Worksheet 4(a)

Student Name: Rahul Saxena **UID:** 24MCI10204
Branch: MCA(AI&ML) **Section/Group:** 3-B
Semester: 1st semester **Date of Performance:** 10/09/2024
Subject Name: Desing and Analysis of Algorithm Lab **Subject Code:** 24CAP-612

Aim/Overview of the practical:

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Task To be done:

- Sort all edges in increasing order of their weight.
- Pick the smallest edge. Check if adding this edge forms a cycle using a disjoint-set/union-find data structure.
- Add the edge to the MST if it doesn't form a cycle.
- Repeat steps 2 and 3 until there are (V - 1) edges in the MST, where V is the number of vertices in the graph.

Source Code:

```
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;
    }
    public int compareTo(Edge compareEdge) {
        return this.weight - compareEdge.weight;
    }
}
class Graph {
    int V, E;
    Edge[] edge;
    Graph(int v, int e) {
```

```
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
            edge[i] = new Edge(0, 0, 0);
    }
    int find(int parent[], int i) {
        if (parent[i] == i)
            return i;
        return find(parent, parent[i]);
    }
    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]++;
        }
    }
    void KruskalMST() {
        Edge[] result = new Edge[V];
        int e = 0;
        int i = 0;
        Arrays.sort(edge);
        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) {
            Edge nextEdge = edge[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 Minimum Cost Spanning Tree:");
        int minCost = 0;
        for (i = 0; i < e; ++i) {
            System.out.println(result[i].src + " -- " + result[i].dest + " == " + result[i].weight);
            minCost += result[i].weight;
        }
        System.out.println("Minimum Cost Spanning Tree: " + minCost);
    }
}
class KruskalAlgorithm {
    public static void main(String[] args) {
        int V = 4;
        int E = 5;
        Graph graph = new Graph(V, E);
        graph.edge[0] = new Edge(0, 1, 10);
        graph.edge[1] = new Edge(0, 2, 6);
        graph.edge[2] = new Edge(0, 3, 5);
        graph.edge[3] = new Edge(1, 3, 15);
        graph.edge[4] = new Edge(2, 3, 4);
        graph.KruskalMST();
    }
}
```

Output:

```
Edges in the Minimum Cost Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19

Process finished with exit code 0
```

Learning Outcome:

- **Understanding Kruskal's Algorithm:** Learn how to apply Kruskal's algorithm to find the Minimum Cost Spanning Tree (MST) of a graph. The algorithm focuses on greedily selecting the smallest edges, ensuring no cycles are formed in the process.
- **Using Union-Find:** Gain hands-on experience with the disjoint-set data structure (union-find) to detect cycles efficiently in an undirected graph.
- **Graph Representation:** Learn how to represent an undirected graph using an edge list and implement operations like sorting edges and handling connected components dynamically.
- **Time Complexity:** Understand the time complexity and efficiency of Kruskal's algorithm for computing MST in sparse graphs.