https://course.acciojob.com/idle?question=504c7464-d786-42b5-8807-4ff8e3b64918

- MEDIUM
- Max Score: 40 Points
-
-
-

Redundant Connection

In this problem, a tree is an undirected graph that is connected and has no cycles.

You are given a graph that started as a tree with `n` nodes labeled from `1` to `n`, with one additional edge added. The added edge has two different vertices chosen from `1` to `n`, and was not an edge that already existed. The graph is represented as an array `edges` of length `n` where `edges[i]` = [a_i, b_i] indicates that there is an edge between nodes a_i and b_i in the graph.

Return an edge that can be removed so that the resulting graph is a tree of `n` nodes. If there are multiple answers, return the answer that occurs last in the input.

Note: You just need to complete findRedundantConnection() function and returns an array representing removed edge.

Input Format

The first line contains a single integer `n` where `n` denotes the number of nodes as well as edges in the graph. Next `n` lines contain an array representing the edge between two nodes .

Output Format

Print an `edge` that can be removed

Example 1

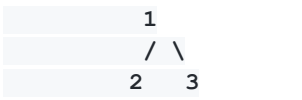
Input

```
3
1 2
1 3
2 3
```

Output

```
2 3
```

Explanation




```
After removing an edge [2,3]
We can convert this undirected graph into a tree.
```

Example 2

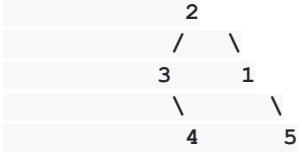
Input

```
5
1 2
2 3
3 4
1 4
1 5
```

Output

```
1 4
```

Explanation



```
After removing an edge [1,4]
We can convert this undirected graph into a tree.
```

Constraints

3 <= n <= 1000

1<= a_i, b_i <= n

The given graph is connected

Topic Tags

- **Graphs**

My code

```
// n java
import java.util.*;
class Solution{

    static int[] parent, rank;
        public static int find(int x) {
                if(parent[x] == x)
                        return x;

                int res = find(parent[x]);

                parent[x] = res;

                return res;
```

```
    }

    public static void union(int x, int y) {
        int px = find(x);
        int py = find(y);

        if(rank[px] < rank[py]) {
            parent[px] = py;
        } else if(rank[px] > rank[py]) {
            parent[py] = px;
        } else {
            parent[py] = px;
            rank[px]++;
        }
    }

public static int[] findRedundantConnection(int[][] edges){
    //write your code here

        int n = edges.length;

        parent = new int[n];
        rank = new int[n];

        for(int i = 0; i < n; i++) {
            parent[i] = i;
            rank[i] = 1;
        }

        for(int i = 0; i < n; i++) {
```

```
                int x = edges[i][0] - 1, y = edges[i][1] - 1;

                int px = find(x);
                int py = find(y);

                if(px != py) {
                    union(px, py);
                } else {
                    return new int[]{x+1, y+1};
                }
            }

            // this will never be called;
            return new int[]{0, 0};

    }
}

public class Main {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        //int m = sc.nextInt();
        int[][] edges = new int[n][2] ;
        for(int i = 0 ; i < n ; ++i){
            edges[i][0] = sc.nextInt();
            edges[i][1] = sc.nextInt();
        }
        int[] ans = Solution.findRedundantConnection(edges);
        for(int i = 0 ; i < 2 ; ++i){
```

```
            System.out.print(ans[i] + " ");
        }
    }
}
```