684. Redundant Connection

DescriptionHintsSubmissionsDiscussSolution

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

The given input is a graph that started as a tree with N nodes (with distinct values 1, 2, ..., 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 resulting graph is given as a 2D-array of `edges`. Each element of `edges` is a pair `[u, v]` with `u < v`, that represents an **undirected** edge connecting nodes `u` and `v`.

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 given 2D-array. The answer edge `[u, v]` should be in the same format, with `u < v`.

Example 1:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given undirected graph will be like this:
  1
 / \
2 - 3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [1,4], [1,5]]
Output: [1,4]
Explanation: The given undirected graph will be like this:
5 - 1 - 2
    |   |
    4 - 3
```

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

Update (2017-09-26):
We have overhauled the problem description + test cases and specified clearly the graph is an ***undirected*** graph. For the ***directed*** graph follow up please see **Redundant Connection II**). We apologize for any inconvenience caused.

Seen this question in a real interview before?

```
class Solution {
```

```
public:

    int root(vector<int> &parent, int f)
    {
        if(f!=parent[f])
        {
            parent[f] = root(parent,parent[f]);
        }
        return parent[f];
    }

    vector<int> findRedundantConnection(vector<vector<int>>& edges)
{
        int n = edges.size();
        vector<int> parent(2001,0);
        for (int i = 1; i <= n; i++) parent[i] = i;
        for(auto edge:edges)
        {
            int u = edge[0];
            int v = edge[1];
            int pu = root(parent,u);
            int pv = root(parent,v);
            if(pu==pv){
                return edge;
            }
            parent[pv]=pu;
        }
        return {};
    }
};
```

685. Redundant Connection II

DescriptionHintsSubmissionsDiscussSolution

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 1, 2, ..., 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, v]` that represents a **directed** edge connecting nodes `u` and `v`, where `u` is a parent of child `v`.

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:

```
Input: [[1,2], [1,3], [2,3]]
Output: [2,3]
Explanation: The given directed graph will be like this:
  1
 / \
v   v
2-->3
```

Example 2:

```
Input: [[1,2], [2,3], [3,4], [4,1], [1,5]]
Output: [4,1]
Explanation: The given directed graph will be like this:
5 <- 1 -> 2
     ^    |
     |    v
     4 <- 3
```

Note:

- The size of the input 2D-array will be between 3 and 1000.
- Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

Seen this question in a real interview before?

```
There are two cases for the tree structure to be invalid.
1) A node having two parents;
   including corner case: e.g. [[4,2],[1,5],[5,2],[5,3],[2,4]]
2) A circle exists
```

If we can remove exactly 1 edge to achieve the tree structure, a single node can have at most two parents. So my solution works in two steps.

```
1) Check whether there is a node having two parents.
    If so, store them as candidates A and B, and set the second edge invalid.
2) Perform normal union find.
    If the tree is now valid
           simply return candidate B
    else if candidates not existing
           we find a circle, return current edge;
    else
           remove candidate A instead of B.
```

```
class Solution {
public:

    int root(vector<int> &parent,int k)
    {
        if(parent[k]!=k)
        {
            parent[k] = root(parent,parent[k]);
        }
        return parent[k];
    }

    vector<int>
findRedundantDirectedConnection(vector<vector<int>>& edges) {
        int n = edges.size();
        vector<int> parent(n+1,0);
        vector<int> candA,candB;
        for(auto &edge:edges)
        {
            if(parent[edge[1]]==0)
            {
                parent[edge[1]]=edge[0];
            }else{
                candA = {parent[edge[1]],edge[1]};
                candB = edge;
                edge[1] = 0;
            }
        }
        for(int i=1;i<=n;++i) parent[i] = i;
        for(auto edge:edges)
        {
            if(edge[1]==0) continue;
            int u=edge[0];
            int v=edge[1];
            int pu = root(parent,u);
            int pv = root(parent,v);
            if(pu==pv)
            {
                if(candA.size()==0) return edge;
                return candA;
            }
            parent[pv]=pu;
        }
        return candB;

    }
};
```