并查集

5929. 处理含限制条件的好友请求

给你一个整数 `n` ，表示网络上的用户数目。每个用户按从 `0` 到 `n - 1` 进行编号。

给你一个下标从 **0** 开始的二维整数数组 `restrictions` ，其中 `restrictions[i] = [$x_i$, $y_i$]` 意味着用户 `$x_i$` 和用户 `$y_i$` **不能** 成为 **朋友** ，不管是 **直接** 还是通过其他用户 **间接** 。

最初，用户里没有人是其他用户的朋友。给你一个下标从 **0** 开始的二维整数数组 `requests` 表示好友请求的列表，其中 `requests[j] = [$u_j$, $v_j$]` 是用户 `$u_j$` 和用户 `$v_j$` 之间的一条好友请求。

如果 `$u_j$` 和 `$v_j$` 可以成为 **朋友** ，那么好友请求将会 **成功** 。每个好友请求都会按列表中给出的顺序进行处理（即，`requests[j]` 会在 `requests[j + 1]` 前）。一旦请求成功，那么对所有未来的好友请求而言，`$u_j$` 和 `$v_j$` 将会 **成为直接朋友** 。

返回一个 **布尔数组** `result` ，其中元素遵循此规则：如果第 `j` 个好友请求 **成功** ，那么 `result[j]` 就是 `true` ；否则，为 `false` 。

注意：如果 `$u_j$` 和 `$v_j$` 已经是直接朋友，那么他们之间的请求将仍然 **成功** 。

示例 1：

```
输入：n = 3, restrictions = [[0,1]], requests = [[0,2], [2,1]]
输出：[true,false]
解释：
请求 0 ：用户 0 和 用户 2 可以成为朋友，所以他们成为直接朋友。
请求 1 ：用户 2 和 用户 1 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (1--2--0) 。
```

题目分析：

https://leetcode-cn.com/problems/process-restricted-friend-requests/solution/bing-cha-ji-ha-xi-biao-zui-xiang-xi-zui-1ym8q/

1. 此题的数据结构为图
2. 一个联通分量中有一个大哥，该大哥维护着该联通分量中的朋友集合和敌人集合
3. 当两个人要建立关系时，就要看彼此大哥之间是否相容

代码：

```
class Solution {
    // 每个人都有一个“大哥”
    vector<int> parent;
    // 我们可以通过`root()`函数递归找到这个圈子里最大的大哥，也就是圈子的头头。
    // 头头没有大哥（parent[x] == -1），或者他的大哥就是他自己（parent[x] == x）
    int root(int x) { return (parent[x] == -1 || parent[x] == x) ? x : root(parent[x]); };
public:
    vector<bool> friendRequests(int n, vector<vector<int>>& restrictions, vector<vector<int>>& requests) {
        // 维护自己朋友圈的朋友名单`friends`，
        vector<unordered_set<int>> friends(n);
        for (int i = 0; i < n; i++) {
            friends[i].insert(i);
        }
        // 维护自己朋友圈的仇人名单`restricts`，
        vector<unordered_set<int>> restricts(n);
        for (auto res : restrictions) {
            restricts[res[0]].insert(res[1]);
```

```
            restricts[res[1]].insert(res[0]);
        }
        // 开始的时候大家都没有大哥（-1）
        parent = vector<int>(n, -1);
        // 我们假定所有交友请求都能成功
        vector<bool> result(requests.size(), true);
        for (int i = 0; i < requests.size(); i++) {
            // 对于每个交友请求`requests[i]`，先找到两个人的大哥x和y。
            int x = root(requests[i][0]), y = root(requests[i][1]);
            // 我判断大哥的方式很粗暴，谁的数字大谁就是大哥。
            // 保证`x`比`y`大。
            if (x < y) swap(x, y);
            [&]{
                // 头头`x`首先查看`y`提交的朋友名单`friends[y]`
                for (auto people : friends[y]) {
                    // 如果有一个`people`出现在`x`维护的仇人名单
`restricts[x]`中
                    if (restricts[x].count(people) != 0) {
                        // 交朋友就失败了
                        result[i] = false;
                        // 立刻滚蛋（相当于goto匿名函数末尾）
                        return;
                    }
                }
                // `x`把`y`上交的仇人名单`restricts[y]`添加到自己的仇人名单里。
                restricts[x].insert(restricts[y].begin(),
restricts[y].end());
                // `x`把`y`上交的朋友名单`friends[y]`添加到自己的朋友名单里。
                friends[x].insert(friends[y].begin(), friends[y].end());
                // `y`拜`x`为大哥，这样，`y`的小弟们都会跟着认`x`为头头。
                parent[y] = x;
            }();
        }
        return result;
    }
};
```

内化代码：

```
```
class Solution {
private:
    vector<int>parent;//起初的大哥都位无
    int root(int son){
        //递归求大哥
        if(parent[son]==-1||parent[son]==son)return son;
        else return root(parent[son]);
    }
```
```

```
public:
    vector<bool> friendRequests(int n, vector<vector<int>>&
restrictions, vector<vector<int>>& requests) {
        //并查集
        //大哥那里维护着每个朋友圈的朋友信息和敌人信息
        //谁的序号大，谁就是大哥
        vector<unordered_set<int>>friends(n);
         for (int i = 0; i < n; i++) {
            friends[i].insert(i);
        }
        vector<unordered_set<int>>enemy(n);
        for(auto res:restrictions){
            enemy[res[0]].insert(res[1]);
            enemy[res[1]].insert(res[0]);
        }


        parent.resize(n,-1);//起初的大哥都位无
        int size=requests.size();
        vector<bool>result(size,false);
        for(int i=0;i<size;i++){

            int x=requests[i][0],y=requests[i][1];
            if(enemy[x].count(y))continue;
            int p1=root(x),p2=root(y);
            if(p1<p2)swap(p1,p2);//求大哥中的大哥
            //检查p1的仇人谱系中是否有p2的朋友
            bool flag=true;
            for(auto s:enemy[p2]){
                if(friends[p1].count(s)){
                    flag=false;
                    break;
                }
            }
            if(!flag)continue;
            for(auto s:enemy[p1]){
                if(friends[p2].count(s)){
                    flag=false;
                    break;
                }
            }
            if(!flag)continue;
            //否则合并
            //更新大大哥p1的敌人列表和朋友列表

            for(auto s:enemy[p2])enemy[p1].insert(s);
            for(auto f:friends[p2])friends[p1].insert(f);
            friends[p1].insert(x);
            friends[p1].insert(y);
```

```
                //更新p2的大哥
                parent[p2]=p1;
                result[i]=true;
            }
            return result;
        }
};
```