# Department of Computing

# School of Electrical Engineering and Computer Science

**CS-250: Data Structure and Algorithms**

**Class: BSCS 13A**

# Lab 13: MST using Prim’s and Kruskal’s Algorithms

**Date: 16th December, 2024**

**Time: 10 am - 1 pm**

**Lab Engineer: Ms Areeba Rameen**

# Name: Muhammad Muneeb ur Rehman

# Qalam: 458696

# Lab 13: MST using Prim’s and Kruskal’s Algorithms

**Question:** Write a program to implement Minimum Spanning Tree (MST) for a connected, undirected graph using both **Prim's Algorithm** and **Kruskal's Algorithm**. Your implementation should:

1. Take the graph as input as an adjacency list or an edge list.
2. Display the MST edges and the MST's total cost for both algorithms.
3. Allow students to analyse the performance of the two algorithms on the same input graph.

**Input Example:**

1. Number of vertices and edges in the graph.
2. List of edges in the format: **(u, v, weight)** for each edge.

**Output Example:**

* **For Prim's Algorithm**: Show the edges in the MST and the total cost.
* **For Kruskal's Algorithm**: Show the edges in the MST and the total cost.

**Code:**

#include <iostream>

#include <vector>

#include <queue>

#include <tuple>

#include <limits>

#include <algorithm>

using namespace std;

const int INF = numeric\_limits<int>::max(); // Representation for no edge

// Prim's Algorithm using Adjacency Matrix

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

    vector<bool> inMST(V, false);    // To keep track of vertices in MST

    vector<int> key(V, INF);        // Minimum edge weight to connect to MST

    vector<int> parent(V, -1);      // Store MST edges

    int totalCost = 0;

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

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

        int u = -1;

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

        for (int v = 0; v < V; ++v) {

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

                u = v;

            }

        }

        inMST[u] = true;

        totalCost += key[u];

        // Update key and parent for adjacent vertices

        for (int v = 0; v < V; ++v) {

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

                key[v] = adjMatrix[u][v];

                parent[v] = u;

            }

        }

    }

    // Output MST

    cout << "Prim's MST Edges:\n";

    for (int i = 1; i < V; ++i) {

        if (parent[i] != -1) {

            cout << parent[i] << " - " << i << " (Weight: " << adjMatrix[parent[i]][i] << ")\n";

        }

    }

    cout << "Total Cost of MST (Prim's): " << totalCost << endl;

}

// Kruskal's Algorithm using Edge List

struct Edge {

    int u, v, weight;

    bool operator<(const Edge &other) const {

        return weight < other.weight;

    }

};

class DSU {

    vector<int> parent, rank;

public:

    DSU(int n) {

        parent.resize(n);

        rank.resize(n, 0);

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

    }

    int find(int x) {

        if (parent[x] != x) parent[x] = find(parent[x]);

        return parent[x];

    }

    bool unite(int x, int y) {

        int rootX = find(x);

        int rootY = find(y);

        if (rootX == rootY) return false;

        if (rank[rootX] > rank[rootY]) parent[rootY] = rootX;

        else if (rank[rootX] < rank[rootY]) parent[rootX] = rootY;

        else {

            parent[rootY] = rootX;

            rank[rootX]++;

        }

        return true;

    }

};

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

    sort(edges.begin(), edges.end());

    DSU dsu(V);

    vector<pair<int, int>> mstEdges;

    int totalCost = 0;

    for (auto &edge : edges) {

        if (dsu.unite(edge.u, edge.v)) {

            mstEdges.push\_back({edge.u, edge.v});

            totalCost += edge.weight;

        }

    }

    // Output MST

    cout << "Kruskal's MST Edges:\n";

    for (auto &[u, v] : mstEdges) {

        cout << u << " - " << v << endl;

    }

    cout << "Total Cost of MST (Kruskal's): " << totalCost << endl;

}

int main() {

    int V = 5; // Number of vertices

    vector<vector<int>> adjMatrix = {

        {0, 2, 0, 6, 0},

        {2, 0, 3, 8, 5},

        {0, 3, 0, 0, 7},

        {6, 8, 0, 0, 9},

        {0, 5, 7, 9, 0}

    };

    // Convert adjacency matrix to edge list for Kruskal's Algorithm

    vector<Edge> edges;

    for (int i = 0; i < V; ++i) {

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

            if (adjMatrix[i][j] != 0) {

                edges.push\_back({i, j, adjMatrix[i][j]});

            }

        }

    }

    cout << "--- Running Prim's Algorithm ---\n";

    primMST(V, adjMatrix);

    cout << "\n--- Running Kruskal's Algorithm ---\n";

    kruskalMST(V, edges);

    return 0;

}

**Output:**

**A screenshot of a computer

Description automatically generated**