题目描述：

有一棵二叉树，每个节点由一个大写字母标识(最多 26 个节点）。现有两组字母，分别表示后序遍历（左孩子->右孩子->父节点）和中序遍历（左孩子->父节点->右孩子）的结果，请输出层次遍历的结果。

输入描述：

输入为两个字符串，分别是二叉树的后续遍历和中序遍历结果。

输出描述：

输出二叉树的层次遍历结果。

补充说明：

示例 1

输入：

```
CBEFDA CBAEDF
```

输出：

```
ABDCEF
```

说明：

二叉树为：

```
   A

  /\

 B  D

/  /\

C  E  F
```

```
import java.util.*;

// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    private String postorder;
    private String inorder;
    private class TreeNode{
        public char val;
        public TreeNode left;
        public TreeNode right;
        public TreeNode(char val){
            this.val = val;
```

```
        }
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String postorderStr = in.next();
        String inorderStr = in.next();

        Main mainClass = new Main();
        mainClass.postorder = postorderStr;
        mainClass.inorder = inorderStr;
        TreeNode root = mainClass.buildTree(0, postorderStr.length() - 1, 0, inorderStr.length() -
1);

        System.out.println(mainClass.traversal(root));
    }

    private TreeNode buildTree(int postLeft, int postRight, int inLeft, int inRight){
        TreeNode root = new TreeNode(postorder.charAt(postRight));
        int inorderRootIndex = 0;
        for(inorderRootIndex = inLeft; inorderRootIndex <= inRight; inorderRootIndex++){
            if(inorder.charAt(inorderRootIndex) == postorder.charAt(postRight))
                break;
        }

        int leftNodeNum = inorderRootIndex - inLeft;
        int rightNodeNum = inRight - inorderRootIndex;
        if(leftNodeNum > 0)
            root.left = buildTree(postLeft, postLeft + leftNodeNum - 1, inLeft,
inorderRootIndex - 1);
        if(rightNodeNum > 0)
            root.right = buildTree(postLeft + leftNodeNum, postRight - 1, inorderRootIndex + 1,
inRight);

        return root;
    }

    private String traversal(TreeNode root){
        String ans = "";
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while(!queue.isEmpty()){
            TreeNode node = queue.poll();
            ans = ans + node.val;
```

```
                if(node.left != null)
                    queue.offer(node.left);
                if(node.right != null)
                    queue.offer(node.right);
            }
            return ans;
        }
    }
```