Lab Report-08

(Kruskal's Algorithm)

CSE-2212 (Design and Analysis of Algorithms Lab)

Submitted By:

Name: Eyasir Ahamed
Exam Roll: 413
Class Roll: 15
Registration No: 202004017

Submitted To:

Sharad Hasan
Ex. Lecturer
Dept. of CSE
Sheikh Hasina University, Netrokona

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SHEIKH HASINA UNIVERSITY
NETROKONA, BANGLADESH

#8_ Kruskal's Algorithm

Problem Definition

Given a connected, undirected graph with weighted edges, the problem is to find a minimum spanning tree (MST), which is a subset of the edges that connects all the vertices together without any cycles and with the minimum possible total edge weight.

Formal Statement of Algorithm (Kruskal's Algorithm):

- Initialize an empty list of edges to store all the edges of the graph.
- Traverse through each vertex of the graph.
- For each vertex, traverse through its adjacency list to get all its adjacent vertices along with the edge weights.
- Add each edge (vertex pair with its weight) to the list of edges.
- Sort the list of edges in non-decreasing order of their weights.
- Initialize a Disjoint Set data structure with the number of vertices in the graph.
- Initialize the total weight of the MST to 0.
- Iterate through each edge in the sorted list:
    - Check if adding the current edge to the MST forms a cycle or not by checking if the endpoints of the edge belong to the same connected component in the Disjoint Set.

    - If adding the edge does not form a cycle, union the endpoints in the Disjoint Set and add the weight of the edge to the total weight of the MST.
- After considering all edges, the total weight of the MST is obtained.

Complexity Analysis:

- Constructing the list of edges takes O(E) time, where E is the number of edges in the graph.
- Sorting the list of edges takes O(E log E) time.
- Initializing the Disjoint Set data structure takes O(V) time, where V is the number of vertices in the graph.
- Iterating through all edges and performing union-find operations takes O(E log V) time.
- Overall, the time complexity of the algorithm is O(E log E) or O(E log V), depending on the implementation of the disjoint set data structure.

Actual Code and Output

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

class DisjointSet {
    vector<int> rank, parent, size;
public:
    DisjointSet(int n) {
        rank.resize(n + 1, 0);
        parent.resize(n + 1);
        size.resize(n + 1);
        for (int i = 0; i <= n; i++) {
            parent[i] = i;
            size[i] = 1;
        }
    }

    int findUPar(int node) {
        if (node == parent[node])
            return node;
        return parent[node] = findUPar(parent[node]);
    }

    void unionByRank(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (rank[ulp_u] < rank[ulp_v]) {
            parent[ulp_u] = ulp_v;
        }
        else if (rank[ulp_v] < rank[ulp_u]) {
            parent[ulp_v] = ulp_u;
        }
        else {
            parent[ulp_v] = ulp_u;
            rank[ulp_u]++;
        }
    }

    void unionBySize(int u, int v) {
        int ulp_u = findUPar(u);
        int ulp_v = findUPar(v);
        if (ulp_u == ulp_v) return;
        if (size[ulp_u] < size[ulp_v]) {
            parent[ulp_u] = ulp_v;
            size[ulp_v] += size[ulp_u];
        }
        else {
            parent[ulp_v] = ulp_u;
            size[ulp_u] += size[ulp_v];
        }
    }
};
```

```cpp
class Solution
{
public:
    //Function to find sum of weights of edges of the Minimum Spanning Tree.
    int spanningTree(int V, vector<vector<int>> adj[])
    {
        vector<pair<int, pair<int, int>>> edges;
        for (int i = 0; i < V; i++) {
            for (auto it : adj[i]) {
                int adjNode = it[0];
                int wt = it[1];
                int node = i;

                edges.push_back({wt, {node, adjNode}});
            }
        }
        DisjointSet ds(V);
        sort(edges.begin(), edges.end());
        int mstWt = 0;
        for (auto it : edges) {
            int wt = it.first;
            int u = it.second.first;
            int v = it.second.second;

            if (ds.findUPar(u) != ds.findUPar(v)) {
                mstWt += wt;
                ds.unionBySize(u, v);
            }
        }
        return mstWt;
    }
};

int main() {

    int V = 5;
    vector<vector<int>> edges = {{0, 1, 2}, {0, 2, 1}, {1, 2, 1}, {2, 3, 2}, {3, 4, 1}, {4, 2, 2}};
    vector<vector<int>> adj[V];
    for (auto it : edges) {
        vector<int> tmp(2);
        tmp[0] = it[1];
        tmp[1] = it[2];
        adj[it[0]].push_back(tmp);

        tmp[0] = it[0];
        tmp[1] = it[2];
        adj[it[1]].push_back(tmp);
    }

    Solution obj;
    int mstWt = obj.spanningTree(V, adj);
    cout << "The sum of all the edge weights: " << mstWt << endl;
    return 0;
}
```

```
The sum of all the edge weights: 5
[Finished in 1.3s]
```