**Experiment-4**

**Aim-** Write a program to find the Minimum Cost Spanning Tree (MST) of a given undirected graph using either Prim's Algorithm or Kruskal's Algorithm. The program should be menu-driven, allowing the user to select which algorithm to use for solving the problem

**Theory-**

A **Minimum Cost Spanning Tree (MST)** connects all vertices of a graph with the smallest possible total edge weight. **Prim's Algorithm** builds the MST by expanding a tree one edge at a time from an arbitrary starting vertex, selecting the smallest edge connecting to the tree. **Kruskal's Algorithm**, on the other hand, sorts all edges by weight and adds them one by one, ensuring no cycles are formed. Both algorithms efficiently solve the MST problem in undirected, weighted graphs.

**Software Used –** Visual Studio Code

**Code-**

#include <iostream>

#include <vector>

#include <algorithm>

#include <climits>

using namespace std;

// Structure to represent an edge for Kruskal's algorithm

struct Edge {

int u, v, weight;

bool operator<(const Edge& other) const {

return weight < other.weight;

}

};

// Disjoint set (Union-Find) to detect cycles in Kruskal's algorithm

class DisjointSet {

vector<int> parent, rank;

public:

DisjointSet(int n) {

parent.resize(n);

rank.resize(n, 0);

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

}

int find(int u) {

if (u != parent[u]) {

parent[u] = find(parent[u]);

}

return parent[u];

}

void unionSets(int u, int v) {

int rootU = find(u);

int rootV = find(v);

if (rootU != rootV) {

if (rank[rootU] > rank[rootV]) {

parent[rootV] = rootU;

} else if (rank[rootU] < rank[rootV]) {

parent[rootU] = rootV;

} else {

parent[rootV] = rootU;

rank[rootU]++;

}

}

}

};

// Function to run Kruskal's algorithm

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

sort(edges.begin(), edges.end()); // Sort edges by weight

DisjointSet ds(V); // Initialize disjoint set

vector<Edge> mst; // Store MST edges

int totalCost = 0;

for (Edge& edge : edges) {

if (ds.find(edge.u) != ds.find(edge.v)) {

ds.unionSets(edge.u, edge.v);

mst.push\_back(edge);

totalCost += edge.weight;

}

}

cout << "Minimum Cost Spanning Tree using Kruskal's Algorithm: \n";

for (Edge& e : mst) {

cout << e.u << " -- " << e.v << " == " << e.weight << endl;

}

cout << "Total Cost: " << totalCost << endl;

}

// Function to run Prim's algorithm

void primMST(int V, vector<vector<pair<int, int>>>& adj) {

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

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

vector<bool> inMST(V, false);

key[0] = 0; // Start from the first vertex

int totalCost = 0;

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

// Find the vertex with the minimum key value not in MST

int u = -1;

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

if (!inMST[i] && (u == -1 || key[i] < key[u])) {

u = i;

}

}

// Include u in MST

inMST[u] = true;

totalCost += key[u];

// Update key values of adjacent vertices

for (auto& [v, weight] : adj[u]) {

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

key[v] = weight;

parent[v] = u;

}

}

}

// Print the MST

cout << "Minimum Cost Spanning Tree using Prim's Algorithm: \n";

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

cout << parent[i] << " -- " << i << " == " << key[i] << endl;

}

cout << "Total Cost: " << totalCost << endl;

}

void addEdge(int u, int v, int weight, vector<vector<pair<int, int>>>& adj, vector<Edge>& edges) {

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

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

edges.push\_back({u, v, weight});

}

int main() {

int V, E;

cout << "Enter number of vertices: ";

cin >> V;

cout << "Enter number of edges: ";

cin >> E;

vector<vector<pair<int, int>>> adj(V); // Adjacency list for Prim's

vector<Edge> edges; // Edge list for Kruskal's

cout << "Enter the edges (u, v, weight): \n";

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

int u, v, weight;

cin >> u >> v >> weight;

addEdge(u, v, weight, adj, edges);

}

int choice;

cout << "\nChoose the algorithm:\n";

cout << "1. Prim's Algorithm\n";

cout << "2. Kruskal's Algorithm\n";

cin >> choice;

if (choice == 1) {

primMST(V, adj);

} else if (choice == 2) {

kruskalMST(V, edges);

} else {

cout << "Invalid choice!" << endl;

}

return 0;

}

**Output-**

