**Kruskal’s Algorithm**:

#include<bits/stdc++.h>

using namespace std;

struct Edge{

int src,dest,weight;

};

int find(vector<int>&parent,int i){

if(parent[i]==i)

return i;

return parent[i]=find(parent,parent[i]);

}

void unionSets(vector<int>&parent,int x,int y){

int setx=find(parent,x);

int sety=find(parent,y);

parent[setx]=sety;

}

int kruskalMST(vector<Edge>&edges,int vertices){

sort(edges.begin(),edges.end(),

[] (const Edge &a , const Edge &b ){

return a.weight <b.weight;

});

int totalCost=0;

vector<int>parent(vertices);

for(int i=0;i<vertices;i++){

parent[i]=i;

}

for(const auto &edge : edges){

int srcParent=find(parent,edge.src);

int destParent=find(parent,edge.dest);

if(srcParent != destParent){

unionSets(parent,srcParent,destParent);

totalCost+=edge.weight;

}

}

return totalCost;

}

int main(){

int vertices,edgeCount;

cin>>vertices>>edgeCount;

vector<Edge>edges(edgeCount);

for(int i=0;i<vertices;i++){

cin>>edges[i].src>>edges[i].dest>>edges[i].weight;

}

int totalCost=kruskalMST(edges,vertices);

cout<<totalCost<<endl;

return 0;

}

**//Printing all paths of MST**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

int u, v, w;

bool operator<(const Edge &e) const {

return w < e.w;

}

};

int find(vector<int> &parent, int x) {

return parent[x] == x ? x : (parent[x] = find(parent, parent[x]));

}

int main() {

int n, e;

cin >> n >> e;

if (n < 0) {

cout << "Invalid input" << endl;

return 0;

}

vector<Edge> edges(e);

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

cin >> edges[i].u >> edges[i].v >> edges[i].w;

sort(edges.begin(), edges.end());

vector<int> parent(n);

for (int i = 0; i < n; i++) parent[i] = i;

for (auto &edge : edges) {

int u = find(parent, edge.u), v = find(parent, edge.v);

if (u != v) {

parent[u] = v;

cout << edge.u << " " << edge.v << " " << edge.w << endl;

}

}

}

**Prim’s Algorithm:**

#include <bits/stdc++.h>

using namespace std;

void primMST(int n, int e, vector<vector<pair<int, int>>> &graph) {

if (n < 0) {

cout << "Invalid input" << endl;

return;

}

if (n == 1) {

cout << "0" << endl; // Single city, no cables needed

return;

}

vector<bool> inMST(n, false);

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

pq.push({0, 0}); // (weight, start node)

int totalWeight = 0;

int edgesUsed = 0;

while (!pq.empty() && edgesUsed < n - 1) {

int weight = pq.top().first, u = pq.top().second;

pq.pop();

if (inMST[u]) continue;

inMST[u] = true;

totalWeight += weight;

edgesUsed++;

for (const auto &edge : graph[u]) {

int v = edge.first, w = edge.second;

if (!inMST[v]) {

pq.push({w, v});

}

}

}

cout << totalWeight << endl;

}

int main() {

int n, e;

cin >> n;

if (n < 0) {

cout << "Invalid input" << endl;

return 0;

}

cin >> e;

vector<vector<pair<int, int>>> graph(n);

for (int i = 0; i < e; i++) {

int u, v, w;

cin >> u >> v >> w;

graph[u].push\_back({v, w});

graph[v].push\_back({u, w}); // \*\*Add both directions (Undirected Graph)\*\*

}

primMST(n, e, graph);

return 0;

}

**// Prints the edges and total cost**

#include <iostream>

#include <vector>

#include <climits>

using namespace std;

typedef pair<int, int> Pair;

vector<Pair> primMST(vector<vector<Pair>>& adj, int vertices, int &totalWeight) {

vector<Pair> mst;

vector<int> key(vertices, INT\_MAX);

vector<int> parent(vertices, -1);

vector<bool> inMST(vertices, false);

int src = 0;

key[src] = 0;

totalWeight = 0;

for (int count = 0; count < vertices - 1; ++count) {

int minKey = INT\_MAX, u = -1;

for (int v = 0; v < vertices; ++v) {

if (!inMST[v] && key[v] < minKey) {

minKey = key[v];

u = v;

}

}

if (u == -1) continue;

inMST[u] = true;

for (auto& neighbor : adj[u]) {

int v = neighbor.first;

int weight = neighbor.second;

if (!inMST[v] && weight < key[v]) {

parent[v] = u;

key[v] = weight;

}

}

}

for (int i = 1; i < vertices; ++i) {

if (parent[i] != -1) {

mst.push\_back({parent[i], i});

totalWeight += key[i];

}

}

return mst;

}

int main() {

int vertices, edges;

cin >> vertices >> edges;

vector<vector<Pair>> adj(vertices);

for (int i = 0; i < edges; ++i) {

int u, v, weight;

cin >> u >> v >> weight;

adj[u].push\_back({v, weight});

adj[v].push\_back({u, weight});

}

int totalWeight = 0;

vector<Pair> mst = primMST(adj, vertices, totalWeight);

for (const auto& edge : mst) {

cout << edge.first << " -- " << edge.second << endl;

}

cout << "Total cost of MST: " << totalWeight << endl;

return 0;

}

**Max flow min cut algorithm:**

//Max Flow Min cut Algorithm

#include <bits/stdc++.h>

using namespace std;

bool bfs(const vector<vector<int>>& residualGraph, int s, int t, vector<int>& parent) {

int n = residualGraph.size();

vector<bool> visited(n, false);

queue<int> q;

q.push(s);

visited[s] = true;

parent[s] = -1;

while (!q.empty()) {

int u = q.front();

q.pop();

for (int v = 0; v < n; ++v) {

if (!visited[v] && residualGraph[u][v] > 0) {

q.push(v);

parent[v] = u;

visited[v] = true;

}

}

}

return visited[t];

}

int maxFlowMinCut(vector<vector<int>>& graph, int s, int t) {

int n = graph.size();

vector<vector<int>> residualGraph = graph;

vector<int> parent(n);

int maxFlow = 0;

while (bfs(residualGraph, s, t, parent)) {

int pathFlow = INT\_MAX;

for (int v = t; v != s; v = parent[v]) {

int u = parent[v];

pathFlow = min(pathFlow, residualGraph[u][v]);

}

for (int v = t; v != s; v = parent[v]) {

int u = parent[v];

residualGraph[u][v] -= pathFlow;

residualGraph[v][u] += pathFlow;

}

maxFlow += pathFlow;

}

return maxFlow;

}

int main() {

int n, m;

if (!(cin >> n >> m) || n <= 0 || m < 0) {

cout << "Invalid input" << endl;

return 1;

}

vector<vector<int>> graph(n, vector<int>(n, 0));

for (int i = 0; i < m; ++i) {

int u, v, capacity;

if (!(cin >> u >> v >> capacity) || u < 1 || u > n || v < 1 || v > n || capacity < 0) {

cerr << "Invalid input" << endl;

return 1;

}

graph[u - 1][v - 1] = capacity; // Adjust for 0-based indexing

}

int s, t;

if (!(cin >> s >> t) || s < 1 || s > n || t < 1 || t > n || s == t) {

cerr << "Invalid input";

return 1;

}

int maxFlow = maxFlowMinCut(graph, s - 1, t - 1); // Adjust for 0-based indexing

cout << maxFlow << endl;

return 0;

}