# Practical 6: Greedy - Kruskal using Union Find

## Aim

To **Understand** and **Implement** the Kruskal's Algorithm using Union Find Greedy Approach, analyse space and time complexity of it.

## Algorithm

1. **Sort** the edges of the graph by weight in non-decreasing order.
2. **Initialize** an empty set of edges S.
3. **Initialize** an empty Union-Find data structure with V disjoint sets, where V is the number of vertices in the graph.
4. For each edge **e = (u, v)** in the sorted list of edges:
   a. **Find** the sets that u and v belong to using the **find()** operation of the Union-Find data structure.
   b. If the sets are different, **add** e to S and **merge** the sets using the **union()** operation.
   c. If the sets are the same, **skip** e to avoid creating a cycle.
5. **Return** S, which contains the edges of the minimum spanning tree.

## Program

```
import java.util.*;

public class Kruskal {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of vertices: ");
        int V = sc.nextInt();
        System.out.print("Enter the number of edges: ");
        int E = sc.nextInt();

        Graph graph = new Graph(V, E);

        // Adding edges
        for (int i = 0; i < E; i++) {
            System.out.println("Enter the source, destination, and weight of edge " + (i + 
1) + ":");
            graph.edges[i].src = sc.nextInt();
            graph.edges[i].dest = sc.nextInt();
            graph.edges[i].weight = sc.nextInt();
        }

        graph.kruskal();
    }
}

class Graph {
    static class Edge implements Comparable<Edge> {
        int src, dest, weight;
        public int compareTo(Edge other) {
            return weight - other.weight;
        }
    }

    int V, E;
    Edge[] edges;
```

```java
    Graph(int v, int e) {
        V = v;
        E = e;
        edges = new Edge[E];
        for (int i = 0; i < E; ++ i)
            edges[i] = new Edge();
    }

    int find(int[] parent, int i) {
        if (parent[i] == -1)
            return i;
        return find(parent, parent[i]);
    }

    void union(int[] parent, int x, int y) {
        int xset = find(parent, x);
        int yset = find(parent, y);
        parent[xset] = yset;
    }

    void kruskal() {
        Edge[] result = new Edge[V];
        int e = 0;
        int i = 0;
        for (i = 0; i < V; ++ i)
            result[i] = new Edge();

        Arrays.sort(edges);

        int[] parent = new int[V];
        Arrays.fill(parent, -1);

        i = 0;
        while (e < V - 1) {
            Edge next_edge = edges[i ++];
            int x = find(parent, next_edge.src);
            int y = find(parent, next_edge.dest);

            if (x != y) {
                result[e ++] = next_edge;
                union(parent, x, y);
            }
        }

        int finalWeight = 0;
        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);
            finalWeight = finalWeight + result[i].weight;
        }
        System.out.println("Total Weight of MST :: " + finalWeight);

    }
}
```

**Output:**

```
Enter the number of vertices: 6
Enter the number of edges: 8
Enter the source, destination, and weight of edge 1:
0 1 4
Enter the source, destination, and weight of edge 2:
0 2 4
Enter the source, destination, and weight of edge 3:
1 2 2
Enter the source, destination, and weight of edge 4:
2 3 3
Enter the source, destination, and weight of edge 5:
2 4 4
Enter the source, destination, and weight of edge 6:
2 5 2
Enter the source, destination, and weight of edge 7:
3 4 3
Enter the source, destination, and weight of edge 8:
4 5 3
Edges in the MST ::
1 - 2: 2
2 - 5: 2
2 - 3: 3
3 - 4: 3
0 - 1: 4
Total Weight of MST :: 14
```

## Analysis of Algorithm

**Time Complexity:**

The time complexity of Kruskal's algorithm is ***O(E log E)***, where E is the number of edges in the graph. This is because the algorithm sorts the edges in the graph by weight, which takes O(E log E) time using an efficient sorting algorithm such as **quick sort** or **merge sort**.

After the edges are sorted, the algorithm iterates through them in increasing order of weight and performs a union-find operation to determine whether adding the edge to the MST would create a cycle. The **union-find operation takes O(log V) time**, where V is the number of vertices in the graph.

Since Kruskal's algorithm performs the union-find operation at most E times, the total time complexity of the algorithm is **O(E log E + E log V)**, which can be simplified to **O(E log E)** since E >= V-1 in a connected graph.

**Space Complexity:**

The space complexity of the algorithm is O(V) to store the parent array in the union-find data structure.