# **DAA Lab 5: MST using Union Find**

### Meet Jain, 23BCP093

**Theory:**

Kruskal’s algorithm is a **greedy algorithm** used to find the **Minimum Spanning Tree (MST)** of a connected, weighted, and undirected graph. The MST is a subset of the edges that connects all vertices with the minimum possible total edge weight and without cycles.

**Steps in Kruskal’s Algorithm**

1. **Sort all edges** in ascending order of weight.
2. **Initialize a disjoint-set (Union-Find)** to keep track of connected components.
3. **Iterate through sorted edges:**
   * If adding an edge does not form a cycle, include it in the MST.
   * Otherwise, discard the edge.
4. **Continue until MST has (V - 1) edges**, where V is the number of vertices.

**Code:**

#include <iostream>

using namespace std;

#define MAX\_NODES 100

#define MAX\_EDGES 1000

struct Edge {

    int src, dest, weight;

};

class DisjointSet {

private:

    int parent[MAX\_NODES];

    int rank[MAX\_NODES];

public:

    DisjointSet(int n) {

        for (int i = 0; i < n; i++) {

            parent[i] = i;

            rank[i] = 0;

        }

    }

    int find(int node) {

        if (parent[node] != node)

            parent[node] = find(parent[node]);

        return parent[node];

    }

    void unionSets(int u, int v) {

        int rootU = find(u);

        int rootV = find(v);

        if (rootU != rootV) {

            if (rank[rootU] > rank[rootV])

                parent[rootV] = rootU;

            else if (rank[rootU] < rank[rootV])

                parent[rootU] = rootV;

            else {

                parent[rootV] = rootU;

                rank[rootU]++;

            }

        }

    }

};

void bubbleSort(Edge edges[], int E) {

    for (int i = 0; i < E - 1; i++) {

        for (int j = 0; j < E - i - 1; j++) {

            if (edges[j].weight > edges[j + 1].weight) {

                Edge temp = edges[j];

                edges[j] = edges[j + 1];

                edges[j + 1] = temp;

            }

        }

    }

}

void kruskalMST(Edge edges[], int V, int E) {

    bubbleSort(edges, E);

    DisjointSet ds(V);

    Edge result[MAX\_NODES];

    int edgeCount = 0;

    for (int i = 0; i < E && edgeCount < V - 1; i++) {

        Edge nextEdge = edges[i];

        int root1 = ds.find(nextEdge.src);

        int root2 = ds.find(nextEdge.dest);

        if (root1 != root2) {

            result[edgeCount++] = nextEdge;

            ds.unionSets(root1, root2);

        }

    }

    cout << "Edges in Minimum Spanning Tree:\n";

    for (int i = 0; i < edgeCount; i++)

        cout << result[i].src << " - " << result[i].dest << " : " << result[i].weight << endl;

}

int main() {

    int V = 4, E = 5;

    Edge edges[MAX\_EDGES] = {

        {0, 1, 10},

        {0, 2, 6},

        {0, 3, 5},

        {1, 3, 15},

        {2, 3, 4}

    };

    kruskalMST(edges, V, E);

    return 0;

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Time Complexity Analysis**

1. **Sorting the Edges:**
   * Sorting **E** edges takes **O(E²)** with **Bubble Sort**
   * If we use **Merge Sort or Quick Sort**, it takes **O(E log E)**.
2. **Union-Find Operations:**
   * Each edge requires find() and union().
   * Using a simple Union-Find without optimizations, each operation takes **O(V)** in the worst case.
   * Since we process **E** edges, this results in **O(EV)** in the worst case.

**Overall Time Complexity**

* **With Bubble Sort:** **O(E² + EV) = O(E²)**
* **With Efficient Sorting:** **O(E log E + EV) = O(E log E) (if V is small) or O(EV) (if V is large)**

Since **E is at most V²**, in the worst case, the complexity can go up to **O(V²)**.