https://course.acciojob.com/idle?question=fead3653-183f-4eac-94cc-fee94b777e16

MEDIUM

Max Score: 30 Points

Binary Tree from Inorder and Postorder

You are given two arrays of size `N` each. They represent the postorder and inorder traversals of a binary tree.

You need to construct the original binary tree with their help and return a pointer to the head of the tree.

NOTE: You need to complete the given function. The input and printing of output will be handled by the driver code.

Input Format

For each test case: You are given an integer `N`, denoting the size of the tree.

The second line contains `N` space-separated integers denoting the postorder traversal.

The third line contains `N` space-separated integers denoting the in-order traversal.

Output Format

For each test case, return the root of the binary tree. For reference, preorder traversal is shown in the output.

Example 1

Input

```
5

23 24 11 35 12 10

23 11 24 10 35 12
```

Output

```
10 11 23 24 12 35
```

Explanation

```
The tree looks like this:


    10

   /  \

  11   12

 / \   /

23 24 35
```

You can check the preorder from the given tree.

Example 2

Input

```
4

4 3 2 1

1 2 4 3
```

Output

```
1 2 3 4
```

Explanation

```
The tree looks like this:

1

 \

  2

   \

    3

   /

  4
```

You can check the preorder from the given tree.

Constraints

1 <= N <= 10000

1 <= pos[i] <= 100000

1 <= in[i] <= 100000

Topic Tags

Trees

My code

```
// in java
import java.util.*;

class Node {
    int data;
    Node left;
    Node right;

    Node(int value) {
        data = value;
        left = null;
        right = null;
    }
}

class Main {
    public void preOrder(Node root) {
        if (root == null) return;

        System.out.print(root.data + " ");
        preOrder(root.left);

        preOrder(root.right);
    }

    public static void main(String args[]) {
```

```
        Scanner sc = new Scanner(System.in);
        Main ip = new Main();
        int T = 1;
        while (T > 0) {
            int n = sc.nextInt();
            int[] inorder = new int[n];
            int[] postorder = new int[n];
            for (int i = 0; i < n; i++) postorder[i] = sc.nextInt();
            for (int i = 0; i < n; i++) inorder[i] = sc.nextInt();
            Solution g = new Solution();
            Node root = g.buildTree(inorder, postorder, n);
            ip.preOrder(root);
            System.out.println();

            T--;
        }
        sc.close();
    }
}

class Solution {
        private Node build(int[] postorder, int posIdx, int[] inorder, int
inStart, int inEnd){
        if(inStart > inEnd || posIdx < 0) return null;
        Node root = new Node(postorder[posIdx]);
        int i = 0;
        for(i = inStart; i <= inEnd; i ++){
            if(inorder[i] == postorder[posIdx]) break;
        }
```

```
        root.right = build(postorder, posIdx - 1, inorder, i + 1, inEnd);
        root.left = build(postorder, posIdx - 1 - (inEnd - i), inorder,
inStart, i - 1);
        return root;
    }
    Node buildTree(int in[], int post[], int n){
        //Write code here
                return build(post, post.length - 1, in, 0, in.length -1);
    }
}
```