105. Construct Binary Tree from Preorder and Inorder Traversal

link

explain link

这个非递归, 用map存一下, 每次判断在左边还是右边, 如果是左边, 就top.next, 如果是在右边就while循环pop到top是上一个他的右边, current.right就是current.

这里判断stack.peek().val, 如果下一个元素是第一个在他右边的元素, 当前元素就应该是他的父节点, 判断栈顶元素peek来决定循环走向, 还有就是栈运行到root会变空.

```
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        return helper(preorder, inorder);
    }
    public TreeNode helper(int[] preorder, int[] inorder) {
        if(preorder.length == 0) return null;
        Map<Integer, Integer> map = new HashMap();
        int len = preorder.length;
        Stack<TreeNode> stack = new Stack();
        for(int i = 0; i < len; ++i){
            map.put(inorder[i], i);
        }
        TreeNode root = new TreeNode(preorder[0]);
        stack.push(root);
        for(int i = 1; i < len; ++i){
            int val = preorder[i];
            TreeNode current = new TreeNode(val);
            if(map.get(stack.peek().val) > map.get(val)){
                stack.peek().left = current;
                stack.push(current);
            }else{
                TreeNode parent = null;
                //运行到root.right会出现栈是空的情况.
                while(!stack.isEmpty() &&
//这里判断stack.peek().val, 如果下一个元素是第一个在他右边的元素, 当前元素就应该是他的父节点.
                      map.get(stack.peek().val) < map.get(val)){
                    parent = stack.pop();
                }
                parent.right = current;
                stack.push(current);
            }
        }
        return root;
```

```
    }
```

更好的

这个就是按顺序处理preorder, node相当于一个标识, 判断是左边还是右边`stack.size()>0 && stack.peek().val == inorder[j]`表示如果peak的值与下面的相等了, 表示已经入栈并且加在左子树了, 此时pop到peak最后相等此时: 栈为空, 是根节点, 栈非空表示这个值的左子树已经处理完了, 接下来该处理他的右子树.

```
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        if(inorder==null || inorder.length==0) return null;
        int j=0;
        TreeNode root = new TreeNode(preorder[0]);
        Stack<TreeNode> stack = new Stack();
        stack.push(root);
        TreeNode node = null;
        for(int i=1; i<preorder.length; i++){
            TreeNode cur = new TreeNode(preorder[i]);
            while(stack.size()>0 && stack.peek().val == inorder[j]){
                node = stack.pop();
                j++;
            }
            if(node!=null){
                node.right = cur;
                node=null;
            }
            else{
                stack.peek().left = cur;
            }
            stack.push(cur);
        }
        return root;
    }
```

解释

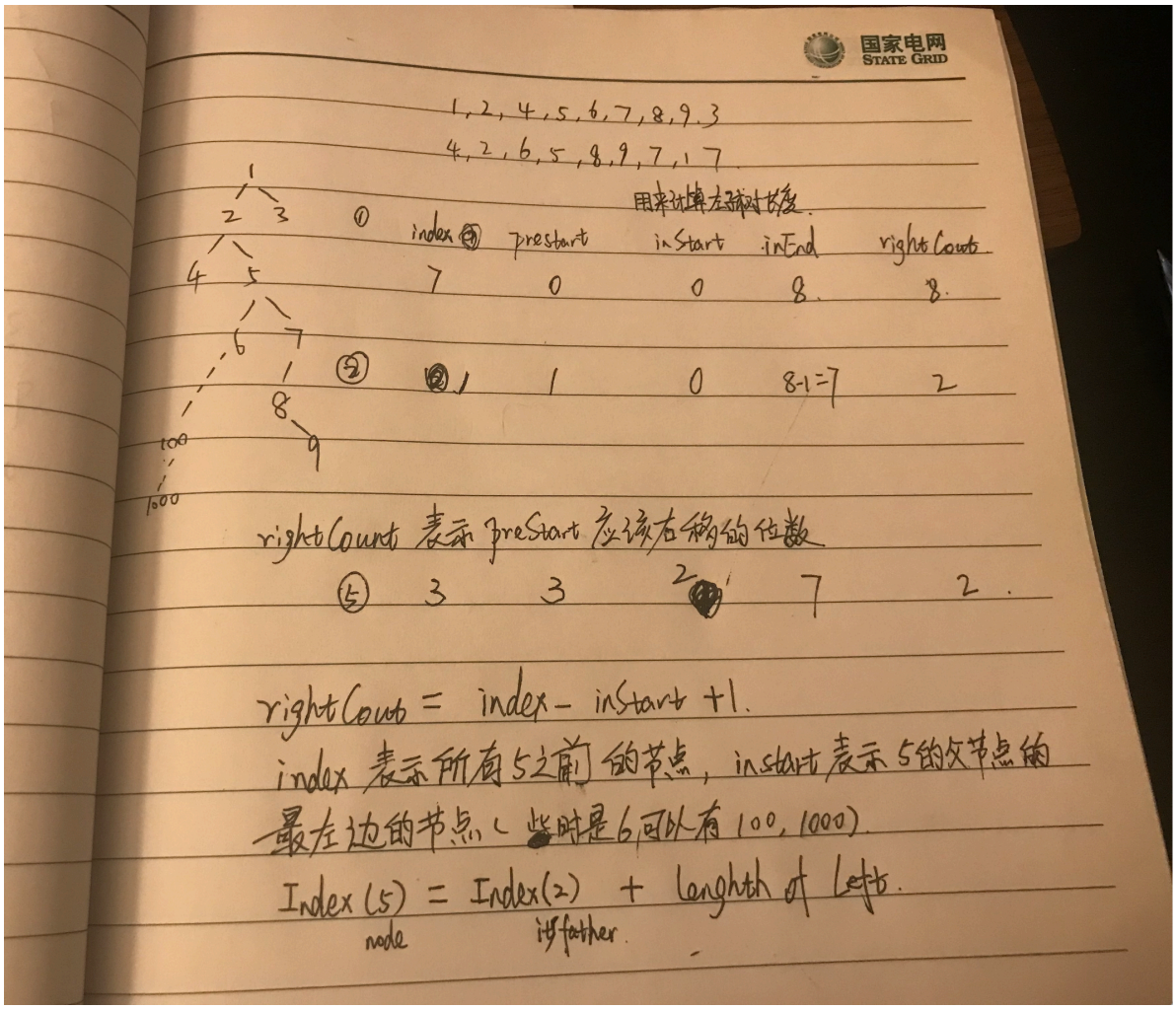
```
class Solution {
    public TreeNode buildTree(int[] preorder, int[] inorder) {
        // deal with edge case(s)
        if (preorder.length == 0) {
            return null;
        }

        // build a map of the indices of the values as they appear in the inorder
    array
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < inorder.length; i++) {
            map.put(inorder[i], i);
        }

        // initialize the stack of tree nodes
```

```
        Stack<TreeNode> stack = new Stack<>();
        int value = preorder[0];
        TreeNode root = new TreeNode(value);
        stack.push(root);

        // for all remaining values...
        for (int i = 1; i < preorder.length; i ++) {
            // create a node
            value = preorder[i];
            TreeNode node = new TreeNode(value);

            if (map.get(value) < map.get(stack.peek().val)) {
// the new node is on the left of the last node,
// so it must be its left child (that's the way preorder works)
                stack.peek().left = node;
            } else {
// the new node is on the right of the last
// so it must be the right child of either the last
// or one of the last
// pop the stack until we either run out of
// or the node at the top of the stack is to the right of the new node
                TreeNode parent = null;
                while(!stack.isEmpty() && map.get(value) >
    map.get(stack.peek().val)) {
                    parent = stack.pop();
                }
                parent.right = node;
            }
            stack.push(node);
        }

        return root;
    }
}
```

递归写法 比较难懂

```
    public TreeNode helper2(int[] preorder, int preStart, int[] inorder, int
    inStart, int inEnd){
        if(preStart >= preorder.length || inStart > inEnd) return null;
        int val = preorder[preStart];
        TreeNode root = new TreeNode(val);
        int index = getIndex(inorder, inStart, inEnd, val);
//inStart代表父节点左边的节点个数,左子树完全用不上,用来算右子树的长度用的.
//所以最左边的左子树全是0, 有右子树才更新.
        int leftSubTreeCount = index - inStart;
        root.left = helper2(preorder, preStart + 1, inorder, inStart, index - 1);
        root.right = helper2(preorder, preStart + leftSubTreeCount + 1, inorder,
    index + 1, inEnd);
        return root;
    }
    public int getIndex(int[] array, int start,int end, int val){
        for(int i = start; i <= end; ++i){
            if(array[i] == val)
                return i;
        }
        return - 1;
```

}


国家电网
STATE GRID
1,2,4,5,6,7,8,9,3
4,2,6,5,8,9,7,1,7
用来计算左子树长度
index
preStart
inStart
inEnd
rightCount
7 0 0 8 8
1 1 0 8-1=7 2
rightCount 表示 preStart 应该右移的位数
3 3 2 7 2
rightCount = index - inStart + 1.
index 表示所有5之前的节点，inStart 表示5的父节点的最左边的节点，此时是6，可以有100，1000.
Index(5) = Index(2) + length of Left.
node
its father