889. Construct Binary Tree from Preorder and Postorder Traversal

Return any binary tree that matches the given preorder and postorder traversals.

Values in the traversals `pre` and `post` are distinct positive integers.

Input: pre = [1,2,4,5,3,6,7], post = [4,5,2,6,7,3,1] **Output:** [1,2,3,4,5,6,7]

- `1 <= pre.length == post.length <= 30`
- `` `pre[]` and `post[]` are both permutations of `1, 2, ..., pre.length`. ``
- `It is guaranteed an answer exists. If there exists multiple answers, you can return any of them.`

```
def constructFromPrePost(self, pre: List[int], post: List[int]) ->
TreeNode:
        if len(pre)==0:
            return None
        if len(pre)==1:
            return TreeNode(pre[0])
        temp = TreeNode(pre[0])
        idx = post.index(pre[1])
        temp.left = self.constructFromPrePost(pre[1:idx+2],post[:idx+1])
        temp.right = self.constructFromPrePost(pre[idx+2:],post[idx+1:-1])
        return temp
```

This is very important question

Official Solution:

Intuition

A preorder traversal is:

- `(root node) (preorder of left branch) (preorder of right branch)`

While a postorder traversal is:

- `(postorder of left branch) (postorder of right branch) (root node)`

For example, if the final binary tree is `[1, 2, 3, 4, 5, 6, 7]` (serialized), then the preorder traversal is `[1] + [2, 4, 5] + [3, 6, 7]`, while the postorder traversal is `[4, 5, 2] + [6, 7,`

`3] + [1]`.

If we knew how many nodes the left branch had, we could partition these arrays as such, and use recursion to generate each branch of the tree.

Algorithm

Let's say the left branch has LLL nodes. We know the head node of that left branch is `pre[1]`, but it also occurs last in the postorder representation of the left branch. So `pre[1] = post[L-1]` (because of uniqueness of the node values.) Hence, `L = post.indexOf(pre[1]) + 1`.

Now in our recursion step, the left branch is represnted by `pre[1 : L+1]` and `post[0 : L]`, while the right branch is represented by `pre[L+1 : N]` and `post[L : N-1]`.

```
class Solution(object):
    def constructFromPrePost(self, pre, post):
        if not pre: return None
        root = TreeNode(pre[0])
        if len(pre) == 1: return root

        L = post.index(pre[1]) + 1
        root.left = self.constructFromPrePost(pre[1:L+1], post[:L])
        root.right = self.constructFromPrePost(pre[L+1:], post[L:-1])
        return root
```