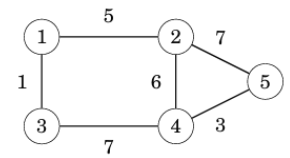
# Đề: Cho một đồ thị vô hướng có hướng có trọng số không âm. Hãy tìm đường đi có trọng số ngắn nhất từ điểm A đến B.

## Ý tưởng:

Giả sử: tìm đường đi từ 1 đến 5

* Tạo pre[] để lưu đường đi; Tạo adj[] để lưu đỉnh kề;



* Từ 1 xét các adj[1]={1,2}, pre[1] ={1}
* Xét đỉnh 3 (ưu tiên trọng số bé) adj[3]={4} Vì đỉnh 1 qua rồi pre[3]={1}
* Xét đỉnh 4 adj[4]={2,5} Vì đỉnh 3 qua rồi pre[4]={3}
* Xét đỉnh 5 pre[5]={4}

## Mã Giả

* DIJKSTRA(s, t):
* Tạo mảng d[1..n], gán tất cả = oo
* d[s] ← 0
* pre[s] ← s
* Khởi tạo hàng đợi ưu tiên p (ưu tiên giá trị nhỏ nhất)
* Đưa (0, s) vào p
* While(p không rỗng)
* Lấy phần tử đầu (kc, u) ra khỏi p
* Nếu kc > d[u] thì bỏ qua (vì đã tìm đường tốt hơn rồi)
* Với mỗi cặp (v, w) trong adj[u]: // v: đỉnh kề, w: trọng số cạnh
* Nếu d[v] > d[u] + w thì:
* d[v] ← d[u] + w
* pre[v] ← u
* Đưa (d[v], v) vào p
* In ra d[t] (độ dài đường đi ngắn nhất từ s đến t)
* Khởi tạo danh sách path rỗng
* Lặp:
* Thêm t vào path
* Nếu t == s thì dừng vòng lặp
* t ← pre[t] // lùi ngược theo mảng tiền nhiệm
* Đảo ngược path
* In ra các đỉnh trong path

## Độ phức tạp

O((n + m) log n)

## Giải thích

1. Tại sao lại dùng priority\_queue?

Tại vì tối ưu được thời gian tìm kiếm

1. greater<pair<int,int>> có ý nghĩa gì và tại sao lại dùng?

* Nếu không có greater thì sẽ mặc định mà max-heap. Nhưng ta muốn sử dụng min heap nên phải thêm greater<pair<int,int>>
* Đảm bảo rằng phần từ đầu hàng đợi là khoảng cách bé nhất

1. Tại sao chúng ta phải cần reverse lại path?

Trong quá trình duyệt chúng ta chỉ biết được trước khi tới đỉnh u là đỉnh v. Và hiện tại sau khi duyệt hết thì chúng ta đang ở đỉnh đích (đỉnh cần tìm). Vì thế chúng ta chỉ có thể truy ngược lại nhờ có mảng pre mà thôi. Mà nếu như thế thì path sẽ bị ngược.

## Code

#include<iostream>

#include<fstream>

#include<vector>

#include<queue>

#include <algorithm>

using namespace std;

int n,m,s,t;

int pre[100];

vector<pair<int,int>> adj[100];

void doc()

{

    ifstream file;

    file.open("inputDijkstra.txt");

    file>>n>>m>>s>>t;

    for(int i=1;i<=m;i++)

    {

        int x,y,z; file>>x>>y>>z;

        adj[x].push\_back({y,z});

adj[y].push\_back({x,z});

    }

}

void Dijkstra(int s, int t)

{

    vector<long long> d(n+1,1e9);

    d[s]=0;

    pre[s]=s;

    priority\_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> p;

    p.push({0,s});

    while(!p.empty())

    {

        pair<int,int> top = p.top();

        p.pop();

        int kc = top.first;// kc

        int u = top.second;//dinh

        if(kc>d[u]) continue;

        for(auto i:adj[u])

        {

            int v = i.first;

            int w = i.second;

            if( d[v]>d[u]+w)

                {

                    d[v]=d[u]+w;

                    p.push({d[v],v});

                    pre[v]=u;

                }

        }

    }

    cout<<d[t]<<endl;

    vector<int>path;

    while(1)

    {

        path.push\_back(t);

        if(t==s) break;

        t=pre[t];

    }

    reverse(begin(path),end(path));

    for(int x:path)

    {

        cout<<x<<" ";

    }

}

int main()

{

    doc();

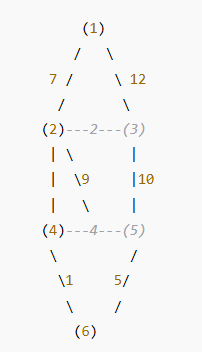
    Dijkstra(s,t);

    return 0; }

## Input

Dòng đầu nhập số đỉnh, số cạnh, đỉnh bắt đầu, đỉnh kết thúc

N(N<=100) dòng sau là danh sách cạnh và trọng số tương ứng mỗi cạnh

6 8 1 5

1 2 7

1 3 12

2 3 2

2 4 9

3 5 10

4 6 1

5 4 4

5 6 5

## Output

