# Specyfikacja funkcjonalna projektu 1

Sebastian Pietrykowski, Paweł Borkowski
Grupa projektowa nr 3

## 1 Cel projektu

Celem projektu będzie napisanie programu, który znajduje najkrótszą możliwą ścieżkę pomiędzy dwoma wybranymi wierzchołkami oraz sprawdza, czy graf jest spójny. Potrafi on generować grafy o zadanej liczbie kolumn, wierszy oraz zakresie wartości, z którego będzie losowana waga krawędzi, podawanych przez użytkownika. Program dodatkowo wyposażony jest w możliwość zapisu wygenerowanego grafu do pliku oraz odczytu grafu z takiego pliku. W programie wykorzystywane są dwa algorytmy:

- Algorytm Dijkstry - algorytm dzięki, któremu wyszukiwana jest najkrótsza ścieżka.
- Algorytm BFS - algorytm umożliwiający sprawdzenie czy graf jest spójny.

Program działa w trybie wsadowym.

## 2 Dane wejściowe

Przy uruchamianiu programu możemy podać :

- plik tekstowy z zapisanym grafem

Przykładowy zapis grafu umieszczony jest poniżej:

```
7 4
  1 :0.8864916775696521 4 :0.2187532451857941
  5 :0.2637754478952221 2 :0.6445273453144537 0 :0.4630166785185348
  6 :0.8650384424149676 3 :0.42932761976709255 1 :0.6024952385895536
  7 :0.5702072705027322 2 :0.86456124269257
  8 :0.9452864187437506 0 :0.8961825862332892 5 :0.9299058855442358
  1 :0.5956443807073741 9 :0.31509645530519625 6 :0.40326574227480094 4 :0.44925728962449873
  10 :0.7910000224849713 7 :0.7017066711437372 2 :0.20056970253149548 5 :0.3551383541997829
  6 :0.9338390704123928 3 :0.797053444490967 11 :0.7191822139832875
  4 :0.7500681437013168 12 :0.5486221194511974 9 :0.25413610146892474
  13 :0.8647843756083231 5 :0.8896910556803207 8 :0.4952122733888106 10 :0.40183865613683645
  14 :0.5997502519024634 6 :0.5800735782304424 9 :0.7796297161425758 11 :0.3769093717781341
  15 :0.3166804339669712 10 :0.14817882621967496 7 :0.8363991936747263
  13 :0.5380334165340379 16 :0.8450927265651617 8 :0.5238810833905587
  17 :0.5983997022381085 9 :0.7870744571266874 12 :0.738310558943156 14 :0.45746700405234864
  10 :0.8801737147065481 15 :0.6153113201667844 18 :0.2663754517229303 13 :0.22588495147495308
  19 :0.9069409600272764 11 :0.7381164412958352 14 :0.5723418590602954
  20 :0.1541384547533948 17 :0.3985282545552262 12 :0.29468967639003735
  21 :0.7576872377752496 13 :0.4858285745038984 16 :0.28762266137392745 18 :0.6264588252010738
  17 :0.6628790185051667 22 :0.9203623808816617 14 :0.8394013782615275 19 :0.27514794195197545
  18 :0.6976948178131532 15 :0.4893608558927002 23 :0.5604145612239925
  24 :0.8901867253885717 21 :0.561967244435089 16 :0.35835658210649646
  17 :0.8438726714274797 20 :0.3311114339467634 25 :0.7968809594947989 22 :0.9281943906422196
  21 :0.6354858042070723 23 :0.33441278736675584 18 :0.43027465583738667 26 :0.3746522679684584
  27 :0.8914256412658524 22 :0.8708278171237049 19 :0.4478162295166256
```

20 :0.35178269705930043 25 :0.2054048551310126
21 :0.6830700124292063 24 :0.3148089827888376 26 :0.5449034876557145
27 :0.2104213229517653 22 :0.8159939689806697 25 :0.4989269533310492
26 :0.44272335750313074 23 :0.4353604625664018

Dwie cyfry na początku pliku oznaczają odpowiednio liczbę kolumn oraz liczbę wierszy.
Każda następna linia reprezentuje wierzchołek, od którego zaczyna się krawędź, rozpoczynając numerację od wierzchołka nr 0. W liniach podane są numery wierzchołków, w których kończy się krawędź. Po odstępie podane są wagi dla konkretnych krawędzi, zaczynające się od przedrostka ":". Każdy wpis w linii oddzielony jest odstępem.

Dla przykładu:
Linia 2: 1 :0.8864916775696521 4 :0.2187532451857941 - Oznacza to, że wagi krawędzi, które łączą wierzchołki 0 z 1 oraz 0 z 4 wynoszą odpowiednio 0.8864916775696521 oraz 0.2187532451857941.

Uwagi:
1. Wierzchołki w grafie numerujemy od lewej do prawej zaczynając od numeru 0.
2. Wagi w grafie nie mogą być ujemne.

# 3 Argumenty wywołania programu

Program akceptuje następujące argumenty wywołania:

- -i input-file nazwa pliku z danymi wejściowymi; jeżeli określony, nie generuje pliku z grafem, określonym w -o output-file;
- -o output-file nazwa pliku przechowującego wygenerowany graf; pomijany jeżeli wczytano plik z danymi wejściowymi w -i input-file;
  Jeżeli nie podano ani -i input-file, ani -o output-file, to następuje generowanie grafu;
- -c columns liczba kolumn w generowanym grafie; domyślna wartość columns=5;
- -r rows liczba wierszy w generowanym grafie; domyślna wartość rows=5;
- -f from-weight dolna granica wagi generowanej dla krawędzi w generowanym grafie; program generuje wagi w zakresie (from-weight,to-weight); domyślna wartość from-weight=0;
- -t to-weight górna granica wagi generowanej dla krawędzi w generowanym grafie; program generuje wagi w zakresie (from-weight,to-weight); domyślna wartość to-weight=1;
- -m 1|2|3 (mode) pozwala na wybór trybu działania programu:

  1. W przypadku generowania grafu: program generuje graf z wszystkimi możliwymi krawędziami (pomiędzy punktami sąsiadującymi poziomo lub pionowo) oraz z losowymi wagami. W przypadku czytania grafu: jeżeli graf nie posiada wszystkich możliwych krawędzi (pomiędzy punktami sąsiadującymi poziomo lub pionowo), program kończy działanie;
  2. W przypadku generowania grafu: program generuje graf spójny z losowymi wagami. W przypadku czytania grafu: jeżeli graf jest niespójny, program kończy działanie;
  3. W przypadku generowania grafu: program generuje graf z losowo występującymi krawędziami (spójny lub niespójny) oraz losowymi wagami. W przypadku czytania grafu: nie wpływa na działanie programu;

  Domyślna wartość mode=3.

- -s start-vertex-number -e end-vertex-number określa wierzchołki, pomiędzy którymi ma zostać wyznaczona najkrótsza możliwa ścieżka - od wierzchołka start-vertex-number do wierzchołka end-vertex-number;

- `-n 0|1` (check-connectivity) określa, czy program ma sprawdzić spójność grafu dla użytkownika; 0 - nie, 1 - tak; domyślna wartość `0`;
- `-p 0|1` (print-weights) określa, czy wypisane mają zostać wagi dla krawędzi w najkrótszej możliwej ścieżce; 0 - nie, 1 - tak; domyślna wartość `1`;

Ogólny schemat wywołania:
./projekt1 [ -i input-file | [ [-o output-file] [ [-c columns] [-r rows] [-f from-weight] [-t to-weight] ] ] ] [-m 1|2|3] [-s start-vertex-number -e end-vertex-number] [-n 0|1] [-p 0|1]

Przykładowe wywołania programu:

- ./projekt1 -o plik.txt -c 10 -r 10 -f 0 -t 2 -m 1 -n 0 -s 0 -e 12
  Program wygeneruje graf do pliku plik.txt o 10 kolumnach i 10 rzędach oraz wagach w zakresie (0,2). Wybrano -m 1, więc graf wygeneruje się z wszystkimi możliwymi krawędziami (pomiędzy punktami sąsiadującymi poziomo lub pionowo) oraz z losowymi wagami. Program nie wydrukuje komunikatu, czy graf jest spójny. Znajdzie najkrótszą możliwą ścieżkę z wierzchołka 0 do wierzchołka 12. Domyślnie wydrukuje wagi dla krawędzi w najkrótszej możliwej ścieżce.
- ./projekt1 -o plik.txt -c 7 -r 9 -m 3 -n 1 -s 3 -e 20 -p 0
  Program wygeneruje graf do pliku plik.txt o 7 kolumnach i 9 rzędach oraz wagach w domyślnym zakresie (0,1). Wybrano -m 3, więc graf wygeneruje się z losowo występującymi krawędziami (spójny lub niespójny) oraz z losowymi wagami. Program wydrukuje komunikat, czy graf jest spójny. Znajdzie najkrótszą możliwą ścieżkę z wierzchołka 3 do wierzchołka 20. Nie wydrukuje wag dla krawędzi w najkrótszej możliwej ścieżce.
- ./projekt1 -i plik.txt -m 1 -n 0 -s 1 -e 8
  Program odczyta graf z pliku plik.txt. Jeżeli graf nie posiada wszystkich możliwych krawędzi (pomiędzy punktami sąsiadującymi poziomo lub pionowo), to program przerwie działanie. Program nie wydrukuje komunikatu, czy graf jest spójny. Znajdzie najkrótszą możliwą ścieżkę z wierzchołka 1 do wierzchołka 8. Domyślnie wydrukuje wagi dla krawędzi w najkrótszej możliwej ścieżce.
- ./projekt1 -i plik.txt -m 3 -s 12 -e 15
  Program odczyta graf z pliku plik.txt. Wybrany tryb nie wpływa na działanie programu. Domyślnie nie zostanie wydrukowany komunikat, czy graf jest spójny oraz domyślnie wydrukuje wagi dla krawędzi w najkrótszej możliwej ścieżce. Znajdzie najkrótszą możliwą ścieżkę z wierzchołka 1 to wierzchołka 12 do wierzchołka 15.

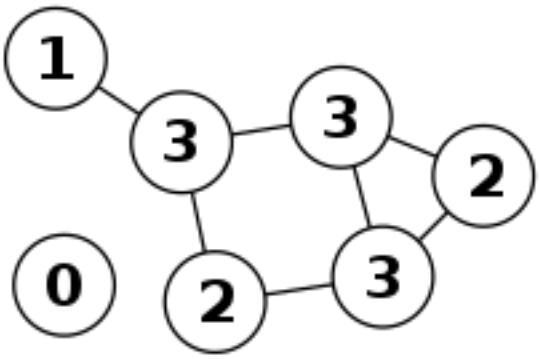
# 4 Teoria

**Graf**

Matematyczna struktura, oznaczana: G =(V,E) składająca się z niepustego, skończonego zbioru wierzchołków V oraz zbioru połączeń między nimi, zwanymi krawędziami E.

**Graf spójny i niespójny**

Graf nazywamy spójnym jeśli istnieje droga pomiędzy każdym wierzchołkiem grafu. Jeśli w danym grafie dwa punkty nie są ze sobą połączone, to taki graf nazywamy niespójnym.




Rys.1 Przykładowy graf niespójny

**Algorytm BFS**

Algorytm przeszukiwania wszerz (Breadth-first search - BFS) jest jednym z najprostszych algorytmów przeszukiwana grafu. Działanie algorytmu głównie polega na dodawaniu kolejnych wierzchołkow do kolejki, a następnie usuwaniu ich przy przejściu do kolejnej warstwy grafu. Czynności te powtarzamy aż do przejścia całego grafu.

Złożoność czasowa tego algorytmu wynosi O(V+E).
Gdzie: V - liczba wierzchołków, E - liczba krawędzi w grafie.

Algorytm:

```
szukaj_wszerz( Graf G, Wierzchołek s ):
    inicjuj Color c[ 0...G.liczbaWierzchołków()-1 ]    //kolory:
                    BIAŁY - nie odwiedzono wierzchołka,
                    SZARY - odwiedzono wierzchołek, ale nie odwiedzono sąsiednich wierzchołków,
                    CZARNY - odwiedzono wierzchołek i wszystkie sąsiednie wierzchołki
    inicjuj Integer poprzednik[ 0...G.liczbaWierzchołków()-1 ]    //poprzedniki
    inicjuj Integer l[ 0...G.liczbaWierzchołków()-1 ]    //odległość od punktu START
    inicjuj kolejkę First-In, First-Out FIFO<Wierzchołek>
    dla każdego Wierzchołka w z G.wierzchołki() wykonaj    //inicjuj początkowe wartości
        c[w] ← BIAŁY
        l[w] ← INFINITY
        poprzednik[w] ← -1
    c[s] ← SZARY    //odwiedź wierzchołek s
    l[s] ← 0
    FIFO.put(s)
    dopóki FIFO nie jest pusta wykonuj
        Wierzchołek w ← FIFO.get()
        dla każdego Wierzchołka v z G.sąsiednie(w) wykonaj    //odwiedź sąsiadów wierzchołka w
            jeżeli c[v] = BIAŁY wykonaj
                c[v] ← SZARY
                l[v] ← l[w]+1
                poprzednik[v] ← w
                FIFO.put(v)
        c[w] ← CZARNY    //odwiedzono wierzchołek w i jego sąsiadów
```

Jeżeli iterując po tablicy poprzednik[] natrafimy na wartość "-1" w innym elemencie niż poprzednik[s] (poprzednik wierzchołka START), to znaczy, że graf jest niespójny. Ponadto w tablicy l[] zapisane są odległości poszczególnych wierzchołków od wierzchołka START.

**Algorytm Dijkstry**

Algorytm Dijkstry służy do znajdywania najkrótszej ścieżki pomiędzy jednym wierzchołkiem a

wszystkimi innymi osiągalnymi wierzchołkami w grafie. W wynikowym zbiorze znajdzie się więc również najkrótsza możliwa ścieżka do zadanego wierzchołka.

Złożoność czasowa tego algorytmu wynosi $O(E \cdot \log V)$
Gdzie: V - liczba wierzchołków, E - liczba krawędzi w grafie.

Funkcje pomocnicze dla algorytmu:

```
inicjujNS1Z( Graf G, Wierzchołek s ):    //inicjuje tablicę poprzedników i odległości
    inicjuj Integer p[ 0...G.liczbaWierzchołków()-1 ]    //poprzedniki
    inicjuj Double o[ 0...G.liczbaWierzchołków()-1 ]    //odległości od punktu START
    dla każdego Wierzchołka w z G.wierzchołki() wykonaj
        o[w] ← INFINITY
        p[w] ← -1
    o[s] ← 0
    zwróć parę <o,p>

relax( Graf G, Wierzchołek u, Wierzchołek v, Double[] o, Integer[] p ):
    / /wyznacza kolejną część drogi
    // u - sąsiad Wierzchołka v, v - potencjalny poprzednik Wierzchołka u
    // o[u] = INFINITY (ścieżka do punktu nie została wyznaczona) lub nowa ścieżka jest krótsza
    jeżeli o[u] > o[v] + G.waga(u,v) to
        o[u] ← o[v] + G.waga(u,v)
        p[u] ← v
```

Algorytm:

```
aDijkstry( Graf G, Wierzchołek s ):
    inicjuj parę <o,p> ← inicjujNS1Z( G, s )
    inicjuj SET<Wierzchołek> w
    inicjuj PRIOTITY_QUEUE<Wierzcholek> q, zawierającą wszystkie elementy z G.wierzchołki(), o
priotytecie 1/o[]

    dopóki q nie jest pusta wykonuj
        Wierzchołek u ← q.get()    //element o najmniejszym koszcie dojścia (najmniejszym o)
        w.add(u)
        dla każdego Wierzchołka v z G.sąsiednie(u) wykonaj
            jeżeli v nie jest w w, wykonaj
                relax( G, v, u, o, p)
```

Drogę z wierzchołka "a" do wierzchołka "b", można znaleźć iterując po poprzednikach "b" (tablica p) - p[b] ->p [ p[b] ] -> p[ p[ p[b] ] ], aż do napotkania "a" jako poprzednika. Jeżeli program napotka na wartość "-1", takie połączenie nie istnieje.

# 5 Komunikaty błędów

Program stara się kontynuować pracę mimo napotkania nieprawidłowych danych.

1. Błędy związane z plikiem wejściowym - czytaniem grafu:
   (a) Nie zadeklarowano ilości kolumn/wierszy w pliku wejściowym: Linia 1: Nie znaleziono ilości kolumn/wierszy. Przerywam działanie. Komunikat pojawia się, gdy w podanych danych wejściowych program nie znajdzie poprawnie zadeklarowanej ilości kolumn/wierszy w 1. linii pliku. Może to wynikać z tego, że zostały one wpisane w niepoprawnym formacie lub plik jest pusty.
   (b) Numer wierzchołka (poprzednika) większy niż columns*rows-1 lub mniejszy od 0: Linia 106: Numer wierzchołka musi być mniejszy od columns*rows i większy lub równy 0. Wczytano:

"104". Wierzchołek poprzednik wraz z jego następnikami został pominięty. Program wykrył wierzchołek (poprzednik) o numerze większym niż columns*rows-1 lub ujemnym (o wartości 104) w linii 106 w danych wejściowych. Program ignoruje nieprawidłowy wierzchołek poprzednik wraz z jego następnikami.

(c) Numer wierzchołka (następnika) większy niż columns*rows-1 lub mniejszy od 0: Linia 24: Numer wierzchołka musi być mniejszy od columns*rows i większy lub równy 0. Wczytano: "7890". Wierzchołek (następnik) został pominięty. Program wykrył wierzchołek (następnik) o numerze większym niż columns*rows-1 lub ujemnym (o wartości 7890) w linii 24 w danych wejściowych. Program ignoruje nieprawidłowy wierzchołek.

(d) Waga krawędzi w danych wejściowych mniejsza/równa 0: Linia 15: Waga krawędzi musi być większa od 0. Wczytano: "-3.3". Krawędź została pominięta. Program wykrył ujemną wagę (o wartości -3.3) w linii 15 w danych wejściowych. Program ignoruje nieprawidłową krawędź, ponieważ uniemożliwiłaby ona znalezienie najkrótszej możliwej drogi za pomocą algorytmu Dijkstry.

(e) Brak poprawnie wczytanych krawędzi: W podanych danych wejściowych brak poprawnie zdefiniowanych krawędzi. Przerywam działanie. Komunikat pojawia się, gdy w podanych danych wejściowych program nie znajdzie poprawnie zdefiniowanych krawędzi. Może to wynikać z tego, że w pliku nie zadeklarowano żadnych krawędzi, albo wszystkie wpisy zawierają błędy.

2. Błędy związane z nieprawidłowymi danymi wprowadzonymi przez użytkownika:

(a) columns lub rows mniejsze/równe 0: Wartość wczytana w -c columns lub -r rows jest mniejsza od 0 lub równa 0. Przerywam działanie. Program wykrył, że użytkownik podał w -c columns lub -r rows liczbę mniejszą od 0 lub równą 0. Ze względu na niepoprawność matematyczną program przerywa działanie.

(b) from-weight lub to-weight mniejsze od 0: Wartość wczytana w -f from-weight lub -t to-weight jest ujemna. Przerywam działanie. Program wykrył, że użytkownik podał w -f from-weight lub -t to-weight liczbę mniejszą od 0. Program ignoruje nieprawidłową krawędź, ponieważ uniemożliwiłaby ona znalezienie najkrótszej możliwej drogi za pomocą algorytmu Dijkstry.

(c) from-weight większe od to-weight: Wartość wczytana w -f from-weight jest większa niż w -t to-weight. Przerywam działanie. Program wykrył, że użytkownik podał w -f from-weight wartość większą niż w -t to-weight. Ze względu na niepoprawność matematyczną program przerywa działanie.

(d) Niepoprawny numer wierzchołka: Zadeklarowano niepoprawny numer wierzchołka w -s start-vertex-number lub -e end-vertex-number. Nie znajdę najkrótszej możliwej ścieżki. Program wykrył, że użytkownik podał w argumencie wywołania programu -s start-vertex-number lub -e end-vertex-number numer wierzchołka większy od columns*rows-1 lub ujemny. Z tego powodu program nie będzie mógł znaleźć najkrótszej możliwej ścieżki z wierzchołka p do wierzchołka q.

(e) Nie podano drugiego wierzchołka: Podano jedynie jeden wierzchołek - nie mogę wyznaczyć najkrótszej możliwej ścieżki. Program wykrył, że użytkownik podał jeden wierzchołek w -s start-vertex-number lub -e end-vertex-number, jednak nie podał drugiego - -s start-vertex-number lub -e end-vertex-number.

# Literatura


[1] Jacek Starzyński. *Prezentacja "Algorytmy dla grafów" na podstawie: Cormen, Leiserson, Rivest, Stein: "Wprowadzenie do algorytmów", WNT 2004*

[2] *Artykuł "Przeszukiwanie wszerz" na stronie Wikipedia*,
[ https://pl.wikipedia.org/wiki/Przeszukiwanie_wszerz ]

[3] *Artykuł "Algorytm Dijkstry" na stronie Wikipedia*,
[ https://pl.wikipedia.org/wiki/Algorytm_Dijkstry ]

Źródło Rys.1.: https://commons.wikimedia.org/wiki/File:UndirectedDegrees.svg