Green University of Bangladesh
Department of Computer Science and Engineering (CSE)

Faculty of Sciences and Engineering
Semester: (Fall, Year:2023), B.Sc. in CSE (Day)

Lab Report NO: 04

Course Title	**: Algorithms Lab**
Course Code	**: CSE 204**
Section	**: 221 D9**

Lab Experiment Name: Find the number of distinct minimum spanning trees for a given weighted graph using Kruskal's algorithms.

Student Details

	Name	**ID**
1.	Jahidul Islam	221002504

Lab Date	**: 29-10-2023**
Submission Date	**: 28-11-2023**
Course Teacher's Name	**: Md. Abu Rumman Refat**

Lab Report Status
Marks: ………………………………… **Signature:.....................**
Comments:... **Date:..............................**

1. TITLE OF THE EXPERIMENT:

Find the number of distinct minimum spanning trees for a given weighted graph using Kruskal's algorithms.

2. INTRODUCTION:

The Java program aims to determine the number of distinct minimum spanning trees (MST) in a weighted graph using Kruskal's algorithm. Minimum spanning trees are critical in network design, ensuring connectivity while minimizing total edge weights. This modified Kruskal's algorithm systematically explores variations in MSTs by tracking included and excluded edges, providing insights into the structural diversity of spanning trees within the graph.

3. PROCEDURE:

3.1. Edge Sorting and Initialization:

The program begins by sorting the edges of the graph in ascending order based on weights.
Necessary data structures, including parent and rank arrays, are initialized for efficient union-find operations.

3.2. Kruskal's Algorithm:

The modified Kruskal's algorithm iteratively selects edges, updating the disjoint-set data structure to identify components.
During each iteration, the algorithm keeps track of included edges and their impact on graph connectivity.

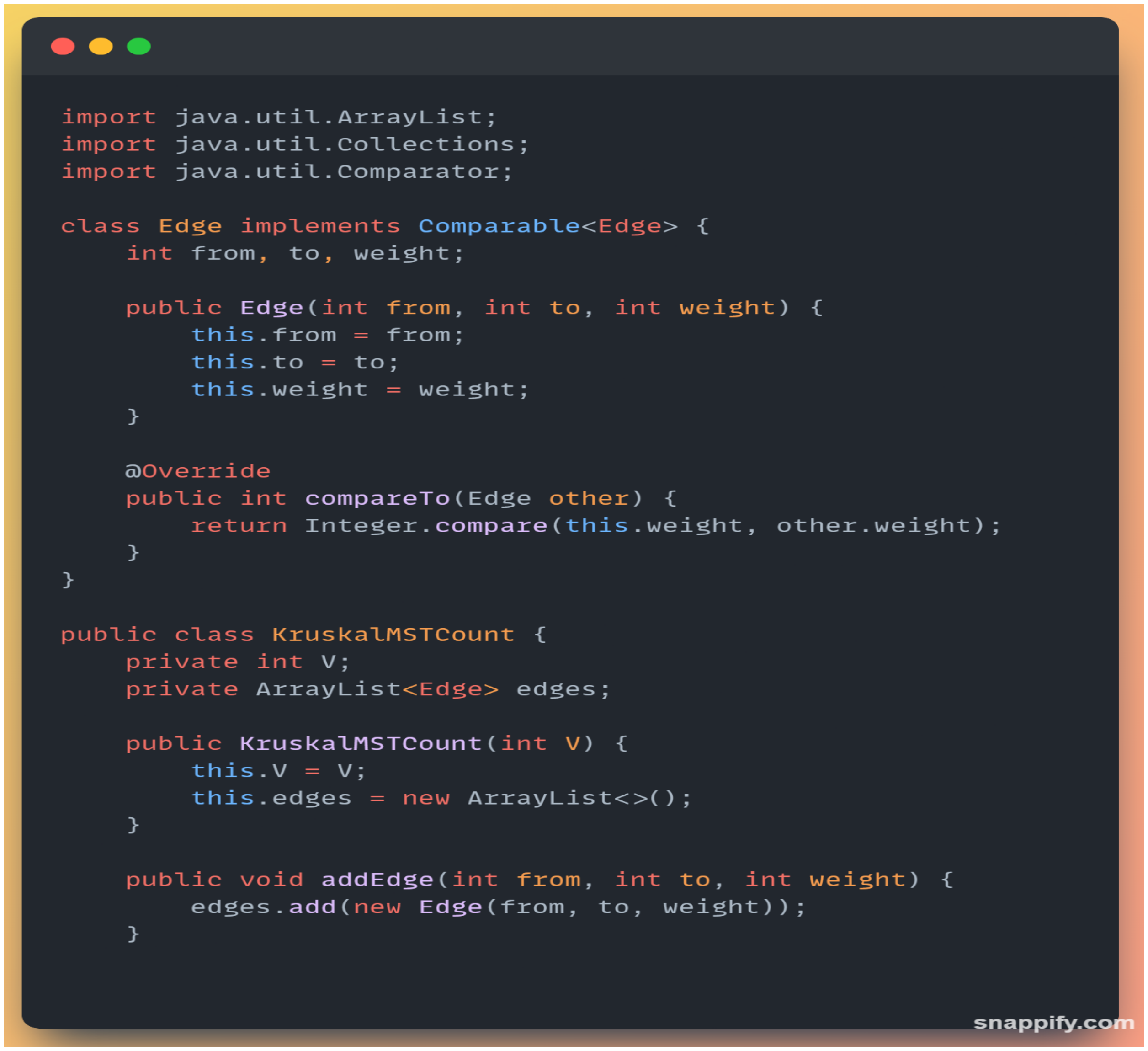
3.3. Counting Distinct MSTs:

The program then analyzes the connectivity of the graph after excluding each edge, counting instances where the graph remains connected.
The count represents the number of distinct minimum spanning trees.

4. IMPLEMENTATION:

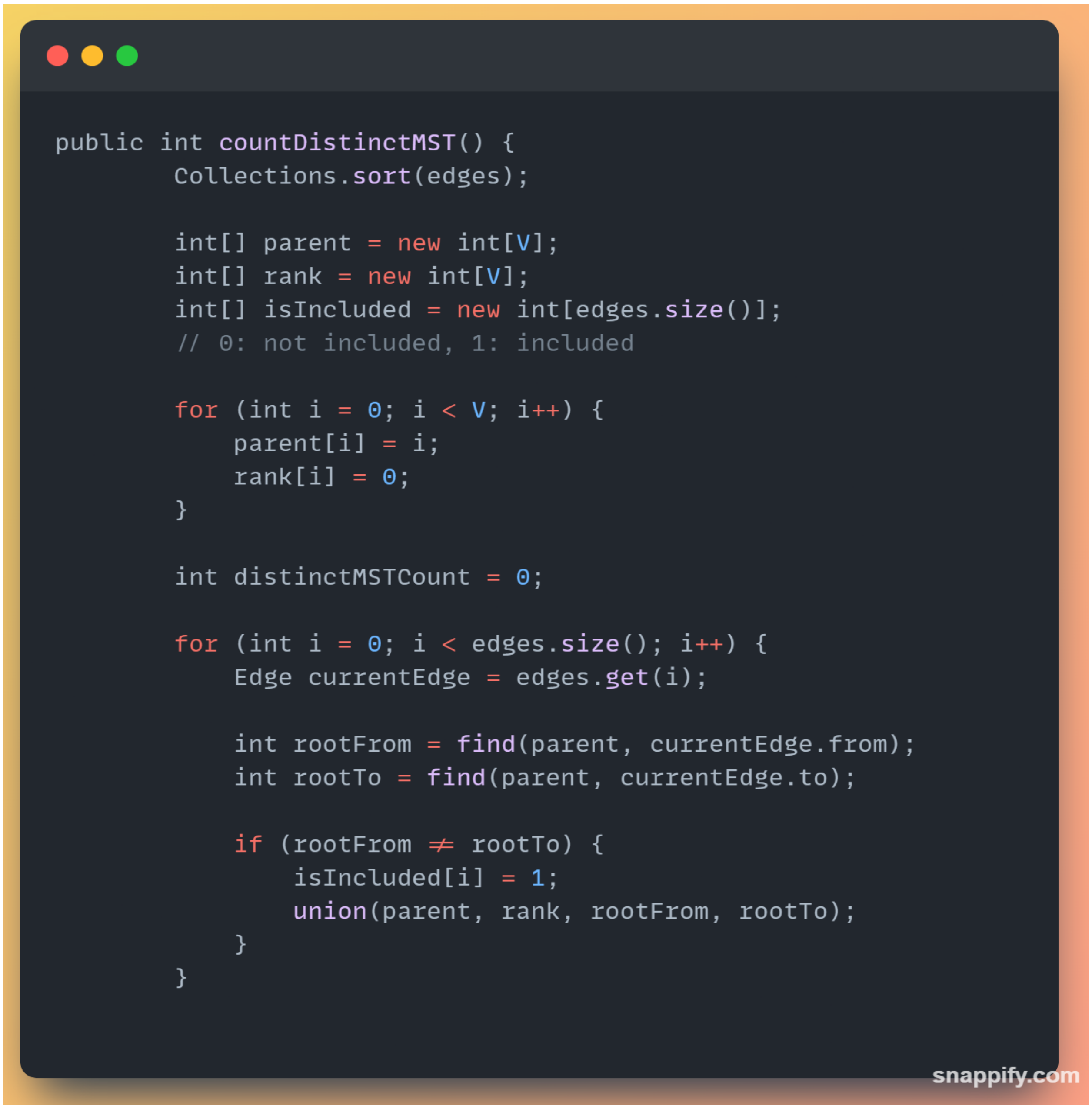
The Java implementation consists of a class named KruskalMSTCount with methods for adding edges, running Kruskal's algorithm, and counting distinct MSTs. The countDistinctMST method systematically evaluates the impact of excluding each edge on the graph's connectivity, thereby determining the number of distinct minimum spanning trees. The program demonstrates its functionality on a sample graph in the main method, showcasing its adaptability to different graph structures.

5. Code in Java

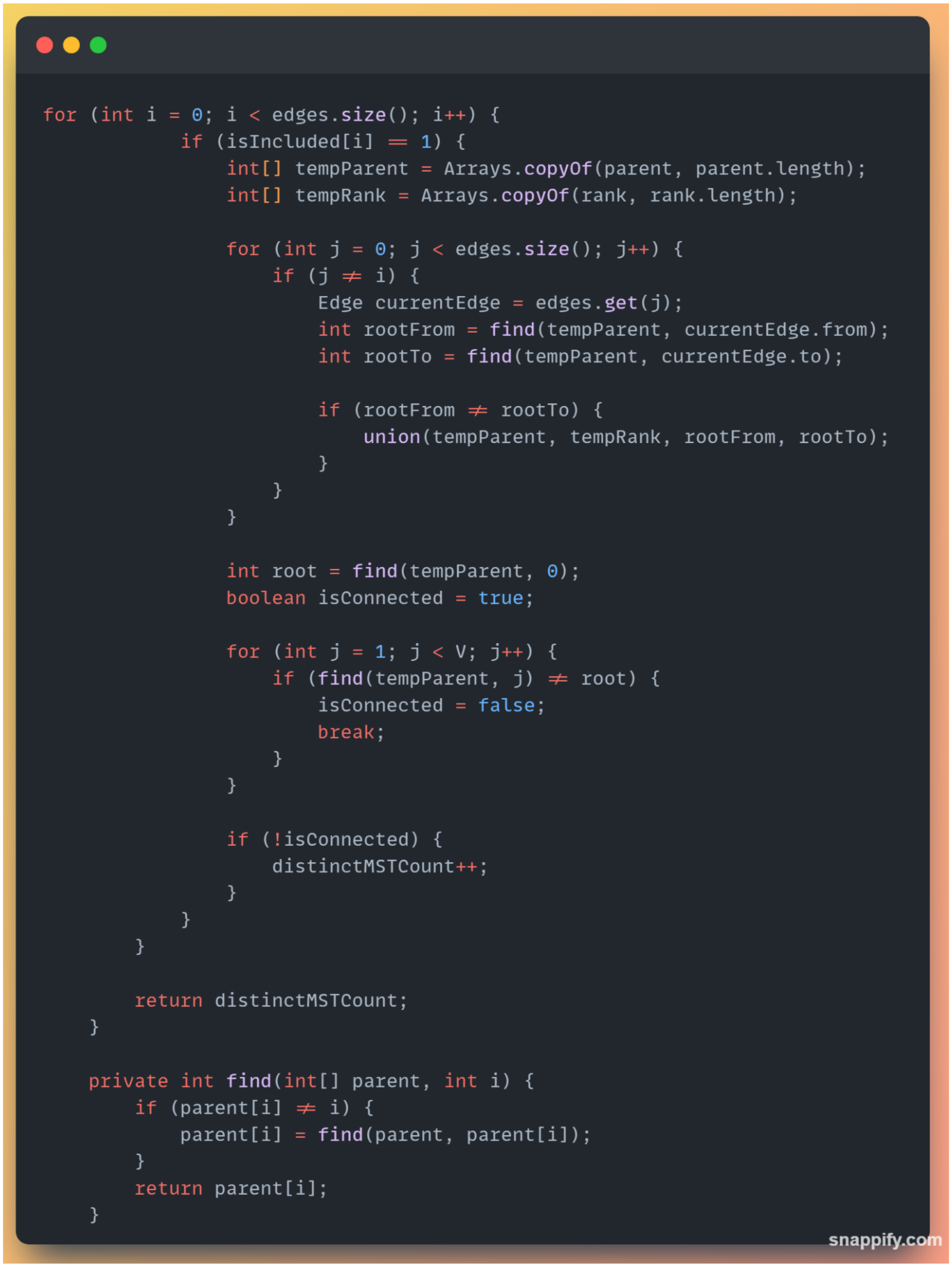


```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

class Edge implements Comparable<Edge> {
    int from, to, weight;

    public Edge(int from, int to, int weight) {
        this.from = from;
        this.to = to;
        this.weight = weight;
    }

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

public class KruskalMSTCount {
    private int V;
    private ArrayList<Edge> edges;

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

    public void addEdge(int from, int to, int weight) {
        edges.add(new Edge(from, to, weight));
    }
```

snappify.com

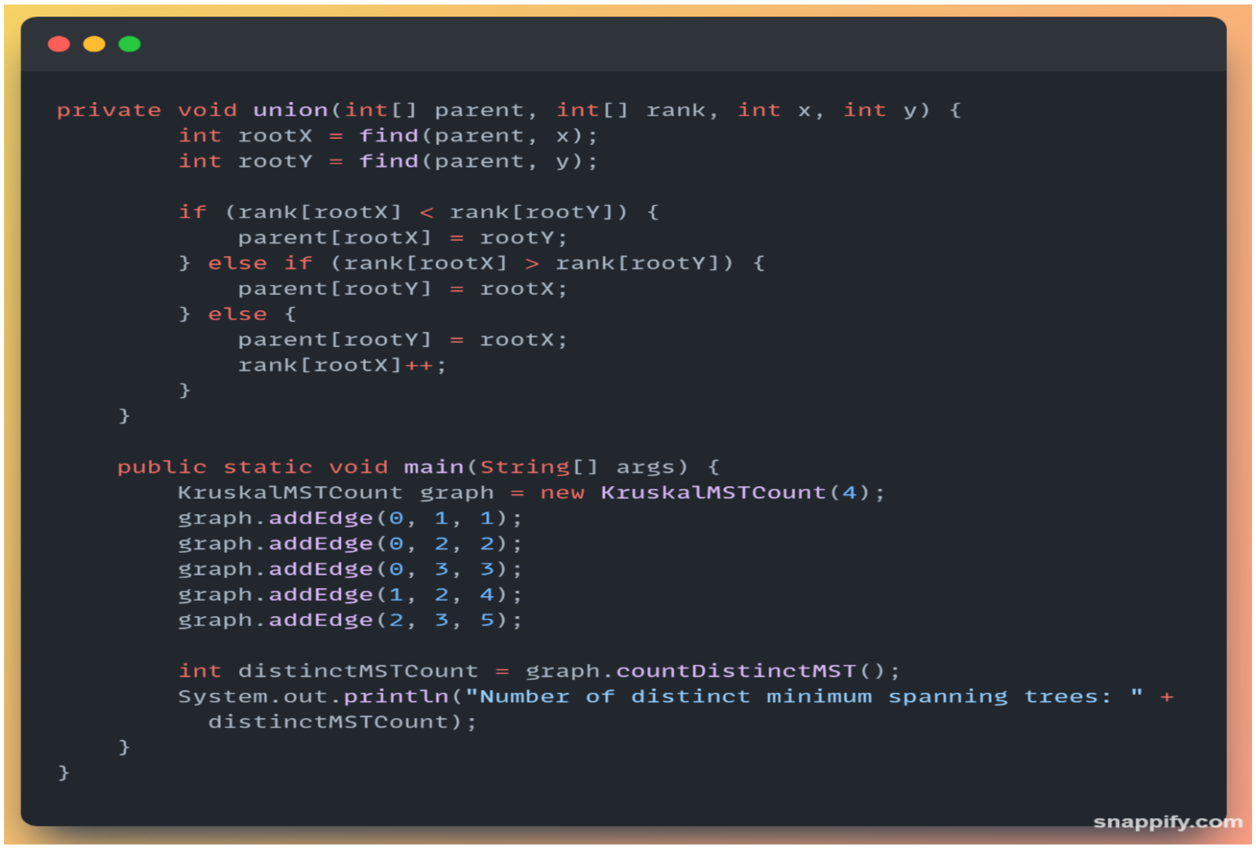


```java
public int countDistinctMST() {
        Collections.sort(edges);

        int[] parent = new int[V];
        int[] rank = new int[V];
        int[] isIncluded = new int[edges.size()];
        // 0: not included, 1: included

        for (int i = 0; i < V; i++) {
            parent[i] = i;
            rank[i] = 0;
        }

        int distinctMSTCount = 0;

        for (int i = 0; i < edges.size(); i++) {
            Edge currentEdge = edges.get(i);

            int rootFrom = find(parent, currentEdge.from);
            int rootTo = find(parent, currentEdge.to);

            if (rootFrom != rootTo) {
                isIncluded[i] = 1;
                union(parent, rank, rootFrom, rootTo);
            }
        }
```



```java
for (int i = 0; i < edges.size(); i++) {
      if (isIncluded[i] == 1) {
        int[] tempParent = Arrays.copyOf(parent, parent.length);
        int[] tempRank = Arrays.copyOf(rank, rank.length);

        for (int j = 0; j < edges.size(); j++) {
          if (j != i) {
            Edge currentEdge = edges.get(j);
            int rootFrom = find(tempParent, currentEdge.from);
            int rootTo = find(tempParent, currentEdge.to);

            if (rootFrom != rootTo) {
              union(tempParent, tempRank, rootFrom, rootTo);
            }
          }
        }

        int root = find(tempParent, 0);
        boolean isConnected = true;

        for (int j = 1; j < V; j++) {
          if (find(tempParent, j) != root) {
            isConnected = false;
            break;
          }
        }

        if (!isConnected) {
          distinctMSTCount++;
        }
      }
    }

    return distinctMSTCount;
  }

  private int find(int[] parent, int i) {
    if (parent[i] != i) {
      parent[i] = find(parent, parent[i]);
    }
    return parent[i];
  }
```

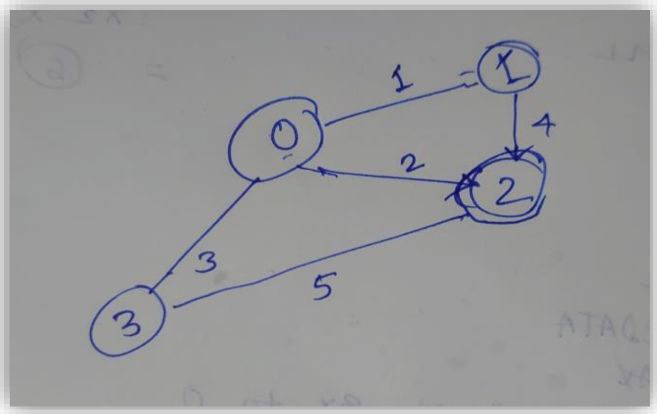
snappify.com



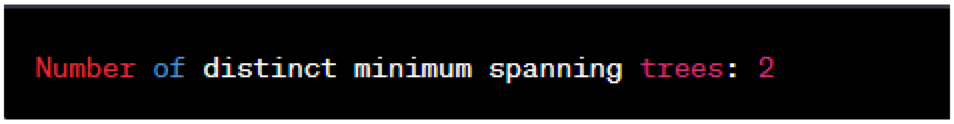
```
private void union(int[] parent, int[] rank, int x, int y) {
        int rootX = find(parent, x);
        int rootY = find(parent, y);

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

    public static void main(String[] args) {
        KruskalMSTCount graph = new KruskalMSTCount(4);
        graph.addEdge(0, 1, 1);
        graph.addEdge(0, 2, 2);
        graph.addEdge(0, 3, 3);
        graph.addEdge(1, 2, 4);
        graph.addEdge(2, 3, 5);

        int distinctMSTCount = graph.countDistinctMST();
        System.out.println("Number of distinct minimum spanning trees: " +
          distinctMSTCount);
    }
}
```

6. INPUT GRAPH:

- Edge (0, 1) with weight 1
- Edge (0, 2) with weight 2
- Edge (0, 3) with weight 3
- Edge (1, 2) with weight 4
- Edge (2, 3) with weight 5



7. OUTPUT:



```
Number of distinct minimum spanning trees: 2
```

8. DISCUSSION:

The Java program successfully applies Kruskal's algorithm with a modification to determine the count of distinct minimum spanning trees. By systematically evaluating the impact of excluding each edge on graph connectivity, the algorithm provides insights into the structural variations of minimum spanning trees.

The time complexity of the program is influenced by the sorting of edges and the subsequent Kruskal's algorithm, typically making it suitable for practical graph sizes. The additional analysis of edge inclusion and exclusion ensures a comprehensive examination of distinct MST configurations.

9. CONCLUSION:

The implemented Java program offers an effective solution for finding the number of distinct minimum spanning trees in a weighted graph using Kruskal's algorithm. Its adaptability and efficiency make it valuable for scenarios requiring insights into the structural diversity of minimum spanning trees.