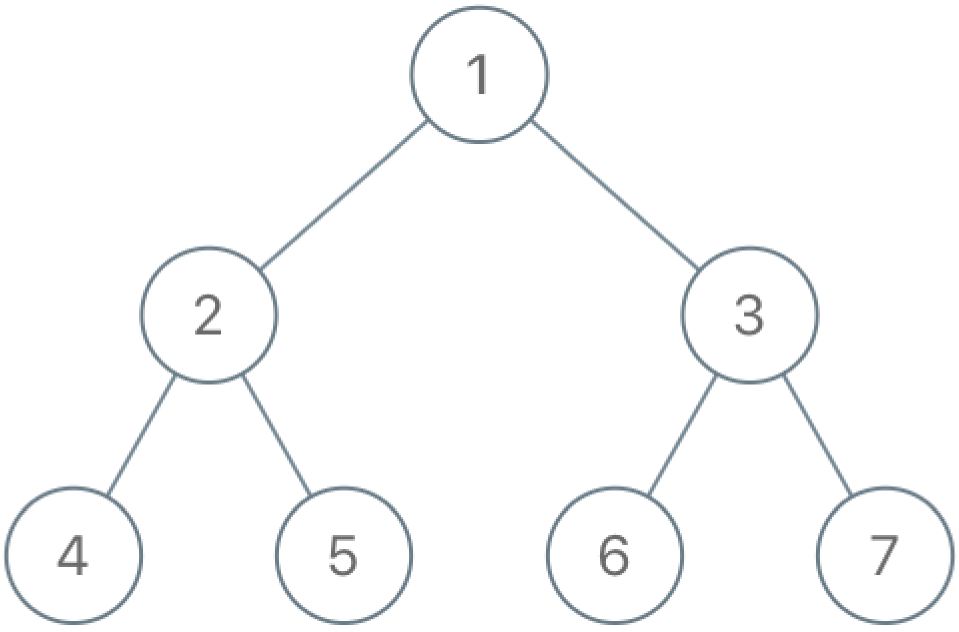
1110. Delete Nodes And Return Forest

Given the `root` of a binary tree, each node in the tree has a distinct value.

After deleting all nodes with a value in `to_delete`, we are left with a forest (a disjoint union of trees).

Return the roots of the trees in the remaining forest. You may return the result in any order.

Example 1:



```
Input: root = [1,2,3,4,5,6,7], to_delete = [3,5]
Output: [[1,2,null,4],[6],[7]]
```

Example 2:

```
Input: root = [1,2,4,null,3], to_delete = [3]
Output: [[1,2,4]]
```

Constraints:

- The number of nodes in the given tree is at most `1000`.
- Each node has a distinct value between `1` and `1000`.
- `to_delete.length <= 1000`
- `to_delete` contains distinct values between `1` and `1000`.

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def delNodes(self, root: TreeNode, to_delete: List[int]) ->
List[TreeNode]:
        to_delete = set(to_delete)
        res = []
        if root.val not in to_delete:
            root = self.dfs(root,to_delete,res)
            res = res + [root]
            return res
        else:
            root1 = self.dfs(root.left,to_delete,res)
            root2 = self.dfs(root.right,to_delete,res)
            root.left = None
            root.right= None
            if root1:
                res = res+[root1]
            if root2:
                res = res+[root2]

            return res

    def dfs(self,root,to_delete,res):
        if root is None:
            return None
        root.left = self.dfs(root.left,to_delete,res)
        root.right = self.dfs(root.right,to_delete,res)
        if root.val in to_delete:
            if root.left!=None:
                res.append(root.left)
            if root.right!=None:
                res.append(root.right)
            root.left,root.right = None,None
            return None
        return root
```

```
# Definition for a binary tree node.
# class TreeNode:
```

```
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def delNodes(self, root: TreeNode, to_delete: List[int]) ->
List[TreeNode]:
        ans = []
        self.helper(root,to_delete,ans)
        if root.val not in to_delete:
            ans.append(root)
        return ans


    def helper(self,root,to_delete,ans):
        if root is None:
            return

        root.left = self.helper(root.left,to_delete,ans)
        root.right = self.helper(root.right,to_delete,ans)

        if root.val in to_delete:
            lf = root.left
            rf = root.right
            root.left = None
            root.right = None
            if lf!=None:
                ans.append(lf)
            if rf!=None:
                ans.append(rf)
            return None
        return root
```