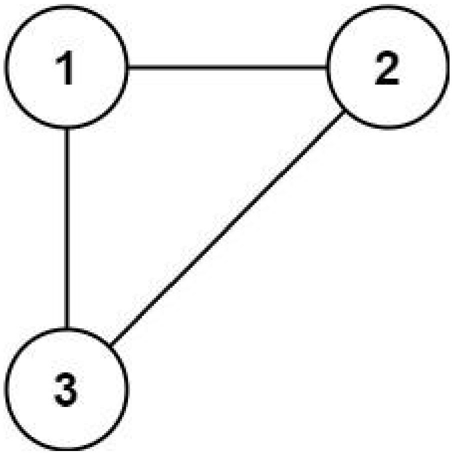
684. Redundant Connection

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.

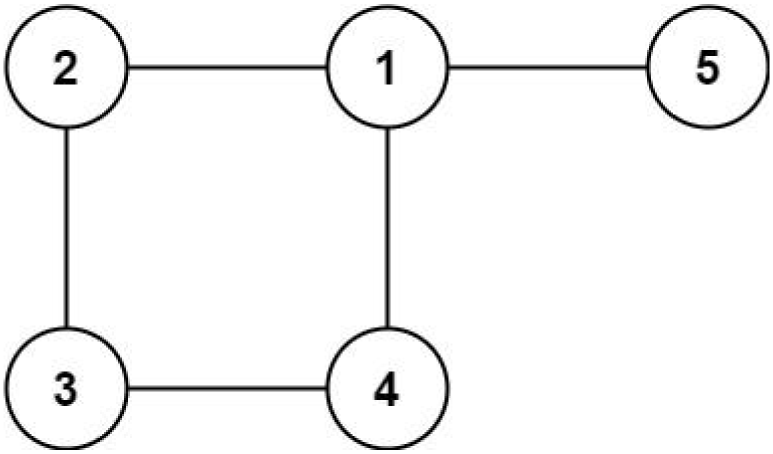
Return *an edge that can be removed so that the resulting graph is a tree of* n *nodes*. If there are multiple answers, return the answer that occurs last in the input.

Example 1:



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

Example 2:



```
Input: edges = [[1,2],[2,3],[3,4],[1,4],[1,5]]
Output: [1,4]
```

Constraints:

- n == edges.length
- 3 <= n <= 1000
- edges[i].length == 2
- 1 <= a_i < b_i <= edges.length
- a_i != b_i
- There are no repeated edges.
- The given graph is connected.

Code:

```
class Solution:
    def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:

        rank = [0]*1001
        parent = [i for i in range(0,1002)]

        def find(x):
            if parent[x]==x:
                return x
            parent[x] = find(parent[x])
            return parent[x]

        def union(x,y):
            a = find(x)
            b = find(y)
            if a == b:
                return False
            elif rank[a]>rank[b]:
                parent[b]=a
            elif rank[b]>rank[a]:
                parent[a]=b
            else:
                parent[b]=a
                rank[a]=a+1
            return True

        for edge in edges:
            if not union(*edge):
                return edge
```