DJ

#include<bits/stdc++.h>

using namespace std;

class Solution

{

public:

*//Function to find the shortest distance of all the vertices*

*//from the source vertex S.*

vector <int> dijkstra(int V, vector<vector<int>> adj[], int S)

{

*// Create a set ds for storing the nodes as a pair {dist,node}*

*// where dist is the distance from source to the node.*

*// set stores the nodes in ascending order of the distances*

set<pair<int,int>> st;

*// Initialising dist list with a large number to*

*// indicate the nodes are unvisited initially.*

*// This list contains distance from source to the nodes.*

vector<int> dist(V, 1e9);

st.insert({0, S});

*// Source initialised with dist=0*

dist[S] = 0;

*// Now, erase the minimum distance node first from the set*

*// and traverse for all its adjacent nodes.*

while(!st.empty()) {

auto it = \*(st.begin());

int node = it.second;

int dis = it.first;

st.erase(it);

*// Check for all adjacent nodes of the erased*

*// element whether the prev dist is larger than current or not.*

for(auto it : adj[node]) {

int adjNode = it[0];

int edgW = it[1];

if(dis + edgW < dist[adjNode]) {

*// erase if it was visited previously at*

*// a greater cost.*

if(dist[adjNode] != 1e9)

st.erase({dist[adjNode], adjNode});

*// If current distance is smaller,*

*// push it into the queue*

dist[adjNode] = dis + edgW;

st.insert({dist[adjNode], adjNode});

}

}

}

*// Return the list containing shortest distances*

*// from source to all the nodes.*

return dist;

}

};

int main()

{

*// Driver code.*

int V = 3, E = 3, S = 2;

vector<vector<int>> adj[V];

vector<vector<int>> edges;

vector<int> v1{1, 1}, v2{2, 6}, v3{2, 3}, v4{0, 1}, v5{1, 3}, v6{0, 6};

int i = 0;

adj[0].push\_back(v1);

adj[0].push\_back(v2);

adj[1].push\_back(v3);

adj[1].push\_back(v4);

adj[2].push\_back(v5);

adj[2].push\_back(v6);

Solution obj;

vector<int> res = obj.dijkstra(V, adj, S);

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

{

cout << res[i] << " ";

}

cout << endl;

return 0;

P

}

#include <bits/stdc++.h>

using namespace std;

class Solution

{

public:

*//Function to find sum of weights of edges of the Minimum Spanning Tree.*

int spanningTree(int V, vector<vector<int>> adj[])

{

priority\_queue<pair<int, int>,

vector<pair<int, int> >, greater<pair<int, int>>> pq;

vector<int> vis(V, 0);

*// {wt, node}*

pq.push({0, 0});

int sum = 0;

while (!pq.empty()) {

auto it = pq.top();

pq.pop();

int node = it.second;

int wt = it.first;

if (vis[node] == 1) continue;

*// add it to the mst*

vis[node] = 1;

sum += wt;

for (auto it : adj[node]) {

int adjNode = it[0];

int edW = it[1];

if (!vis[adjNode]) {

pq.push({edW, adjNode});

}

}

}

return sum;

}

};

int main() {

int V = 5;

vector<vector<int>> edges = {{0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};

vector<vector<int>> adj[V];

for (auto it : edges) {

vector<int> tmp(2);

tmp[0] = it[1];

tmp[1] = it[2];

adj[it[0]].push\_back(tmp);

tmp[0] = it[0];

tmp[1] = it[2];

adj[it[1]].push\_back(tmp);

}

Solution obj;

int sum = obj.spanningTree(V, adj);

cout << "The sum of all the edge weights: " << sum << endl;

return 0;

}