Redundant Connection

We are given a "tree" in the form of a 2D-array, with `distinct` values for each node.

In the given 2D-array, each element pair `[u, v]` represents that `v` is a **child** of `u` in the tree.

We can remove `exactly` one redundant pair in this "tree" to make the result a (rooted) tree.

You need to find and output such a pair. If there are multiple answers for this question, output the one appearing last in the 2D-array. There is always at least one answer.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: Original tree will be like this:
  1
 / \
2 - 3
```

Example 2:

```
Input: [[1,2], [1,3], [3,1]]
Output: [3,1]
Explanation: Original tree will be like this:
  1
 / \\
2   3
```

Note:

- The size of the input 2D-array will be between 1 and 1000.
- Every integer represented in the 2D-array will be between 1 and 2000.

Solution 1

@administrators
@contributors
As the title says, my solution is not accepted. Why does this happen? I think we should remove [4,2] as my solution. What is wrong?

Input: [[2,3],[5,2],[1,5],[4,2],[4,1]]
Output: [4,2]
Expected: [4,1]

```
class Solution {
public:
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {
        map<int, int> parent;
        for (auto e : edges) {
            if (parent.find(e[1]) != parent.end()) {
                return e;
            }
            if (parent.find(e[0]) != parent.end()) {
                if (parent[e[0]] == e[1])
                    return e;
            }
            parent[e[1]] = e[0];
        }
        set<int> root;
        for (auto e : edges) {
            if (root.find(e[1]) != root.end())
                return e;
            root.insert(e[0]);
        }
        return edges.back();
    }
};
```

written by yanchao_hust original link here

Solution 2

```
class Solution {
    public int[] findRedundantConnection(int[][] edges) {
        int[] parent = new int[2001];
        for (int i = 0; i < parent.length; i++) parent[i] = i;

        for (int[] edge: edges){
            int f = edge[0], t = edge[1];
            if (find(parent, f) == find(parent, t)) return edge;
            else parent[find(parent, f)] = find(parent, t);
        }

        return new int[2];
    }

    private int find(int[] parent, int f) {
        if (f != parent[f]) {
          parent[f] = find(parent, parent[f]);
        }
        return parent[f];
    }
}
```

written by shawngao original link here

Solution 3

```
def findRedundantConnection(self, edges):
    tree = ''.join(map(chr, range(1001)))
    for u, v in edges:
        if tree[u] == tree[v]:
            return [u, v]
        tree = tree.replace(tree[u], tree[v])
```

Start with each node being its own tree, then combine trees by adding the edges. The first (and only) edge closing a loop is the last edge in that loop, which is the edge we must return.

My `tree[u]` denotes the tree of node `u`. I use a string because it has a convenient and fast `replace` method. My title is of course a pun on Union-Find.

written by StefanPochmann original link here