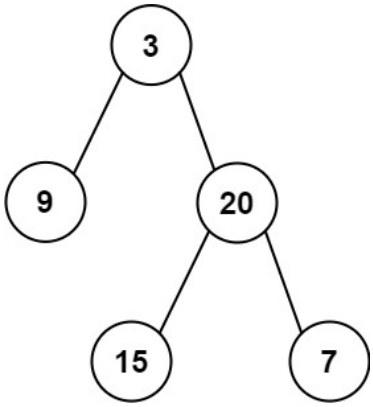
105. Construct Binary Tree from Preorder and Inorder Traversal

Created	@November 2, 2021 9:02 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/
Importance	
Tag	BST NEET Tree
Video	https://www.youtube.com/watch?v=Qc0XLGFnchU, https://www.youtube.com/watch?v=S1wNG5hx-30

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.

Solution

递归

思路

对于任意一颗树而言，前序遍历的形式总是

```
[ 根节点, [左子树的前序遍历结果], [右子树的前序遍历结果] ]
```

即根节点总是前序遍历中的第一个节点。而中序遍历的形式总是

```
[ [左子树的中序遍历结果], 根节点, [右子树的中序遍历结果] ]
```

只要我们在中序遍历中**定位**到根节点，那么我们就可以分别知道左子树和右子树中的节点数目。由于同一颗子树的前序遍历和中序遍历的长度显然是相同的，因此我们就可以对应到前序遍历的结果中，对上述形式中的所有**左右括号**进行定位。

这样以来，我们就知道了左子树的前序遍历和中序遍历结果，以及右子树的前序遍历和中序遍历结果，我们就可以递归地对构造出左子树和右子树，再将这两颗子树接到根节点的左右位置。

细节

在中序遍历中对根节点进行定位时，一种简单的方法是直接扫描整个中序遍历的结果并找出根节点，但这样做的时间复杂度较高。我们可以考虑使用哈希表来帮助我们快速地定位根节点。对于哈希映射中的每个键值对，键表示一个元素（节点的值），值表示其在中序遍历中的出现位置。在构造二叉树的过程之前，我们可以对中序遍历的列表进行一遍扫描，就可以构造出这个哈希映射。在此后构造二叉树的过程中，我们就只需要 $O(1)$ 的时间对根节点进行定位了。

下面的代码给出了详细的注释。

```
# 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 buildTree(self, preorder: List[int], inorder: List[int]) -> Optional[TreeNode]:
        if not preorder or not inorder:
            return None

        root_value = preorder[0]
        root = TreeNode(root_value)
        inorder_index = inorder.index(root_value)

        root.left = self.buildTree(preorder[1:inorder_index+1], inorder[:inorder_index])
```

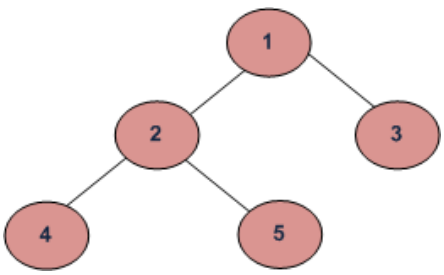
```
        root.right = self.buildTree(preorder[inorder_index+1:], inorder[inorder_index+1:])

        return root
```

```
/**
 * 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 buildTree(int[] preorder, int[] inorder) {
        if (preorder == null || inorder == null || preorder.length == 0 || preorder.length != inorder.length) return null;
        return buildTreeHelper(preorder, inorder, 0, 0, preorder.length-1);
    }

    public TreeNode buildTreeHelper(int[] preorder, int[] inorder, int pre_st, int in_st, int in_end) {
        if (pre_st > preorder.length || in_st > in_end) return null;
        TreeNode current = new TreeNode(preorder[pre_st]);
        int idx = in_st;
        while (idx < in_end) {
            if (inorder[idx] == preorder[pre_st]) break;
            idx++;
        }
        current.left = buildTreeHelper(preorder, inorder, pre_st+1, in_st, idx-1);
        current.right = buildTreeHelper(preorder, inorder, pre_st+(idx-in_st)+1, idx+1, in_end);

        return current;
    }
}
```

Tree Traversals (Inorder, Preorder and Postorder)

- Difficulty Level : **Easy**
- Last Updated : 21 Oct, 2021

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. Following are the generally used ways for traversing trees.



Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer **Complete Interview Preparation Course.**

In case you wish to attend **live classes** with experts, please refer **DSA Live Classes for Working Professionals** and **Competitive Programming Live for Students**.

Depth First Traversals:

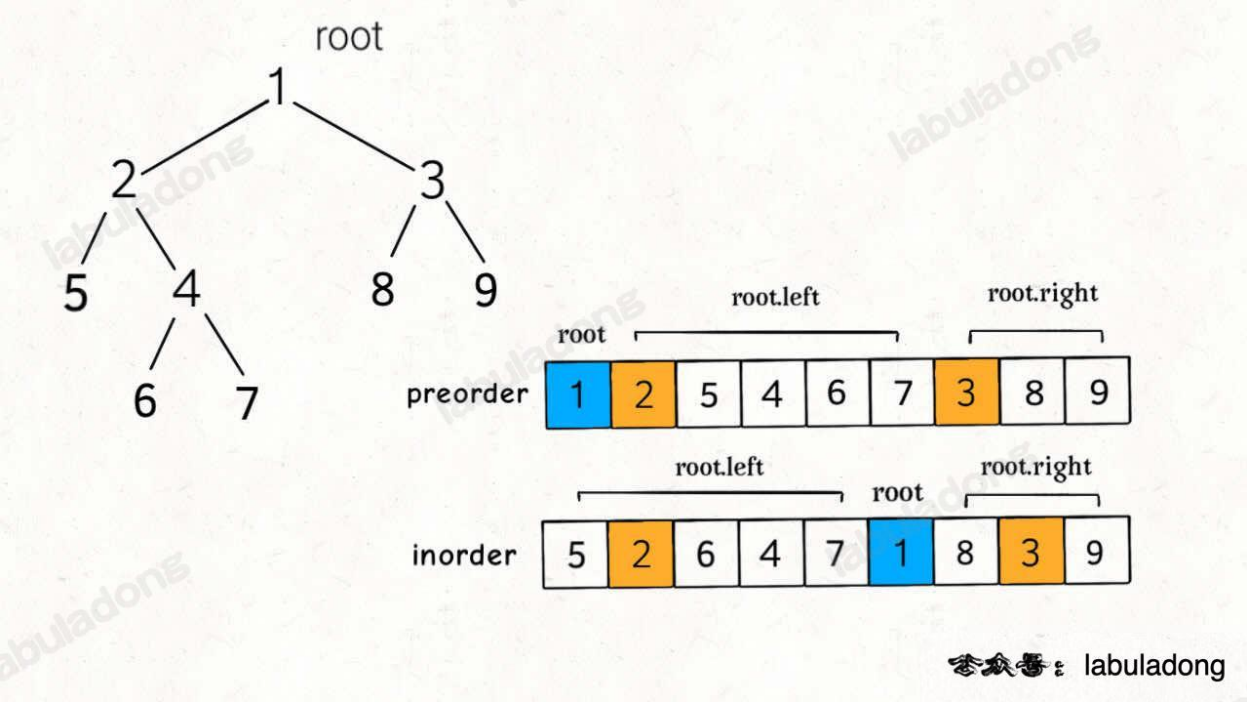
(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

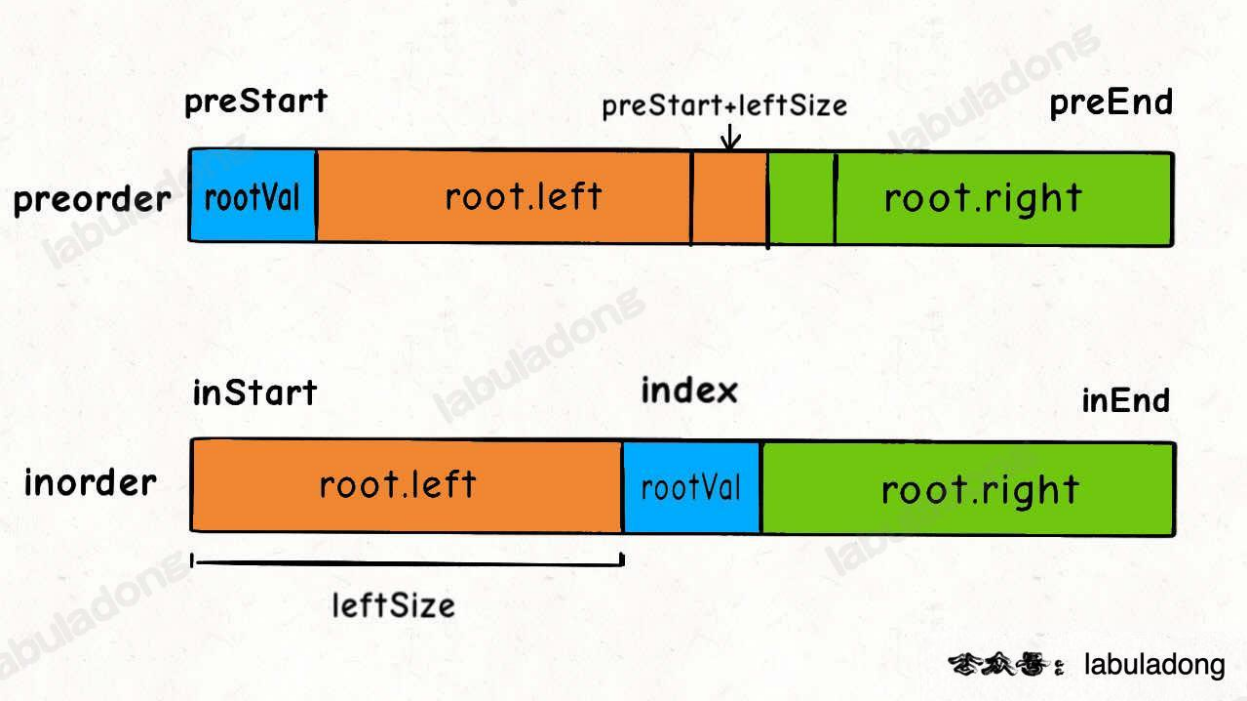
(c) Postorder (Left, Right, Root) : 4 5 2 3 1

(d) Breadth-First or Level Order Traversal: 1 2 3 4 5

二叉树的前序和中序遍历结果的特点如下：



前序遍历结果第一个就是根节点的值，然后再根据中序遍历结果确定左右子树的节点。




```
class Solution {
    // 存储 inorder 中值到索引的映射
    HashMap<Integer, Integer> valToIndex = new HashMap<>();

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        for (int i = 0; i < inorder.length; i++) {
            valToIndex.put(inorder[i], i);
        }
        return build(preorder, 0, preorder.length - 1,
                     inorder, 0, inorder.length - 1);
    }

    /*
       定义：前序遍历数组为 preorder[preStart..preEnd]，
       中序遍历数组为 inorder[inStart..inEnd]，
       构造这个二叉树并返回该二叉树的根节点
    */
    TreeNode build(int[] preorder, int preStart, int preEnd,
                   int[] inorder, int inStart, int inEnd) {
        if (preStart > preEnd) {
            return null;
        }

        // root 节点对应的值就是前序遍历数组的第一个元素
        int rootVal = preorder[preStart];
        // rootVal 在中序遍历数组中的索引
        int index = valToIndex.get(rootVal);

        int leftSize = index - inStart;

        // 先构造出当前根节点
        TreeNode root = new TreeNode(rootVal);
        // 递归构造左右子树
        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;
```

```
    }
}
// 详细解析参见：
// https://labuladong.github.io/article/?qno=105
```