题目地址(997. 找到小镇的法官)

https://leetcode-cn.com/problems/find-the-town-judge/

题目描述

在一个小镇里，按从 1 到 n 为 n 个人进行编号。传言称，这些人中有一个是小镇上的秘密法官。

如果小镇的法官真的存在，那么：

小镇的法官不相信任何人。
每个人（除了小镇法官外）都信任小镇的法官。
只有一个人同时满足条件 1 和条件 2 。

给定数组 trust，该数组由信任对 trust[i] = [a, b] 组成，表示编号为 a 的人信任编号为 b 的人。

如果小镇存在秘密法官并且可以确定他的身份，请返回该法官的编号。否则，返回 -1。

示例 1:

输入: n = 2, trust = [[1,2]]
输出: 2

示例 2:

输入: n = 3, trust = [[1,3],[2,3]]
输出: 3

示例 3:

输入: n = 3, trust = [[1,3],[2,3],[3,1]]
输出: -1

示例 4:

输入: n = 3, trust = [[1,2],[2,3]]
输出: -1

示例 5:

输入: n = 4, trust = [[1,3],[1,4],[2,3],[2,4],[4,3]]
输出: 3

提示：

```
1 <= n <= 1000
0 <= trust.length <= 104
trust[i].length == 2
trust[i] 互不相同
trust[i][0] != trust[i][1]
1 <= trust[i][0], trust[i][1] <= n
```

前置知识

- 图

标签

- 图

难度

- 简单

入选理由

- 最简单的图题目，适合用来开头

公司

- 暂无

思路

我们可以将小镇中的人们之间的信任关系抽象为图的边，那么图中的点自然就是小镇中的人。这样问题就转化为**求图中入度（或出度）为 n - 1 并且出度（或入度）为 0**的点。

究竟是入度还是出度取决于你对边的定义。比如我定义：a 信任 b 表示图中有一条从顶点 a 到顶点 b 的有向边，那么此时我们要找的是**入度为 n - 1 并且出度为 0**的点。反之，我定义：a 信任 b 表示图中有一条从顶点 b 到顶点 a 的有向边，那么此时我们要找的是**出度为 n - 1，入度为 0**的点。

这里我们不妨使用第一种定义方式，即找图中入度为 n - 1 ，出度为 0 的点。

算法：

- 初始化长度为 n 的两个数组 in_degree 和 out_degree，分别表示入度和出度信息，比如 in_degree[i] 表示顶点 i 的入度为 in_degress[i]。其中 n 为人数，也就是图中的顶点数。

- 接下来根据题目给的 trust 关系建图。由于我们定义图的方式为**a 信任 b 表示图中有一条从顶点 a 到顶点 b 的有向边**。因此如果 a 信任 b，那么 a 的出度 + 1，b 的入度 +1 。

- 最后遍历 in_degree 和 out_degree 找到满足 in_degree[i] 为 n - 1，并且 out_degress[i] 为 0 的点，返回即可。如果没有这样的点返回 -1。

关键点

- 将问题抽象为图，问题转为求图的入度和出度

代码

- 语言支持：Python3，CPP Code,JS Code, Java Code

Python3 Code:

```
class Solution:
    def findJudge(self, N, trust):
        in_degree = [0] * (N + 1)
        out_degree = [0] * (N + 1)
        for a, b in trust:
            in_degree[b] += 1
            out_degree[a] += 1
        for i in range(1, N + 1):
            if in_degree[i] == N - 1 and out_degree[i] == 0:
                return i
        return -1
```

我们也可以直接统计入度和出度的差，因为我们要找的是入度和出度差为 n -1 的点。这样可以将两个数组降低为一个数组，不过复杂度是一样的。

Python3 Code:

```
class Solution:
    def findJudge(self, N, trust):
        count = [0] * (N + 1)
        for i, j in trust:
            count[i] -= 1
            count[j] += 1
        for i in range(1, N + 1):
            if count[i] == N - 1:
```

```
                return i
        return -1
```

CPP Code:

```
class Solution {
public:
    int findJudge(int n, vector<vector<int>>& trust) {
        if (trust.empty() && n == 1) return 1;
        unordered_map<int, int> count;
        for (auto& relation : trust)
        {
            count[relation[0]] += -1;
            count[relation[1]] += 1;
        }
        int no_k = -1;
        for (auto& kvp : count)
        {
            if (kvp.second == (n-1)) no_k = kvp.first;
        }
        return -1;
    }
};
```

JS Code:

```
/**
 * @param {number} n
 * @param {number[][]} trust
 * @return {number}
 */
var findJudge = function (n, trust) {
  const count = new Array(n + 1).fill(0);
  for (const edge of trust) {
    const x = edge[0];
    const y = edge[1];
    count[y]++;
    count[x]--;
  }
  for (let i = 1; i <= n; ++i) {
    if (count[i] === n - 1) {
      return i;
    }
  }
  return -1;
};
```

Java Code:

```
class Solution {
    public int findJudge(int n, int[][] trust) {
        int[] count = new int[n + 1];
        for (int[] edge : trust) {
            int x = edge[0];
            int y = edge[1];
            ++count[y];
            --count[x];
        }
        for (int i = 1; i < n + 1; i++) {
            if (count[i] == n - 1) {
                return i;
            }
        }
        return -1;
    }
}
```

复杂度分析

令 n 为数组长度。

- 时间复杂度：$O(n)$
- 空间复杂度：$O(n)$

> 此题解由 力扣刷题插件 自动生成。

力扣的小伙伴可以关注我，这样就会第一时间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

关注公众号力扣加加，努力用清晰直白的语言还原解题思路，并且有大量图解，手把手教你识别套路，高效刷题。

欢迎长按关注

上一页 下一页