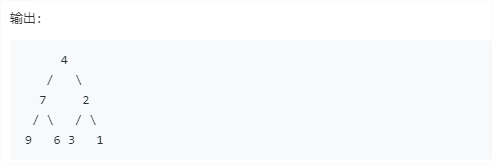
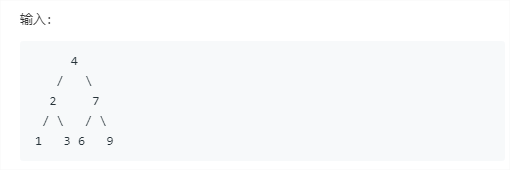
Leetcode 总结版~树

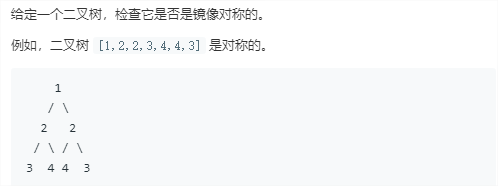
：

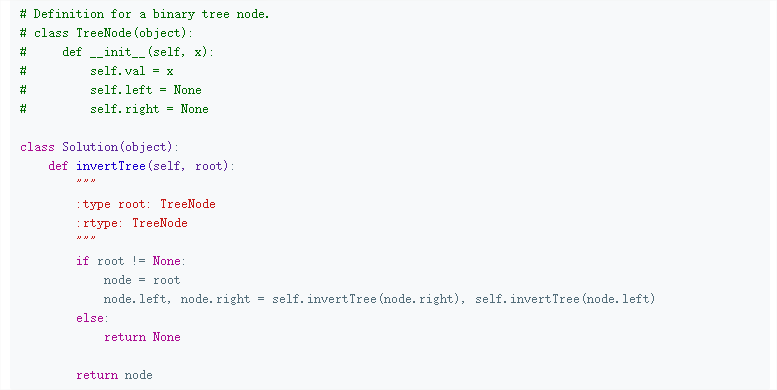
1. **翻转二叉树**



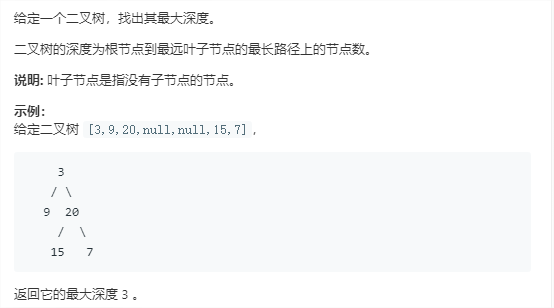


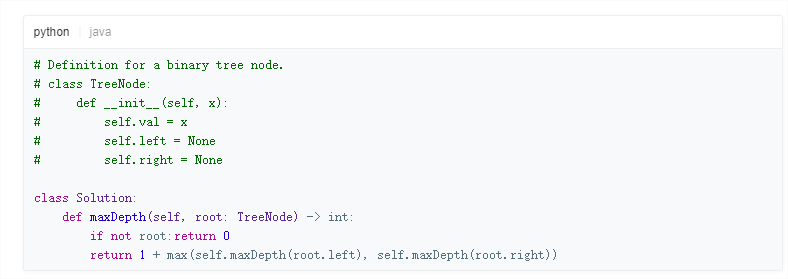
1. **对称二叉树**



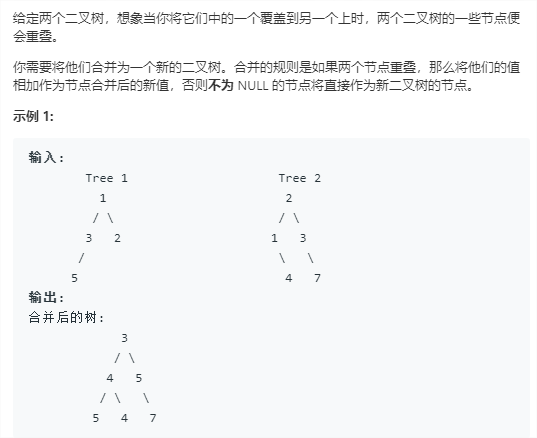


1. **二叉树的最大深度**



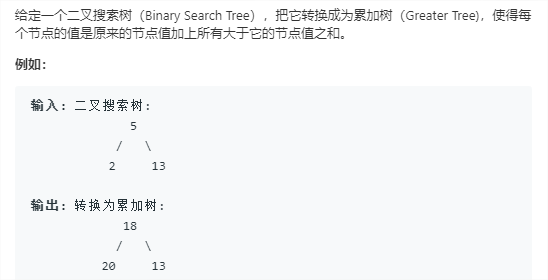


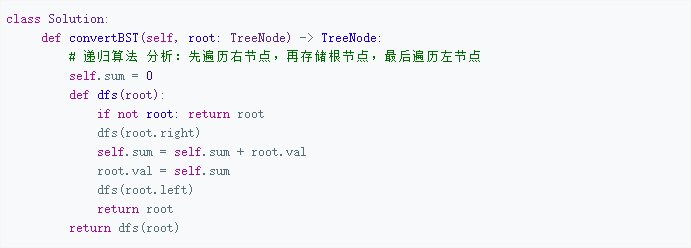
1. **合并二叉树**





1. **把二叉搜索树转为累加树**





1. **路径总和3**

给定一个二叉树，它的每个结点都存放着一个整数值。

找出路径和等于给定数值的路径总数。

路径不需要从根节点开始，也不需要在叶子节点结束，但是路径方向必须是向下的（只能从父节点到子节点）。

二叉树不超过1000个节点，且节点数值范围是 [-1000000,1000000] 的整数。



1. **路径总和2：**

给定一个二叉树和一个目标和，找到所有从根节点到叶子节点路径总和等于给定目标和的路径。**说明:** 叶子节点是指没有子节点的节点。



1. **二叉树的直径**

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过根结点。



return max(self.diameterOfBinaryTree(root.left), self.diameterOfBinaryTree(root.right), depth(root.right)+depth(root.left))

1. **前序遍历，中序遍历，后序遍历，层次遍历**

前序：



中序：



后序：

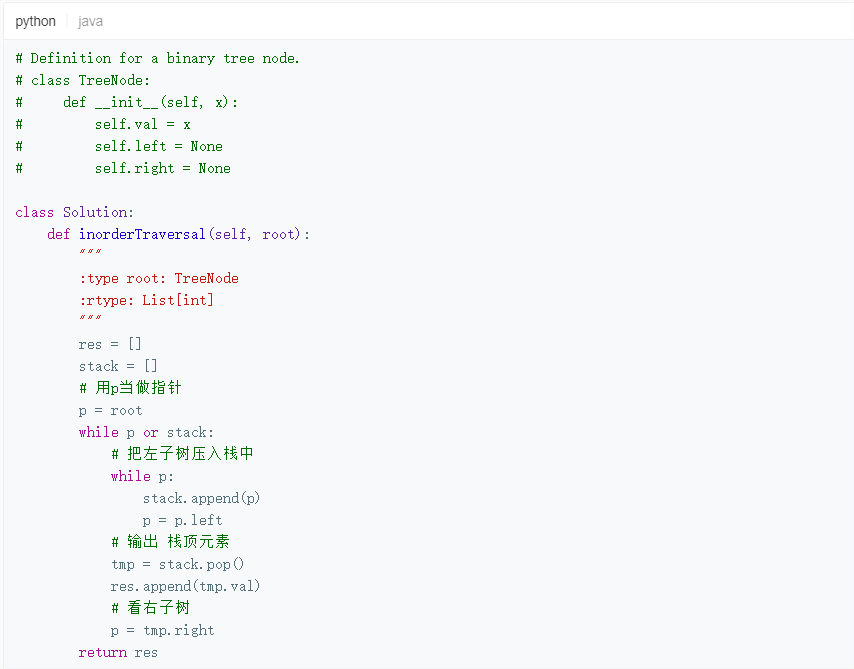


层次遍历：（BFS）



非递归版本：

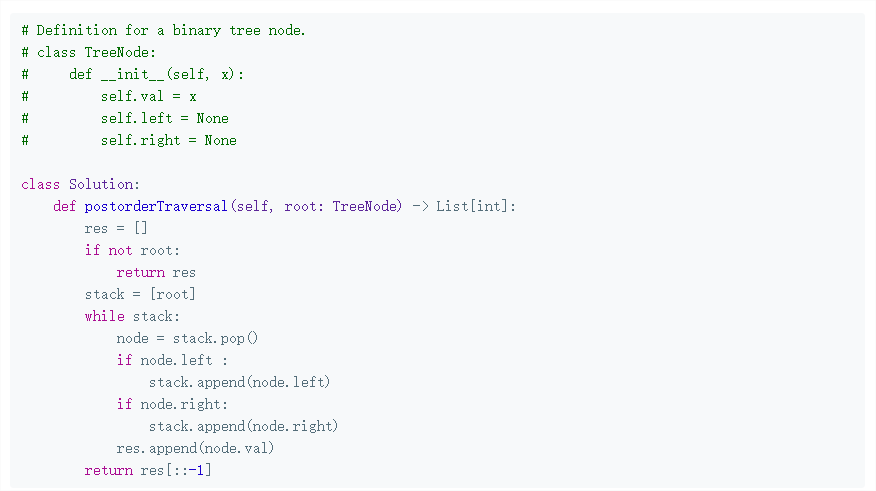
中序遍历：



前序遍历：迭代:使用栈来完成,我们先将根节点放入栈中,然后将其弹出,依次将该弹出的节点的右节点,和左节点,\*\*注意顺序,\*\*是右,左,为什么?因为栈是先入后出的,我们要先输出右节点,所以让它先进栈.



后序遍历：



1. **从前序中序建造二叉树**



1. **验证二叉搜索树**

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

节点的左子树只包含小于当前节点的数。

节点的右子树只包含大于当前节点的数。

所有左子树和右子树自身必须也是二叉搜索树。



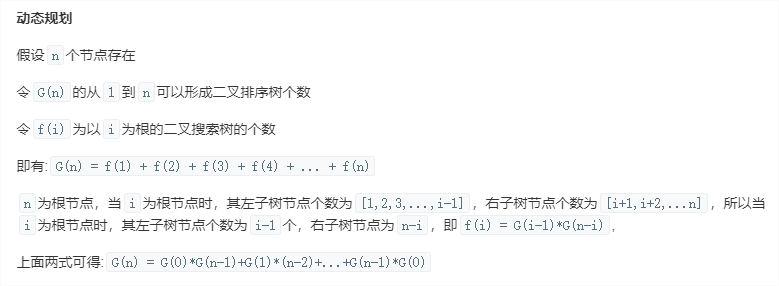
1. **将有序数组转为二叉搜索树**

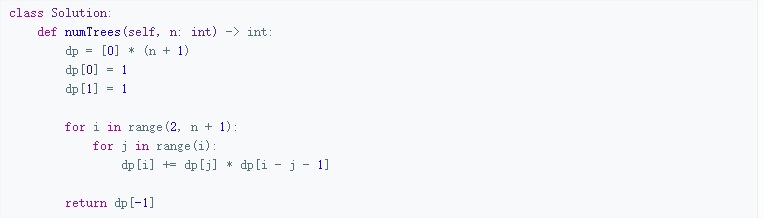
将一个按照升序排列的有序数组，转换为一棵高度平衡二叉搜索树。本题中，一个高度平衡二叉树是指一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过 1。



1. **不同的二叉搜索树**

给定一个整数 n，求以 1 ... n 为节点组成的二叉搜索树有多少种？

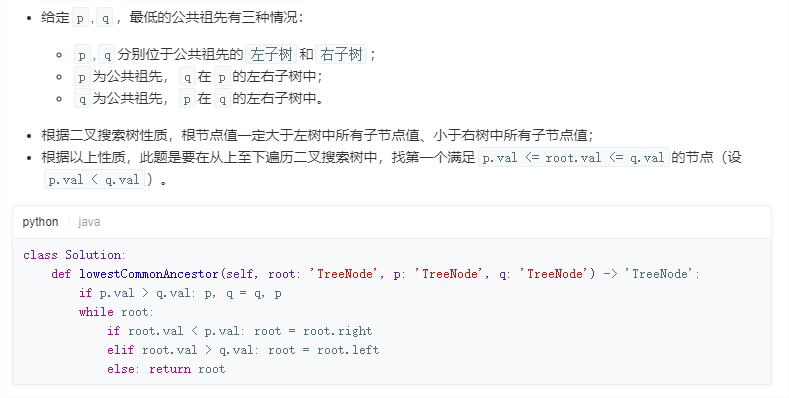




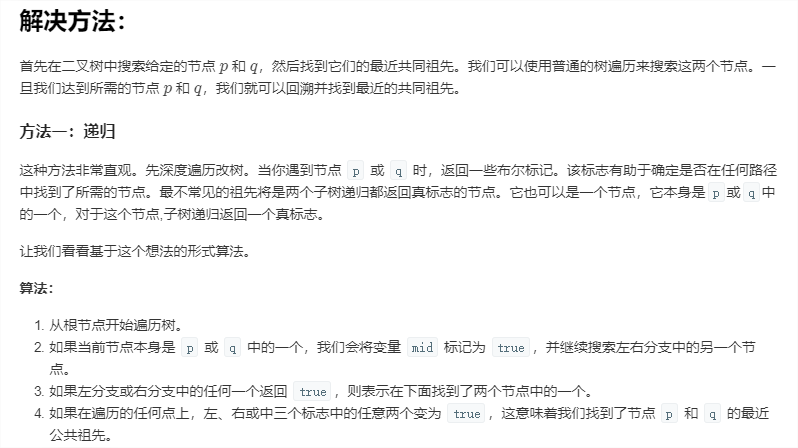
1. **二叉树第k小元素**

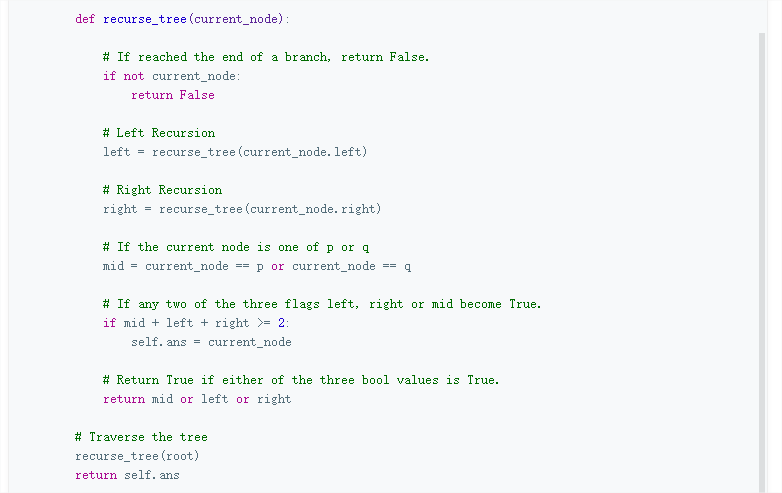


1. **二叉搜索树的最近公共祖先**



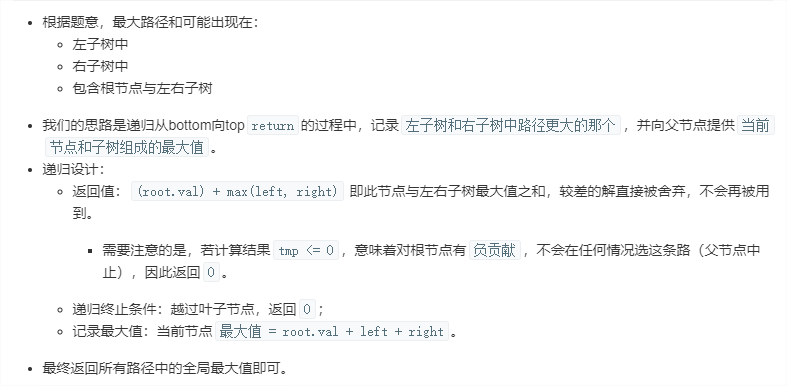
1. **二叉树的最近公共祖先**





1. **二叉树的最大路径和**

给定一个**非空**二叉树，返回其最大路径和。本题中，路径被定义为一条从树中任意节点出发，达到任意节点的序列。该路径**至少包含一个**节点，且不一定经过根节点。



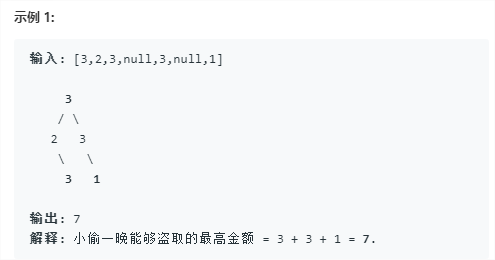


1. **二叉树转化为链表**



1. **打家劫舍3**

聪明的小偷意识到“这个地方的所有房屋的排列类似于一棵二叉树”。 如果两个直接相连的房子在同一天晚上被打劫，房屋将自动报警。计算在不触动警报的情况下，小偷一晚能够盗取的最高金额。





1. **二叉树的序列化与反序列化**

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

