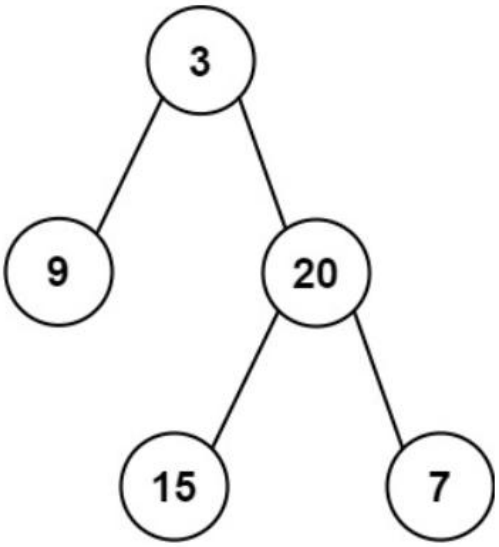
105. 从前序与中序遍历序列构造二叉树

已解答

中等 相关标签 相关企业

给定两个整数数组 `preorder` 和 `inorder` ，其中 `preorder` 是二叉树的**先序遍历**，`inorder` 是同一棵树的**中序遍历**，请构造二叉树并返回其根节点。

示例 1:



输入: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
输出: [3,9,20,null,null,15,7]

示例 2:

解题思路一：

递归：

根据二叉树前序中序特点可知，前序的第一个节点为根节点，我们在中序中找到它的根节点的位置 index，就可以计算其左子树的大小 leftsize=index-inorder,那么它的右子树的在中序的起始位置就是 index+1 到 inend,，在前序的起始位置是 prestart+leftsize+1,preend；左子树在前序就有起点是 prestart+1,终点是 prestart+leftsize,它在中序的位置是 inostart,index-1。现在已经知道了起始位置，那么我们就可以将其当参数，模仿根节点的创建，在函数中递归返回即可

代码：

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
```

```
            return construct(preorder, 0, preorder.size() - 1, inorder, 0, inorder.size() - 1);
        }
    private:
        TreeNode* construct(vector<int>pre, int prestart, int preend, vector<int>ino, int inostart, int inoend)
        {
            if (prestart > preend || inostart > inoend)
                return nullptr;
            //从根节点开始构建
            TreeNode* root = new TreeNode(pre[prestart]);
            if (prestart == preend) {
                return root;
            }
            // 前序遍历中根节点之后的第一个节点是左子树的根节点
            int leftRootVal = pre[prestart];

            // 找到根节点在中序遍历中的位置
            int leftRootIndex = inostart;
            while (ino[leftRootIndex] != leftRootVal) {
                leftRootIndex++;
            }
            // 计算左子树的大小
            int leftSize = leftRootIndex - inostart;

            // 递归构建左子树和右子树
            root->left = construct(pre, prestart + 1, prestart + leftSize, ino, inostart, leftRootIndex-1);
            root->right = construct(pre, prestart + leftSize + 1, preend, ino, leftRootIndex + 1, inoend );

            return root;

        }
        };
```

解题思路二：

迭代法：

对于前序的任意两个连续的节点，它们之间的关系有两种。第一种最常见的是 v 是 u 的左孩

子，第二种就是 u 没有左孩子，它是 v 的某个祖先的右孩子或者是 u 自己的右孩子，如果 u 没有右孩子，向上回溯，直到第一个有右孩子且 u 不是该节点的右儿子的子树

使用一个辅助栈进行迭代更新，栈中存储的是没有考虑过右孩子的节点，当栈为空时即这可能是根的右孩子或者说节点没入栈。首先根节点入栈，根节点是前序第一个元素，然后初始化索引 index 指向中序第一个元素，如果当前栈顶元素不等于索引指向的值，前序的元素成为当前元素的左孩子，前序的元素入栈成为新的栈顶，如果当前栈顶元素等于索引指向的值，说明当前栈顶元素没有左孩子了，此时开始考虑右孩子，而栈中每一个元素都是没有考虑孩子的可以把 index 不断向右移动，并与栈顶节点进行比较。如果 index 对应的元素恰好等于栈顶节点，那么说明我们在中序遍历中找到了栈顶节点，所以将 index 增加 1 并弹出栈顶节点，直到 index 对应的元素不等于栈顶节点。按照这样的过程，我们弹出的最后一个节点 x 就是 10 的双亲节点

代码：

```
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
      //处理异常条件
        if (!preorder.size()) {
            return nullptr;
        }
        //根节点一定是前序第一个节点
        TreeNode* root = new TreeNode(preorder[0]);
        //辅助栈提供应该进入的没考虑过右孩子的节点
        stack<TreeNode*> stk;
        stk.push(root);
        //初始化指向为中序的第一个元素
        int inorderIndex = 0;
        //由于中序的特点，当前元素的左边一定是它的左孩子或左孩子的右节点
        for (int i = 1; i < preorder.size(); ++i) {
            int preorderVal = preorder[i];
            TreeNode* node = stk.top();
            //如果当前栈顶元素不等于索引指向的值，入栈，成为左子树
            if (node->val != inorder[inorderIndex]) {
                node->left = new TreeNode(preorderVal);
                stk.push(node->left);
            }
            //当前栈顶元素等于索引指向的值，弹出栈顶，弹出一次索引++，直到当前元素等于索引指向的元素
            //或者当前栈为空，为空说明是根的右子
```

```
                else {
                    while (!stk.empty() && stk.top()->val == inorder[inorderIndex]) {
                        node = stk.top();
                        stk.pop();
                        ++inorderIndex;
                    }
                    //最后一次弹出的元素是当前元素的父亲，当前元素是它的右子，当前元素入栈
                    node->right = new TreeNode(preorderVal);
                    stk.push(node->right);
                }
            }
            return root;
        }
    };
```