

东哥手把手帮你刷通二叉树|第二期

Original labuladong labuladong 9/22

收录于话题

#手撕力扣高频面试题 58 #框架思维系列 20

学算法认准 labuladong

东哥带你手把手撕力扣 😊

读完本文，你能去力扣解决如下题目：

654.最大二叉树（难度 **Medium**）

105.从前序与中序遍历序列构造二叉树（难度 **Medium**）

106.从中序与后序遍历序列构造二叉树（难度 **Medium**）

上篇文章 [手把手教你刷二叉树（第一篇）](#) 连刷了三道二叉树题目，很多读者直呼内行。其实二叉树相关的算法真的不难，本文再来三道，手把手带你看看树的算法到底怎么做。

先来复习一下，我们说过写树的算法，关键思路如下：

把题目的要求细化，搞清楚根节点应该做什么，然后剩下的事情抛给前/中/后序的遍历框架就行了，我们千万不要跳进递归的细节里，你的脑袋才能压几个栈呀。

也许你还不太理解这句话，我们下面来看例子。

构造最大二叉树

先来道简单的，这是力扣第 654 题，题目如下：

654. 最大二叉树

难度 中等

👍 190



给定一个不含重复元素的整数数组。一个以此数组构建的最大二叉树定义如下：

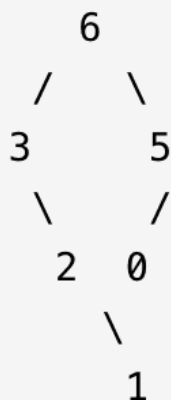
1. 二叉树的根是数组中的最大元素。
2. 左子树是通过数组中最大值左边部分构造出的最大二叉树。
3. 右子树是通过数组中最大值右边部分构造出的最大二叉树。

通过给定的数组构建最大二叉树，并且输出这个树的根节点。

示例：

输入：[3,2,1,6,0,5]

输出：返回下面这棵树的根节点：



函数签名如下：

```
TreeNode constructMaximumBinaryTree(int[] nums);
```

按照我们刚才说的，先明确根节点做什么？对于构造二叉树的问题，根节点要做的就是想办法把自己构造出来。

我们肯定要遍历数组把找到最大值 `maxVal`，把根节点 `root` 做出来，然后对 `maxVal` 左边的数组和右边的数组进行递归调用，作为 `root` 的左右子树。

按照题目给出的例子，输入的数组为 `[3,2,1,6,0,5]`，对于整棵树的根节点来说，其实在做这件事：

```
TreeNode constructMaximumBinaryTree([3,2,1,6,0,5]) {
    // 找到数组中的最大值
    TreeNode root = new TreeNode(6);
    // 递归调用构造左右子树
    root.left = constructMaximumBinaryTree([3,2,1]);
    root.right = constructMaximumBinaryTree([0,5]);
    return root;
}
```

再详细一点，就是如下伪码：

```
TreeNode constructMaximumBinaryTree(int[] nums) {
    if (nums is empty) return null;
    // 找到数组中的最大值
    int maxVal = Integer.MIN_VALUE;
    int index = 0;
    for (int i = 0; i < nums.length; i++) {
        if (nums[i] > maxVal) {
            maxVal = nums[i];
            index = i;
        }
    }

    TreeNode root = new TreeNode(maxVal);
    // 递归调用构造左右子树
    root.left = constructMaximumBinaryTree(nums[0..index-1]);
    root.right = constructMaximumBinaryTree(nums[index+1..nums.length-1]);
    return root;
}
```

看懂了吗？对于每个根节点，只需要找到当前 `nums` 中的最大值和对应的索引，然后递归调用左右数组构造左右子树即可。

明确了思路，我们可以重新写一个辅助函数 `build`，来控制 `nums` 的索引：

```
/* 主函数 */
TreeNode constructMaximumBinaryTree(int[] nums) {
    return build(nums, 0, nums.length - 1);
}

/* 将 nums[lo..hi] 构造成符合条件的树，返回根节点 */
TreeNode build(int[] nums, int lo, int hi) {
    // base case
```

```
    if (lo > hi) {
        return null;
    }

    // 找到数组中的最大值和对应的索引
    int index = -1, maxVal = Integer.MIN_VALUE;
    for (int i = lo; i <= hi; i++) {
        if (maxVal < nums[i]) {
            index = i;
            maxVal = nums[i];
        }
    }

    TreeNode root = new TreeNode(maxVal);
    // 递归调用构造左右子树
    root.left = build(nums, lo, index - 1);
    root.right = build(nums, index + 1, hi);

    return root;
}
```

至此，这道题就做完了，还是挺简单的对吧，下面看两道更困难一些的。

通过前序和中序遍历结果构造二叉树

经典问题了，面试/笔试中常考，力扣第 105 题就是这个问题：

105. 从前序与中序遍历序列构造二叉树

难度 中等

👍 679



根据一棵树的前序遍历与中序遍历构造二叉树。

注意：

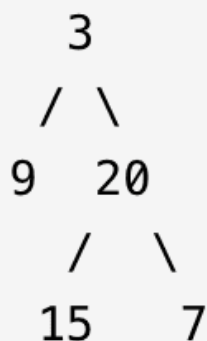
你可以假设树中没有重复的元素。

例如，给出

前序遍历 preorder = [3,9,20,15,7]

中序遍历 inorder = [9,3,15,20,7]

返回如下的二叉树：



函数签名如下：

```
TreeNode buildTree(int[] preorder, int[] inorder);
```

废话不多说，直接来想思路，首先思考，根节点应该做什么。

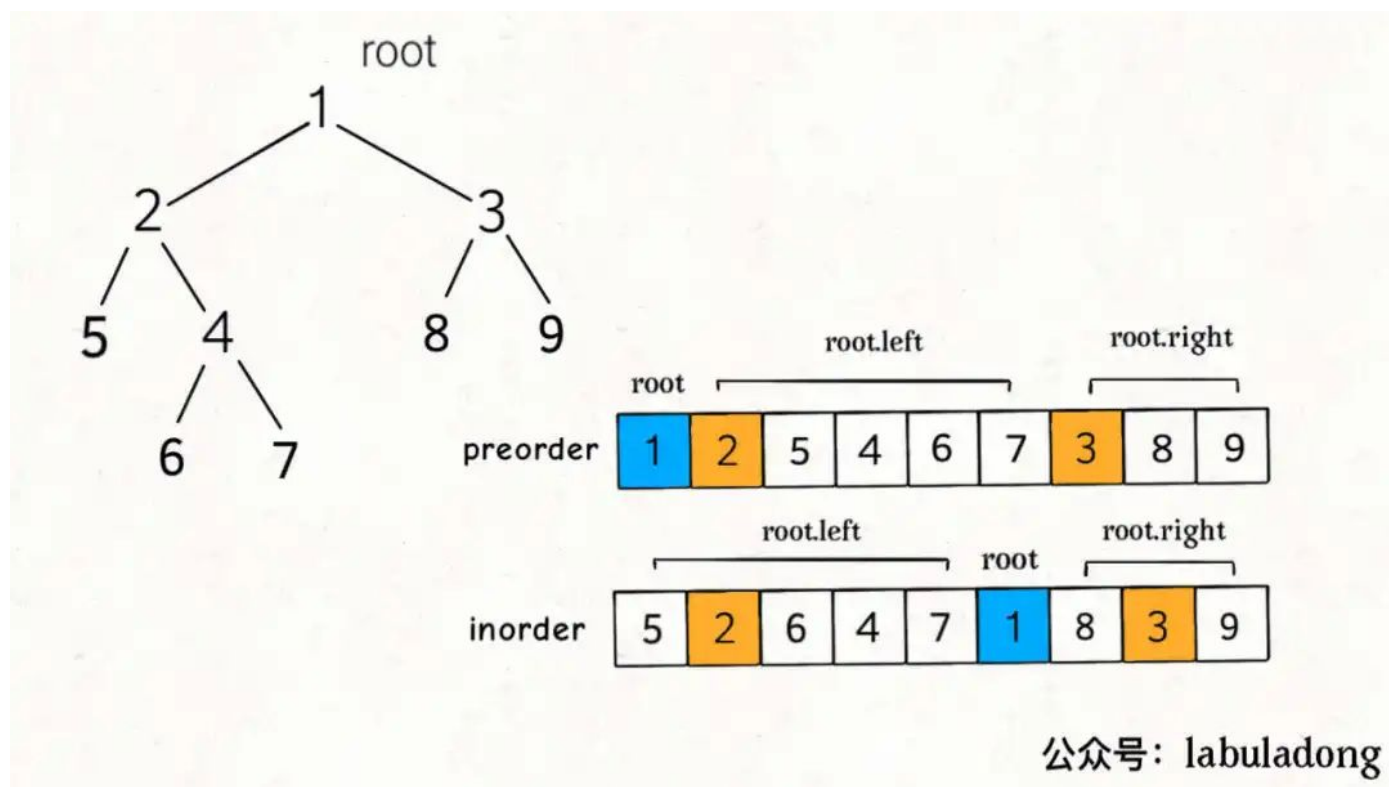
类似上一题，我们肯定要想办法确定根节点的值，把根节点做出来，然后递归构造左右子树即可。

我们先来回顾一下，前序遍历和中序遍历的结果有什么特点？

```
void traverse(TreeNode root) {
    // 前序遍历
    preorder.add(root.val);
    traverse(root.left);
    traverse(root.right);
}

void traverse(TreeNode root) {
    traverse(root.left);
    // 中序遍历
    inorder.add(root.val);
    traverse(root.right);
}
```

前文 [二叉树就那几个框架](#) 写过，这样的遍历顺序差异，导致了 `preorder` 和 `inorder` 数组中的元素分布有如下特点：



找到根节点是很简单的，前序遍历的第一个值 `preorder[0]` 就是根节点的值，关键在于如何通过根节点的值，将 `preorder` 和 `postorder` 数组划分成两半，构造根节点的左右子树？

换句话说，对于以下代码中的 `?` 部分应该填入什么：

```

/* 主函数 */
TreeNode buildTree(int[] preorder, int[] inorder) {
    return build(preorder, 0, preorder.length - 1,
                inorder, 0, inorder.length - 1);
}

/*
    若前序遍历数组为 preorder[preStart..preEnd],
    后续遍历数组为 postorder[postStart..postEnd],
    构造二叉树, 返回该二叉树的根节点
*/
TreeNode build(int[] preorder, int preStart, int preEnd,
               int[] inorder, int inStart, int inEnd) {
    // 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 = 0;
    for (int i = inStart; i <= inEnd; i++) {
        if (inorder[i] == rootVal) {
            index = i;
            break;
        }
    }

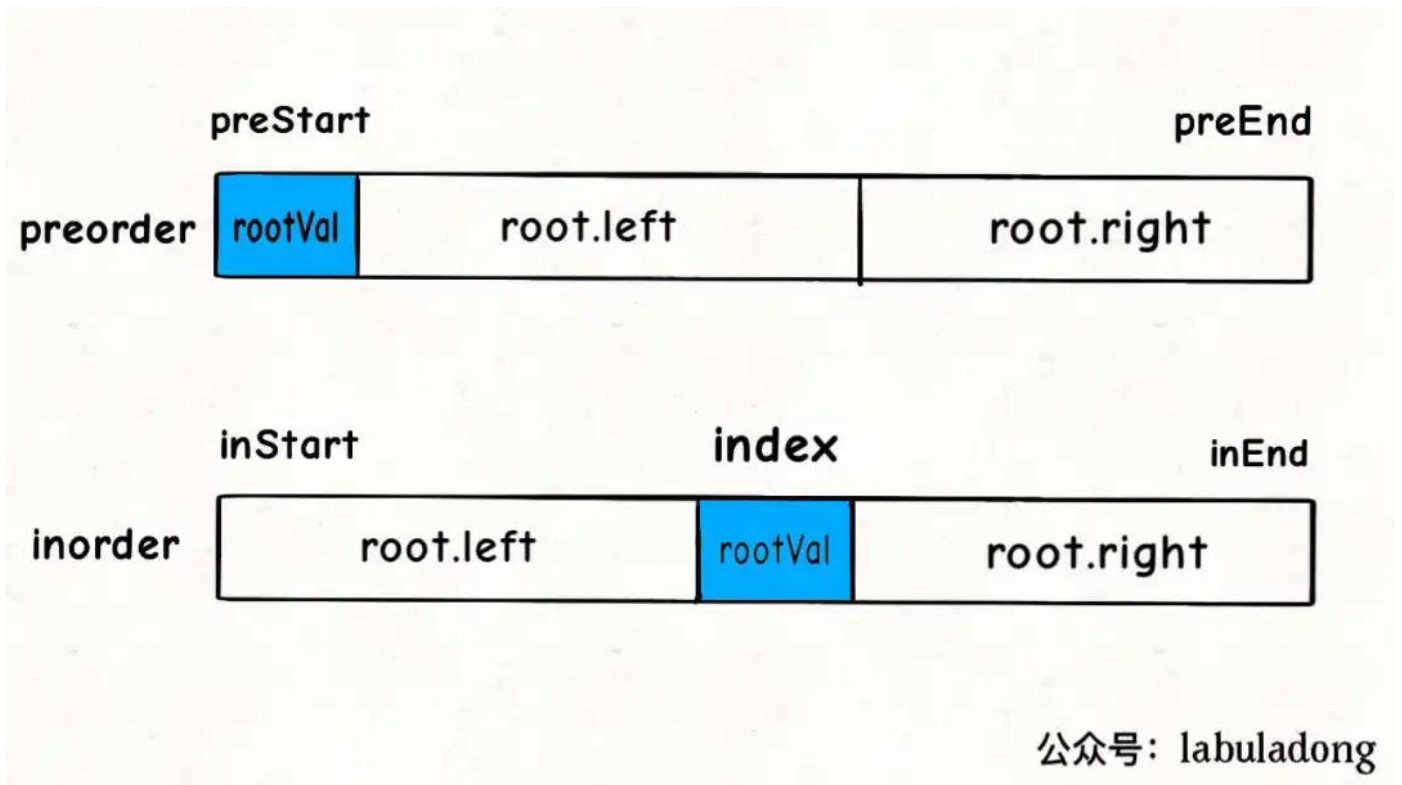
    TreeNode root = new TreeNode(rootVal);
    // 递归构造左右子树
    root.left = build(preorder, ?, ?,
                     inorder, ?, ?);

    root.right = build(preorder, ?, ?,
                      inorder, ?, ?);

    return root;
}

```

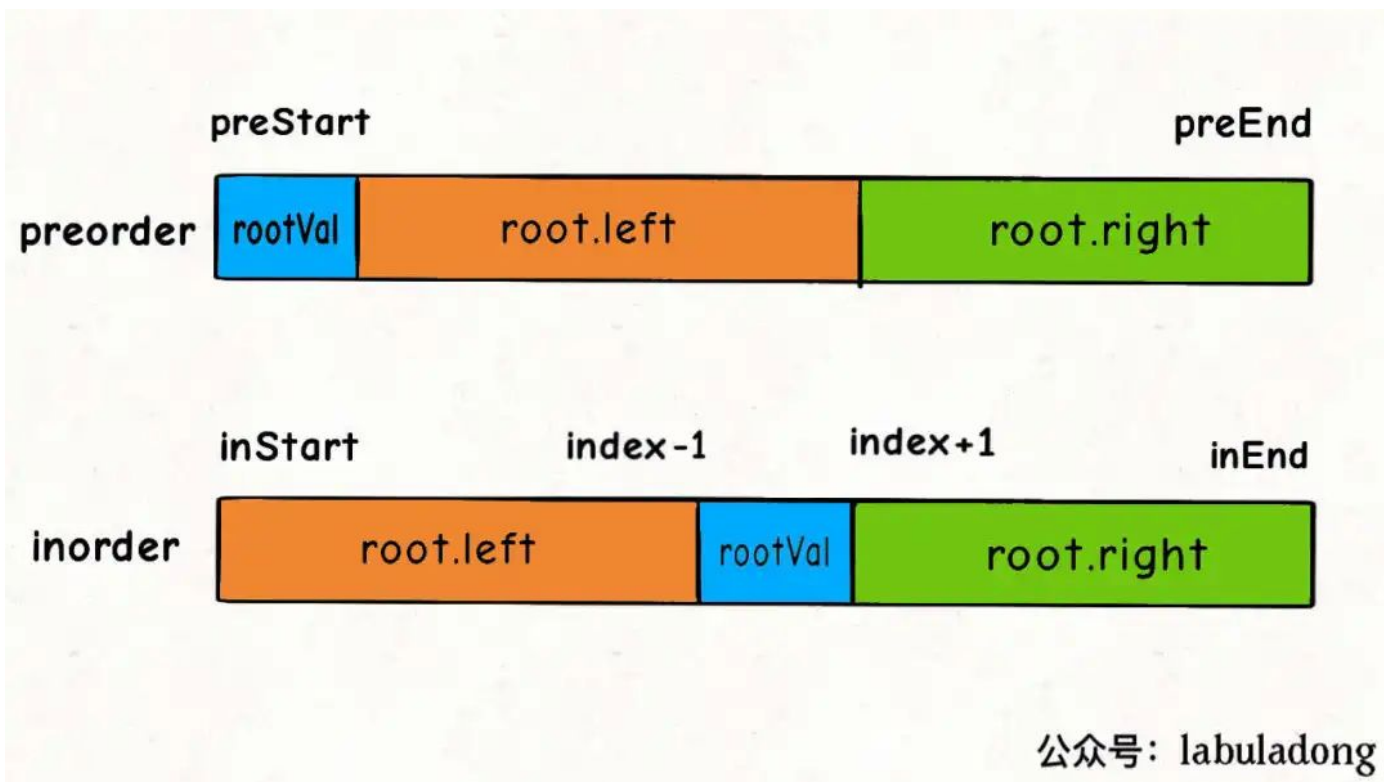
对于代码中的 `rootVal` 和 `index` 变量, 就是下图这种情况:



现在我们来看图做填空题，下面这几个问号处应该填什么：

```
root.left = build(preorder, ?, ?,  
                  inorder, ?, ?);  
  
root.right = build(preorder, ?, ?,  
                   inorder, ?, ?);
```

对于左右子树对应的 **inorder** 数组的起始索引和终止索引比较容易确定：

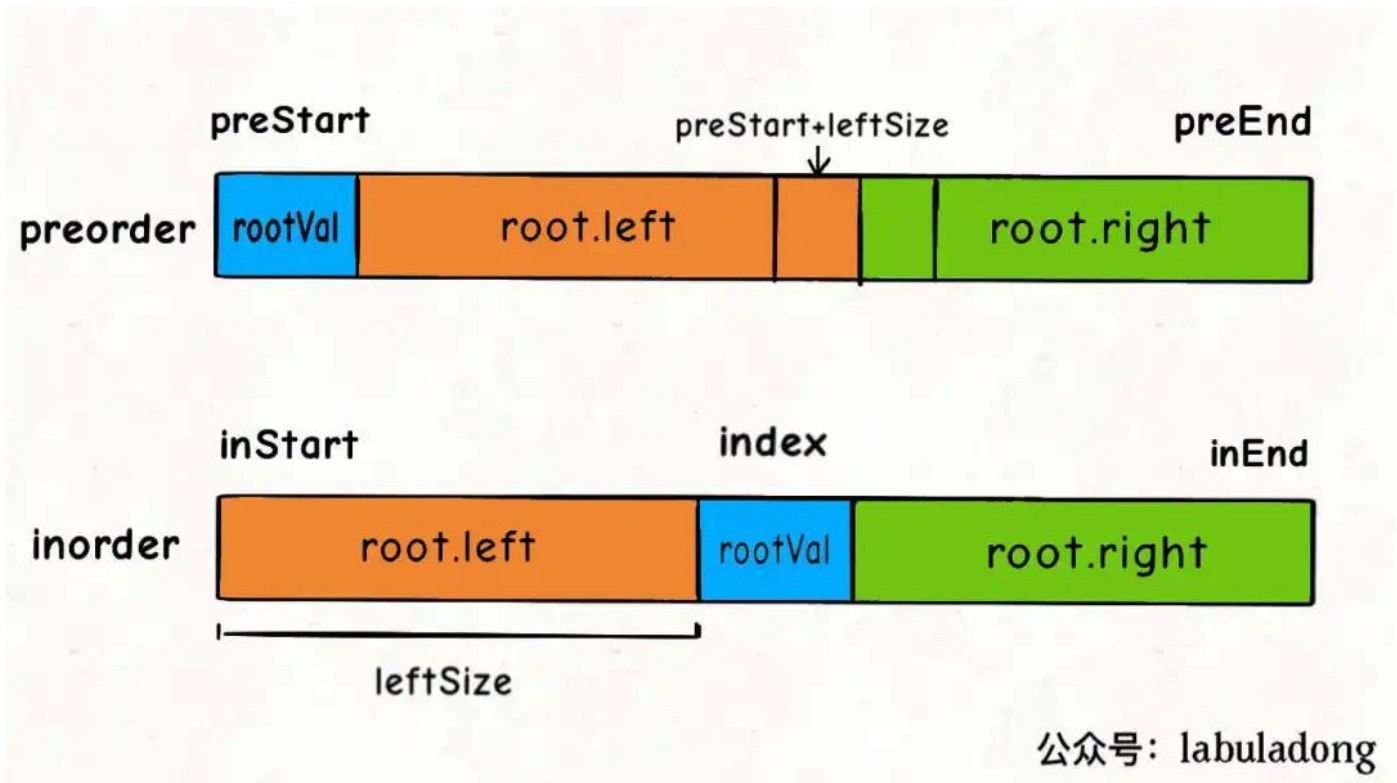


```
root.left = build(preorder, ?, ?,  
                  inorder, inStart, index - 1);
```

```
root.right = build(preorder, ?, ?,  
                   inorder, index + 1, inEnd);
```

对于 `preorder` 数组呢？如何确定左右数组对应的起始索引和终止索引？

这个可以通过左子树的节点数推导出来，假设左子树的节点数为 `leftSize`，那么 `preorder` 数组上的索引情况是这样的：



看着这个图就可以把 **preorder** 对应的索引写进去了：

```
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);
```

至此，整个算法思路就完成了，我们再补一补 base case 即可写出解法代码：

```
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 = 0;
    for (int i = inStart; i <= inEnd; i++) {
        if (inorder[i] == rootVal) {
            index = i;
            break;
        }
    }
}
```

```
    }  
}  
  
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;  
}
```

我们的主函数只要调用 **build** 函数即可，你看着函数这么多参数，解法这么多代码，似乎比我们上面讲的那道题难很多，让人望而生畏，实际上呢，这些参数无非就是控制数组起止位置的，画个图就能解决了。

通过后序和中序遍历结果构造二叉树

类似上一题，这次我们利用**后序**和**中序**遍历的结果数组来还原二叉树，这是力扣第106题：

106. 从中序与后序遍历序列构造二叉树

难度 中等

👍 296

☆

📄

文A

🔔

根据一棵树的中序遍历与后序遍历构造二叉树。

注意：

你可以假设树中没有重复的元素。

例如，给出

```
中序遍历 inorder = [9,3,15,20,7]
后序遍历 postorder = [9,15,7,20,3]
```

返回如下的二叉树：

```
      3
     / \
    9  20
     / \
    15  7
```

函数签名如下：

```
TreeNode buildTree(int[] inorder, int[] postorder);
```

类似的，看下后序和中序遍历的特点：

```
void traverse(TreeNode root) {
    traverse(root.left);
    traverse(root.right);
    // 前序遍历
}
```

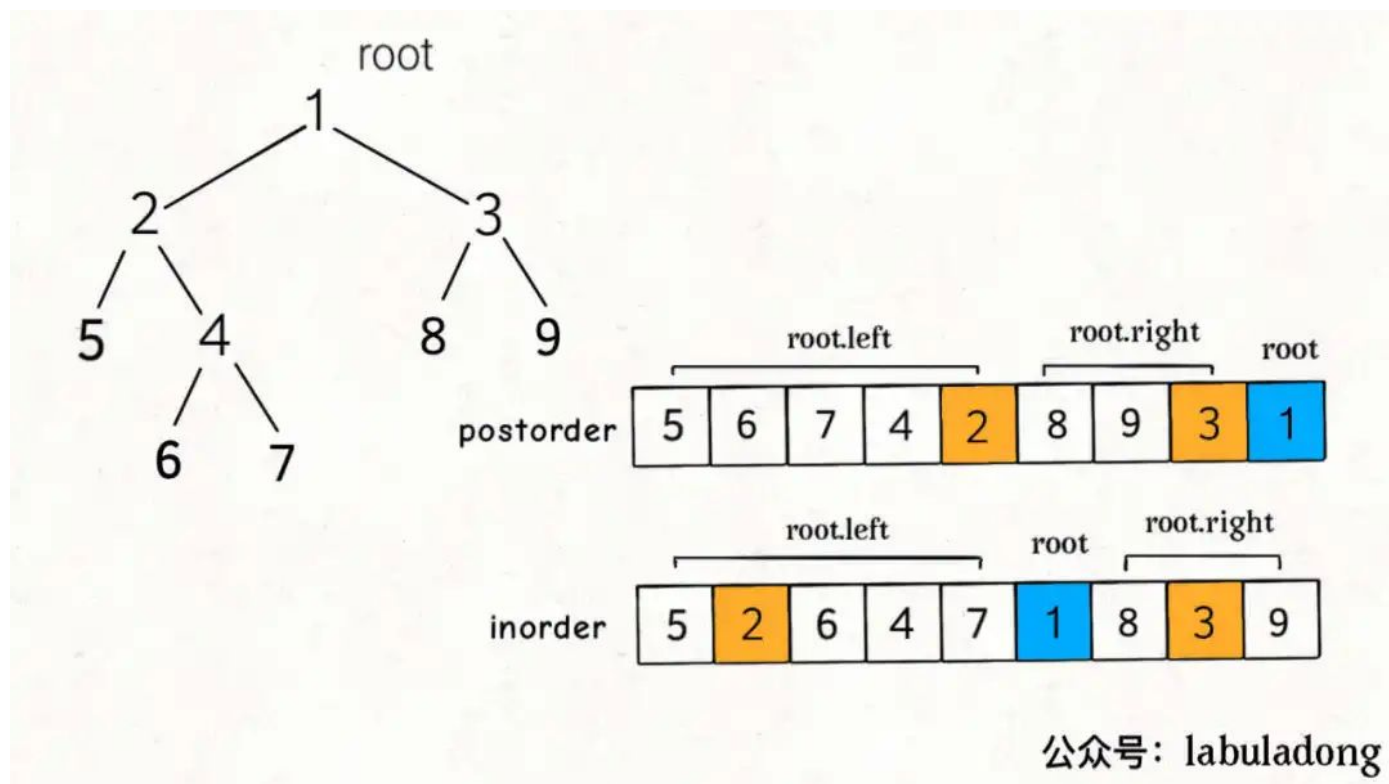
```

        postorder.add(root.val);
    }

    void traverse(TreeNode root) {
        traverse(root.left);
        // 中序遍历
        inorder.add(root.val);
        traverse(root.right);
    }

```

这样的遍历顺序差异，导致了 `preorder` 和 `inorder` 数组中的元素分布有如下特点：



这道题和上一题的关键区别是，后序遍历和前序遍历相反，根节点对应的值为 `postorder` 的最后一个元素。

整体的算法框架和上一题非常类似，我们依然写一个辅助函数 `build`：

```

TreeNode buildTree(int[] inorder, int[] postorder) {
    return build(inorder, 0, inorder.length - 1,
                postorder, 0, postorder.length - 1);
}

TreeNode build(int[] inorder, int inStart, int inEnd,
               int[] postorder, int postStart, int postEnd) {
    // root 节点对应的值就是后序遍历数组的最后一个元素

```

```

    int rootVal = postorder[postEnd];
    // rootVal 在中序遍历数组中的索引
    int index = 0;
    for (int i = inStart; i <= inEnd; i++) {
        if (inorder[i] == rootVal) {
            index = i;
            break;
        }
    }

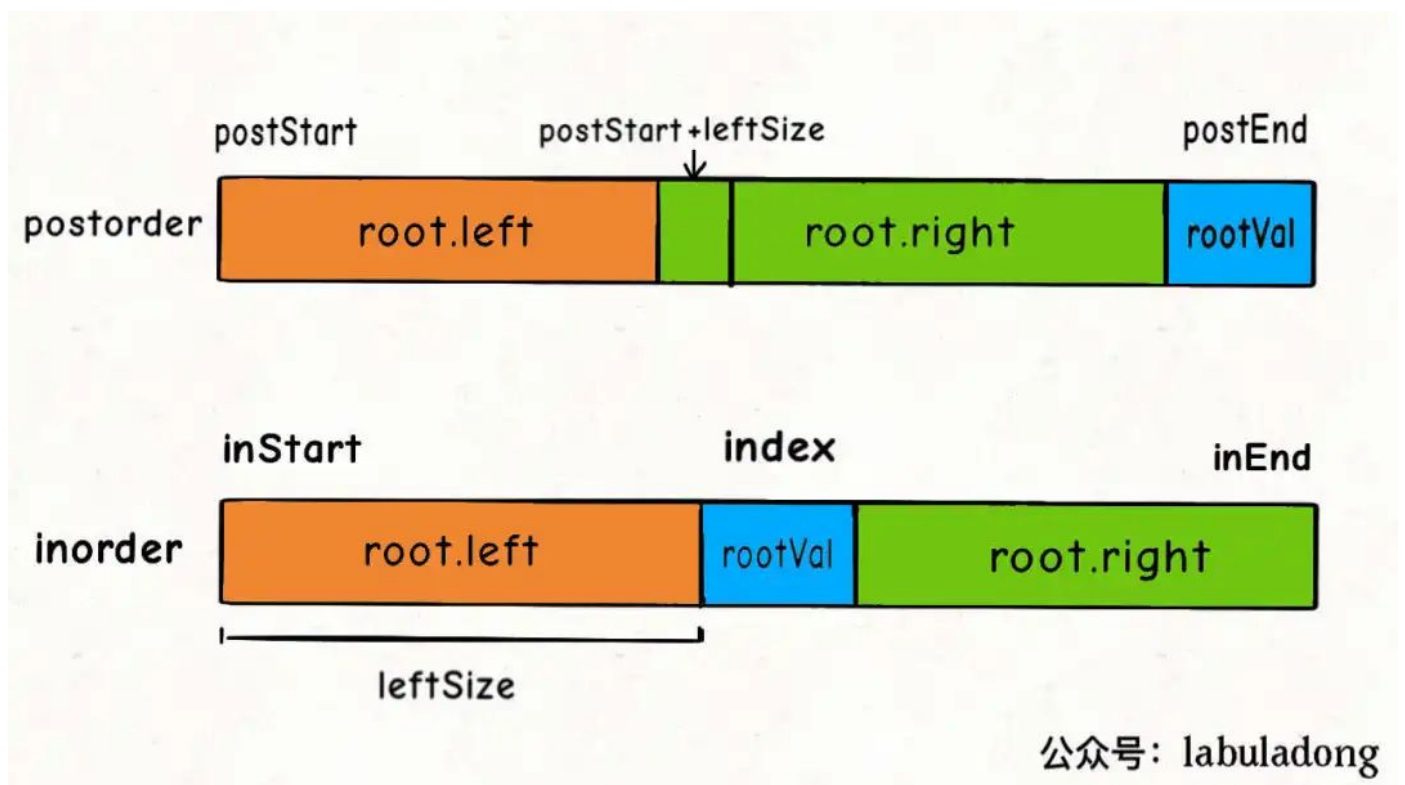
    TreeNode root = new TreeNode(rootVal);
    // 递归构造左右子树
    root.left = build(preorder, ?, ?,
                     inorder, ?, ?);

    root.right = build(preorder, ?, ?,
                      inorder, ?, ?);

    return root;
}

```

现在 `postoder` 和 `inorder` 对应的状态如下：



我们可以按照上图将问号处的索引正确填入：

```

int leftSize = index - inStart;

root.left = build(inorder, inStart, index - 1,
                  postorder, postStart, postStart + leftSize - 1);

```

```
root.right = build(inorder, index + 1, inEnd,
                  postorder, postStart + leftSize, postEnd - 1);
```

综上，可以写出完整的解法代码：

```
TreeNode build(int[] inorder, int inStart, int inEnd,
               int[] postorder, int postStart, int postEnd) {

    if (inStart > inEnd) {
        return null;
    }
    // root 节点对应的值就是后序遍历数组的最后一个元素
    int rootVal = postorder[postEnd];
    // rootVal 在中序遍历数组中的索引
    int index = 0;
    for (int i = inStart; i <= inEnd; i++) {
        if (inorder[i] == rootVal) {
            index = i;
            break;
        }
    }
    // 左子树的节点个数
    int leftSize = index - inStart;
    TreeNode root = new TreeNode(rootVal);
    // 递归构造左右子树
    root.left = build(inorder, inStart, index - 1,
                     postorder, postStart, postStart + leftSize - 1);


    root.right = build(inorder, index + 1, inEnd,
                      postorder, postStart + leftSize, postEnd - 1);

    return root;
}
```

有了前一题的铺垫，这道题很快就解决了，无非就是 `rootVal` 变成了最后一个元素，再改改递归函数的参数而已，只要明白二叉树的特性，也不难写出来。

最后呼应下前文，做二叉树的问题，关键是把题目的要求细化，搞清楚根节点应该做什么，然后剩下的事情抛给前/中/后序的遍历框架就行了。

现在你是否明白其中的玄妙了呢？

往期回顾 

状态压缩技巧：动态规划的降维打击

我作了首诗，保你闭着眼睛也能写对二分查找

我写了套框架，把滑动窗口算法变成了默写题

BFS 算法框架套路详解

东哥手把手带你套框架刷通二叉树第一期

学好算法全靠套路，认准 labuladong，知乎、B站账号同名。

《labuladong的算法小抄》即将出版，公众号后台回复关键词「pdf」下载，回复「进群」可加入刷题群。



微信搜一搜

Q labuladong

收录于话题 #手撕力扣高频面试题

58个

上一篇

下一篇