**Assignment 4**

**Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments and links should represent the distance between them. Find a minimum spanning tree using Kruskal’s algorithm or using Prim’s algorithm.**

#include <iostream>

using namespace std;

#define MAX 100

class SimpleGraph {

struct Edge {

int from;

int to;

int weight;

};

Edge edges[MAX];

int parent[MAX];

int totalEdges;

int totalNodes;

public:

SimpleGraph(int nodes, int edgesCount) {

totalNodes = nodes;

totalEdges = edgesCount;

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

parent[i] = -1;

}

}

void insertEdge(int index, int a, int b, int w) {

edges[index].from = a;

edges[index].to = b;

edges[index].weight = w;

}

void sortEdgesByWeight() {

// Simple bubble sort

for (int i = 0; i < totalEdges - 1; i++) {

for (int j = 0; j < totalEdges - i - 1; j++) {

if (edges[j].weight > edges[j + 1].weight) {

Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

}

}

}

}

int findRoot(int node) {

if (parent[node] == -1)

return node;

return parent[node] = findRoot(parent[node]); // Path compression

}

void joinSets(int root1, int root2) {

parent[root1] = root2;

}

void createMST() {

sortEdgesByWeight();

int totalWeight = 0;

cout << "\nEdges in the Minimum Spanning Tree:\n";

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

int u = edges[i].from;

int v = edges[i].to;

int w = edges[i].weight;

int rootU = findRoot(u);

int rootV = findRoot(v);

if (rootU != rootV) {

cout << u << " - " << v << " : " << w << "\n";

totalWeight += w;

joinSets(rootU, rootV);

}

}

cout << "Total Weight of MST = " << totalWeight << "\n";

}

};

int main() {

int edgesCount, nodesCount;

cout << "Enter number of nodes (departments): ";

cin >> nodesCount;

cout << "Enter number of edges (paths between departments): ";

cin >> edgesCount;

if (edgesCount < nodesCount - 1) {

cout << "Warning: A Minimum Spanning Tree requires at least (nodes - 1) edges.\n";

}

SimpleGraph g(nodesCount, edgesCount);

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

int a, b, w;

cout << "Enter edge " << i + 1 << " (format: node1 node2 distance): ";

cin >> a >> b >> w;

if (a < 1 || b < 1 || a > nodesCount || b > nodesCount) {

cout << "Invalid node number. Please enter nodes between 1 and " << nodesCount << ".\n";

i--; // re-do this edge

continue;

}

g.insertEdge(i, a, b, w);

}

g.createMST();

return 0;

}