

57、城市聚集度，考点 or 实现——数据结构/并查集

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DPi (Degree of Polymerization) , DPi = 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DPj = min(DP1,DP2 ... DPn))

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：DPi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，求解多个子树节点数的问题。

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 <= N <= 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 <= x, y <= 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

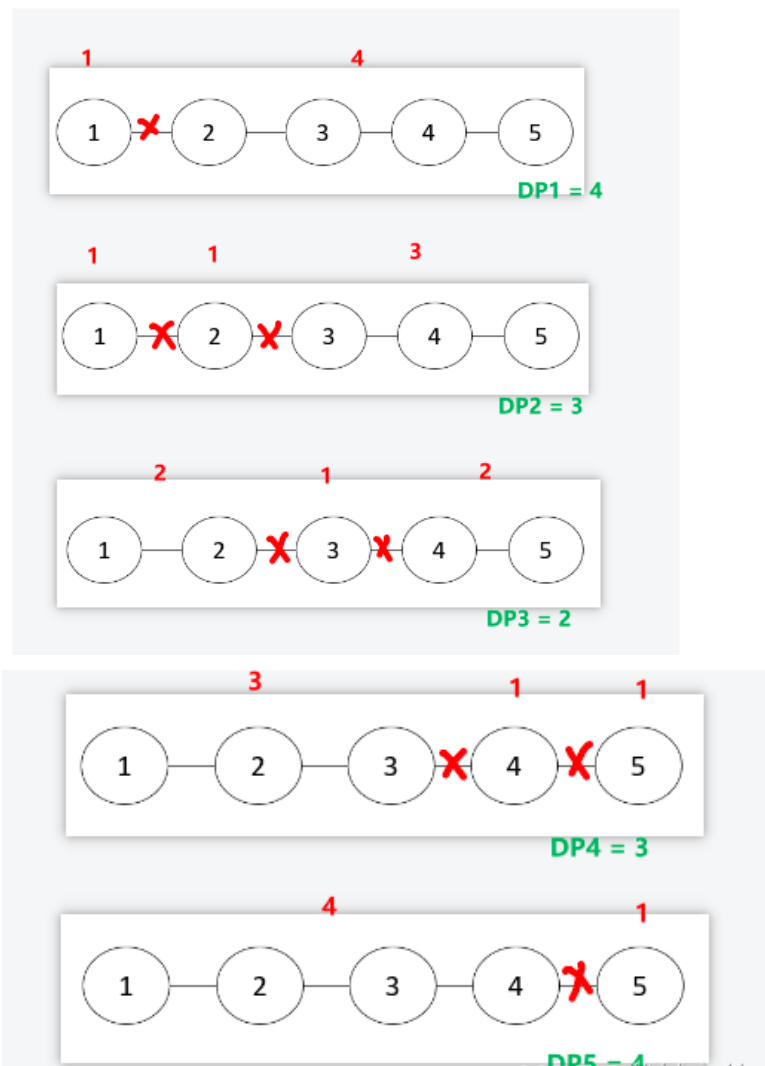
输入	5 1 2 2 3 3 4 4 5
输出	3
说明	<p>输入表示的是如下地图：</p> <div></div> <p>CSDN @伏城之外</p> <p>对于城市3，切断通往3的所有道路后，形成2个城市群 [(1, 2) , (4, 5)], 其聚集度分别都是2。DP3 = 2。</p> <p>对于城市4，切断通往城市4的所有道路后，形成2个城市群 [(1, 2, 3) , (5)], DP4 = max (3, 1) = 3。</p> <p>依次类推，切断其它城市的所有道路后，得到的DP都会大于2，因为城市3就是满足条件的城市，输出是3。</p>

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

题目解析

用例1的意思如下：

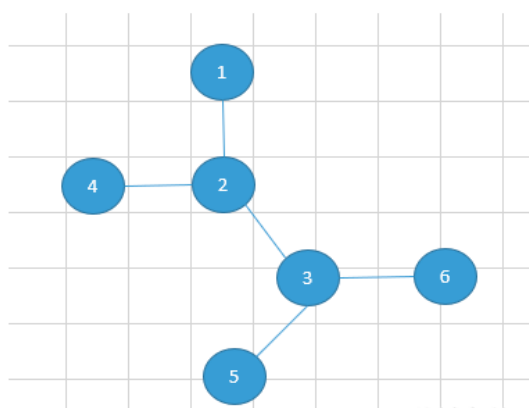
$DP_i = \max(\text{城市群1的城市个数}, \text{城市群2的城市个数}, \dots, \text{城市群m的城市个数})$

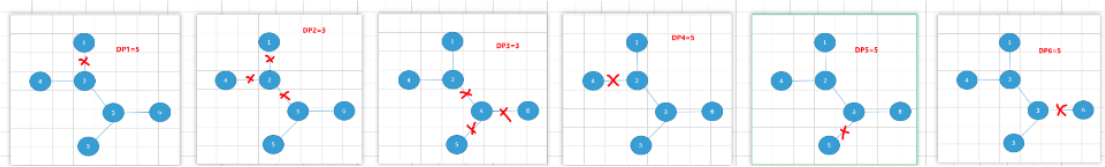


题目要求找出地图上DP值最小的城市（即找到城市j，使得 $DP_j = \min(DP_1, DP_2, \dots, DP_n)$ ），

因此DP3最小，输出DP3的3。

用例2图示如下

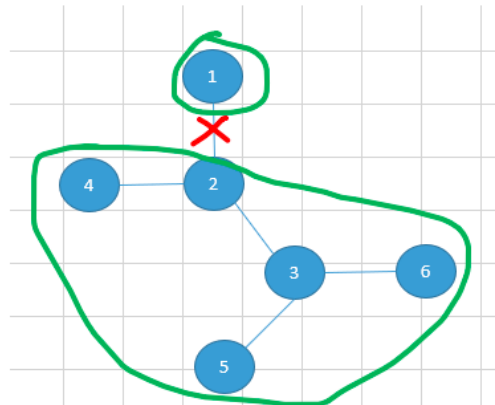




输出2, 3。

通过上面两个示例的图示可以发现，其实本题就是求解：图的 **连通分量** Q 。

本题的难点在于，分量关系没有直接给出，需要我们自己推导，其次是需要我们统计出每个连通分量的节点个数。



比如，上面且1的所有联系，则会产生两个连通分量，分别是[1], [4,2,3,5,6]，这两个连通分量的节点个数分别为1, 5。

求解图的连通分量，我们一般使用 **并查集** Q 。

但是，这里我们不能直接根据输入的连接关系，来产生并查集，因为输入的连接关系没有被切断。

本题是要我们尝试切断每一个城市的连接，因此我们遍历出每一个城市，比如城市2，然后遇到和城市2相关的连接后，我们就跳过并查集的合并操作，这样就能产生切断效果。

我们模拟下用例2

初始时，每个城市的父都是自己

city	1	2	3	4	5	6
fa_city	1	2	3	4	5	6

输入的连接关系如下

1 2

2 3

2 4

3 5

3 6

现在我们要切断城市2的所有联系，则遇到和2相关的连接时就不进行合并操作

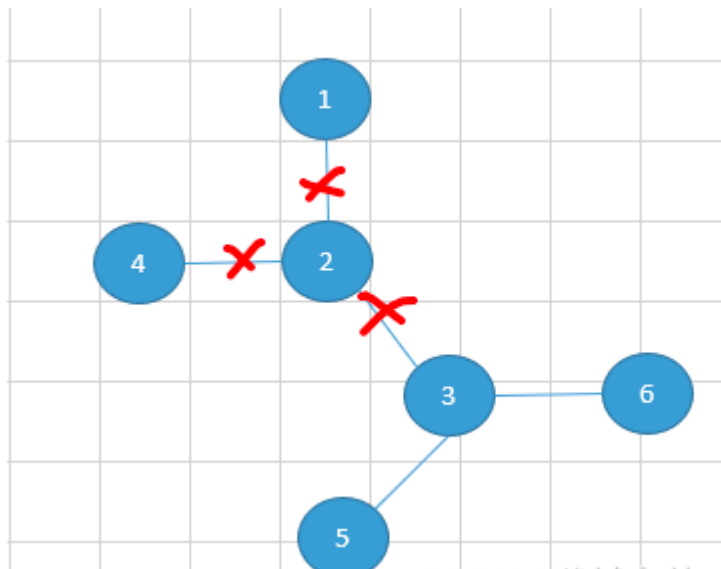
因此上述连接关系只需要考虑

3 5

3 6

这样的话，最终并查集的合并结果为

city	1	2	3	4	5	6
fa_city	1	2	3	4	3	3



可以发现，父为3的连通分量具有最多城市数量

这样的话，我们就可以求解出每个DPi了，最后求其中最小的DP对应i即可。

JavaScript算法源码

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
15     n = parseInt(lines[0]);
16   }
17
18   if (n && lines.length === n) {
19
20     if (n && lines.length === n) {
21       lines.shift();
22       const relations = lines.map((line) => line.split(" ").map(Number));
23
24       console.log(getMinDP(relations, n));
25
26       lines.length = 0;
27     }
28   });
29
30 function getMinDP(relations, n) {
31   // min用于保存DPj，即城市聚集度
32   let min = Infinity;
33   // city用于保存j，即城市序号
34   let city = [];
35
36   // 遍历每个城市 1~n
37   for (let i = 1; i <= n; i++) {
38     const ufs = new UnionFindSet(n);
39     for (let [x, y] of relations) {
40       // 切断城市的所有道路，即忽略和城市i有联系的合并操作
41       if (x === i || y === i) continue;
42       ufs.union(x, y);
43     }
44
45     // 统计各个连通分量自身的城市个数
46     const count = {};
47     ufs.fa.forEach((f) => {
```

```

46     f = ufs.find(f);
47     count[f] ? count[f]++ : (count[f] = 1);
48 });
49
50 // 取最多城市个数作为当前的切断城市的聚集度
51 const dp = Object.values(count).sort((a, b) => b - a)[0];
52
53 if (dp < min) {
54     min = dp;
55     city = [i];
56 } else if (dp === min) {
57     city.push(i);
58 }
59 }
60
61 return city.join(" ");
62 }
63
64 /* 并查集 */
65 class UnionFindSet {
66     constructor(n) {
67         this.fa = new Array(n + 1).fill(0).map((_, idx) => idx);
68         this.count = n;
69     }
70
71     find(x) {
72         if (x !== this.fa[x]) {
73             return (this.fa[x] = this.find(this.fa[x]));
74         }
75         return x;
76     }
77
78     union(x, y) {
79         let x_fa = this.find(x);
80         let y_fa = this.find(y);
81
82         if (x_fa !== y_fa) {
83             this.fa[y_fa] = x_fa;
84             this.count--;
85         }
86     }
87 }

```

Java算法源码

```
1 import java.util.ArrayList;
2 import java.util.HashMap;
3 import java.util.Scanner;
4 import java.util.StringJoiner;
5
6 public class Main {
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         int n = sc.nextInt();
10
11         int[][] relations = new int[n - 1][2];
12         for (int i = 0; i < n - 1; i++) {
13             relations[i][0] = sc.nextInt();
14             relations[i][1] = sc.nextInt();
15         }
16
17         System.out.println(getResult(n, relations));
18     }
19
20     public static String getResult(int n, int[][] relations) {
21         int min = Integer.MAX_VALUE;
22         ArrayList<Integer> city = new ArrayList<>();
23
24         for (int i = 1; i <= n; i++) {
25             UnionFindSet ufs = new UnionFindSet(n + 1);
26             for (int[] relation : relations) {
27                 int x = relation[0], y = relation[1];
28                 if (x == i || y == i) continue;
```

```
29     ufs.union(x, y);
30 }
31
32 HashMap<Integer, Integer> count = new HashMap<>();
33 for (int f : ufs.fa) {
34     f = ufs.find(f);
35     count.put(f, count.getDefault(f, 0) + 1);
36 }
37
38 int dp = count.values().stream().max((a, b) -> a - b).orElse(0);
39
40 if (dp < min) {
41     min = dp;
42     city = new ArrayList<>();
43     city.add(i);
44 } else if (dp == min) {
45     city.add(i);
46 }
47 }
48
49 StringJoiner sj = new StringJoiner(" ");
50 for (Integer c : city) {
51     sj.add(c + "");
52 }
53 return sj.toString();
54 }
55 }
```



```
56
57 class UnionFindSet {
58     int[] fa;
59
60     public UnionFindSet(int n) {
61         this.fa = new int[n];
62         for (int i = 0; i < n; i++) fa[i] = i;
63     }
64
65     public int find(int x) {
66         if (this.fa[x] != x) {
67             return this.fa[x] = this.find(this.fa[x]);
68         }
69         return x;
70     }
71
72     public void union(int x, int y) {
73         int x_fa = this.find(x);
74         int y_fa = this.find(y);
75
76         if (x_fa != y_fa) {
77             this.fa[y_fa] = x_fa;
78         }
79     }
80 }
```

Python算法源码

```
1 import sys
2
3 # 输入获取
4 n = int(input())
5 relations = [list(map(int, input().split())) for i in range(n - 1)]
6
7
8 # 并查集
9 class UnionFindSet:
10     def __init__(self, n):
11         self.fa = [i for i in range(n)]
12         self.count = n
13
14     def find(self, x):
15         if x != self.fa[x]:
16             self.fa[x] = self.find(self.fa[x])
17         return self.fa[x]
18     return x
19
20     def union(self, x, y):
21         x_fa = self.find(x)
22         y_fa = self.find(y)
23         if x_fa != y_fa:
24             self.fa[y_fa] = x_fa
25             self.count -= 1
26
```

```
27
28 # 算法入口
29 def getResult(n, relations):
30     minV = sys.maxsize
31     city = []
32
33     for i in range(1, n + 1):
34         ufs = UnionFindSet(n + 1)
35
36         for x, y in relations:
37             if x == i or y == i:
38                 continue
39             ufs.union(x, y)
40
41         count = {}
42         for x in ufs.fa:
43             f = ufs.find(x)
44             if count.get(f) is None:
45                 count[f] = 1
46             else:
47                 count[f] += 1
48
49         tmp = list(count.values())
50         tmp.sort(reverse=True)
51         dp = tmp[0]
52
53         if dp < minV:
```

```
54         minV = dp
55         city = [i]
56         elif dp == minV:
57             city.append(i)
58
59         return " ".join(map(str, city))
60
61
62 # 算法调用
63 print(getResult(n, relations))
```