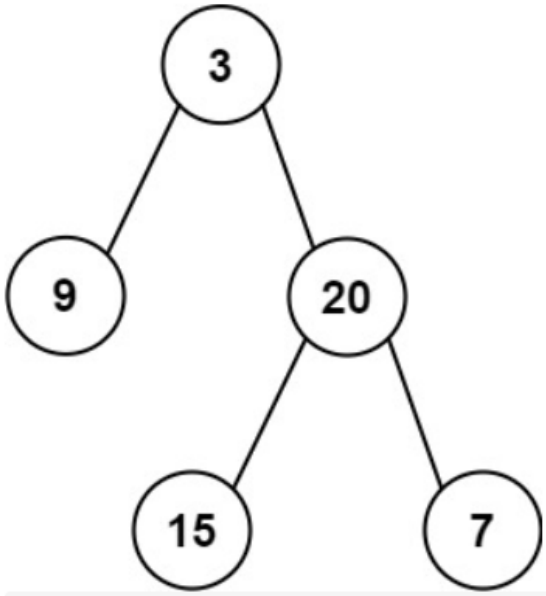
第30题——重建二叉树

输入某二叉树的前序遍历和中序遍历的结果，请构建该二叉树并返回其根节点。

假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

> **示例 1:**
>
> 
>
>
> ```
> Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
> Output: [3,9,20,null,null,15,7]
> ```

> **示例 2:**
>
> ```
> Input: preorder = [-1], inorder = [-1]
> Output: [-1]
> ```

思路:

架设我们的序列如下:

前序	1	2	4	7	3	5	6	8
中序	4	7	2	1	5	3	8	6

因为二叉树前序和中序遍历的规律,我们可以得知,前序序列的第一个元素是头结点,中序序列的头结点的左边的所有元素是左子树,右边所有结点是右子树.而对于每一颗子树,我们同样可以用这条规律将它分为左右子树,以此类推.

显而易见,解决的最好办法就是递归

在解决问题之前,对于一棵树,我们需要先明白几个概念:

```
front_root : 前序序列的头结点

mid_left : 中序序列的左边界

mid_right : 中序序列的有边界
```

递归需要完成以下三件事情:

1. 检查递归条件 : mid_left>left_right
2. 生成当前子树的根结点root
3. 递归遍历左子树
4. 递归遍历右子树
5. 返回当前子树的根节点root

那么问题来了,为什么递归条件是 mid_left>left_right 呢?

```
因为我们的每一次递归,子树的边界都会不断移动,当mid_left=left_right时,恰好是当前结点,而如果
mid_left>left_right则说明这是一个空的结点,我们应该返回nullptr;
```

> 左子树的左边界=中序序列左边界
>
> 左子树的右边界=中序序列根节点-1

> 右子树的左边界=中序序列根结点+1
>
> 右子树的右边界=中序序列右边界

> 左子树的根结点=前序序列头节点+1
>
> 右子树的根节点=前序序列头节点+左子树长度+1
>
> 左子树长度=左子树右边界-左子树左边界

代码如下:

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
```

```
public:

    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        this->preorder=preorder;//参数传递
        for(int i=0;i<inorder.size();i++)
        {
            map[inorder[i]]=i;//构建哈希映射
        }
        //调用递归
        return createTree(0,0,inorder.size()-1);
    }
private:
    vector<int>preorder;
    unordered_map<int,int>map;
    TreeNode* createTree(int front_root,int mid_left,int mid_right)
    {
        if(mid_left>mid_right)
        {
            return nullptr;
        }
        TreeNode* root= new TreeNode(preorder[front_root]);//生成根结点

        int mid_root=map[preorder[front_root]];//获得中序序列根节点的下标

        //递归左子树
        root->left=createTree(front_root+1,mid_left,mid_root-1);

        //递归右子树
        root->right=createTree(front_root+(mid_root-mid_left)+1,mid_root+1,mid_right);

        return root;//返回当前结点
    }
};
```