**代码思路：**

定义一个树类，内含三个参数：root(存储根节点数据),left(左子树),right(右子树)。

1. **树的构建**

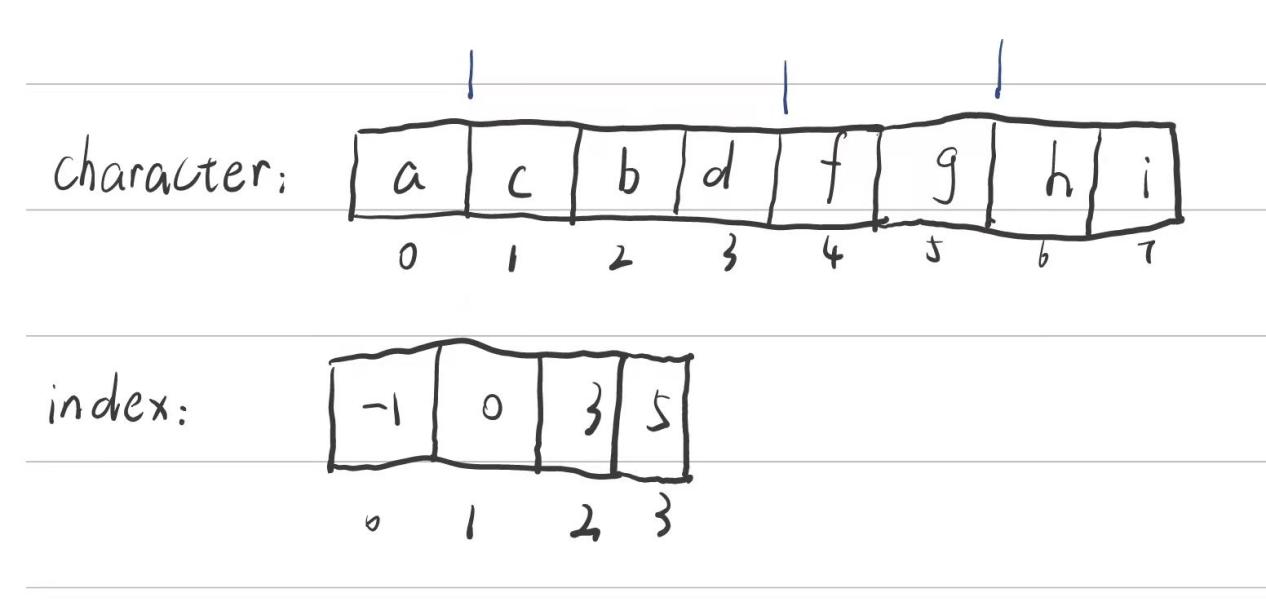
定义Character为存储树的栈

列表index存储’|’所在的位置，index[0]=-1为哨兵，防止数组越界

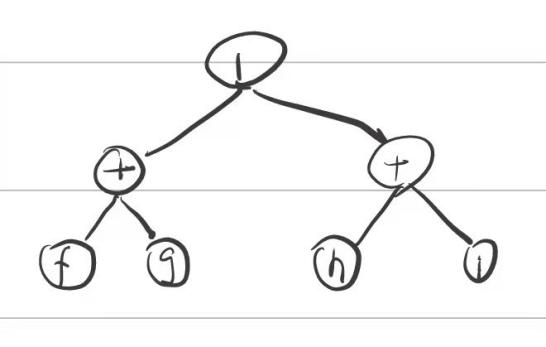
从左至右对正则表达式字符串进行遍历，分情况处理：

1. 遇到字母，将其处理为（root=字母,left=right=None)的二叉树，压入Character中
2. 遇到‘\*’号，因其优先级最高，直接连，将栈中最后一个数据弹出，将（root=’\*’,left=栈中最后一个数据,right=None)的树入栈
3. 遇到’|’号，因其优先级最低，留至最后处理，index.append(i),记录下‘|’所在的位置
4. 遇到‘（’，进入函数Bracket()，对括号内正则表达式字符串处理方式与外部一致，若再遇到’(’则进入下一层递归，遇到‘）’返回
5. 遍历完后，对剩余’|’进行处理：

以正则表达式a|cbd|fg|hi为例，遍历一遍后,Character和index中的元素如图



先分别对Character中(5,7]和（3，5]中的树以‘+’为根节点从后至前进行合并，再对两边以‘|’为根节点进行合并，示例如下：



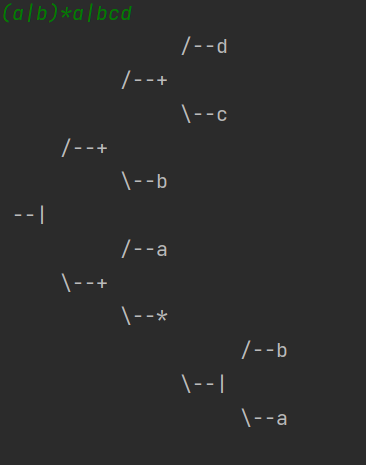
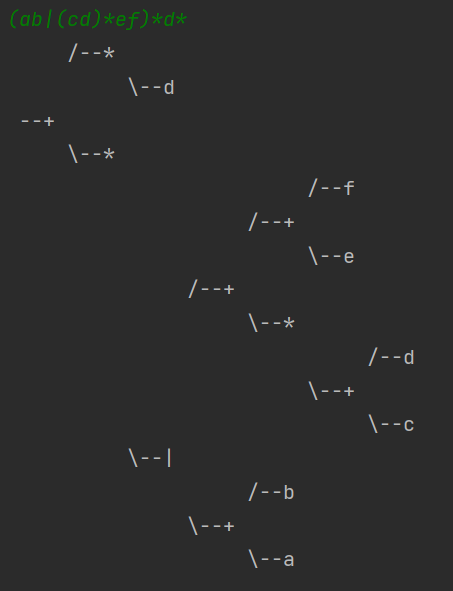
同理，再将(0,3]和(-1,0]之间的树以‘+’为根节点从后至前进行合并，再对两边以‘|’为根节点进行合并，最终一起合并为一棵完整的语法树

**二、树的输出**

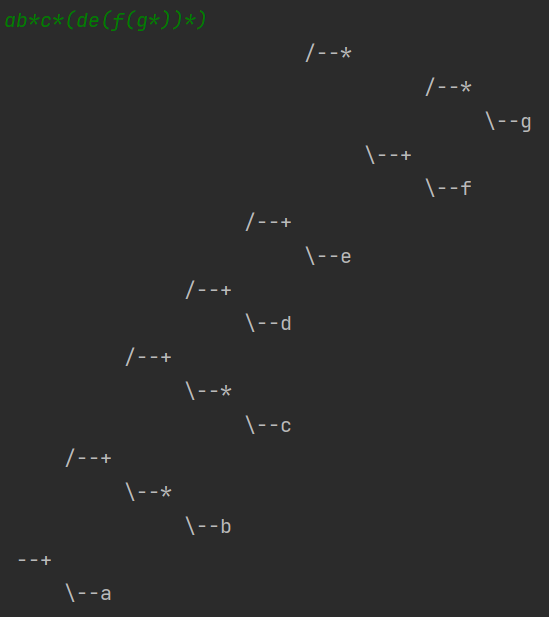
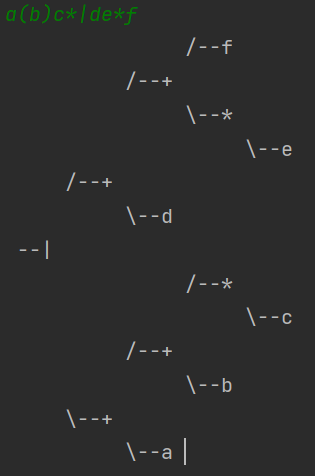
以“右中左”的顺序将整棵树以递归的方式逐行输出，先从根节点开始向右搜索，当右子树为空时回溯至根节点，再向左搜索。

**测试样例：**

(a|b)\*a|bcd (ab|(cd)\*ef)\*d\*

ab\*c\*(de(f(g\*))\*) a(b)c\*|de\*f

a|aa(((a|a\*)\*)\*)\* ((b|c)|d)\*e

