Assignment-16

Implement Kruskal's algorithm in Java to find the Minimum Spanning Tree (MST) of a given undirected, weighted 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;
    }

    // Comparator function used for sorting edges based on their weight
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
}

class Subset {
    int parent, rank;
}
```

```
public class KruskalAlgorithmDemo {
    int vertices, edges;
    Edge edge[];

    // Create a graph with V vertices and E edges
    KruskalAlgorithmDemo(int v, int e) {
        vertices = v;
        edges = e;
        edge = new Edge[e];
        for (int i = 0; i < e; ++i) {
            edge[i] = new Edge(0, 0, 0);
        }
    }

    // Find set of an element i (uses path compression technique)
    int find(Subset subsets[], int i) {
        if (subsets[i].parent != i)
            subsets[i].parent = find(subsets, subsets[i].parent);
        return subsets[i].parent;
    }

    // A function that does union of two sets of x and y (uses union by
rank)
    void union(Subset subsets[], int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);
```

```
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

// The main function to construct MST using Kruskal's algorithm
void KruskalMST() {
    Edge result[] = new Edge[vertices];
    int e = 0; // Index used to pick the next edge
    int i = 0; // Index used to sort the edges
    for (i = 0; i < vertices; ++i)
        result[i] = new Edge(0, 0, 0);

    Arrays.sort(edge);

    Subset subsets[] = new Subset[vertices];
    for (i = 0; i < vertices; ++i)
        subsets[i] = new Subset();

    for (int v = 0; v < vertices; ++v) {
```

```
            subsets[v].parent = v;
            subsets[v].rank = 0;
        }

        i = 0;

        while (e < vertices - 1) {
            Edge next_edge = new Edge(0, 0, 0);
            next_edge = edge[i++];

            int x = find(subsets, next_edge.src);
            int y = find(subsets, next_edge.dest);

            if (x != y) {
                result[e++] = next_edge;
                union(subsets, x, y);
            }
        }

        System.out.println("Following are the edges in the constructed
MST");
        for (i = 0; i < e; ++i)
            System.out.println(result[i].src + " -- " +
                    result[i].dest + " == " + result[i].weight);
    }

    public static void main(String[] args) {
```

```
        int vertices = 9; // Number of vertices in graph
        int edges = 14; // Number of edges in graph
        KruskalAlgorithmDemo graph = new
KruskalAlgorithmDemo(vertices, edges);

        graph.edge[0] = new Edge(7, 6, 1);

        graph.edge[1] = new Edge(8, 2, 2);
        graph.edge[2] = new Edge(6, 5, 2);

        graph.edge[3] = new Edge(0, 1, 4);
        graph.edge[4] = new Edge(2, 5, 4);

        graph.edge[5] = new Edge(8, 6, 6);

        graph.edge[6] = new Edge(2, 3, 7);
        graph.edge[7] = new Edge(7, 8, 7);

        graph.edge[8] = new Edge(0, 7, 8);
        graph.edge[9] = new Edge(1, 2, 8);

        graph.edge[10] = new Edge(3, 4, 9);

        graph.edge[11] = new Edge(5, 4, 10);

        graph.edge[12] = new Edge(1, 7, 11);
```

```
        graph.edge[13] = new Edge(3, 5, 14);



        graph.KruskalMST();
    }
}
```