Worksheet No. - 3

Student Name: Devanand Utkarsh
UID: 24MCA20454
Branch: MCA
Section/Group: 6 (B)
Semester: II
Date of Performance: 20-02-2025
Subject Name: DAA LAB
Subject Code: 24CAP612

Aim/Overview of the practical: Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Objective: To find the Minimum Cost Spanning Tree (MCST) of a given undirected graph using Kruskal's algorithm. The algorithm follows a greedy approach, sorting edges by weight and using the find and union method to avoid cycles. The goal is to understand MST construction and analyze its efficiency.

Input/Apparatus Used: IntelliJ Idea as code editior.

Algorithms Steps:

1. **Sort:** all edges in non-decreasing order based on their weight**.**
2. **Initialize:** a parent array where each vertex is its own parent.
3. **Iterate:** through the sorted edges and check if they form a cycle using the Union-Find data structure**.**
4. **Include:** an edge in the MST if it does not create a cycle**.**
5. **Repeat:** until we have `V-1` edges in the MST (where `V` is the number of vertices).

Procedure/Algorithm/Code:

```
package graph;

import java.util.*;

class Edge implements Comparable<Edge> {
    int src, dest, weight;

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

```
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }

    @Override
    public int compareTo(Edge other) {
        return this.weight - other.weight;
    }
}

class DisjointSet {
    int[] parent, rank;

    public DisjointSet(int n) {
        parent = new int[n];
        rank = new int[n];
        for (int i = 0; i < n; i++) parent[i] = i;
    }

    public int find(int node) {
        if (parent[node] != node) {
            parent[node] = find(parent[node]); // Path compression
        }
        return parent[node];
    }

    public void union(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]++;
            }
        }
```

```
    }
}

public class KruskalMST {

    public static List<Edge> kruskalMST(List<Edge> edges, int V) {
        List<Edge> mst = new ArrayList<>();
        Collections.sort(edges); // Sort edges by weight
        DisjointSet ds = new DisjointSet(V);

        for (Edge edge : edges) {
            if (ds.find(edge.src) != ds.find(edge.dest)) { // Check if cycle forms
                mst.add(edge);
                ds.union(edge.src, edge.dest);
                if (mst.size() == V - 1) break; // Stop when MST is complete
            }
        }
        return mst;
    }


    public static void main(String[] args) {
        int V = 4; // Number of vertices
        List<Edge> edges = new ArrayList<>();

        // Adding edges (source, destination, weight)
        edges.add(new Edge(0, 1, 10));
        edges.add(new Edge(0, 2, 6));
        edges.add(new Edge(0, 3, 5));
        edges.add(new Edge(1, 3, 15));
        edges.add(new Edge(2, 3, 4));

        List<Edge> mst = kruskalMST(edges, V);

        System.out.println("Graph Edges ");
        for (Edge edge : edges) {
            System.out.println(edge.src + " - " + edge.dest + " : " + edge.weight);
        }

        System.out.println("Minimum Spanning Tree:");
```

```
        for (Edge edge : mst) {
            System.out.println(edge.src + " - " + edge.dest + " : " + edge.weight);
        }
    }

}
```

Output:

```
C:\Users\HELLO\.jdks\openjdk-22.0.2\bin\java.exe "
Graph Edges
2 - 3 : 4
0 - 3 : 5
0 - 2 : 6
0 - 1 : 10
1 - 3 : 15
Minimum Spanning Tree:
2 - 3 : 4
0 - 3 : 5
0 - 1 : 10

Process finished with exit code 0
```

Learning outcomes (What I have learnt):

1. Sorting helps process edges in optimal order.
2. find() and union() prevent cycles efficiently.
3. Not all edges need to be processed; MST stops early.
4. The final MST is unique for a given set of weights.

Evaluation Grid:

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.	Demonstration and Performance		12
2.	Worksheet		8
3.	Viva		10