

12.3 构建二叉树问题

? Question

给定一棵二叉树的前序遍历 `preorder` 和中序遍历 `inorder`，请从中构建二叉树，返回二叉树的根节点。假设二叉树中没有值重复的节点（如图 12-5 所示）。

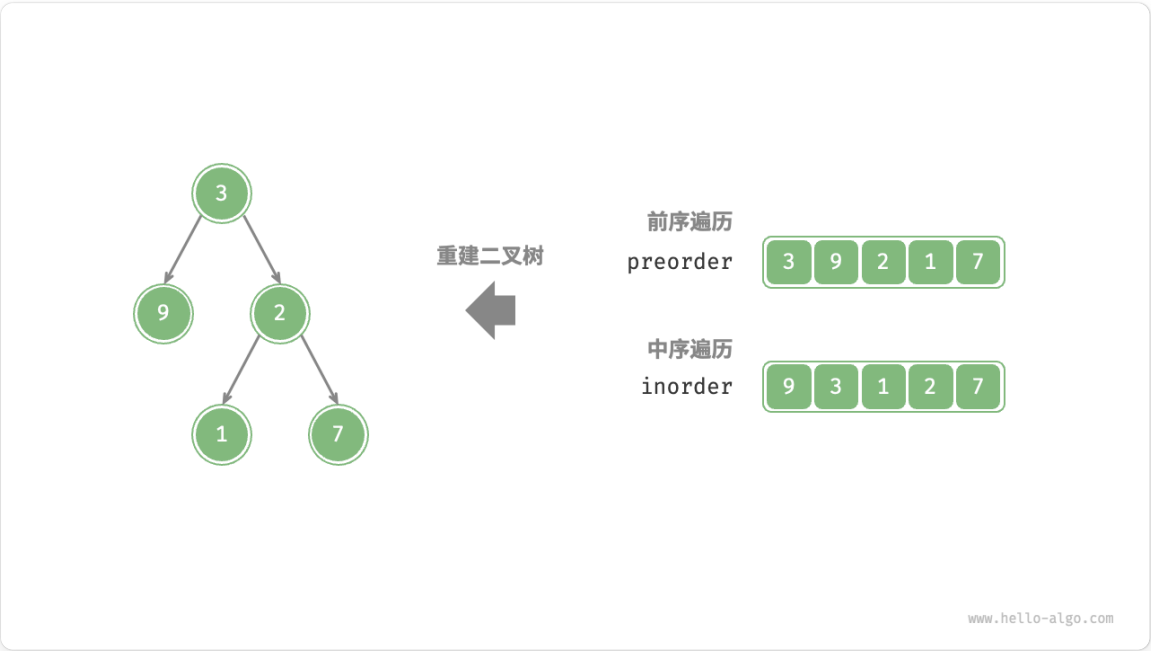


图 12-5 构建二叉树的示例数据

1. 判断是否为分治问题

原问题定义为从 `preorder` 和 `inorder` 构建二叉树，是一个典型的分治问题。

- **问题可以分解：**从分治的角度切入，我们可以将原问题划分为两个子问题：构建左子树、构建右子树，加上一步操作：初始化根节点。而对于每棵子树（子问题），我们仍然可以复用以上划分方法，将其划分为更小的子树（子问题），直至达到最小子问题（空子树）时终止。
- **子问题是独立的：**左子树和右子树是相互独立的，它们之间没有交集。在构建左子树时，我们只需关注中序遍历和前序遍历中与左子树对应的部分。右子树同理。
- **子问题的解可以合并：**一旦得到了左子树和右子树（子问题的解），我们就可以将它们链接到根节点上，得到原问题的解。

2. 如何划分子树

根据以上分析，这道题可以使用分治来求解，但如何通过前序遍历 preorder 和中序遍历 inorder 来划分左子树和右子树呢？

根据定义，preorder 和 inorder 都可以划分为三个部分。

- 前序遍历：[根节点 | 左子树 | 右子树]，例如图 12-5 的树对应 [3 | 9 | 2 1 7]。
- 中序遍历：[左子树 | 根节点 | 右子树]，例如图 12-5 的树对应 [9 | 3 | 1 2 7]。

以上图数据为例，我们可以通过图 12-6 所示的步骤得到划分结果。

- 前序遍历的首元素 3 是根节点的值。
- 查找根节点 3 在 inorder 中的索引，利用该索引可将 inorder 划分为 [9 | 3 | 1 2 7]。
- 根据 inorder 的划分结果，易得左子树和右子树的节点数量分别为 1 和 3，从而可将 preorder 划分为 [3 | 9 | 2 1 7]。

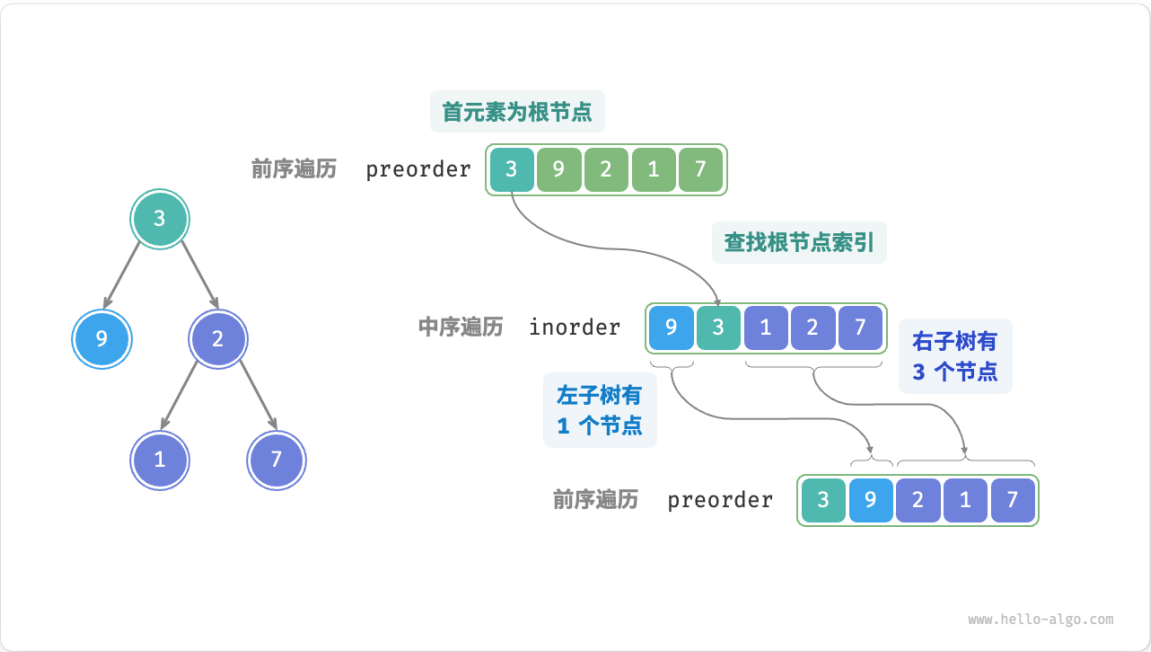


图 12-6 在前序遍历和中序遍历中划分子树

3. 基于变量描述子树区间

根据以上划分方法，我们已经得到根节点、左子树、右子树在 preorder 和 inorder 中的索引区间。而为了描述这些索引区间，我们需要借助几个指针变量。

- 将当前树的根节点在 `preorder` 中的索引记为 i 。
- 将当前树的根节点在 `inorder` 中的索引记为 m 。
- 将当前树在 `inorder` 中的索引区间记为 $[l, r]$ 。

如表 12-1 所示，通过以上变量即可表示根节点在 `preorder` 中的索引，以及子树在 `inorder` 中的索引区间。

表 12-1 根节点和子树在前序遍历和中序遍历中的索引

	根节点在 <code>preorder</code> 中的索引	子树在 <code>inorder</code> 中的索引区间
当前树	i	$[l, r]$
左子树	$i + 1$	$[l, m - 1]$
右子树	$i + 1 + (m - l)$	$[m + 1, r]$

请注意，右子树根节点索引中的 $(m - l)$ 的含义是“左子树的节点数量”，建议结合图 12-7 理解。

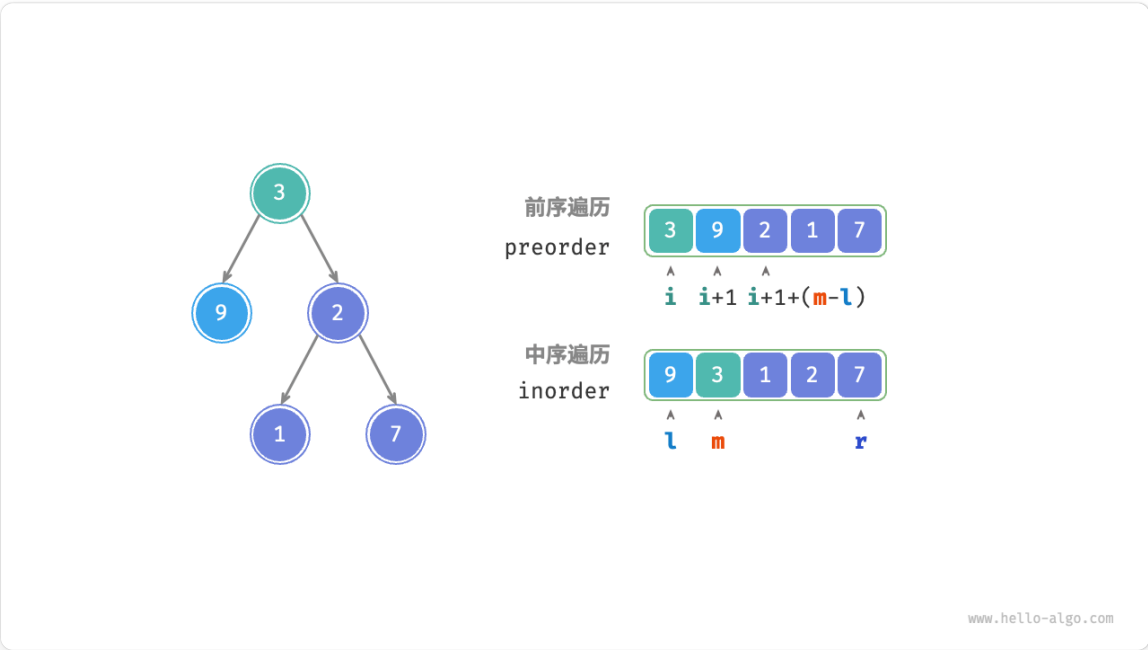


图 12-7 根节点和左右子树的索引区间表示

4. 代码实现

为了提升查询 m 的效率，我们借助一个哈希表 `hmap` 来存储数组 `inorder` 中元素到索引的映射：

Python

build_tree.py

```
def dfs(
    preorder: list[int],
    inorder_map: dict[int, int],
    i: int,
    l: int,
    r: int,
) -> TreeNode | None:
    """构建二叉树：分治"""
    # 子树区间为空时终止
    if r - l < 0:
        return None
    # 初始化根节点
    root = TreeNode(preorder[i])
    # 查询 m，从而划分左右子树
    m = inorder_map[preorder[i]]
    # 子问题：构建左子树
    root.left = dfs(preorder, inorder_map, i + 1, l, m - 1)
    # 子问题：构建右子树
    root.right = dfs(preorder, inorder_map, i + 1 + m - l, m + 1, r)
    # 返回根节点
    return root

def build_tree(preorder: list[int], inorder: list[int]) -> TreeNode | None:
    """构建二叉树"""
    # 初始化哈希表，存储 inorder 元素到索引的映射
    inorder_map = {val: i for i, val in enumerate(inorder)}
    root = dfs(preorder, inorder_map, 0, 0, len(inorder) - 1)
    return root
```

图 12-8 展示了构建二叉树的递归过程，各个节点是在向下“递”的过程中建立的，而各条边（引用）是在向上“归”的过程中建立的。

<1>

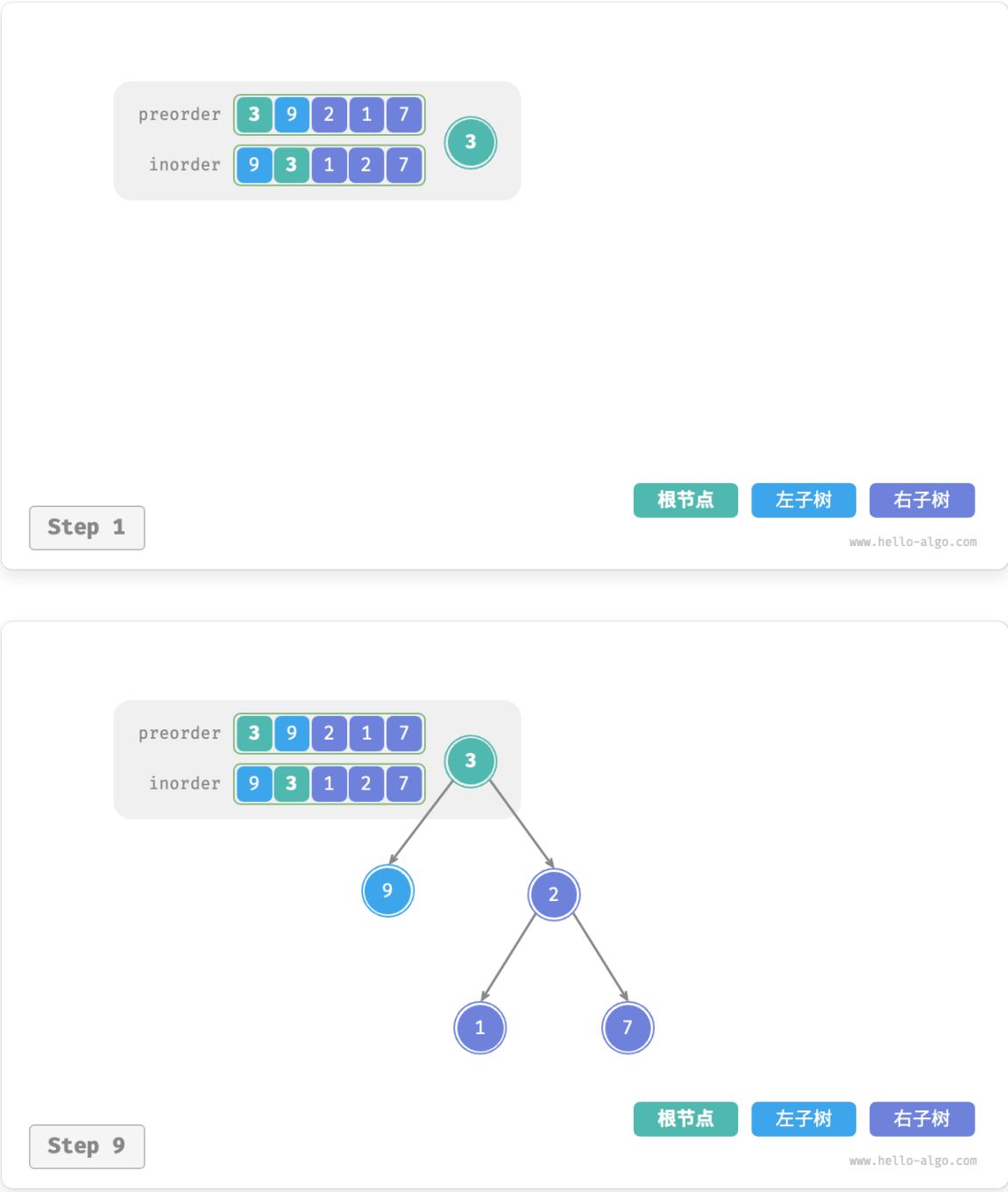


图 12-8 构建二叉树的递归过程

每个递归函数内的前序遍历 preorder 和中序遍历 inorder 的划分结果如图 12-9 所示。

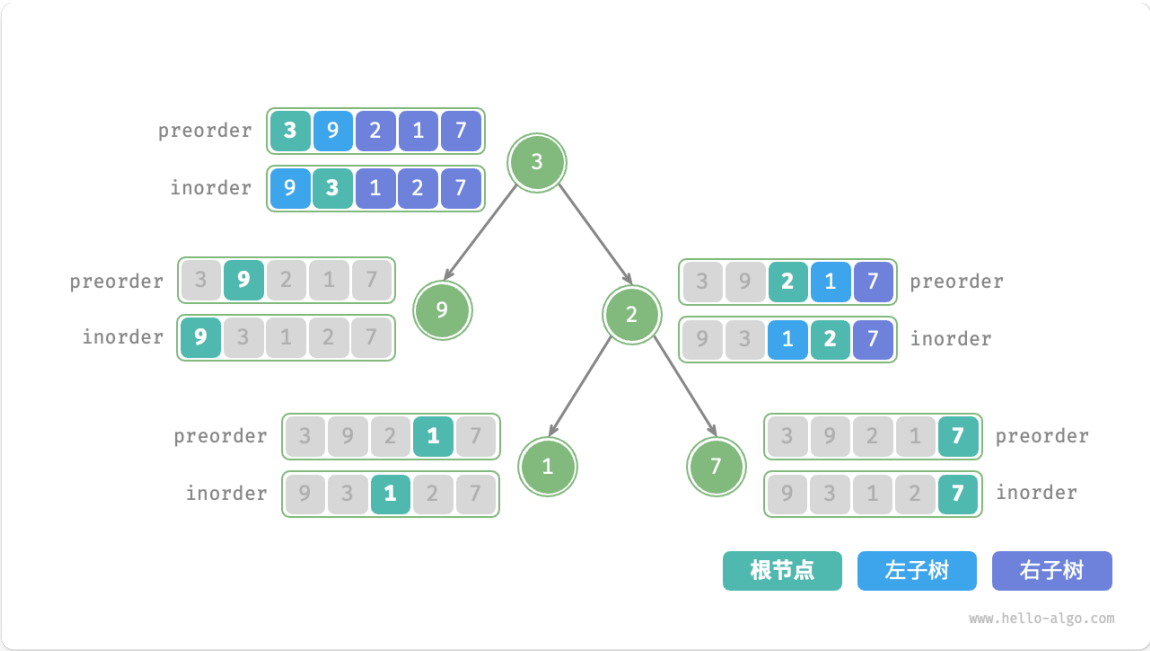


图 12-9 每个递归函数中的划分结果

设树的节点数量为 n ，初始化每一个节点（执行一个递归函数 `dfs()`）使用 $O(1)$ 时间。因此总体时间复杂度为 $O(n)$ 。

哈希表存储 `inorder` 元素到索引的映射，空间复杂度为 $O(n)$ 。在最差情况下，即二叉树退化为链表时，递归深度达到 n ，使用 $O(n)$ 的栈帧空间。因此总体空间复杂度为 $O(n)$ 。

欢迎在评论区留下你的见解、问题或建议

45 个表情



 18

 6

 7

 6

 6

 2

17 条评论 · 25 条回复 – 由 giscus 提供支持

最早

最新


输入

预览

Aa

登录后可发表评论

使用 GitHub 登录




NormanCarrie 22 天前


```
class TreeNode {
  constructor(val) {
    this.val = val;
    this.left = null;
    this.right = null;
  }
}

function buildTree(preorder, inorder) {
  if (preorder.length === 0) {
    return null;
  }
  let root = new TreeNode(preorder[0]);
  let index = inorder.indexOf(preorder[0]);
  root.left = buildTree(preorder.slice(1, index + 1), inorder.slice(0, index));
  root.right = buildTree(preorder.slice(index + 1), inorder.slice(index + 1));
  return root;
}
```

↑ 1



1 条回复



NormanCarrie 22 天前

我损失一点空间[汗]



Asunami 28 天前

如果二叉树中有值重复的节点的情况该如何实现呢

↑ 1



2 条回复



Asunami 27 天前

已编辑

```
static List<TreeNode> buildTrees(int[] preorder, int[] inorder) {
    Map<Integer, List<Integer>> hmap = new HashMap<>();
    // computeIfAbsent方法,存在key则返回,否则添加
    for (int i = 0; i < inorder.length; i++) {
        hmap.computeIfAbsent(inorder[i], k -> new ArrayList<>())
    }
    return ndfs(preorder, hmap, 0, 0, inorder.length - 1);
}

static List<TreeNode> ndfs(int[] preorder, Map<Integer, List<Integer>> hmap, int l, int r) {
    List<TreeNode> res = new ArrayList<>();
    if (r < l) {
        res.add(null);
        return res;
    }
    TreeNode root = new TreeNode(preorder[l]);
    for (Integer m : hmap.get(preorder[l])) {
        if (m >= l && m <= r) {
            for (TreeNode left : ndfs(preorder, hmap, l + 1, m))
                for (TreeNode right : ndfs(preorder, hmap, m + 1, r))
                    root.right = right;
            root.left = left;
            res.add(root);
        }
    }
    return res;
}
```



Asunami 27 天前

这是我根据leetcode上的题解凑出来的答案,但是我对于if(m>=l && m<=r)这个条件不太理解



Coolllh 4 月 3 日

已编辑

```
public static void main(String[] args) {
    System.out.println(new BuildTree().buildTreeByPost(new int[]{8, 6, 5, 4, 3, 2, 1}, new int[]{1, 2, 3, 4, 5, 6, 8}));
}

public TreeNode buildTreeByPost(int[] postorder, int[] inorder) {
    Map<Integer, Integer> inorderMap = new HashMap<>();
    System.out.println(Arrays.toString(inorder));
    for (int i = 0; i < inorder.length; i++) {
        inorderMap.put(inorder[i], i);
    }
    System.out.println(inorderMap);
    return dfsPost(postorder, inorderMap, postorder.length - 1, 0, inorder.length - 1);
}

/**
 *
 * @param postorder 后序遍历的结果
 * @param inorderMap 中序遍历的结果Map集合
 * @param rootInPost 要构建的树的根节点在后序遍历中的索引
 * @param left 要构建的树的最左子节点在中序遍历中的索引
 * @param right 要构建的树的最右子节点在中序遍历中的索引
 * @return 要构建的树的根节点
 */
public TreeNode dfsPost(int[] postorder, Map<Integer, Integer> inorderMap, int rootInPost, int left, int right) {
    if (left > right) return null;

    TreeNode treeNode = new TreeNode();

    treeNode.val = postorder[rootInPost];
    // 根节点在中序遍历中的索引
    int rootInIn = inorderMap.get(postorder[rootInPost]);
    // 右子树节点的数量
    int rightTreeNodeNum = right - rootInIn;
    treeNode.left = dfsPost(postorder, inorderMap, rootInPost - rightTreeNodeNum, left, rootInIn);
    treeNode.right = dfsPost(postorder, inorderMap, rootInPost - 1, rootInIn + 1, right);
    return treeNode;
}
```

↑ 1

2 条回复

Coolllh 4 月 3 日

昨天看到评论区有一个写了中序+后序的，我就想着自己也试着写一下，和前序+后序的思路是一样的，
写完这个中序+后序的感觉对构件树的递归过程有了一些想法



Coolllh 4 月 3 日

我认为递归构建树，每一次递归，他的目的实际上只是为了构建一个节点，调用一次dfs方法实际上只构建了一个节点，给这个节点的value属性左子节点，右子节点都填上对应的属性，最后返回到最外层的时候，实际上得到的就是最上方的根节点，而且构建树的过程我想应该中序遍历是必备的，因为没有中序遍历则无法分辨出左子树和右子树，前序和后序遍历的区别我想应该就是在，先遍历根节点还是最后遍历根节点，k神能指点一下我说的对不QWQ



Coolllh 4 月 2 日

已编辑

```
public class BuildTree {  
    /**  
     *  
     * @param preorder 二叉树的前序遍历结果  
     * @param inorderMap 二叉树的中序遍历结果  
     * @param rootInPre 要构建的子树根节点在前序遍历中的索引  
     * @param left 要构建的子树的左子节点在中序遍历中的索引  
     * @param right 要构建的子树的右子节点在中序遍历中的索引  
     * @return 构建出的树根节点  
     */  
    public TreeNode dfs(int[] preorder, Map<Integer,Integer> inorderMap,  
        if (left>right){  
            //左子节点的索引大于右子节点的索引，即无子节点  
            return null;  
        }  
        TreeNode treeNode=new TreeNode();  
        treeNode.val =preorder[rootInPre];  
        //根节点在中序遍历中的索引  
        int rootInIn=inorderMap.get(preorder[rootInPre]);  
        treeNode.left=dfs(preorder,inorderMap,rootInPre+1, left, rootInIn-1  
        treeNode.right=dfs(preorder,inorderMap,rootInPre+1+rootInIn-left,  
        System.out.println(treeNode);  
        return treeNode;  
    }  
  
    public TreeNode buildTree(int[] preorder,int[] inorder){  
        Map<Integer,Integer>inorderMap=new HashMap<>();  
        for (int i = 0; i < inorder.length; i++) {  
            inorderMap.put(inorder[i],i);  
        }  
    }  
}
```

```
        return dfs(preorder,inorderMap,0,0,inorder.length-1);
    }

    public static void main(String[] args) {
        new BuildTree().buildTree(new int[]{3,9,2,1,7},new int[]{9,3,1,2,
    }

}

    public TreeNode() {

    }

    public int val;
    public TreeNode left;
    public TreeNode right;

    public TreeNode(int value, TreeNode left, TreeNode right) {
        this.val = value;
        this.left = left;
        this.right = right;
    }
}
```

↑ 1



1 条回复

Coolllh 4 月 2 日

我根据这里的思路写了一个一样的，并且对其中的一些字段做上了注解，没注解刚看的时候有点一头雾水的



Coolllh 4 月 2 日

当前树是什么意思，没太看懂

↑ 1



0 条回复

xxc111111 3 月 16 日

12.3的代码有点难，如果有需要的小伙伴可以邮箱私我，我自己画了个步骤图，可以方

方便大家

史垚群

↑ 1



1 条回复

big-good-time 4 月 9 日

好哥哥, 180089009@qq.com



Purdre 2023 年 12 月 12 日

已编辑

```
/* 构建二叉树: 分治 */
TreeNode *dfs(int *preorder, int *inorderMap, int i, int l, int r, int size)
// 子树区间为空时终止
if (r - l < 0)
    return NULL;
// 初始化根节点
TreeNode *root = (TreeNode *)malloc(sizeof(TreeNode));
root->val = preorder[i];
root->left = NULL;
root->right = NULL;
// 查询 m , 从而划分左右子树
int m = inorderMap[preorder[i]];
// 子问题: 构建左子树
root->left = dfs(preorder, inorderMap, i + 1, l, m - 1, size);
// 子问题: 构建右子树
root->right = dfs(preorder, inorderMap, i + 1 + m - l, m + 1, r, size);
// 返回根节点
return root;
}
```

思考不下去, 没想明白它这个“递”的过程会怎样结束, 就是可以知道如果左子树 (右子树) 为空的话就就不在进行构建左子树 (右子树), 比如当 $m-l=0$ 时说明没有左子树了, 就是在建立节点9的那个函数 (也就是第二函数), 里面还会有`root->left=dfs(preoder,inorderMap,i+1,l,m-1,size)`///此时 i 为2, l 为0, $m-1$ 为0,很混乱。我的想法是需要加`if (m!=l)`在前面加`if (m!=l)`///如果相等代表左子树没了就不再行。应该是还没理解, 所以来请教一下

↑ 1



2 条回复

hpstory 2023 年 12 月 13 日

已编辑

Hi @Purdre , 两种写法是一样的, `TreeNode`类型的左右节点默认值都是`NULL`, 如果加`if (m != l)`, 则当前节点的`left`会使用默认值`NULL`, 如果不加, `dfs`返回的结

果也是NULL。



480284856 10 天前

在以3为根节点，递归到其左子树时，递归函数的中的部分参数为：

- 1. i=1
- 2. l=0
- 3. r=0

进入后，此时根节点为9，计算得到**m=0**.

再进行递归，进入节点9的左子树，此时递归函数中的部分参数为：

- 1. i=2
- 2. l=0
- 3. r=-1



eastcukt 2023 年 12 月 5 日

讲解很棒，也希望加些更详细的说明。比如这里中序是用来确定左右子树长度，进而在前序中确定左右子树(的根节点)下标。后序加中序重建也类似的道理。不过简洁点也好，要是当年在学数据结构时能看到hello算法就好了。。。

↑ 1



1

1 条回复



krahets 2023 年 12 月 5 日

谢谢建议！



amlei 2023 年 11 月 24 日

Hi, 请问有考虑添加：已知 后序和中序遍历 构建树的情形吗？我在构建时遇到下面一个问题，以本章后序遍历树为例： [9,1,7,2,3]

- 1. 以 postorder or inorder 为索引？
 - 显然遵照后序遍历： [左子树 | 右子树 | 根节点] ，建立 inorder 索引后的 inorderMap 中直接能够找到根节点 (postorderSize - 1)，但对于其左子树与右子树来说本人无法得出如何获取它们的方式。

- 以 postorder 建立索引，却丢失了根节点

↑ 1 

4 条回复

hpstory 2023 年 11 月 24 日

@amlei 力扣上面有类似的题目[LeetCode 106](#) 供参考

  3

chen-xiao-yu 2023 年 11 月 27 日

后序遍历的时候，[左子树 | 右子树 | 根节点]，实际上可以说不成立(当然严格意义上说是成立的)。

- 比如你后序遍历出来的结果，根节点是3，前面 7，2并不是他的左节点和右节点。
- 因为实际上还有可能只有左子树或者右子树。
- 唯一能确定的只有中序遍历中，根节点的前一个节点是他的左节点，后序遍历最后一个节点是根节点。
- 至于有没有右节点，你还得看后续遍历中根节点的前一个节点和中序遍历中根节点的前一个节点是否一样，如果一样，说明没有右子树，如果不一样，才说明有右子树，这个时候紧挨着根节点的才是右节点。



chen-xiao-yu 2023 年 11 月 27 日

哦，前面说那也都不对，根节点前面一个节点也可能是他的左节点的右子树里面的节点



amlei 2023 年 11 月 28 日

已编辑

@hpstory 感谢给出的参考！@chen-xiao-yu 感谢提供思路！

我结合本章的代码风格构建前序树的代码已实现，只需修改少部分即可实现：

dfs() 函数：

```
// 子问题：构建右子树
root->right = dfsPre(inorderMap, postorder, i - 1, m + 1, r, size);
// 子问题：构建左子树
root->left = dfsPre(inorderMap, postorder, i - (r - m) - 1, l, m - 1);
```

build() 函数:

```
TreeNode *root = dfsPre(inorderMap, postorder, inorderSize - 1, 0, ...
```

完整代码:

```
/* 构建二叉树-已知后序与中序 */
TreeNode *dfsPre(int *inorderMap, const int *postorder, int i, int l, int r) {
    // 子树区间为空时终止
    if (r - l < 0) {
        return NULL;
    }

    // 初始化根节点，以后序遍历的特性，最后一个元素即为根节点
    TreeNode *root = newTreeNode(postorder[i]);
    root->left = NULL;
    root->right = NULL;

    // 查询 m 从后序遍历索引到中序遍历，查看是否有左、右节点
    int m = inorderMap[postorder[i]];

    // 子问题：构建右子树
    root->right = dfsPre(inorderMap, postorder, i - 1, m + 1, r, si);
    // 子问题：构建左子树
    root->left = dfsPre(inorderMap, postorder, i - (r - m) - 1, l, i);

    // 返回根节点
    return root;
}

// 构建前序树
TreeNode *buildTreePre(const int *inorder, int inorderSize, const int *postorder) {
    // 初始化哈希表，存储 inorder 元素到索引的映射
    int *inorderMap = (int *) malloc(sizeof(int) * MAX_SIZE);
    for (int i = 0; i < inorderSize; i++) {
        inorderMap[inorder[i]] = i;
    }

    TreeNode *root = dfsPre(inorderMap, postorder, inorderSize - 1, 0, ...);
    free(inorderMap);
    return root;
}
```

测试用例:

```
1. int inorder[] = {9,3,1,2,7}; int postorder[] = {9,1,7,2,3};
```

```
2. int inorder[] = {15,9,10,3,20,5,7,8,4}; int postorder[] =  
   {15,10,9,5,4,8,7,20,3};
```



2

xhddx 2023 年 11 月 23 日

```
root.left = dfs(preorder, inorderMap, i + 1, l, m - 1);
```

如果 左子树在inorder里的索引是[l, m - 1], 那 这个i+1参数是什么含义呀, 左子树的根节点在preOrder中的索引? 也不对呀。这里没太理解

↑ 1



1

1 条回复

xhddx 2023 年 11 月 23 日

哎呦, 还真是左子树的根节点在preOrder中的索引



1

gh9991 2023 年 10 月 23 日

图12-8也太酷了吧, hhh, 厉害

↑ 1



1

0 条回复

mcthesw 2023 年 10 月 23 日

看理论部分感觉很难, 但是代码实现居然如此简洁, 这就是分治的魅力吗

↑ 1



1 条回复



krahets 2023 年 10 月 23 日

递归代码一般都有这个特点~



newcakes 2023 年 10 月 13 日

用这种方式构建的树节点的值不能是一样的吧! 因为map的key值必须唯一!


```
// 查询 m, 从而划分左右子树
```

```
int m = inorderMap[preorder[i]];
```


我想请教一下k佬，如果用unordered_multimap来实现树中结点的值可以是相同的，那该如何实现呢？
因为一个key值对应两个数组下标，该如何操作？还是说要另辟蹊径，用其他方式。

↑ 1



 2

 1

2 条回复

newcakes 2023 年 10 月 14 日

c++语言呀！



krahets 2023 年 10 月 23 日

你说的对。这道题目假设树中没有重复的节点，否则也无法完成划分了～



Rzzzzzz 2023 年 9 月 28 日

k神, 如果把9去掉, 这个树退化成只有一边, 根节点就不存在左子树, 也就L的初始值不存在, i+1也不是左子树的根, 这个算法是不是就不适用了

↑ 1



2 条回复



krahets 2023 年 9 月 28 日

不会的，这个算法适用于任意二叉树。建议运行源代码，改一下 test case 试试



chenghui-ch 2023 年 12 月 17 日

还要用inorder限制左子树的范围 (m-1)，r-l<0了。



BigDevil82 2023 年 9 月 14 日

一个小改进，dfs 中没有用到 inorder，可以去掉

↑ 1



1 条回复



krahets 2023 年 9 月 15 日



kranets 2023 年 9 月 15 日

谢谢建议，已优化



zhuasdfg 2023 年 8 月 9 日

难点是如何把过程用编程语言表示出来，用指针把自己想要的操作表达出来，还得注意他们之间的约束关系

↑ 1



2

0 条回复

danntee 2023 年 8 月 3 日

考研挺喜欢考这种题的

↑ 1



2

4 条回复

bushilao 2023 年 9 月 28 日

$i + 1 + m - l$ 右子数为什么会用到这么奇怪的表达式，直接用 m 不可以吗

