****

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Spring, Year: 2025), B.Sc. in CSE (Day)**

**Lab Report NO 02**

**Course Title: Algorithm Lab**

**Course Code: CSE 208**

**Section: D9**

**Lab Experiment Name:** MST usingKruskal’s Algorithm and Prim's Algorithm.

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
| **1.** | Rukonuzzaman Topu | 232002280 |

**Lab Date : 26/02/25**

**Submission Date : 05/03/25**

**Course Teacher’s Name : Farjana Akter Jui**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature: .....................**  **Comments: .............................................. Date: ..............................** |

**1. INTRODUCTION**

Kruskal's and Prim's algorithms are well-known techniques for determining the Minimum Spanning Tree (MST) of a weighted graph. Both methods seek to minimize the overall edge weight while ensuring all vertices remain connected. When multiple edges share the same weight, the count of distinct MSTs becomes significant, as different selections of these edges can still satisfy the spanning tree conditions. The number of unique MSTs is determined by the various ways these edges can be chosen while maintaining the tree structure. Both algorithms account for this by keeping track of and multiplying the number of possible selections at each step.

**2. OBJECTIVES**

The main goals of this lab report are:

• To implement Kruskal’s and Prim’s algorithms to determine the Minimum Spanning

Tree of of a given weighted graph.

* To compute and analyze the number of distinct MSTs by examining edges with identical weights and evaluating their possible combinations during the construction of the MST.

**3. IMPLEMENTATION**

Task 1: Find the number of distinct minimum spanning trees for a given weighted graph using Kruskal’s Algorithm.

import java.util.\*;

class KruskalMST {

    static class Edge implements Comparable<Edge> {

        int src, dest, weight;

        Edge(int src, int dest, int weight) {

            this.src = src;

            this.dest = dest;

            this.weight = weight;

        }

        public int compareTo(Edge other) {

            return Integer.compare(this.weight, other.weight);

        }

    }

    static class DisjointSet {

        int[] parent, rank;

        DisjointSet(int n) {

            parent = new int[n];

            rank = new int[n];

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

        }

        int find(int u) {

            if (parent[u] != u)

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

            return parent[u];

        }

        boolean union(int u, int v) {

            int rootU = find(u), rootV = find(v);

            if (rootU == rootV) return false;

            if (rank[rootU] > rank[rootV]) parent[rootV] = rootU;

            else if (rank[rootU] < rank[rootV]) parent[rootU] = rootV;

            else { parent[rootV] = rootU; rank[rootU]++; }

            return true;

        }

    }

    private int V;

    private List<Edge> edges;

    KruskalMST(int V) {

        this.V = V;

        this.edges = new ArrayList<>();

    }

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

        edges.add(new Edge(src, dest, weight));

    }

    int findMSTWeight() {

        Collections.sort(edges);

        DisjointSet ds = new DisjointSet(V);

        int mstWeight = 0, edgeCount = 0;

        for (Edge edge : edges) {

            if (ds.union(edge.src, edge.dest)) {

                mstWeight += edge.weight;

                edgeCount++;

                if (edgeCount == V - 1) break;

            }

        }

        return (edgeCount == V - 1) ? mstWeight : -1;

    }

    int countDistinctMSTs() {

        int mstWeight = findMSTWeight();

        if (mstWeight == -1) return 0;

        Map<Integer, Integer> edgeCountByWeight = new HashMap<>();

        for (Edge edge : edges) {

            edgeCountByWeight.put(edge.weight, edgeCountByWeight.getOrDefault(edge.weight, 0) + 1);

        }

        int count = 1;

        for (int countEdges : edgeCountByWeight.values()) {

            count \*= factorial(countEdges);

        }

        return count;

    }

    private int factorial(int n) {

        int fact = 1;

        for (int i = 2; i <= n; i++) fact \*= i;

        return fact;

    }

    public static void main(String[] args) {

        KruskalMST graph = new KruskalMST(4);

        graph.addEdge(0, 1, 2);

        graph.addEdge(1, 2, 2);

        graph.addEdge(2, 3, 2);

        graph.addEdge(0, 3, 4);

        graph.addEdge(0, 2, 5);

        graph.addEdge(1, 3, 5);

        System.out.println("Number of distinct MSTs: " + graph.countDistinctMSTs());

    }

}

Task 2: Find the number of distinct minimum spanning trees for a given weighted graph using Prim's Algorithm.

import java.util.\*;

class PrimMST {

static class Edge {

int src, dest, weight;

Edge(int src, int dest, int weight) {

this.src = src;

this.dest = dest;

this.weight = weight;

}

}

private int V;

private List<Edge> edges;

private Map<Integer, List<Edge>> adjList;

PrimMST(int V) {

this.V = V;

this.edges = new ArrayList<>();

this.adjList = new HashMap<>();

}

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

edges.add(new Edge(src, dest, weight));

adjList.computeIfAbsent(src, k -> new ArrayList<>()).add(new Edge(src, dest, weight));

adjList.computeIfAbsent(dest, k -> new ArrayList<>()).add(new Edge(dest, src, weight));

}

int findMSTWeight() {

boolean[] inMST = new boolean[V];

int[] key = new int[V];

Arrays.fill(key, Integer.MAX\_VALUE);

key[0] = 0;

PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.comparingInt(i -> key[i]));

pq.offer(0);

int mstWeight = 0;

while (!pq.isEmpty()) {

int u = pq.poll();

if (inMST[u]) continue;

inMST[u] = true;

mstWeight += key[u];

for (Edge edge : adjList.get(u)) {

int v = edge.dest;

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

key[v] = edge.weight;

pq.offer(v);

}

}

}

return mstWeight;

}

int countDistinctMSTs() {

int mstWeight = findMSTWeight();

if (mstWeight == Integer.MAX\_VALUE) return 0;

boolean[] inMST = new boolean[V];

int[] key = new int[V];

Arrays.fill(key, Integer.MAX\_VALUE);

key[0] = 0;

PriorityQueue<Integer> pq = new PriorityQueue<>(Comparator.comparingInt(i -> key[i]));

pq.offer(0);

Map<Integer, Integer> weightChoices = new HashMap<>();

while (!pq.isEmpty()) {

int u = pq.poll();

if (inMST[u]) continue;

inMST[u] = true;

for (Edge edge : adjList.get(u)) {

int v = edge.dest;

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

key[v] = edge.weight;

pq.offer(v);

weightChoices.put(edge.weight, weightChoices.getOrDefault(edge.weight, 0) + 1);

}

}

}

int distinctMSTCount = 1;

for (int count : weightChoices.values()) {

distinctMSTCount \*= factorial(count);

}

return distinctMSTCount;

}

private int factorial(int n) {

int fact = 1;

for (int i = 2; i <= n; i++) {

fact \*= i;

}

return fact;

}

public static void main(String[] args) {

PrimMST graph = new PrimMST(4);

graph.addEdge(0, 1, 1);

graph.addEdge(1, 2, 1);

graph.addEdge(2, 3, 2);

graph.addEdge(0, 3, 1);

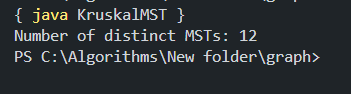
graph.addEdge(0, 2, 3);

System.out.println("Number of distinct MSTs: " + graph.countDistinctMSTs());

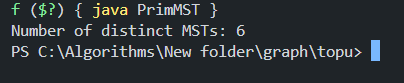
}

}

Output1:

****

**Output 2:**

****

**4. DISCUSSION**

Both Kruskal’s and Prim’s algorithms efficiently determine a graph’s Minimum Spanning Tree (MST). However, when calculating the number of distinct MSTs, it is crucial to consider the edges with identical weights that can be selected at each step. This adds complexity to the problem but also makes it more intriguing, as it incorporates a combinatorial element into MST formation.