Problem 889: Construct Binary Tree from Preorder and Postorder Traversal

Problem Information

Difficulty: Medium
Acceptance Rate: 78.04%
Paid Only: No
Tags: Array, Hash Table, Divide and Conquer, Tree, Binary Tree

Problem Description

Given two integer arrays, `preorder` and `postorder` where `preorder` is the preorder traversal of a binary tree of **distinct** values and `postorder` is the postorder traversal of the same tree, reconstruct and return _the binary tree_.

If there exist multiple answers, you can **return any** of them.

Example 1:

Input: preorder = [1,2,4,5,3,6,7], postorder = [4,5,2,6,7,3,1] **Output:** [1,2,3,4,5,6,7]

Example 2:

Input: preorder = [1], postorder = [1] **Output:** [1]

Constraints:

* `1 <= preorder.length <= 30` * `1 <= preorder[i] <= preorder.length` * All the values of `preorder` are **unique**. * `postorder.length == preorder.length` * `1 <= postorder[i] <= postorder.length` * All the values of `postorder` are **unique**. * It is guaranteed that `preorder` and `postorder` are the preorder traversal and postorder traversal of the same binary tree.

Code Snippets

C++:

```
/**
* Definition for a binary tree node.
* struct TreeNode {
* int val;
* TreeNode *left;
* TreeNode *right;
* TreeNode() : val(0), left(nullptr), right(nullptr) {}
* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
* TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
right(right) {}
* };
*/
class Solution {
public:
TreeNode* constructFromPrePost(vector<int>& preorder, vector<int>& postorder)
{

}
};
```

Java:

```
/**
* Definition for a binary tree node.
* public class TreeNode {
* int val;
* TreeNode left;
* TreeNode right;
* TreeNode() {}
* TreeNode(int val) { this.val = val; }
* TreeNode(int val, TreeNode left, TreeNode right) {
* this.val = val;
* this.left = left;
* this.right = right;
* }
* }
*/
class Solution {
public TreeNode constructFromPrePost(int[] preorder, int[] postorder) {
```

```
}
}
```

Python3:

```
# 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 constructFromPrePost(self, preorder: List[int], postorder: List[int]) ->
Optional[TreeNode]:
```