1. 题目
2. 题干

输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。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。

1. 示例

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：

3

/ \

9 20

/ \

15 7

1. 题解
2. 思路

前序遍历就是先访问树的根节点，再访问树的左子树，最后访问树的右子树。中序遍历是先访问树的左子树，再访问树的根节点，最后访问树的右子树。可以看到，前序遍历是按照根节点，左子树，右子树排列的；而中序遍历是按照左子树，根节点，右子树排列的。

因此，我们可以从前序遍历获取最前面的数，一定是根节点，根节点将中序遍历一分为二，左边是左子树，右边是右子树。获取根节点在中序遍历的索引后，我们可以计算得到左子树和右子树的节点数量。因此又可以在前序序列中找寻左子树和右子树的根节点了，如果左子树的节点数不为0，则左子树的根节点就是前序遍历中根节点的下一个数；而右子树的根节点在前序遍历中的索引是（根节点+左子树节点数）。这样就能递归下去了，每次递归就是创建一个新的树，需要每次从前序遍历和中序遍历中取出相应的数字，因此，我们需要记录每次取出的前序遍历和中序遍历的数字段的左右区间。题目的要求是重建二叉树，因此每次获取一个节点时都要重新创建节点，并且连接根节点，左子树和右子树。

1. 代码实现

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode(int x) { val = x; }

 \* }

 \*/

class Solution {

    public TreeNode buildTree(int[] preorder, int[] inorder) {

        if(preorder == null || inorder == null){

            return null;

        }

        int length = preorder.length;

        //Map存储inorder的索引

        Map<Integer,Integer> store = new HashMap<>();

        for(int i = 0; i < length; ++i){

            store.put(inorder[i], i);

        }

        TreeNode root = dfs(preorder, 0, length-1, inorder, 0, length-1 , store);

        return root;

    }

    public TreeNode dfs(int[] preorder, int preStart, int preEnd, int[] inorder, int inStart, int inEnd, Map<Integer,Integer> store)    {

        if(preStart > preEnd){

            return null;

        }

        int rootVal = preorder[preStart];

        TreeNode root = new TreeNode(rootVal);

        //只有一个root

        if(preStart == preEnd){

            return root;

        }else{

            //获取root在中序遍历中的索引

            int rootIndex = store.get(rootVal);

            //从中序遍历中获取左子树和右子树的节点数

            int leftNum = rootIndex - inStart, rightNum = inEnd - rootIndex;

            //创建左子树

            TreeNode leftNode= dfs(preorder, preStart+1, preStart+leftNum,  inorder, inStart, rootIndex-1, store);

            //创建右子树

            TreeNode rightNode = dfs(preorder, preEnd-rightNum+1, preEnd, inorder, rootIndex+1, inEnd, store);

            root.left = leftNode;

            root.right = rightNode;

            return root;

        }

    }

}