#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Structure to represent an edge in the graph

struct Edge {

int src, dest, weight;

};

// Structure to represent a subset for union-find

struct Subset {

int parent, rank;

};

class Graph {

int V;

vector<Edge> edges;

public:

Graph(int V) : V(V) {}

void addEdge(int src, int dest, int weight) {

Edge edge = {src, dest, weight};

edges.push\_back(edge);

}

// Find set of an element i

int find(vector<Subset>& subsets, int i) {

if (subsets[i].parent != i)

subsets[i].parent = find(subsets, subsets[i].parent);

return subsets[i].parent;

}

// Perform union of two sets

void Union(vector<Subset>& subsets, int x, int y) {

int xroot = find(subsets, x);

int yroot = find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

else if (subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

// Kruskal's algorithm to find MST

void KruskalMST() {

vector<Edge> result;

int e = 0;

int i = 0;

// Sort all edges in non-decreasing order of their weight

sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {

return a.weight < b.weight;

});

// Allocate memory for creating V subsets

vector<Subset> subsets(V);

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

subsets[v].parent = v;

subsets[v].rank = 0;

}

// Number of edges to be taken is equal to V-1

while (e < V - 1 && i < edges.size()) {

Edge next\_edge = edges[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

if (x != y) {

result.push\_back(next\_edge);

Union(subsets, x, y);

e++;

}

}

// Print the MST

cout << "Edges in the MST:\n";

int totalCost = 0;

for (i = 0; i < result.size(); i++) {

cout << result[i].src << " - " << result[i].dest << " : " << result[i].weight << "\n";

totalCost += result[i].weight;

}

cout << "Total cost of leasing phone lines for MST: " << totalCost << "\n";

}

};

int main() {

int V, E;

cout << "Enter the number of cities (vertices) and the number of connections (edges): ";

cin >> V >> E;

Graph g(V);

cout << "Enter the source, destination, and cost for each connection:\n";

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

int src, dest, weight;

cin >> src >> dest >> weight;

g.addEdge(src, dest, weight);

}

// Find minimum spanning tree using Kruskal's algorithm

g.KruskalMST();

return 0;

}

Enter the number of cities (vertices) and the number of connections (edges): 5 7

Enter the source, destination, and cost for each connection:

0 1 10

0 2 6

0 3 5

1 3 15

2 3 4

1 2 8

3 4 7

Edges in the MST:

2 - 3 : 4

0 - 3 : 5

1 - 3 : 5

3 - 4 : 7

Total cost of leasing phone lines for MST: 21