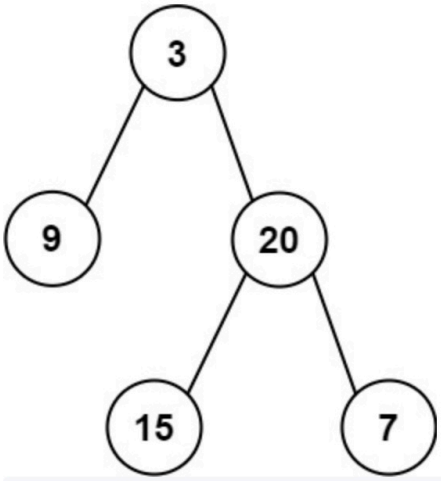
105. Construct Binary Tree from Preorder and Inorder Traversal

Medium 4754 123 Add to List Share

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*.

Example 1:



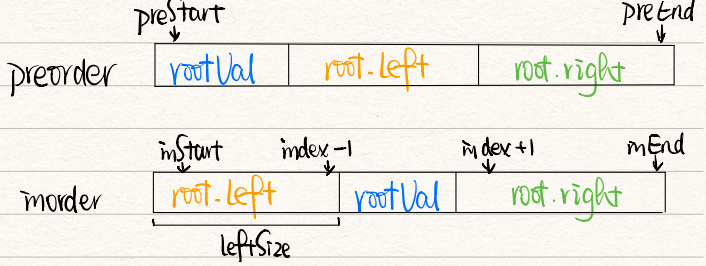

```
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
```

Example 2:

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

Constraints:

- `1 <= preorder.length <= 3000`
- `inorder.length == preorder.length`
- `-3000 <= preorder[i], inorder[i] <= 3000`
- `preorder` and `inorder` consist of **unique** values.
- Each value of `inorder` also appears in `preorder`.
- `preorder` is **guaranteed** to be the preorder traversal of the tree.
- `inorder` is **guaranteed** to be the inorder traversal of the tree.



```cpp
/**
 * 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* buildTree(vector<int>& preorder, vector<int>& inorder) {
        TreeNode *root = build(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1);
        return root;
    }

private:
    TreeNode* build(vector<int>& preorder, int preStart, int preEnd, vector<int>& inorder, int inStart, int inEnd) {
        // base case
        if (preStart > preEnd && inStart > inEnd) {
            return NULL;
        }

        // root value is the first element in preorder
        int rootVal = preorder[preStart];
        TreeNode *root = new TreeNode(rootVal);

        // find the index of root value in inorder
        int index = -1;
        for (int i = inStart; i <= inEnd; i++) {
            if (inorder[i] == rootVal) {
                index = i;
                break;
            }
        }

        // calculate the leftSize
        int leftSize = index - inStart;

        root->left = build(preorder, preStart + 1, preStart + leftSize, inorder, inStart, index - 1);
        root->right = build(preorder, preStart + leftSize + 1, preEnd, inorder, index + 1, inEnd);

        return root;
    }
};
```