ASSIGNMENT - 8

ANIKEIT SETHI (190001003)

The given problem is a graph problem here we have to find the minimum cost of making the road to reach each city.

I.e given a graph with the weighted edge we have to find the edges such that the cost of travelling all the vertices is minimum.

Basically, we have to find the minimum spanning tree of the given graph.

We will use Prim's Minimum Spanning Tree Algorithm to solve this problem. We will consider each city as the vertex of the graph and each road as weighted edges.

ALGORITHM :

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which form a tree that includes every vertex has the minimum sum of weights among all the trees that can be formed from the graph It falls under a class of algorithms called [greedy algorithms](https://www.programiz.com/dsa/greedy-algorithm) that find the local optimum in the hopes of finding a global optimum.

We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

The steps for implementing Prim's algorithm are as follows:

* Initialize the minimum spanning tree with a vertex chosen at random.
* Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree

Keep repeating step 2 until we get a minimum spanning tree

Code: -

#include <bits/stdc++.h>

using namespace std;

void primsMST(vector <vector<pair<int,int>>> adj, int v) //we are using prims algorithm for finding the minimum spanning tree

{

priority\_queue< pair<int,int>, vector <pair<int,int>> , greater<pair<int,int>> > pq; //defining priority\_queue

pq.push(make\_pair(0, 1)); //pushing the source vertex

vector <int> MST(v+1);

vector <int> parent(v+1);

vector <int> min\_weight(v+1, INT\_MAX); // minimum weight of edge from a vertex to its parent

min\_weight[1]=0;

while(!pq.empty())

{

int u = pq.top().second; // minimum cost edge from the previously included vertices in the queue

pq.pop();

MST[u]=1;

for(int i=0;i<adj[u].size();i++) //traversing the adjacent nodes of u

{

int adjv=adj[u][i].first;

int weight = adj[u][i].second;

if(MST[adjv]==0 && min\_weight[adjv]>weight) //if we find a vertex with minimum cost than the previous, we push the node in queue

{

parent[adjv]=u;

min\_weight[adjv]=weight;

pq.push(make\_pair(weight,adjv));

}

}

}

for(int i=2;i<=v;i++)

{

cout<<i<<" "<<parent[i]<<" "<<min\_weight[i]<<endl;

}

}

int main()

{

int v,e; //number of vertices and the number of edges

cin>>v>>e;

vector <vector<pair<int, int>>> adj(v+1);

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

{

int a,b,weight;

cin>>a>>b>>weight;

adj[a].push\_back(make\_pair(b,weight)); //filling the adjacency list

adj[b].push\_back(make\_pair(a,weight));

}

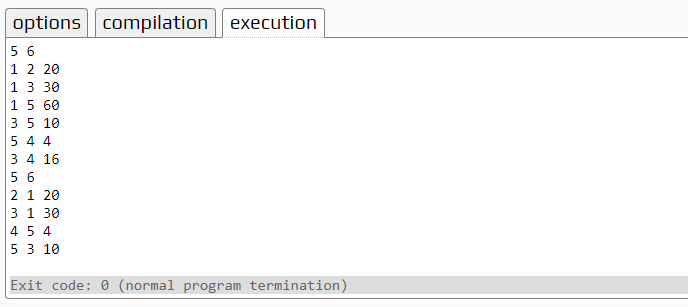
cout<< v<<" "<<e<<endl;

primsMST(adj,v);

return 0;

}

Output: -



TIME COMPLEXITY: - O(ElogV)

The time complexity of the above code is O(ElogV).

Where E is the number of edges and V is the number of vertices.

GRAPH: -

