Task 2: 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.

Code-:

```
package com.wipro.assignment;

import java.util.Arrays;

public class KruskalMST {

    static class Edge implements
Comparable<Edge> {
        int source, destination,
weight;

        public Edge(int i, int j, int
k) {
         // TODO Auto-generated
constructor stub
      }

      @Override
```

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

    public static void
KruskalMST(Edge[] edges, int V) {
        // Sort edges by weight
(ascending order)
        Arrays.sort(edges);

        Subset[] subsets = new
Subset[V];

        // Create `V` subsets with
single elements
        for (int i = 0; i < V; ++i) {
            subsets[i] = new
Subset(i, 1);
        }

        int e = 0;  // To count the
number of edges added
```

```
        int result = 0;  // To store
the weight of MST

        // Pick edges one by one in
sorted order
        while (e < V - 1) {
            Edge nextEdge =
edges[e++];

            int x = find(subsets,
nextEdge.source);
            int y = find(subsets,
nextEdge.destination);

            // If including this edge
doesn't cause a cycle
            if (x != y) {
                Union(subsets, x, y);
                result +=
nextEdge.weight;
            }
        }

        System.out.println("Following
are the edges in constructed MST");
        for (Edge edge : edges) {
```

```
            int x = find(subsets,
edge.source);
            int y = find(subsets,
edge.destination);
            if (x != y) {

System.out.println(edge.source + " --
" + edge.destination + "   " +
edge.weight);
            }
        }
        System.out.println("Total
weight of MST: " + result);
    }

    // Utility function to find the
root with path compression
    static int find(Subset[] subsets,
int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent =
find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }
```

```
    // Union function using Union by
Rank
    static void Union(Subset[]
subsets, int x, int y) {
        int xroot = find(subsets, x);
        int yroot = find(subsets, y);

        // Attach the smaller rank
tree under the root of the larger
rank tree
        if (subsets[xroot].rank <
subsets[yroot].rank) {
            subsets[xroot].parent =
yroot;
        } else if
(subsets[xroot].rank >
subsets[yroot].rank) {
            subsets[yroot].parent =
xroot;
        } else {
            // If ranks are same,
then make one as root and increment
its rank by 1
            subsets[yroot].parent =
xroot;
            subsets[xroot].rank++;
        }
```

```
    }

    public static void main(String[]
args) {
        /*
         * Let us create following
graph
         *     2    3
         *   (0)--(1)--(2)
         *    | / \   |
         *  4/   \5  |
         *         (3)
         */
        int V = 4;  // Number of
vertices in graph
        Edge[] edges = new Edge[5];

        edges[0] = new Edge(0, 1, 2);
        edges[1] = new Edge(0, 3, 4);
        edges[2] = new Edge(1, 2, 3);
        edges[3] = new Edge(1, 3, 5);
        edges[4] = new Edge(2, 3, 1);

        KruskalMST(edges, V);
    }

    static class Subset {
```

```
        int parent, rank;

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

Output: -