题目：

在本问题中，树指的是一个连通且无环的无向图。

输入一个图，该图由一个有着N个节点（节点值不重复1，2，...，N）的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。

结果图是一个以边组成的二维数组。每一个边的元素是一对[u，v]，满足 u < v，表示连接顶点u 和v的无向图的边。

返回一条可以删去的边，使得结果图是一个有着N个节点的树。如果有多个答案，则返回二维数组中最后出现的边。答案边 [u，v]应满足相同的格式 u < v。

示例：

输入: [[1,2], [1,3], [2,3]]
输出: [2,3]
解释: 给定的无向图为:
  1
 / \
2 - 3

输入: [[1,2], [2,3], [3,4], [1,4], [1,5]]
输出: [1,4]
解释: 给定的无向图为:
5 - 1 - 2
    | |
    4 - 3

类型/方法：

连通且无环的无向图/并查集

思路分析：

方法一：比较常见的思路，使用并查集。
方法二：topo排序，根据节点的度来剪枝，最后所有节点度为2，形成环。

方法一：并查集思路：
遍历edges中所有边，将边的两个节点加入到并查集进行合并
    作以下判断：
    01. 如果两个节点根节点不同（不属于同一个并查集），说明还未形成环，继续遍历
    02. 如果两个节点根节点相同（属于同一个并查集），说明形成环，直接返回该边的两个节点，程序结束

算法流程：

（实际）代码：

```
class Solution:
    def findRedundantConnection(self, edges: List[List[int]]) -> List[int]:
        # 并查集的方法：
        # ------------------------
        p = [*range(len(edges)+1)]      # 并查集元素初始化 # 建立集合，每一个点的集合
        # 合并
        def f(x):                       # 递归修改所属集合
            if p[x] != x:
                p[x] = f(p[x])
            # else:
            #     return p[x]
            return p[x]
        for x ,y in edges:              # 遍历边
            px, py = f(x), f(y)
        #   # 查询
            if px != py:                # 《合并》检查集合，如果集合不同就合并
                p[py] = px
            else:
                return [x,y]            # 《检查》集合相同就返回答案
```