

Probl... < > ⚡ 🔒 Submit 📄 ⚙️ ⚙️ 0 ⏱ ⌂

Description Editorial Solutions Submissions

685. Redundant Connection II

Solved ✓

Hard Topics 🔒

In this problem, a rooted tree is a **directed** graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with n nodes (with distinct values from 1 to n), with one additional directed edge added. The added edge has two different vertices chosen from 1 to n , and was not an edge that already existed.

The resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair $[u_i, v_i]$ that represents a **directed** edge connecting nodes u_i and v_i , where u_i is a parent of child v_i .

Return *an edge that can be removed so that the resulting graph is a rooted tree of n nodes*. If there are multiple answers, return the answer that occurs last in the given 2D-array.

Example 1:

```
graph TD; 1((1)) --> 2((2)); 1((1)) --> 3((3)); 2((2)) --> 3((3))
```

Input: `edges = [[1,2],[1,3],[2,3]]`
Output: [2,3]

</> Code

C# 🔒 Auto

```
1 public class Solution {
2     public int[] FindRedundantConnection(int[][] edges) {
3         int n = edges.Length;
4         int[] parent = new int[n];
5         for (int i = 0; i < n; i++) {
6             parent[i] = i;
7         }
8
9         for (int i = 0; i < n; i++) {
10            int u = edges[i][0];
11            int v = edges[i][1];
12
13            if (findParent(parent, u) == findParent(parent, v)) {
14                return edges[i];
15            } else {
16                union(parent, u, v);
17            }
18        }
19
20        return null;
21    }
22
23    private int findParent(int[] parent, int u) {
24        if (parent[u] != u) {
25            parent[u] = findParent(parent, parent[u]);
26        }
27        return parent[u];
28    }
29
30    private void union(int[] parent, int u, int v) {
31        int pu = findParent(parent, u);
32        int pv = findParent(parent, v);
33
34        if (pu < pv) {
35            parent[pv] = pu;
36        } else {
37            parent[pu] = pv;
38        }
39    }
40}
```

Saved

Testcase > Te