Name	Surti Keyur
Roll No.	25MCE026
Branch	M.Tech (CSE)

Experiment 5

Aim: Implement Kruskal's algorithm to find MST using greedy approach

Overview

Kruskal's Algorithm is a **Greedy Algorithm** for finding the MST.

- It sorts all edges in non-decreasing order of weights.
- Iteratively, the shortest edge is selected, provided it does not form a cycle with the edges already included in MST.
- Cycle detection is efficiently handled using **Disjoint Set Union (DSU)**.
- The algorithm stops when the MST contains exactly V – 1 edges (where V is the number of vertices).

Source Code

```
#include <bits/stdc++.h>
using namespace std;

// Structure to represent an edge
struct Edge {
    int u, v, w;
    bool operator<(const Edge& other) const {
        return w < other.w; // sort by weight
    }
};

// Disjoint Set Union (Union-Find) with path compression
class DSU {
    vector<int> parent, rank;
public:
    DSU(int n) {
```

```
        parent.resize(n);
        rank.resize(n, 0);
        for (int i = 0; i < n; i++) parent[i] = i;
    }

    int find(int x) {
        if (parent[x] != x)
            parent[x] = find(parent[x]); // path compression
        return parent[x];
    }

    bool unite(int x, int y) {
        int xr = find(x), yr = find(y);
        if (xr == yr) return false; // cycle
        if (rank[xr] < rank[yr]) swap(xr, yr);
        parent[yr] = xr;
        if (rank[xr] == rank[yr]) rank[xr]++;
        return true;
    }
};

// Kruskal's Algorithm
void kruskalMST(int V, vector<Edge>& edges) {
    sort(edges.begin(), edges.end()); // step 1: sort by weight

    DSU dsu(V);
    vector<Edge> mst;
    int totalWeight = 0;

    for (auto& e : edges) {
        if (dsu.unite(e.u, e.v)) { // if no cycle
            mst.push_back(e);
            totalWeight += e.w;
        }
    }

    // Print result
    cout << "Edges in MST:\n";
    for (auto& e : mst) {
        cout << e.u << " -- " << e.v << " == " << e.w << "\n";
```

```
    }
    cout << "Total Weight of MST = " << totalWeight << endl;
}

// Driver Code
int main() {
    int V, E;
    cout << "Enter number of vertices and edges: ";
    cin >> V >> E;

    vector<Edge> edges(E);
    cout << "Enter edges (u v w):\n";
    for (int i = 0; i < E; i++) {
        cin >> edges[i].u >> edges[i].v >> edges[i].w;
    }

    kruskalMST(V, edges);
    return 0;
}
```

Output

```
PS E:\DSA\Experiments\EXP_5\output> cd 'e:\DSA\Experiments\EXP_5\output'
PS E:\DSA\Experiments\EXP_5\output> & .\'kruskal_mst.exe'
Enter number of vertices and edges: 4 5
Enter edges (u v w):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in MST:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total Weight of MST = 19
PS E:\DSA\Experiments\EXP_5\output>
```

Time Complexity

- Sorting edges: O(E log E)
- Union-Find operations: Nearly O(E) (amortized)
- Total Complexity: O(E log E)

Space Complexity

- Storing edges: O(E)
- DSU parent and rank arrays: O(V)
- Total Space: O(V + E)

Conclusion

- Kruskal's Algorithm successfully finds the **Minimum Spanning Tree (MST)** using the **Greedy Approach**.
- The **Disjoint Set Union (DSU)** efficiently avoids cycles during edge selection.
- The MST guarantees the **minimum total weight** required to connect all vertices of the graph.
- Time complexity of O(E log E) makes it suitable for **sparse graphs** (where E is much less than V^2).