

切换主题:

默认主题

▼

标签

- 并查集

难度

- 中等

题目地址（1319. 连通网络的操作次数）



<https://leetcode-cn.com/problems/number-of-operations-to-make-network-connected/>

题目描述

用以太网线缆将 n 台计算机连接成一个网络，计算机的编号从 0 到 $n-1$ 。线缆用 `connections` 表示，其中 `connections[i] = [a, b]` 连接了计算机 a 和 b 。

网络中的任何一台计算机都可以通过网络直接或者间接访问同一个网络中其他任意一台计算机。

给你这个计算机网络的初始布线 `connections`，你可以拔开任意两台直连计算机之间的线缆，并用它连接一对未直连的计算机。请你计算并返回使所有计算机都连通所需的最少操作次数。

示例 1:

输入: $n = 4, connections = [[0,1],[0,2],[1,2]]$

输出: 1

解释: 拔下计算机 1 和 2 之间的线缆，并将它插到计算机 1 和 3 上。

示例 2:

输入: $n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]]$

输出: 2

示例 3:

输入: $n = 6, connections = [[0,1],[0,2],[0,3],[1,2]]$

输出: -1

解释: 线缆数量不足。

示例 4:

输入: $n = 5, connections = [[0,1],[0,2],[3,4],[2,3]]$

输出: 0


提示:

```

1 <= n <= 10^5
1 <= connections.length <= min(n*(n-1)/2, 10^5)
connections[i].length == 2
0 <= connections[i][0], connections[i][1] < n
connections[i][0] != connections[i][1]
没有重复的连接。
两台计算机不会通过多条线缆连接。

```

思路

这题稍微难一点的地方在于问题抽象，不管怎么样，网络总会有部分节点连接形成子网，只要我们找到网络中的子网数目，使得整个网络连通的操作次数其实就是将所有子网  的次数。求子网数量其实就是求图中联通分量的数量，求联通分量可以用 DFS 或者并查集，这里提供并查集解法。

代码

代码支持：JS, Python3, Java, CPP

JS Code:

```

/**
 * @param {number} n
 * @param {number[][]} connections
 * @return {number}
 */
var makeConnected = function(n, connections) {
    // 连接 n 台电脑至少需要 n - 1 根线缆
    if (connections.length < n - 1) {
        return -1;
    }
    // 计算联通分量，最小操作次数就是将联通分量链接的次数
    let father = Array.from({ length: n }, (v, i) => i);
    let count = n;
    for (connection of connections) {
        union(...connection);
    }

    return count - 1;

    function find(v) {
        if (father[v] !== v) {
            father[v] = find(father[v]);
        }
        return father[v];
    }
}

```

```
function union(x, y) {
    if (find(x) !== find(y)) {
        count--;
        father[find(x)] = find(y);
        // 联通分量数减一
    }
}
};
```

Python Code:

```
class Solution:
    def makeConnected(self, n: int, connections: List[List[int]]) -> int:
        root = [i for i in range(n)]

        def find(p):
            while p != root[p]:
                root[p] = root[root[p]]
                p = root[p]

            return p

        def union(p, q):
            root[find(p)] = find(q)

        have = 0

        for connec in connections:
            a, b = connec
            if find(a) != find(b):
                union(a, b)
            else:
                have += 1

        diff_root = set()
        for i in range(n):
            diff_root.add(find(i))

        return len(diff_root) - 1 if have >= len(diff_root) - 1 else -1
```

Java Code:

```
class Solution {
    public int makeConnected(int n, int[][] connections) {
        if (n - 1 > connections.length) {
            return -1;
        }
        UnionFind unionFind = new UnionFind(n);
        for (int[] connection : connections) {
```

```

        unionFind.union(connection[0], connection[1]);
    }
    return unionFind.cnt - 1;
}

class UnionFind {
public:
    int cnt;
    int[] parents;

    public UnionFind(int n) {
        this.cnt = n;
        this.parents = new int[n];
        for (int i = 0; i < n; i++) {
            parents[i] = i;
        }
    }

    public void union(int x, int y) {
        int x_root = find(x);
        int y_root = find(y);
        if (x_root != y_root) {
            cnt--;
        }
        parents[x_root] = y_root;
    }

    public int find(int x) {
        if (x == parents[x]) {
            return x;
        }
        parents[x] = find(parents[x]);
        return parents[x];
    }
}

```

CPP Code:

```

struct UF {
private:
    vector<int> parent;
    int count;
public:

    UF(int n)
    {
        for(int i=0; i< n; i++)
            parent.push_back(i);
        count = n;
    }
}

```

```

void union(int i, int j)
{
    int parentI = find(i);
    int parentJ = find(j);
    if(parentI != parentJ)
    {
        parent[parentI] = parentJ;
        count--;
    }
}

int find(int i)
{
    while(parent[i] != i)
    {
        parent[i] = parent[parent[i]];
        i = parent[i];
    }
    return i;
}

int size()
{
    return count;
}

};

class Solution {
public:
    int makeConnected(int n, vector<vector<int>>& connections) {

        // generate tree. and tree
        // connections number is n-1;
        if(connections.size()<n-1) // is impossible.
            return -1;

        UF uf(n);
        for(int i=0; i< connections.size(); i++)
        {
            uf.union(connections[i][0], connections[i][1]);
        }

        return uf.size()-1;
    }
};

```

复杂度分析

令 v 图的点的个数，也就是计算机的个数。

- 时间复杂度：由于仅仅使用了路径压缩，因此合并与查找时间复杂度为 $O(\log x)$ 和 $O(\log y)$ ， x 和 y 分别为合并与查找的次數。

- 空间复杂度: $O(v)$ 。

上一页

下一页



© 2020 lucifer. 保留所有权利