Day 16

Task 1: Kruskal’s Algorithm for MST

Implement Kruskal’s algorithm to find the minimum spanning tree of a given connected, undirected graph with non-negative edge weights

A)

Java code :

```
package Day16;
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 Graph {
   int V, E;
   List<Edge> edges;

   public Graph(int V, int E) {
      this.V = V;
      this.E = E;
      edges = new ArrayList<>();
   }

   public void addEdge(int src, int dest, int weight) {
      edges.add(new Edge(src, dest, weight));
   }

   public List<Edge> getEdges() {
      return edges;
```

```
    }
}

class Subset {
    int parent, rank;

    public Subset(int parent, int rank) {
        this.parent = parent;
        this.rank = rank;
    }
}

public class KruskalAlgorithm1 {
            private int find(Subset[] subsets, int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent = find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }

    private void union(Subset[] subsets, int x, int y) {
        int rootX = find(subsets, x);
        int rootY = find(subsets, y);

        if (subsets[rootX].rank < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        } else if (subsets[rootX].rank > subsets[rootY].rank) {
            subsets[rootY].parent = rootX;
        } else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }

    public List<Edge> kruskalMST(Graph graph) {
        List<Edge> result = new ArrayList<>();
        List<Edge> edges = graph.getEdges();

        Collections.sort(edges);

        Subset[] subsets = new Subset[graph.V];
        for (int v = 0; v < graph.V; ++v) {
            subsets[v] = new Subset(v, 0);
        }

        int e = 0;
        int i = 0;
        while (e < graph.V - 1) {
            Edge nextEdge = edges.get(i++);

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

            if (x != y) {
                result.add(nextEdge);
                union(subsets, x, y);
                e++;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int V = 4;
        int E = 5;
        Graph graph = new Graph(V, E);

        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);

        KruskalAlgorithm1 kruskal = new KruskalAlgorithm1();
        List<Edge> mst = kruskal.kruskalMST(graph);

        System.out.println("Edges in the constructed MST:");
        for (Edge edge : mst) {
            System.out.println(edge.src + " -- " + edge.dest + " == " + edge.weight);
        }
    }
}
```

Explanation:

1.Edge Class: Represents an edge with source, destination, and weight. Implements Comparable to allow sorting by weight.

2.Graph Class: Represents the graph with a list of edges. Contains methods to add edges and retrieve them.

3.Subset Class: Represents subsets for the union-find structure, with parent and rank.

4.KruskalAlgorithm Class:

- find: Uses path compression to find the representative of a subset.

- union: Uses union by rank to merge two subsets.
- kruskalMST: Implements
- Kruskal's algorithm:
- Sorts edges by weight.
- Initializes subsets for union-find.

Iterates over sorted edges and adds them to the MST if they do not form a cycle.

5.Main Method**:** Creates a graph, finds the MST using Kruskal's algorithm, and prints the edges of the MST.

This implementation ensures that the MST is correctly constructed and displays the edges included in the MST along with their weights