Name: Vijay Misal

Div: C Batch: C3

Roll No: 233073

PRN No: 22320079

**Practical No: 7**

**Title:** Represent any real world graph using adjacency list /adjacency matrix. Find minimum spanning tree using Kruskal’s algorithm.

**Code:**

import java.util.\*;

class **Graph** {

    private int V;

    private LinkedList<Edge>[] adjList;

    public **Graph**(int V) {

        this.V = V;

        adjList = new LinkedList[V];

        for (int i = 0; i < V; ++i)

            adjList[i] = new LinkedList<>();

    }

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

        Edge edge = new **Edge**(src, dest, weight);

        adjList[src].**add**(edge);

    }

    public LinkedList<Edge>[] **getAdjList**() {

        return adjList;

    }

}

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;

    }

    public int **compareTo**(Edge compareEdge) {

        return this.weight - compareEdge.weight;

    }

}

public class **KruskalAlgorithm** {

    private int V;

    private LinkedList<Edge>[] adjList;

    public **KruskalAlgorithm**(int V, LinkedList<Edge>[] adjList) {

        this.V = V;

        this.adjList = adjList;

    }

    private int **find**(int[] parent, int i) {

        if (parent[i] == i)

            return i;

        return **find**(parent, parent[i]);

    }

    private void **union**(int[] parent, int[] rank, int x, int y) {

        int xroot = **find**(parent, x);

        int yroot = **find**(parent, y);

        if (rank[xroot] < rank[yroot])

            parent[xroot] = yroot;

        else if (rank[xroot] > rank[yroot])

            parent[yroot] = xroot;

        else {

            parent[yroot] = xroot;

            rank[xroot]++;

        }

    }

    public void **kruskalMST**() {

        List<Edge> edges = new ArrayList<>();

        for (LinkedList<Edge> list : adjList) {

            edges.**addAll**(list);

        }

        Collections.**sort**(edges);

        Edge[] result = new Edge[V];

        int e = 0;

        int i = 0;

        int[] parent = new int[V];

        int[] rank = new int[V];

        for (int v = 0; v < V; ++v) {

            parent[v] = v;

            rank[v] = 0;

        }

        while (e < V - 1 && i < edges.**size**()) {

            Edge nextEdge = edges.**get**(i++);

            int x = **find**(parent, nextEdge.src);

            int y = **find**(parent, nextEdge.dest);

            if (x != y) {

                result[e++] = nextEdge;

**union**(parent, rank, x, y);

            }

        }

        System.out.**println**("Edges in the 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 V = 4;

        Graph graph = new **Graph**(V);

        graph.**addEdge**(0, 1, 10);

        graph.**addEdge**(0, 2, 6);

        graph.**addEdge**(0, 3, 5);

        graph.**addEdge**(1, 3, 15);

        graph.**addEdge**(2, 3, 4);

        LinkedList<Edge>[] adjList = graph.**getAdjList**();

        KruskalAlgorithm kruskal = new **KruskalAlgorithm**(V, adjList);

        kruskal.**kruskalMST**();

    }

}

**Output:**Edges in the MST:

2 - 3 : 4

0 - 3 : 5

0 - 1 : 10