<u>Assignment 6</u>

DESIGN AND ANALYSIS OF ALGORITMS LAB

By Smit Sutariya

Roll no. 21BCP142

Div : 3, G : 5

Submitted To

COMPUTER ENGINEERING

School of Technology, Pandit Deendayal EnergyUniversity

January – 2023

AIM:

Implement Greedy Algorithm: Kruskal using union-find data structure.

Kruskal's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a connected, undirected graph. The MST of a graph is the subset of edges that connect all the vertices of the graph with the minimum possible total edge weight.

The algorithm starts by sorting all the edges in the graph in increasing order of their weight. It then processes the edges one by one, adding each edge to the MST if it does not create a cycle with the edges already included in the MST. To check for cycles, the algorithm uses the union-find data structure, which keeps track of the subsets of vertices that have been connected by edges.

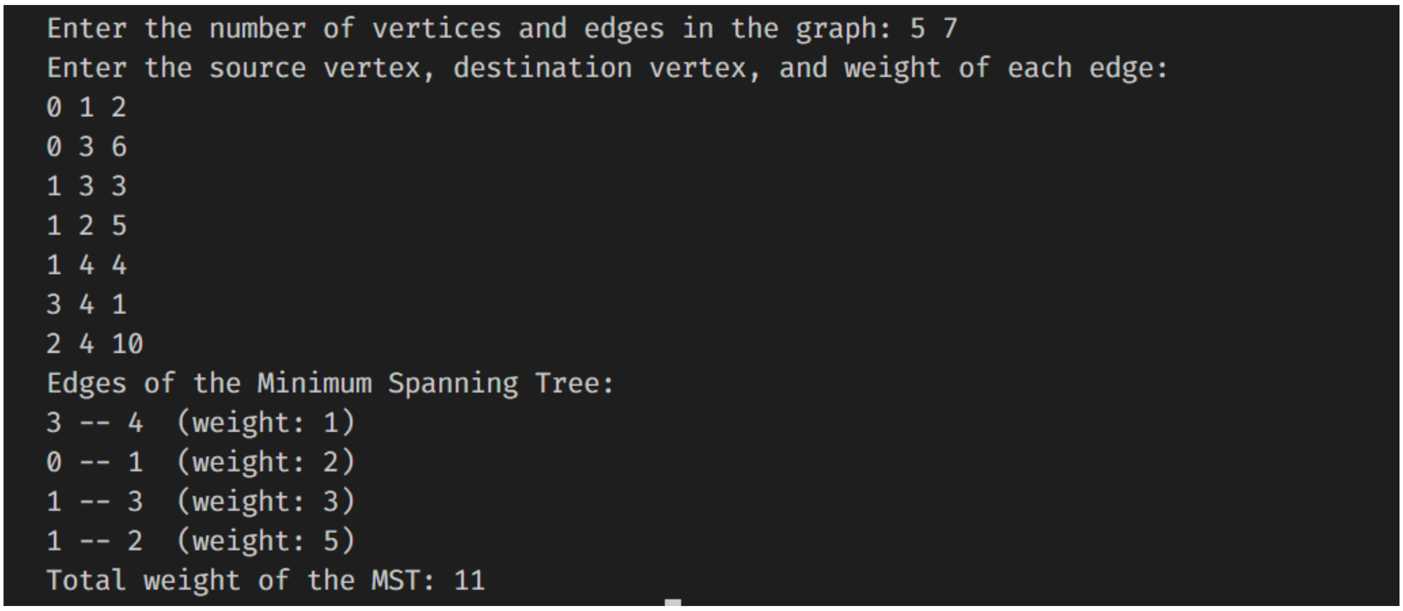
The algorithm terminates when it has added V-1 edges to the MST, where V is the number of vertices in the graph. This is because a connected graph with V vertices always has exactly V-1 edges in its MST.

CODE:

Smit Sutariya **21BCP142**

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

struct Edge {
    int src, dest, weight;
};

struct Subset {
    int parent, rank;
};

bool compareEdges(Edge a, Edge b) {
    return a.weight < b.weight;
}

int find(Subset subsets[], int i) {
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);
    return subsets[i].parent;
}

void unionSubsets(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;
    else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}

vector<Edge> kruskalMST(int V, vector<Edge> edges) {
    vector<Edge> result;
    Subset subsets[V];
    int e = 0;

    sort(edges.begin(), edges.end(), compareEdges);

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

    for (int i = 0; i < edges.size(); i++) {
        int x = find(subsets, edges[i].src);
        int y = find(subsets, edges[i].dest);

        if (x != y) {
            result.push_back(edges[i]);
            unionSubsets(subsets, x, y);
            e++;
        }

        if (e == V - 1)
            break;
    }

    return result;
}

void printMST(vector<Edge> edges) {
    int total_weight = 0;
    cout << "Edges of the Minimum Spanning Tree:" << endl;
    for (int i = 0; i < edges.size(); i++) {
        cout << edges[i].src << " -- " << edges[i].dest << "  (weight: " <<
edges[i].weight << ")" << endl;
        total_weight += edges[i].weight;
    }
    cout << "Total weight of the MST: " << total_weight << endl;
}

int main() {
    int V, E;
    cout << "Enter the number of vertices and edges in the graph: ";
    cin >> V >> E;
    vector<Edge> edges;
    cout << "Enter the source vertex, destination vertex, and weight of each edge:"
<< endl;
    for (int i = 0; i < E; i++) {
        Edge e;
        cin >> e.src >> e.dest >> e.weight;
        edges.push_back(e);
    }
    vector<Edge> mst = kruskalMST(V, edges);
    printMST(mst);

    return 0;
}
```

OUTPUT:



```
Enter the number of vertices and edges in the graph: 5 7
Enter the source vertex, destination vertex, and weight of each edge:
0 1 2
0 3 6
1 3 3
1 2 5
1 4 4
3 4 1
2 4 10
Edges of the Minimum Spanning Tree:
3 -- 4  (weight: 1)
0 -- 1  (weight: 2)
1 -- 3  (weight: 3)
1 -- 2  (weight: 5)
Total weight of the MST: 11
```

ANALYSIS:

The algorithm uses the union-find data structure to check for cycles. Kruskal's algorithm has a time complexity of **O(E log E)** and a space complexity of **O(V + E),** where E is the number of edges in the graph and V is the number of vertices. It is efficient for sparse graphs but may not be the best choice for dense graphs or graphs with large edge weights.