**<http://algorytmy.ency.pl/tutorial/algorytm_dijkstry>**

[**Algorytm Dijkstry**](https://www.algorytm.edu.pl/olimpiada-informatyczna/algorytm-dijkstry)

Artykuł przedstawia algorytm Dijkstry służący do wyznaczania najkrótszych ścieżek w grafie skierowanym (nieskierowanym).

**Zasada działania algorytmu**

Zasada działania algorytmu jest bardzo prosta. W pierwszym kroku ustalamy źródło, czyli taki wierzchołek, z którego będziemy wyznaczać najkrótsze ścieżki do pozostałych. Długość ścieżki w tym przypadku ustawiamy na 0. Dla pozostałych wierzchołków ustawiamy te odległości na nieskończoność, ponieważ nie mamy jeszcze znalezionych długości najkrótszych ścieżek. Przejdź wierzchołka źródłowego, a następnie postępuj według schematu:

1. dodaj do seta (lub kolejki priorytetowej) wszystkie połączone z aktualnym węzłem wierzchołki, ale tylko te, w których zapisana jest odległość od źródła jako nieskończoność. W secie robimy zapis w postaci pary: zapisana długość ścieżki w aktualnie odwiedzonym wierzchołku powiększona o wagę krawędzi oraz numer wierzchołka, z którym nawiązane jest połączenie tą krawędzią,
2. jeśli set jest pusty, to zakończ algorytm, w przeciwnym razie pobierz pierwszy element z seta (jest to najkrótsza odległość od źródła), czyli parę w postaci {dystans, wierzchołek} i zapisz pobraną długość do danego wierzchołka. Zauważ, że jest to możliwie najkrótsza odległość od źródła,
3. przejdź do pobranego wierzchołka,
4. przejdź do kroku pierwszego.

Jeśli **w** to liczba wierzchołków, a **k** to liczba krawędzi, to złożoność algorytmu Dijkstry wynosiO(klogw),

koszt wrzucenia elementu do kolejki/seta jest logarytmiczny. Omawiany algorytm jest algorytmem zachłannym, ponieważ w każdym jego kroku, pobiera najkrótszą ścieżkę z seta, która jest optymalnym rozwiązaniem w danym momencie.

Algorytm Dijkstry nie działa, gdy wagi są ujemne. Można go zastosować zarówno dla grafu skierowanego i nieskierowanego. Jeśli ścieżka do danego wierzchołka nie istnieje, to odległość w nim zapisana będzie wynosiła nieskończoność.

#include<bits/stdc++.h>

#define para pair<int, int>

using namespace std;

const int inf = INT\_MAX; //nieskończoność

//struktura, która będzie przechowywać wierchołki grafu

struct G{

vector <para > pol; //z jakim wierzchołkiem, waga

int odl = inf; //najkrótsza ścieżka od źródła (na początku nieskończoność)

}\*wezel;

void dijkstra(int start)

{

int dystans, a, b;

wezel[start].odl = 0;

set <para > krawedzie; //odleglosc od źródła, ( -> b)

//dodanie wszystkich wag krawędzi oraz numery wierzchołków docelowych

for(int i=0; i<wezel[start].pol.size();i++)

krawedzie.insert({wezel[start].pol[i].second, wezel[start].pol[i].first});

//lub

// krawedzie.insert(make\_pair(wezel[start].pol[i].second, wezel[start].pol[i].first));

while(!krawedzie.empty()) //dopóki w secie są jeszcze jakieś krawędzie do analizy

{

//pobranie najkrótszej ścieżki oraz wierzchołek, do którego ona prowadzi

dystans = krawedzie.begin()->first; //dystans

b = krawedzie.begin()->second; //wierzchołek

wezel[b].odl = min(dystans, wezel[b].odl); //ustawienie długości najkrótszej ścieżki

//usunięcie krawędzi z seta

krawedzie.erase(krawedzie.begin());

for(int i=0; i<wezel[b].pol.size(); i++) //przeglądam wszysktie połączenia z wierzchołka b

//biorę tylko te połączenia z wierzchołkami, do których odleglosć jest równa nieskonczonosć

if(wezel[wezel[b].pol[i].first].odl == inf)

krawedzie.insert(make\_pair(wezel[b].pol[i].second + dystans, wezel[b].pol[i].first));

}

}

int main()

{

int w, k, waga, a, b, start;

//wczytanie liczby wierzchołków oraz liczby krawędzi

cout<<"Podaj liczbę wierzchołków oraz liczbę krawędzi: ";

cin>>w>>k;

wezel = new G[w+1];

for(int i=0; i<k; i++)

{

//(a -> b), waga

cout<<"a -> b (waga): ";

cin>>a>>b>>waga;

wezel[a].pol.push\_back(make\_pair(b, waga));

/\* w przypadku grafu nieskierowanego dodajemy poniższą linijkę

wezel[b].pol.push\_back(make\_pair(a, waga));

\*/

}

cout<<"Podaj wierzchołek początkowy: ";

cin>>start;

dijkstra(start);

cout<<"Najkrótsze ścieżki z wierzchołka źródłowego o numerze "<<start<<": "<<endl;

for(int i=1; i<=w; i++)

if(i != start)

if(wezel[i].odl == inf)

cout<<start<<" -> "<<i<<": nieskończoność"<<endl;

else

cout<<start<<" -> "<<i<<": "<<wezel[i].odl<<endl;

delete [] wezel;

return 0;

}