu danych wejściowych.**

Proszę napisać program, który łamie szyfr Vigenère'a. Oznacza to, że szyfr wyznacza klucz, którym się posłużono do zaszyfrowania tekstu i odszyfrowuje tekst.

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

- `-i` plik wejściowy z tekstem zaszyfrowanym
- `-w` plik wejściowy z tekstem w języku, w którym jest tekst zaszyfrowany
- `-k` plik wyjściowy ze znalezionym kluczem
- `-o` plik wyjściowy z odszyfrowanym tekstem

Uruchomienie programu bez parametrów powoduje wypisanie krótkiej informacji o tym, jak użyć programu.