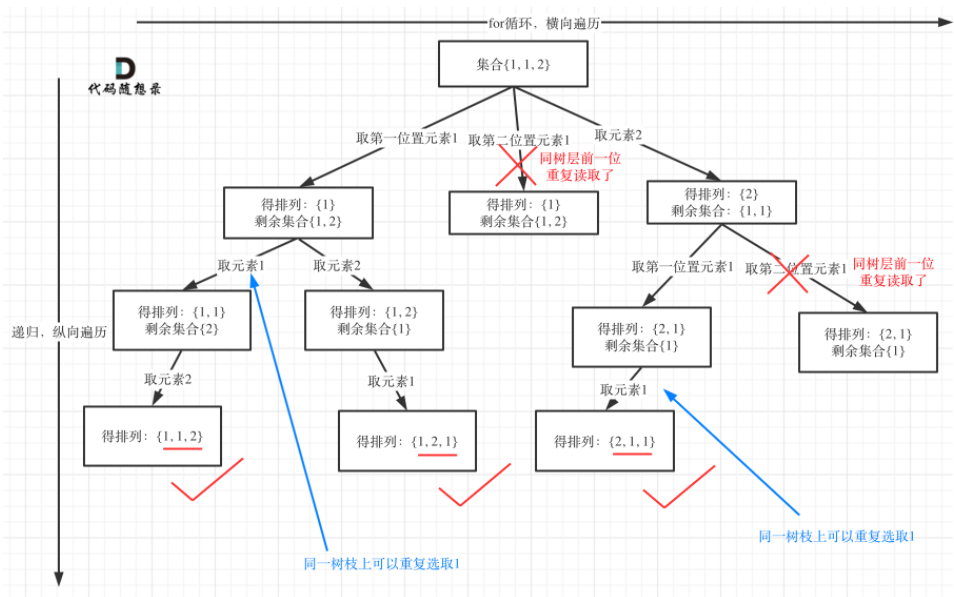
# 47、全排列（有重复元素）

DFS+回溯

如果没有重复元素，设置标志位，若已经选取了该元素，则该元素跳过此轮选择，即if(used[i]){continue;}。每一次都可以从没有选过的元素中选取，所以需要一个循环。

若出现重复元素，则在下图中可以看出，在同一层是不能选取一样的元素，则需要添加判断条件if（i>0&&nums[i-1]==nums[i]&&!nums[i-1]）



# 684、冗余连接

## 并查集

1.集合树：所有节点以代表节点为父节点构成的多叉树

2.节点的代表节点：可以理解为节点的父节点，从当前节点出发，可以向上找到的第一个节点

3.集合的代表节点：可以理解为根节点，意味着该集合内所有节点向上走，最终都能到达的节点

对于同一条边连接的结点，可以将其代表结点合并

<https://leetcode-cn.com/problems/redundant-connection/solution/tong-su-jiang-jie-bing-cha-ji-bang-zhu-xiao-bai-ku/>

<https://leetcode-cn.com/problems/redundant-connection/solution/yi-wen-zhang-wo-bing-cha-ji-suan-fa-by-a-fei-8/>

# 685、冗余连接（有向）

在一棵树上任意有向的连接两个节点，会出现以下情况：

1、根节点有父节点（环路）

2、有入度为2的结点，违背了树的结点（除根节点外）只有一个父节点

3、没有入度为2的结点，但是构成了环路

综合以上三种情况，可以合并情况1和情况3。

如果有入度为2的情况，那么删除的一定是这两条中的一条，如果删了一条，判断这个图是一个树，那么这条边就是答案，同时注意要从后向前遍历，因为如果两天边删哪一条都可以成为树，就删最后那一条；明确没有入度为2的情况，那么一定有有向环，找到构成环的边就是要删除的边。这个就是并查集的操作了。

# 968 监控二叉树

确定了遍历顺序，再看看这个状态应该如何转移，先来看看每个节点可能有几种状态

可以说有如下三种：

该节点无覆盖

本节点有摄像头

本节点有覆盖

我们分别有三个数字来表示

0：该节点无覆盖

1：本节点有摄像头

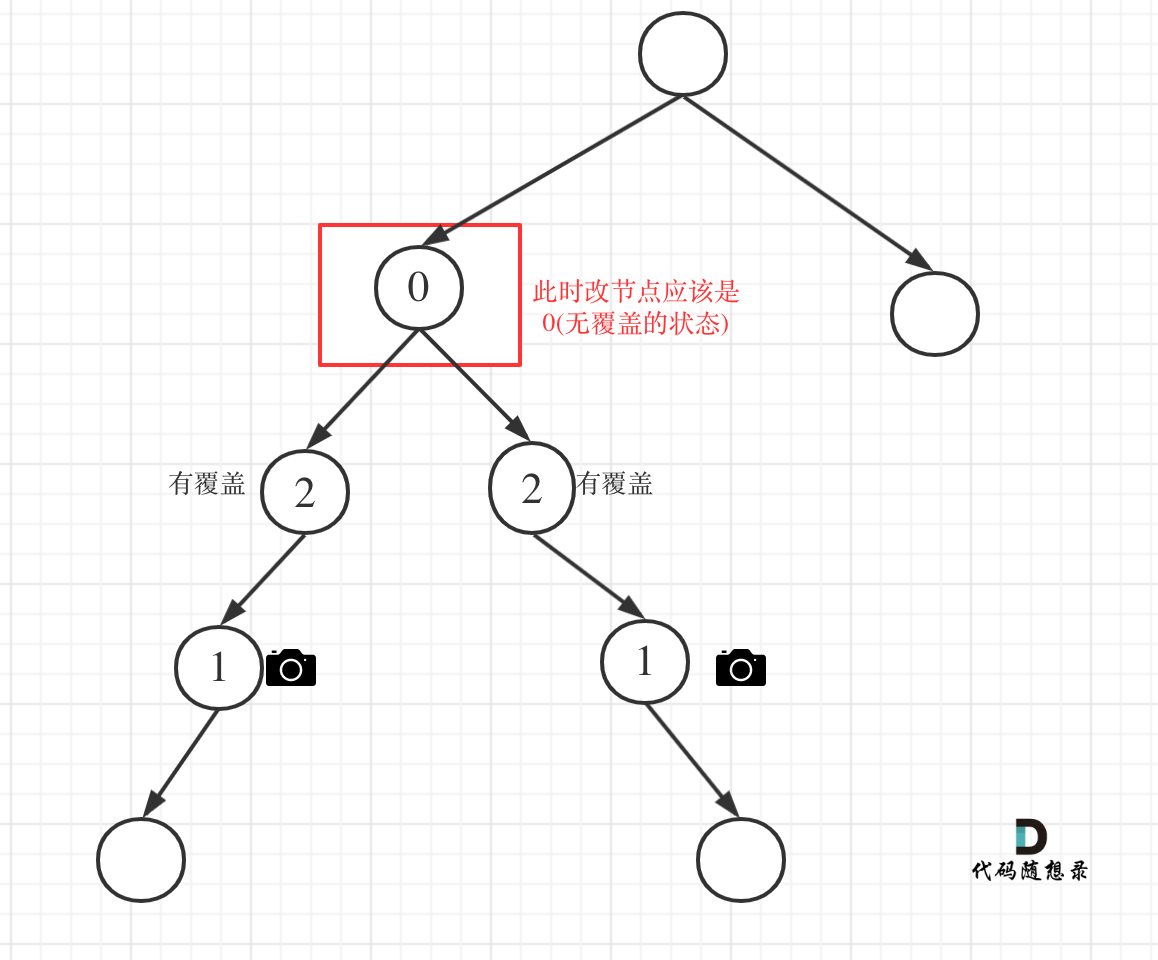
2：本节点有覆盖

空节点的状态只能是有覆盖

那么递归的终止条件应该是遇到了空节点，此时应该返回2（有覆盖）

情况1：左右节点都有覆盖

左孩子有覆盖，右孩子有覆盖，那么此时中间节点应该就是无覆盖的状态了。



情况2：左右节点至少有一个无覆盖的情况

如果是以下情况，则中间节点（父节点）应该放摄像头：

left == 0 && right == 0 左右节点无覆盖

left == 1 && right == 0 左节点有摄像头，右节点无覆盖

left == 0 && right == 1 左节点有无覆盖，右节点摄像头

left == 0 && right == 2 左节点无覆盖，右节点覆盖

left == 2 && right == 0 左节点覆盖，右节点无覆盖

这个不难理解，毕竟有一个孩子没有覆盖，父节点就应该放摄像头。

此时摄像头的数量要加一，并且return 1，代表中间节点放摄像头。

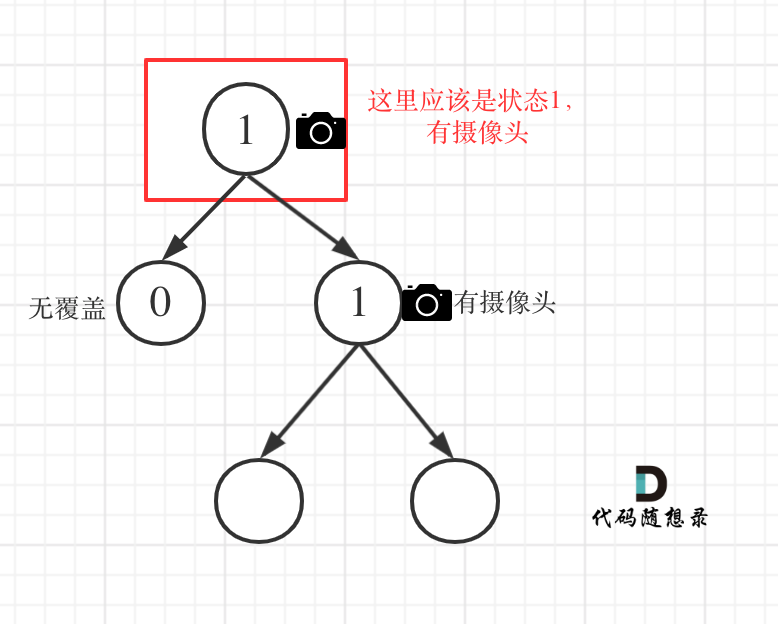
情况3：左右节点至少有一个有摄像头

如果是以下情况，其实就是 左右孩子节点有一个有摄像头了，那么其父节点就应该是2（覆盖的状态）

left == 1 && right == 2 左节点有摄像头，右节点有覆盖

left == 2 && right == 1 左节点有覆盖，右节点有摄像头

left == 1 && right == 1 左右节点都有摄像头



1. 情况4：头结点没有覆盖

以上都处理完了，递归结束之后，可能头结点 还有一个无覆盖的情况，如图：

