Assignment Solutions : Graph - 9 Spanning Tree

Q1 Min Cost to Connect All Points **Leetcode:-1584.**

Solution :

Code :

```
class Solution {
public:
    int minCostConnectPoints(vector<vector<int>>& points) {
        int n = points.size();
        vector<vector<int>> edges;

        // Build all possible edges along with their weights
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                int cost = abs(points[i][0] - points[j][0])
+ abs(points[i][1] - points[j][1]);
                edges.push_back({cost, i, j});
            }
        }

        // Sort edges based on their weights
        sort(edges.begin(), edges.end());

        vector<int> parent(n, -1);
        int minCost = 0, count = 0;

        for (const auto& edge : edges) {
            int cost = edge[0];
            int u = edge[1];
            int v = edge[2];
            if (unionFind(parent, u) != unionFind(parent,
v)) {
                unionMerge(parent, u, v);
                minCost += cost;
                count++;
            }

            if (count == n - 1) {
                break;  // All points connected
            }
        }

        return minCost;
    }

    int unionFind(vector<int>& parent, int x) {
        return parent[x] == -1 ? x : (parent[x] =
unionFind(parent, parent[x]));
    }

    void unionMerge(vector<int>& parent, int x, int y) {
        int rootX = unionFind(parent, x);
        int rootY = unionFind(parent, y);
        if (rootX != rootY) {
            parent[rootX] = rootY;
        }
    }
};
```

Q2 Find Critical and Pseudo-Critical Edges in Minimum Spanning Tree   Leetcode:-1489.

Solution :

Code :

```
class Solution {
public:
    vector<int>parent;
    void disjoint(int size){
        parent.resize(size+1);
        for(int i=0;i<=size;i++)
        parent[i]=(i);
    }
    int find(int u){
        if(parent[u]==u)return u;
        return parent[u] = find(parent[u]);
    }
    void merge(int u,int v){
        int ua = find(u);
        int ub = find(v);
        parent[ua] = ub;
    }
    int help1(vector<vector<int>>& e,int j,int n){
        disjoint(n+1);
        vector<pair<int,pair<int,int >>>v;
        for(int i=0;i<e.size();i++){
            if(i==j)continue;
            v.push_back({e[i][2],{e[i][0],e[i][1]}});
        }
        sort(v.begin(),v.end());
        int mst_weight = 0;
        int edges = 0;
        for(int i=0;i<v.size();i++){
            auto x = v[i];
            int u = find(x.second.first);
            int v = find(x.second.second);
            if(u!=v){
                edges++;
                mst_weight += x.first;
                merge(u,v);
            }
        }
        if(edges!=n-1)return INT_MAX;
        return mst_weight;
    }
    int help2(vector<vector<int>>& e,int j,int n){
        disjoint(n+1);
        int mst_weight = e[j][2];
        merge(e[j][1],e[j][0]);
        vector<pair<int,pair<int,int >>>v;
        for(int i=0;i<e.size();i++){
            if(i==j)continue;
            v.push_back({e[i][2],{e[i][0],e[i][1]}});
        }
        sort(v.begin(),v.end());
        for(int i=0;i<v.size();i++){
            auto x = v[i];
            int u = find(x.second.first);
            int v = find(x.second.second);
            if(u!=v){
                mst_weight += x.first;
                merge(u,v);
            }
        }
        return mst_weight;
    }
    vector<vector<int>>
findCriticalAndPseudoCriticalEdges(int n,
vector<vector<int>>& e) {
        disjoint(n+1);
        int mst_weight = help1(e,-1,n);
        vector<vector<int>>ans;
        vector<int>v1,v2;
        for(int i=0;i<e.size();i++){
            int new_weight1 = help1(e,i,n);
            int new_weight2 = help2(e,i,n);
            if(new_weight1 > mst_weight){
                v1.push_back(i);
            }
           else if(new_weight2 == mst_weight){
                v2.push_back(i);
            }
        }
        ans.push_back(v1);
        ans.push_back(v2);
        return ans;
    }
};
```