**Minimum Cost to Connect All Points**

**Medium**

You are given an array points representing integer coordinates of some points on a 2D-plane, where points[i] = [xi, yi].

The cost of connecting two points [xi, yi] and [xj, yj] is the **manhattan distance** between them: |xi - xj| + |yi - yj|, where |val| denotes the absolute value of val.

Return *the minimum cost to make all points connected.* All points are connected if there is **exactly one** simple path between any two points.

**Example 1:**

Shape

Description automatically generated with medium confidence

**Input:** points = [[0,0],[2,2],[3,10],[5,2],[7,0]]

**Output:** 20

**Explanation:**

A picture containing light

Description automatically generated

We can connect the points as shown above to get the minimum cost of 20.

Notice that there is a unique path between every pair of points.

**Example 2:**

**Input:** points = [[3,12],[-2,5],[-4,1]]

**Output:** 18

**Example 3:**

**Input:** points = [[0,0],[1,1],[1,0],[-1,1]]

**Output:** 4

**Example 4:**

**Input:** points = [[-1000000,-1000000],[1000000,1000000]]

**Output:** 4000000

**Example 5:**

**Input:** points = [[0,0]]

**Output:** 0

**Constraints:**

* 1 <= points.length <= 1000
* -106 <= xi, yi <= 106
* All pairs (xi, yi) are distinct.

class Solution {

public:

    int find(vector<int> &parent, int &x) {

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

        return parent[x];

    }

    void unionSet(vector<int> &parent, vector<int> &rank, int &x, int &y) {

        int rootx=find(parent, x);

        int rooty=find(parent, 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< vector<pair<int, int>> > &graph, vector<bool> &visited, map<int, vector<pair<int, int>> > &mp) {

        visited[i]=true;

        for (auto x : graph[i]) {

            if (!visited[x.first]) dfs(x.first, graph, visited, mp);

            else mp[x.second].push\_back({i, x.first});

        }

    }

    int primsAlgo(vector< vector<pair<int, int>> > &graph, vector<bool> &visited, vector<int> &key, int src) {

        int minCost=0;

        key[src]=0;

        vector<int> parent(key.size(), INT\_MAX);

        priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> > pq;

        pq.push({0, src});

        while (!pq.empty()) {

            pair<int, int> p=pq.top();

            pq.pop();

            if (parent[p.second]!=INT\_MAX and !visited[p.second]) {

                minCost+=p.first;

            }

            visited[p.second]=true;

            for (auto x : graph[p.second]) {

                if (!visited[x.first] and key[x.first]>x.second) {

                    parent[x.first]=p.second;

                    key[x.first]=x.second;

                    pq.push({x.second, x.first});

                }

            }

        }

        return minCost;

    }

    int minCostConnectPoints(vector<vector<int>>& points) {

        vector< vector<pair<int,int>> > graph(points.size());

        map<pair<int, int>, int> m;

        int key=0;

        for (int i=0; i<points.size(); i++) {

            m[{points[i][0], points[i][1]}]=key++;

        }

        for (int i=0; i<points.size()-1; i++) {

            for (int j=i+1; j<points.size(); j++) {

                int dis=abs(points[i][0]-points[j][0])+abs(points[i][1]-points[j][1]);

                graph[m[{points[i][0], points[i][1]}]].push\_back({m[{points[j][0], points[j][1]}], dis});

                graph[m[{points[j][0], points[j][1]}]].push\_back({m[{points[i][0], points[i][1]}], dis});

            }

        }

        /\*vector<int> parent(points.size()), rank(points.size(), 0);

        for (int i=0; i<points.size(); i++) {

            parent[i]=i;

        }

        vector<bool> visited(points.size(), false);

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

        dfs(0, graph, visited, mp);

        int minCost=0, count=0;

        for (auto it : mp) {

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

            for (auto x : p) {

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

                    count++;

                    minCost+=it.first;

                    unionSet(parent, rank, x.first, x.second);

                    if (count==points.size()-1) return minCost;

                }

            }

        }

        return minCost;\*/

        if (points.size()==1) return 0;

        int val=INT\_MAX, src=-1;

        vector<int> keyy(points.size(), INT\_MAX);

        vector<bool> visited(points.size(), false);

        for (int i=0; i<graph.size(); i++) {

            for (auto x : graph[i]) {

                if (x.second<val) {

                    val=x.second;

                    src=i;

                }

            }

        }

        return primsAlgo(graph, visited, keyy, src);

    }

};