算法分析和复杂性理论

第 8 次作业

张瀚文 2201212865

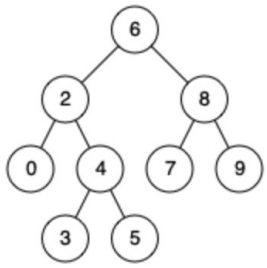
1 二叉搜索树的最近公共祖先（235）

题目描述：

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

例如，给定如下二叉搜索树：root = [6,2,8,0,4,7,9,null,null,3,5]



示例 1：

输入： root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出： 6
解释： 节点 2 和节点 8 的最近公共祖先是 6。

解题思路：
注意到题目中给出的是一棵「二叉搜索树」，因此我们可以快速地找出树中的某个节点以及从根节点到该节点的路径，例如我们需要找到节点 p：

我们从根节点开始遍历；
如果当前节点就是 p，那么成功地找到了节点；
如果当前节点的值大于 p 的值，说明 p 应该在当前节点的左子树，因此将当前节点移动到它的左子节点；
如果当前节点的值小于 p 的值，说明 p 应该在当前节点的右子树，因此将当前节点移动到它的右子节点。

对于节点 q 同理。在寻找节点的过程中，我们可以顺便记录经过的节点，这样就得到了从根节点到被寻找节点的路径。

当我们分别得到了从根节点到 p 和 q 的路径之后，我们就可以很方便地找到它们的最近公共祖先了。显然，p 和 q 的最近公共祖先就是从根节点到它们路径上的「分岔点」，也就是最后一个相同的节点。

运行结果：

测试代码：

```
class Solution:
    def lowestCommonAncestor(self, root: TreeNode, p: TreeNode, q: TreeNode) ->
TreeNode:
        def getPath(root: TreeNode, target: TreeNode) -> List[TreeNode]:
            path = list()
            node = root
            while node != target:
                path.append(node)
                if target.val < node.val:
                    node = node.left
                else:
                    node = node.right
            path.append(node)
            return path

        path_p = getPath(root, p)
        path_q = getPath(root, q)
        ancestor = None
        for u, v in zip(path_p, path_q):
            if u == v:
                ancestor = u
            else:
                break

        return ancestor
```

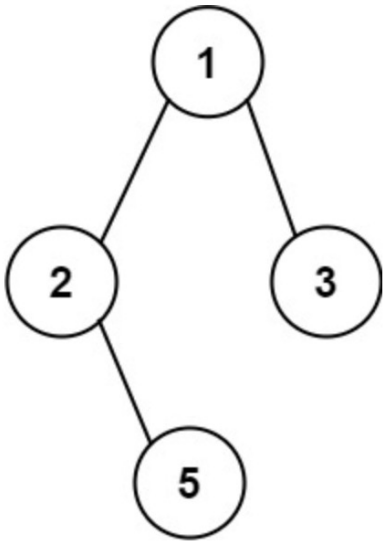
2 二叉树的所有路径（257）

题目描述：

给你一个二叉树的根节点 `root` ，按 **任意顺序** ，返回所有从根节点到叶子节点的路径。

叶子节点 是指没有子节点的节点。

示例 1：




```
输入：root = [1,2,3,null,5]
输出：["1->2->5","1->3"]
```

解题思路：

在深度优先搜索遍历二叉树时，我们需要考虑当前的节点以及它的孩子节点。
如果当前节点不是叶子节点，则在当前的路径末尾添加该节点，并继续递归遍历该节点的每一个孩子节点。
如果当前节点是叶子节点，则在当前路径末尾添加该节点后我们就得到了一条从根节点到叶子节点的路径，将该路径加入到答案即可。
如此，当遍历完整棵二叉树以后我们就得到了所有从根节点到叶子节点的路径。

运行结果：

测试代码：

```
# Definition for a binary tree node.
# class TreeNode:
```

```
#         def __init__(self, val=0, left=None, right=None):
#             self.val = val
#             self.left = left
#             self.right = right
class Solution:
    def binaryTreePaths(self, root):
        """
        :type root: TreeNode
        :rtype: List[str]
        """
        def construct_paths(root, path):
            if root:
                path += str(root.val)
                if not root.left and not root.right:  # 当前节点是叶子节点
                    paths.append(path)  # 把路径加入到答案中
                else:
                    path += '->'  # 当前节点不是叶子节点，继续递归遍历
                    construct_paths(root.left, path)
                    construct_paths(root.right, path)

        paths = []
        construct_paths(root, '')
        return paths
```

3 平衡二叉树（110）

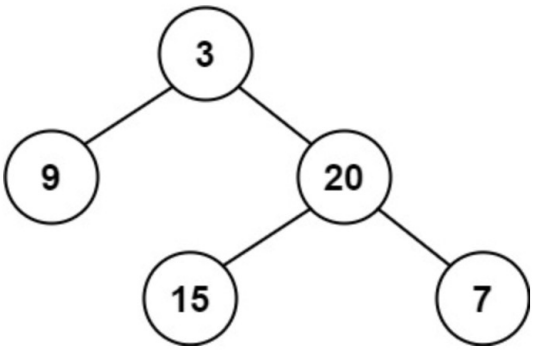
题目描述：

给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

> 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 。

示例 1：

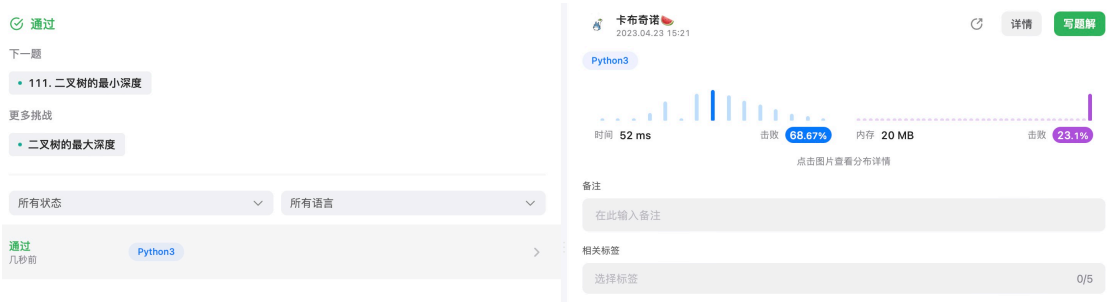



```
输入：root = [3,9,20,null,null,15,7]
输出：true
```

解题思路：

定义函数 height，用于计算二叉树中的任意一个节点 p 的高度。有了计算节点高度的函数，即可判断二叉树是否平衡。具体做法类似于二叉树的前序遍历，即对于当前遍历到的节点，首先计算左右子树的高度，如果左右子树的高度差是否不超过 1，再分别递归地遍历左右子节点，并判断左子树和右子树是否平衡。这是一个自顶向下的递归的过程。

运行结果：



测试代码：

```
# Definition for a binary tree node.
# class TreeNode:
#         def __init__(self, val=0, left=None, right=None):
#                 self.val = val
#                 self.left = left
#                 self.right = right
class Solution:
        def isBalanced(self, root: TreeNode) -> bool:
```

```
        def height(root: TreeNode) -> int:
            if not root:
                return 0
            return max(height(root.left), height(root.right)) + 1

        if not root:
            return True
        return abs(height(root.left) - height(root.right)) <= 1 and
self.isBalanced(root.left) and self.isBalanced(root.right)
```