

【华为OD机考 统一考试机试C卷】城市聚集度/找城市 (C++ Java JavaScript Python)

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份, 华为官方已经将 华为OD机考: OD统一考试 (A卷 / B卷) 切换到 OD统一考试 (C卷) 和 OD统一考试 (D卷) 。根据考友反馈: 目前抽到的试卷为B卷或C卷/D卷, 其中C卷居多, 按照之前的经验C卷D卷部分考题会复用A卷/B卷题, 博主正积极从考过的同学收集C卷和D卷真题, 可以查看下面的真题目录。

真题目录: [华为OD机考机试 真题目录 \(C卷 + D卷 + B卷 + A卷\) + 考点说明](#)

专栏: [2023华为OD机试\(B卷+C卷+D卷\) \(C++JavaJSPy\)](#)

华为OD面试真题精选: [华为OD面试真题精选](#)

在线OJ: [点击立即刷题, 模拟真实机考环境](#) [华为OD机考B卷C卷](#)[华为OD机考华为OD机考B卷](#)[华为OD机试B卷](#)[华为OD机试C卷](#)[华为OD机考C卷](#)[华为OD机考D卷](#)[华为OD机考C卷/D卷答案](#)[华为OD机考C卷/D卷解析](#)[华为OD机考C卷和D卷真题](#)[华为OD机考C卷和D卷题解](#)

题目描述

一张地图上有 n 个城市, 城市和城市之间有且只有一条道路相连: 要么直接相连, 要么通过其它城市中转相连 (可中转一次或多次)。城市与城市之间的道路**都不会成环**。

当切断通往某个城市 i 的所有道路后, 地图上将分为多个连通的城市群, 设该城市 i 的聚集度为 DP_i (Degree of Polymerization), $DP_i = \max$ (城市群1的城市个数, 城市群2的城市个数, ...城市群 m 的城市个数)。

请找出地图上DP值最小的城市 (即找到城市 j , 使得 $DP_j = \min(DP_1, DP_2 \dots DP_n)$)

提示: 如果有**多个城市**都满足条件, 这些城市都要找出来 (**可能存在多个解**)

提示: DP_i 的计算, 可以理解为已知一棵树, 删除某个节点后; 生成的多个子树, 求解多个子数节点数的问题。

输入描述

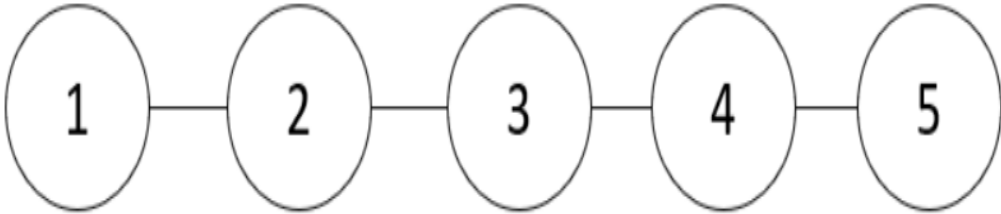
每个样例: 第一行有一个整数 N , 表示有 N 个节点。 $1 \leq N \leq 1000$ 。

接下来的 $N-1$ 行每行有两个整数 x, y , 表示城市 x 与城市 y 连接。 $1 \leq x, y \leq N$

输出描述

输出城市的编号。如果有多个, 按照编号升序输出。

用例

输入	5 1 2 2 3 3 4 4 5
输出	3
说明	<div>输入表示的是如下地图：</div> <div></div> <div>对于城市3，切断通往3的所有道路后，形成2个城市群[（1，2），（4，5）]，其聚集度分别都是2。DP3 = 2。</div> <div>对于城市4，切断通往城市4的所有道路后，形成2个城市群[（1，2，3），（5）]，DP4 = max（3，1）= 3。</div> <div>依次类推，切断其它城市的所有道路后，得到的DP都会大于2，因为城市3就是满足条件的城市，输出是3。</div>
输入	6 1 2 2 3 2 4 3 5 3 6
输出	2 3

说明

将通往2或者3的所有路径切断，最大城市群数量是3，其他任意城市切断后，最大城市群数量都比3大，所以输出2 3

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <unordered_map>
4  #include <algorithm>
5  #include <climits>
6
7  using namespace std;
8
9  class UnionFindSet {
10 public:
11     vector<int> father; // 存储每个节点的父节点
12
13     UnionFindSet(int n) { // 初始化并查集，每个节点的父节点为自己
14         father.resize(n);
15         for (int i = 0; i < n; i++) father[i] = i;
16     }
17
18     int find(int x) { // 查找x的祖先节点，路径压缩优化
19         if (father[x] != x) {
20             return father[x] = find(father[x]);
21         }
22         return x;
23     }
24
25     void unionSet(int x, int y) { // 合并x和y所在的集合
26         int x_fa = find(x);
27         int y_fa = find(y);
28
29         if (x_fa != y_fa) { // 如果x和y不在同一个集合中，则将y的祖先节点设为x的祖先节点
30             father[y_fa] = x_fa;
31         }
32     }
33 };
34
35 int main() {
36
```

```
37     int n;
38     cin >> n;
39
40     vector<vector<int>> relations(n - 1, vector<int>(2)); // 存储n-1条关系
41     for (int i = 0; i < n - 1; i++) { // 输入n-1条关系
42         cin >> relations[i][0] >> relations[i][1];
43     }
44
45     int min_dp = INT_MAX; // 最小的最大连通块大小
46     vector<int> city; // 最小的最大连通块所在的城市
47
48     for (int i = 1; i <= n; i++) { // 枚举每个城市作为特殊城市
49         UnionFindSet ufs(n + 1); // 初始化并查集
50         for (const auto& relation : relations) { // 将与特殊城市相连的边删除
51             int x = relation[0], y = relation[1];
52             if (x == i || y == i) continue;
53             ufs.unionSet(x, y);
54         }
55
56         unordered_map<int, int> count; // 统计每个连通块的大小
57         for (int f : ufs.father) {
58             f = ufs.find(f);
59             count[f]++;
60         }
61
62         int dp = 0; // 最大连通块大小
63         for (const auto& c : count) {
64             dp = max(dp, c.second);
65         }
66
67         if (dp < min_dp) { // 如果当前最大连通块大小比之前的最小值还小, 则更新最小值和最小值所在的城市
68             min_dp = dp;
69             city.clear();
70             city.push_back(i);
71         }
72         else if (dp == min_dp) { // 如果当前最大连通块大小与之前的最小值相等, 则将城市加入最小值所在的城市列表
73             city.push_back(i);
74         }
75     }
76
77 }
```

```
78     for (int c : city) { // 输出最小的最大连通块所在的城市
79         cout << c << " ";
80     }
81     cout << endl;
82
83     return 0;
84 }
```

java

```
1  import java.util.*;
2  import java.lang.*;
3  import java.io.*;
4
5  class Main {
6      static class UnionFindSet {
7          private int[] father; // 存储每个节点的父节点
8
9          public UnionFindSet(int n) { // 初始化并查集, 每个节点的父节点为自己
10             father = new int[n];
11             for (int i = 0; i < n; i++) father[i] = i;
12         }
13
14         public int find(int x) { // 查找x的祖先节点, 路径压缩优化
15             if (father[x] != x) {
16                 father[x] = find(father[x]);
17             }
18             return father[x];
19         }
20
21         public void unionSet(int x, int y) { // 合并x和y所在的集合
22             int x_fa = find(x);
23             int y_fa = find(y);
24
25             if (x_fa != y_fa) { // 如果x和y不在同一个集合中, 则将y的祖先节点设为x的祖先节点
26                 father[y_fa] = x_fa;
27             }
28         }
29     }
30 }
```

```
31 public static void main(String[] args) throws java.lang.Exception {
32     Scanner sc = new Scanner(System.in);
33     int n = sc.nextInt();
34
35     int[][] relations = new int[n - 1][2]; // 存储n-1条关系
36     for (int i = 0; i < n - 1; i++) { // 输入n-1条关系
37         relations[i][0] = sc.nextInt();
38         relations[i][1] = sc.nextInt();
39     }
40
41     int min_dp = Integer.MAX_VALUE; // 最小的最大连通块大小
42     List<Integer> city = new ArrayList<>(); // 最小的最大连通块所在的城市
43
44     for (int i = 1; i <= n; i++) { // 枚举每个城市作为特殊城市
45         UnionFindSet ufs = new UnionFindSet(n + 1); // 初始化并查集
46         for (int[] relation : relations) { // 将与特殊城市相连的边删除
47             int x = relation[0], y = relation[1];
48             if (x == i || y == i) continue;
49             ufs.unionSet(x, y);
50         }
51
52         Map<Integer, Integer> count = new HashMap<>(); // 统计每个连通块的大小
53         for (int f : ufs.father) {
54             f = ufs.find(f);
55             count.put(f, count.getOrDefault(f, 0) + 1);
56         }
57
58         int dp = 0; // 最大连通块大小
59         for (Map.Entry<Integer, Integer> entry : count.entrySet()) {
60             dp = Math.max(dp, entry.getValue());
61         }
62
63         if (dp < min_dp) { // 如果当前最大连通块大小比之前的最小值还小, 则更新最小值和最小值所在的城市
64             min_dp = dp;
65             city.clear();
66             city.add(i);
67         }
68         else if (dp == min_dp) { // 如果当前最大连通块大小与之前的最小值相等, 则将城市加入最小值所在的城市列表
69             city.add(i);
70         }
71     }
72 }
```

```
72     }
73
74     for (int c : city) { // 输出最小的最大连通块所在的城市
75         System.out.print(c + " ");
76     }
77 }
}
```

javaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3      input: process.stdin,
4      output: process.stdout
5  });
6
7  class UnionFindSet {
8      constructor(n) {
9          this.father = new Array(n);
10         for (let i = 0; i < n; i++) {
11             this.father[i] = i;
12         }
13     }
14
15     find(x) {
16         if (this.father[x] !== x) {
17             this.father[x] = this.find(this.father[x]);
18         }
19         return this.father[x];
20     }
21
22     unionSet(x, y) {
23         const x_fa = this.find(x);
24         const y_fa = this.find(y);
25
26         if (x_fa !== y_fa) {
27             this.father[y_fa] = x_fa;
28         }
29     }
30 }
```



```
31 |
32 |
33 | let n;
34 | let relations = [];
35 |
36 | rl.on('line', (line) => {
37 |   if (!n) {
38 |     n = parseInt(line);
39 |   } else {
40 |     const [x, y] = line.split(' ').map(Number);
41 |     relations.push([x, y]);
42 |     if (relations.length === n - 1) {
43 |       rl.close();
44 |     }
45 |   }
46 | });
47 |
48 | rl.on('close', () => {
49 |   let min_dp = Infinity;
50 |   let city = [];
51 |
52 |   for (let i = 1; i <= n; i++) {
53 |     const ufs = new UnionFindSet(n + 1);
54 |     for (const [x, y] of relations) {
55 |       if (x === i || y === i) continue;
56 |       ufs.unionSet(x, y);
57 |     }
58 |
59 |     const count = new Map();
60 |     for (let f of ufs.father) {
61 |       f = ufs.find(f);
62 |       count.set(f, (count.get(f) || 0) + 1);
63 |     }
64 |
65 |     let dp = 0;
66 |     for (const c of count.values()) {
67 |       dp = Math.max(dp, c);
68 |     }
69 |
70 |     if (dp < min_dp) {
71 |       min_dp = dp;
```

```
72     city = [i];
73 } else if (dp === min_dp) {
74     city.push(i);
75 }
76 }
77
78 console.log(city.join(' '));
});
```

python

```
1 class UnionFindSet:
2     def __init__(self, n):
3         self.father = [i for i in range(n)]
4
5     def find(self, x):
6         if self.father[x] != x:
7             self.father[x] = self.find(self.father[x])
8         return self.father[x]
9
10    def unionSet(self, x, y):
11        x_fa = self.find(x)
12        y_fa = self.find(y)
13        if x_fa != y_fa:
14            self.father[y_fa] = x_fa
15
16    n = int(input())
17    relations = [list(map(int, input().split())) for _ in range(n - 1)]
18
19    min_dp = float('inf')
20    city = []
21
22    for i in range(1, n + 1):
23        ufs = UnionFindSet(n + 1)
24        for x, y in relations:
25            if x == i or y == i:
26                continue
27            ufs.unionSet(x, y)
28
29    count = {}
```

```
30     for f in ufs.father:
31         f = ufs.find(f)
32         count[f] = count.get(f, 0) + 1
33
34     dp = max(count.values())
35
36     if dp < min_dp:
37         min_dp = dp
38         city = [i]
39     elif dp == min_dp:
40         city.append(i)
41
42 print(*city)
```

文章目录

[华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷](#)

[题目描述](#)

[输入描述](#)

[输出描述](#)

[用例](#)

[C++](#)

[java](#)

[javaScript](#)

[python](#)

机考真题 华为OD



CSDN @算法大师