**Minimum Spanning Tree**

**Medium**Accuracy: 49.39% Submissions: 34970 Points: 4

Given a weighted, undirected and connected graph of **V** vertices and **E** edges. The task is to find the sum of weights of the edges of the Minimum Spanning Tree.

Diagram

Description automatically generated

**Example 1:**

**Input:**

Diagram

Description automatically generated

**Output:**

4

**Explanation**:

A picture containing diagram

Description automatically generated

The Spanning Tree resulting in a weight

of 4 is shown above.

**Example 2:**

**Input:**

A picture containing chart

Description automatically generated

**Output:**

5

**Explanation**:

Only one Spanning Tree is possible

which has a weight of 5.

**Your task:**  
Since this is a functional problem you don't have to worry about input, you just have to complete the function  **spanningTree()** which takes number of vertices Vandan adjacency matrix adj as input parameters and returns an integer denoting the sum of weights of the edges of the Minimum Spanning Tree. Here adj[i] contains a list of lists containing two integers where the first integer a[i][0] denotes that there is an edge between i and a[i][0] and second integer a[i][1] denotes that the distance between edge i and a[i][0] is a[i][1].

**Expected Time Complexity:**O(ElogV).  
**Expected Auxiliary Space:**O(V2).

**Constraints:**  
2 ≤ V ≤ 1000  
V-1 ≤ E ≤ (V\*(V-1))/2  
1 ≤ w ≤ 1000  
Graph is connected and doesn't contain self loops & multiple edges.

class Solution {

    public:

    //Function to find sum of weights of edges of the Minimum Spanning Tree

    int \*parent, \*rank;

    void disjointSet(int V) {

        parent=new int[V];

        rank=new int[V];

        for (int i=0; i<V; i++) {

            parent[i]=i;

            rank[i]=0;

        }

    }

    int find(int x) {

        if (x!=parent[x]) parent[x]=find(parent[x]);

        return parent[x];

    }

    void unionSet(int &x, int &y) {

        int rootx=find(x);

        int rooty=find(y);

        if (rootx==rooty) return;

        if (rank[rootx]>rank[rooty]) {

            parent[rooty]=rootx;

        }

        else {

            parent[rootx]=rooty;

            if (rank[rootx]==rank[rooty]) rank[rooty]++;

        }

    }

    void dfs(int i, vector<bool> &visited, vector<vector<int>> adj[], map< int, vector<pair<int, int>> > &m) {

        visited[i]=true;

        for (vector<int> v : adj[i]) {

            if (!visited[v[0]]) {

                dfs(v[0], visited, adj, m);

            }

            else m[v[1]].push\_back({i, v[0]});

        }

    }

    int spanningTree(int V, vector<vector<int>> adj[]) {

        // code here

        disjointSet(V);

        int count=0;

        int minSpanningTreeSum=0;

        map< int, vector<pair<int, int>> > m;

        vector<bool> visited(V, false);

        dfs(0, visited, adj, m);

        for (auto it : m) {

            vector<pair<int, int>> v=it.second;

            for (auto x : v) {

                if (find(x.first)!=find(x.second)) {

                    count++;

                    minSpanningTreeSum+=it.first;

                    unionSet(x.first, x.second);

                    if (count==V-1) return minSpanningTreeSum;

                }

            }

        }

        return minSpanningTreeSum;

    }

};