AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

Algorytmy grafowe – minimalne drzewo rozpinające grafu

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

Zad. 1

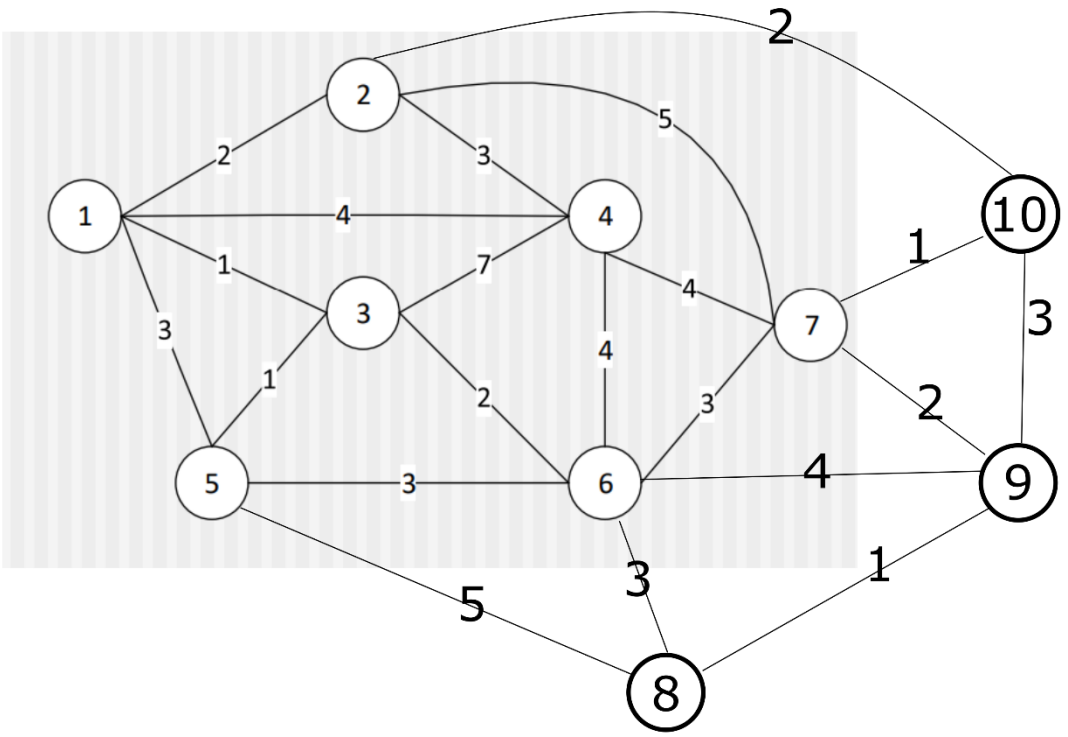
```cpp
#include <iostream>
#include <limits>
#include <map>
#include <list>

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

struct path{
    int origin;
    int destination;
};

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

    // Deklaracja zmiennych
    int suma = 0;
    std::list<path> A = {};
    int alfa[size];
    int beta[size];
    std::list<int> Q;

    // Wpisanie wszystkich wierzchołków do listy do odwiedzenia
    for(size_t i = 0; i < size; i++){
        Q.push_back(i);
        alfa[i] = 0;
        beta[i] = inf;
    }

    // uaktualnienie kosztu przejścia do wierzchołka 1 oraz usunięcie go z
listy
    beta[s - 1] = 0;
    Q.remove(s - 1);
    int u_last = s - 1;

    while (!Q.empty()){

        // Pętla uaktualniająca koszt przejścia do każdego wierzchołka z
nowo przyłączonego
        for(auto u_it = Q.begin();u_it != Q.end(); u_it++){
            int u = *u_it;
            if (graph[u_last][u] != inf and graph[u_last][u] < beta[u]){
                alfa[u] = u_last;
                beta[u] = graph[u_last][u];
            }
        }

        // Poszukiwanie najtańszego do przyłączenia wierzchołka
        int min = inf;
        for(auto u_it = Q.begin();u_it != Q.end(); u_it++){
            int u = *u_it;
            if (beta[u] < min){
                min = beta[u];
                u_last = u;
            }
        }


        // Przyłączenie nowego wierzchołka
```

```
        Q.remove(u_last);
        path temp = {alfa[u_last] + 1, u_last + 1};
        A.push_back(temp);
        suma += graph[alfa[u_last]][u_last];
    }
    return {A, suma};

}



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

    auto ans = DPA(graph, 1);
    std::cout << "suma krawedzi MST: "<< std::get<1>(ans) << std::endl <<
std::endl;

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

Kod. 1 kod źródłowy algorytmu Dijkstry-Prima poszukiwania minimalnego drzewa rozpinającego grafu (MST).

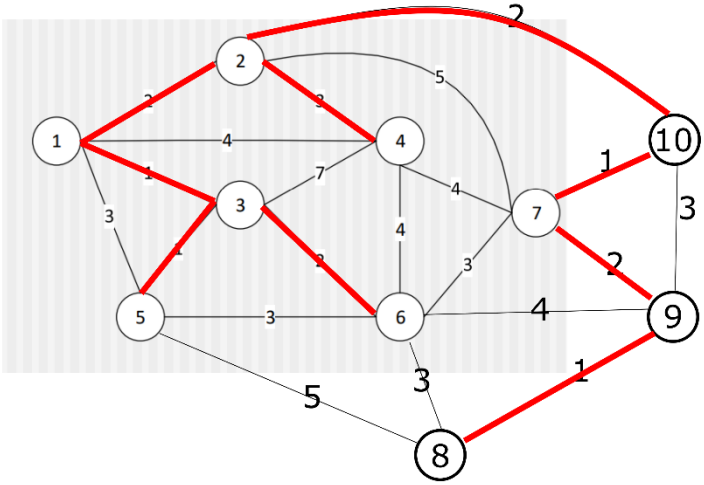
Zad. 2

Dla algorytmu Prima istotne jest by graf był spójny i nieskierowany. Ponieważ jak graf jest niespójny to algorytm będzie zmuszony wpisać w sumie nieskończoność. Nieskierowany ponieważ jest to warunek konieczny bycia drzewem.




```
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 MST: 15

1 -> 3
3 -> 5
1 -> 2
3 -> 6
2 -> 10
10 -> 7
7 -> 9
9 -> 8
2 -> 4
```

Rys. 2 Minimalne drzewo rozpinające.

Zad. 3

Algorytm Kruskala jest algorytmem zachłannym – to znaczy podejmuje najlepiej wyglądającą decyzję i kontynuuje pracę z tym założeniem. Taki algorytm wybiera losową najkrótszą krawędź w grafie i dodaję ją do listy. W momencie gdy jakaś ścieżka zamyka pętlę (łączy dwa odwiedzone wierzchołki) usuwa ją. Robi tak aż znajdzie połączenie wszystkich wierzchołków.

W mojej implementacji algorytmu Kruskala najbardziej złożona była by reprezentacja grafu. Zrobił bym głęboką kopię by móc wykreślać z niej krawędzie łączące dwa odwiedzone wierzchołki.

W innym przypadku przypuszczam, że najbardziej złożona była by iteracja po liście odwiedzonych wierzchołków, na przykład w momencie sprawdzania czy któraś z pozostałych krawędzie łączy dwa wierzchołki już wybrane. Jeśli graf jest gęsty dużą złożonością może się okazać wyszukiwanie najkrótszej krawędzi ponieważ graf będzie miał dużą ich liczbę.

Zad. 4

Moim zdaniem najlepszym przykładem na praktyczne zastosowanie jest sieć energetyczna lub wodociągowa z jedną centralną elektrownią / oczyszczalnią.

- Wagi to koszt połączenia dwóch adresatów.

- Wydaje mi się, że trzeba uwzględnić inne elektrownię / oczyszczalnie albo czy droga nie przechodząca bezpośrednio przez odbiorców nie będzie bardziej optymalna
- Wydaje mi się że oba te problemy są niemożliwe do rozwiązania algorytmem Prima:
    - Budowanie dwóch niezależnych grafów na raz dodając najkrótszą odległość jest możliwe ale czy to nadal będzie podany algorytm
    - Nie widzę sposobu zrealizowania drogi nie przechodzącej bezpośrednio przez wierzchołki z użyciem podanego algorytmu

Wnioski

Zadanie pozwoliło mi przypomnieć sobie działanie na rozmaitych typach danych w c++ oraz sprawiło mi bardzo dużo przyjemności w znajdowaniu i eliminowani błędów. Zapoznanie się z działaniem algorytmamu było ciekawym wyzwaniem umysłowym.