AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

Algorytmy grafowe – najkrótsza ścieżka

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

Zad. 1

Algorytm Floyda-Warshalla

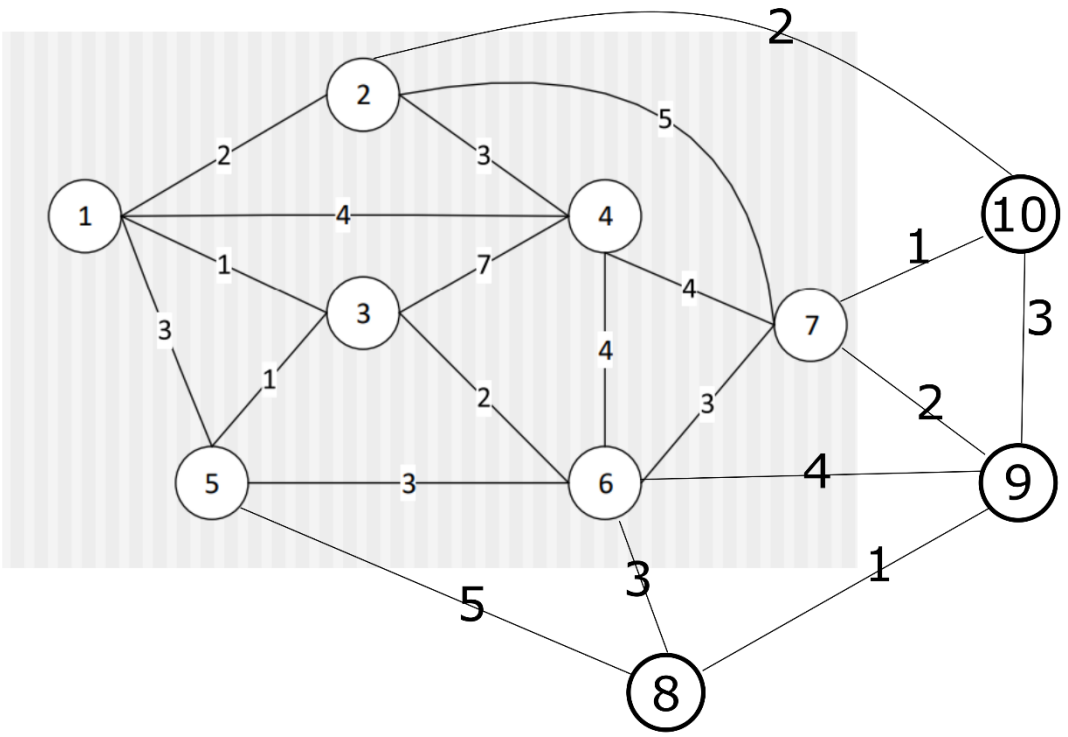
```
#include <iostream>
#include <limits>
#include <map>
#include <list>

int inf = std::numeric_limits<int>::max();

struct path{
    int origin;
    int destination;
};

template <size_t size>
std::tuple<std::list<path>, int> FW (int(&graph)[size][size], int s, int
k){

    // Deklaracja zmiennych
    int d[10][10];
    int r[10][10];
    std::list<path> A = {};

    // przepisywanie ścieżek z grafu, by nie robić zmian grafu
    // plus wypełnianie tablicy r "-1" no "0" to u mnie wierzchołek
    for (size_t y = 0; y < size; y++){
        for (size_t x = 0; x < size; x++){
            r[x][y] = -1;
            d[x][y] = graph[x][y];
            if(x == y){
                d[x][y] = 0;
            }
        }
    }

    for (size_t i = 0; i < size; i++){ // wybieramy wierzchołek, przez
który będziemy przechodzić
        for (size_t y = 0; y < size; y++){ // sprawdzamy dla każdej pary
wierzchołków czy przejście przez "i" skróci
            for (size_t x = 0; x < size; x++){
                if(d[x][i] != inf and  d[i][y] != inf and d[x][y] > d[x][i]
+ d[i][y]){
                    d[x][y] = d[x][i] + d[i][y];
                    r[x][y] = static_cast<int>(i);
                }
            }
        }
    }

    // odzyskuje ścieżkę z tablicy R
    // wrzucam ścieżkę do listy, po czym dla każdego połączenia sprawdzam,
czy przechodzi przez inny
    A.push_back({s + 1, k + 1});

    for(auto i = A.begin(); i != A.end(); i ++){
        path p_old = *i;
        if (r[p_old.origin - 1][p_old.destination - 1] != -1){
            A.erase(i);
            path next = {r[p_old.origin - 1][p_old.destination - 1] + 1,
```

```
p_old.destination};
            path prev = {p_old.origin, r[p_old.origin -
1][p_old.destination - 1] + 1};
            A.push_back(prev);
            A.push_back(next);
            i = A.begin();
        }
    }

    return {A, d[s][k]};

}

int main() {
    int graph[10][10] = {
            {inf,2,  1,  4,   3,   inf, inf, inf, inf, inf},
            {2,  inf,inf,3,   inf, inf, 5,   inf, inf, 2  },
            {1,  inf,inf,7,   1,   2,   inf, inf, inf, inf},
            {4,  3,  7,  inf, inf, 4,   4,   inf, inf, inf},
            {3,  inf,1,  inf, inf, 3,   inf, 5,   inf, inf},
            {inf,inf,2,  4,   3,   inf, 3,   3,   4,   inf},
            {inf,5,  inf,4,   inf, 3,   inf, inf, 2,   1  },
            {inf,inf,inf,inf, 5,   3,   inf, inf, 1,   inf},
            {inf,inf,inf,inf, inf, 4,   2,   1  , inf, 3  },
            {inf,2,  inf,inf, inf, inf, 1,   inf, 3,   inf},
    };

    int s = 1;
    int k = 7;
    auto ans = FW(graph, s, k);
    std::cout << "suma krawedzi dotarcia z "<< s ++  << " do "<< k++ << "
wynosi: "<< std::get<1>(ans) << std::endl;

    for(auto ele : std::get<0>(ans)){
        std::cout << ele.origin << " -> " << ele.destination << std::endl;
    }
}
```

Kod. 1 kod źródłowy algorytmu Floyda-Warshalla poszukiwania minimalnej ścieżki w grafie.

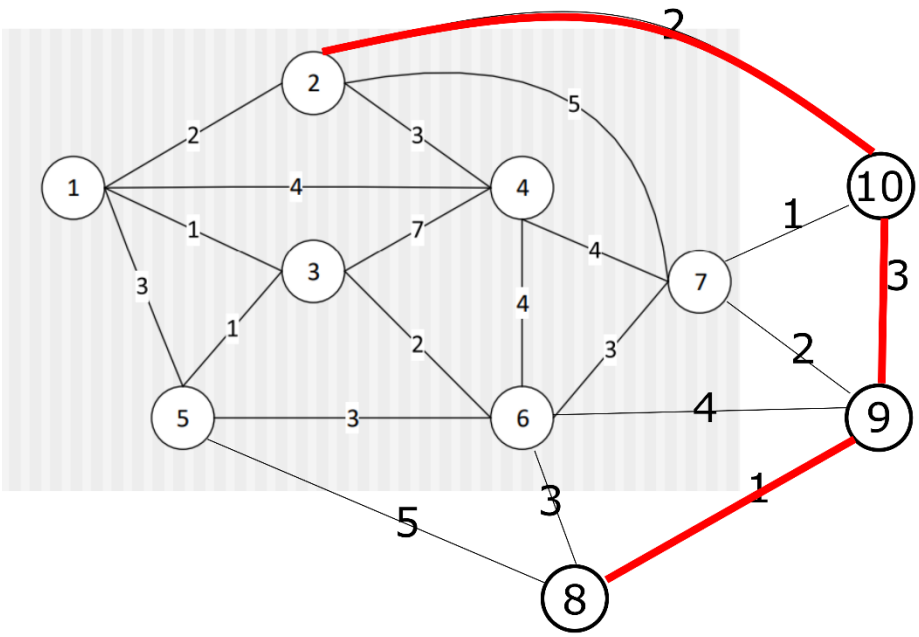
Zad. 2

Dla algorytmu Floyda-Warshalla istotne jest by graf był spójny bez cykli ujemnych. Ponieważ jak graf jest niespójny to zwróci naszą reprezentację nieskończoności. Możliwe jest wystąpienie wagi ujemnej pod warunkiem nie występowania cyklu ujemnego (optymalna trasa krąży w kółko).



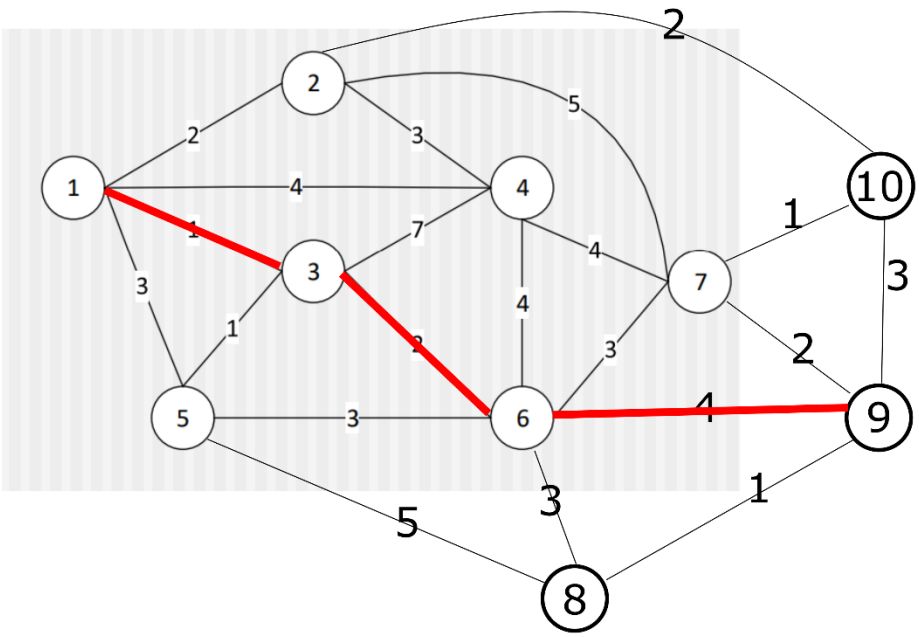

```
int graph[10][10] = {
                    [0]: { [0]: inf, [1]: 2,   [2]: 1,   [3]: 4,   [4]: 3,   [5]: inf, [6]: inf, [7]: inf, [8]: inf, [9]: inf},
                    [1]: { [0]: 2,   [1]: inf, [2]: inf, [3]: 3,   [4]: inf, [5]: inf, [6]: 5,   [7]: inf, [8]: inf, [9]: 2   },
                    [2]: { [0]: 1,   [1]: inf, [2]: inf, [3]: 7,   [4]: 1,   [5]: 2,   [6]: inf, [7]: inf, [8]: inf, [9]: inf},
                    [3]: { [0]: 4,   [1]: 3,   [2]: 7,   [3]: inf, [4]: inf, [5]: 4,   [6]: 4,   [7]: inf, [8]: inf, [9]: inf},
                    [4]: { [0]: 3,   [1]: inf, [2]: 1,   [3]: inf, [4]: inf, [5]: 3,   [6]: inf, [7]: 5,   [8]: inf, [9]: inf},
                    [5]: { [0]: inf, [1]: inf, [2]: 2,   [3]: 4,   [4]: 3,   [5]: inf, [6]: 3,   [7]: 3,   [8]: 4,   [9]: inf},
                    [6]: { [0]: inf, [1]: 5,   [2]: inf, [3]: 4,   [4]: inf, [5]: 3,   [6]: inf, [7]: inf, [8]: 2,   [9]: 1   },
                    [7]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: 5,   [5]: 3,   [6]: inf, [7]: inf, [8]: 1,   [9]: inf},
                    [8]: { [0]: inf, [1]: inf, [2]: inf, [3]: inf, [4]: inf, [5]: 4,   [6]: 2,   [7]: 1   , [8]: inf, [9]: 3   },
                    [9]: { [0]: inf, [1]: 2,   [2]: inf, [3]: inf, [4]: inf, [5]: inf, [6]: 1,   [7]: inf, [8]: 3,   [9]: inf},
                };
```

Rys. 1 Graf oraz jego reprezentacja



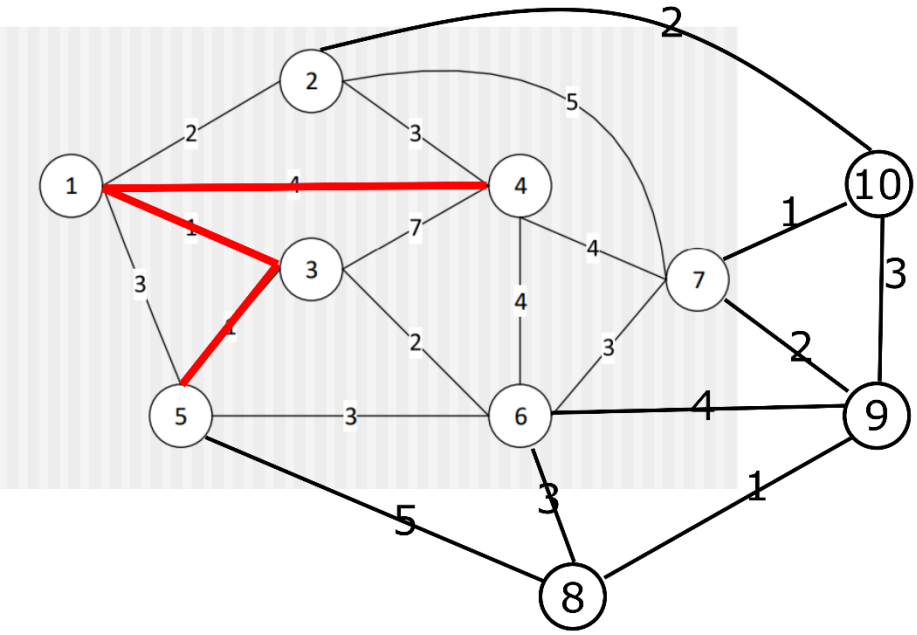

```
suma krawedzi dotarcia z 2 do 8 wynosi: 6
2 -> 10
10 -> 9
9 -> 8
```

Rys. 2 Minimalna trasa z 2 do 8




```
suma krawedzi dotarcia z 9 do 1 wynosi: 7
9 -> 6
6 -> 3
3 -> 1
```

Rys. 3 Minimalna trasa z 9 do 1




```
suma krawedzi dotarcia z 5 do 4 wynosi: 6
5 -> 3
3 -> 1
1 -> 4
```

Rys. 3 Minimalna trasa z 5 do 4

Zad. 3

Algorytm Floyda-Warshalla jest dość złożony obliczeniowo tak czasowo jak i pamięciowo. Pamięciowo wymaga zalokowania dwóch tablic dwuwymiarowych:

- pierwsza służy do zapisywania wartości aktualnej najkrótszej drogi
- druga służy do zapisywania wierzchołków tranzytowych

Czasowo algorytm jest wymagający przez złożoność $o(n^3)$. Dla każdego połączenia rozważa dołączenie każdego wierzchołka w ramach skrócenia odległości.

Niestety w tym algorytmie nie może nas uratować nawet łatwość problemu ponieważ algorytm niezależnie od grafu przeprowadzi tą samą ilość obliczeń:

- Optymistyczna złożoność: $o(n^3)$.
- Pesymistyczna złożoność: $o(n^3)$.
- Średnia złożoność: $o(n^3)$.

Na korzyść algorytmu przemawia fakt, że wystarczy dokonać obliczenia raz po czym dla danego grafu możemy odczytać wartość w czasie stałym lub $o(n)$ (moja implementacja nie umożliwia tego). Dzięki tej zdolności algorytm może być używany w formie jednokrotnego wyliczenia wszystkich możliwych dróg i potem tylko odczytywania.

Wnioski

Zadanie pozwoliło mi przypomnieć sobie działanie na listach w c++. Zadanie sprawiło mi problemy najpierw z usuwaniem elementów z listy oraz nie umiałem znaleźć błędu pojawiającego się przy zmianie numerowania wierzchołków (w algorytmie jest od 0 w reprezentacji od 1). Zapoznanie się z działaniem algorytmu było ciekawym wyzwaniem. Niestety zacząłem robić zadanie nim dowiedziałem się że mamy zrobić tylko jeden algorytm więc wziąłem ten licząc, że następny wezmę Bellmana Forda by dostać równą ilość punktów.