AKADEMIA GÓRNICZO-HUTNICZA im. Stanisława Staszica w Krakowie

**Algorytmy grafowe – minimalne drzewo**

**rozpinające grafu**

Stanisław Olech - 412023

Automatyka i Robotyka

EAIiIB

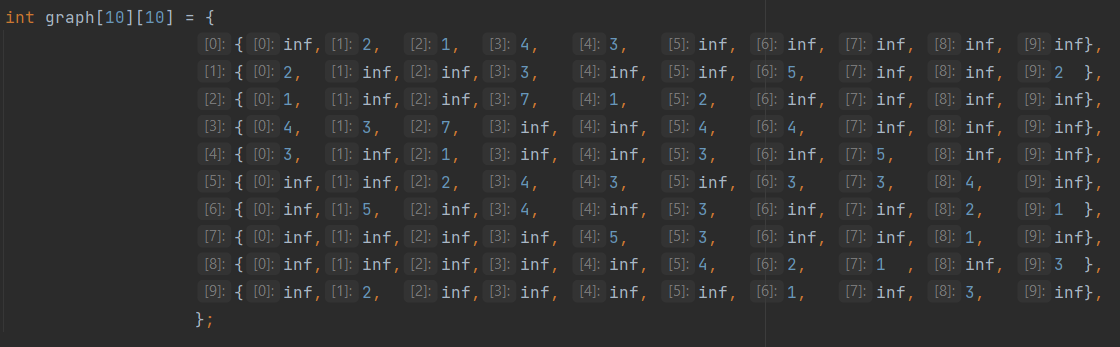
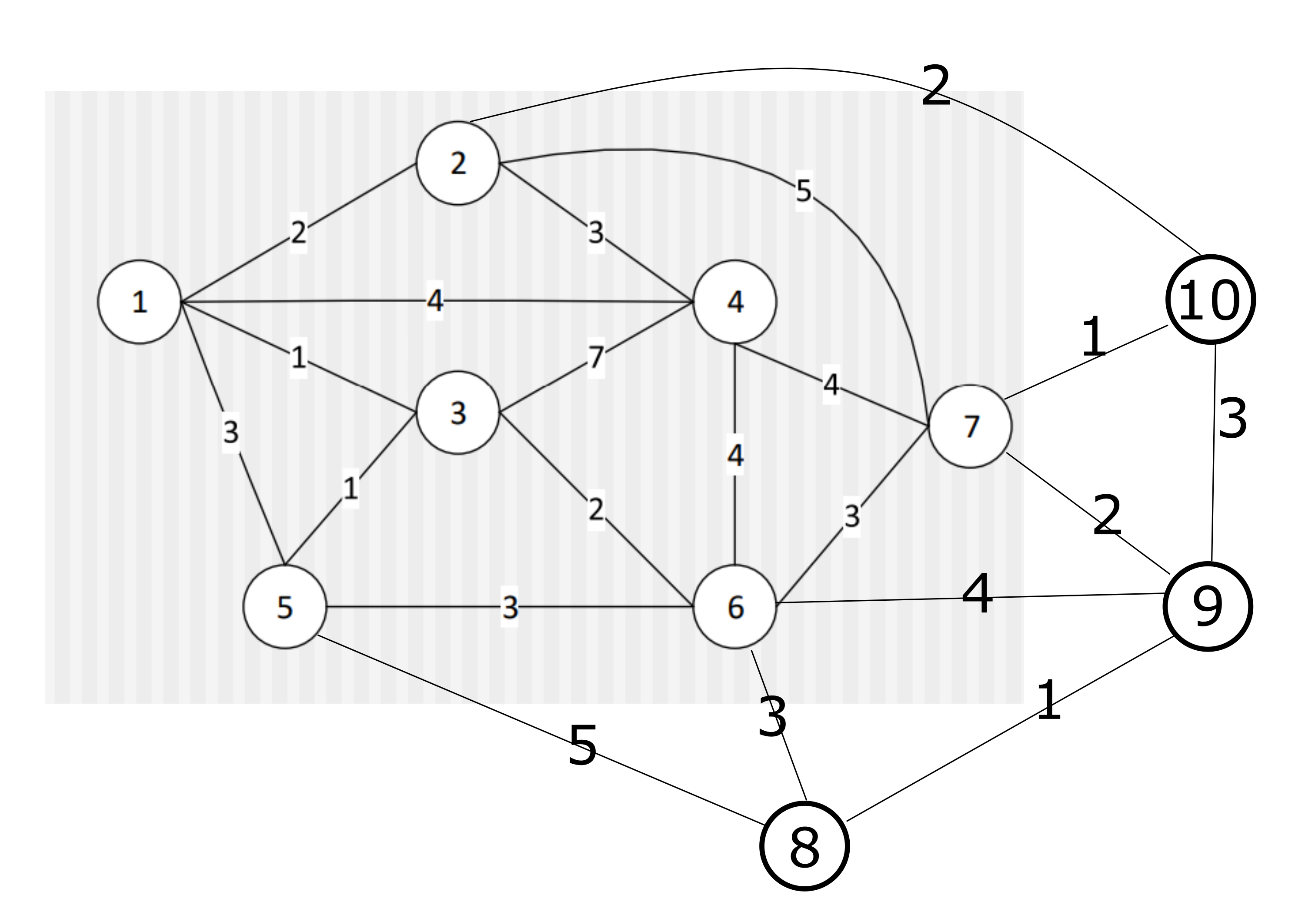
**Zad. 1**

#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;  
 }  
}

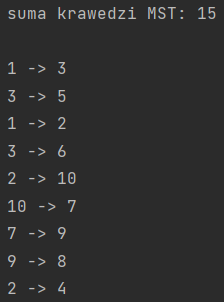
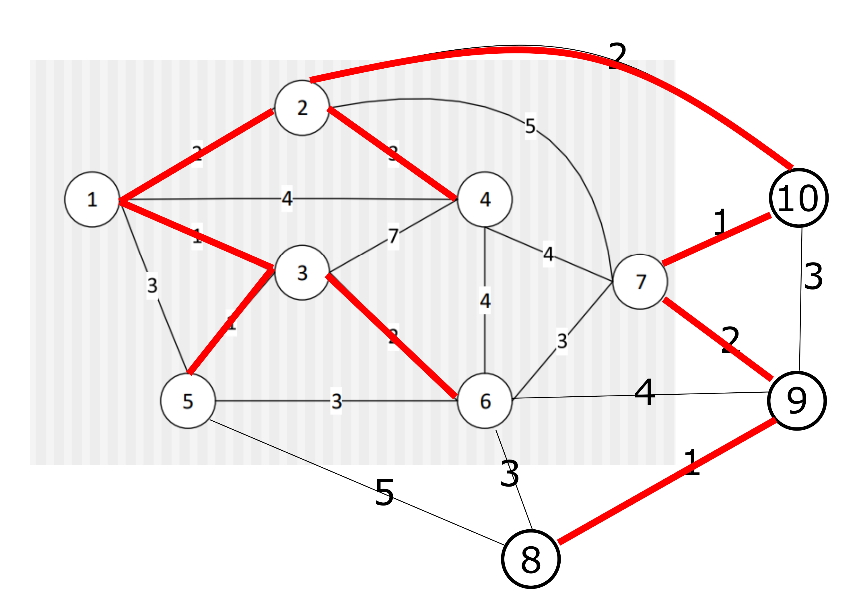
Kod. 1 kod źródłowy algorytmu Dijkstry-Prima poszukiwania minimalnego drzewa rozpinającego grafu (MST).

**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.



Rys. 1 Graf oraz jego reprezentacja



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.