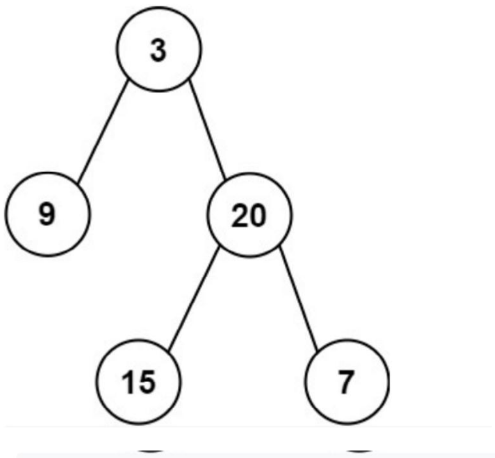
105. Construct Binary Tree from Preorder and Inorder Traversal

30 March 2022 07:08 PM

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.



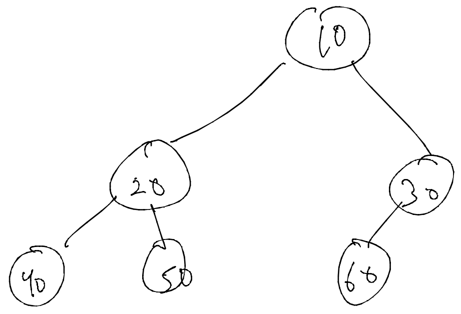
```
Input: preorder = [3,9,20,15,7], inorder =
[9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
```

Example 2:

```
Input: preorder = [-1], inorder = [-1]
Output: [-1]
```

inorder = {40, 20, 50, 10, 60, 30}

preorder = {10, 20, 40, 50, 30, 60}



from preorder

left ← Root → Right

↓

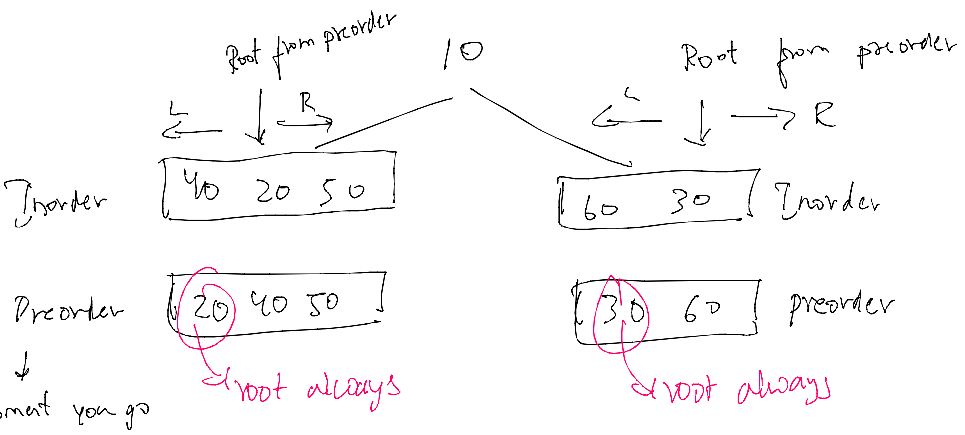
left Root Right

Inorder → [40 20 50 10 60 30]

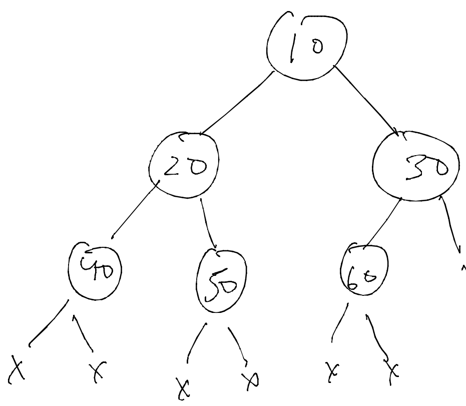
Preorder $\longrightarrow$ [10 $\underbrace{20 \quad 40 \quad 50}$ $\underbrace{30 \quad 60}$]

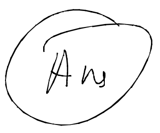
Root Left Right

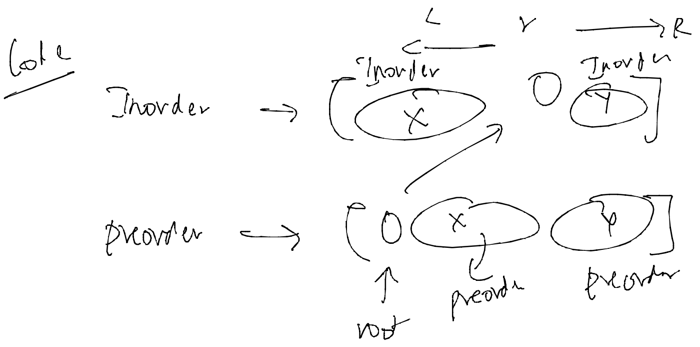
Root always

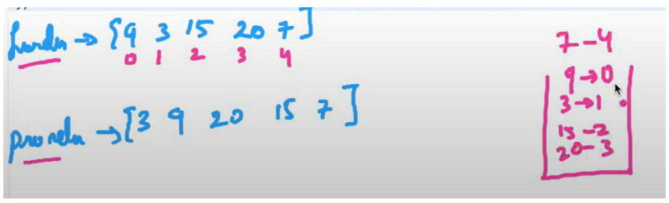


Preorder
↓
The moment you go to left you agin start with root









Inorder → [9 3 15 20 7]
0 1 2 3 4

preorder → [3 9 20 15 7]

7 → 4
9 → 0
3 → 1
15 → 2
20 → 3

```
// Put in-order in HashMap
for(int i = 0; i < inorder.length; i++){
    inMap.put(inorder[i], i);
}
```

```java
class Solution {
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        Map<Integer, Integer> inMap = new HashMap<Integer, Integer>();

        for(int i = 0; i < inorder.length; i++) {
            inMap.put(inorder[i], i);
        }

        TreeNode root = buildTree(preorder, 0, preorder.length - 1, inorder, 0, inorder.length - 1, inMap);
        return root;
    }

    public TreeNode buildTree(int[] preorder, int preStart, int preEnd, int[] inorder, int inStart, int inEnd, 
Map<Integer, Integer> inMap) {
        if(preStart > preEnd || inStart > inEnd) return null;

        TreeNode root = new TreeNode(preorder[preStart]);
        int inRoot = inMap.get(root.val);
        int numsLeft = inRoot - inStart;

        root.left = buildTree(preorder, preStart + 1, preStart + numsLeft, inorder, inStart, inRoot - 1, inMap);
        root.right = buildTree(preorder, preStart + numsLeft + 1, preEnd, inorder, inRoot + 1, inEnd, inMap);

        return root;
    }
}
```