94

给定一个二叉树，返回它的中序 遍历。

示例:

输入: [1,null,2,3]

1

\

2

/

3

输出: [1,3,2]

进阶: 递归算法很简单，你可以通过迭代算法完成吗？

题解：树的中序遍历，左孩子->自身->右孩子。

首先是递归解法：先去向左直到叶节点为空，然后记录父节点值，最后遍历右孩子。

class Solution:

    def inorderTraversal(self, root: TreeNode) -> List[int]:

        res = []

        if root is None:

            return res

        self.mid\_print(root, res)

        return res

    def mid\_print(self, root, res):

        if root is None:

            return

        self.mid\_print(root.left, res)

        res.append(root.val)

        self.mid\_print(root.right, res)

然后是迭代解法：迭代解法需要首先确定循环结束的条件，所有节点遍历完毕。通过使用栈结构来模拟递归函数的实现。根节点放入栈中，栈为空时表示遍历完成。首先判断左孩子是否存在，若存在则放入栈，继续查找左孩子，直到为空，然后存入本节点的值，再继续当前节点的右孩子，如果存在，右孩子的中序仍旧从左孩子开始，所以只需要遍历右孩子即可。从栈中弹出的节点左孩子遍历过但是仍存在，需要标记已经遍历的结果，或者删除已有的树结构

res = []

        if root is None:

            return res

        stack = [root]

        p = root

        while stack:

            while p.left is not None:

                stack.append(p.left)

                p = p.left

            p = stack.pop(-1)

            p.left = None

            res.append(p.val)

            if p.right is not None:

                p = p.right

                stack.append(p)

        return res

96

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？

示例:

输入: 3

输出: 5

解释:

给定 n = 3, 一共有 5 种不同结构的二叉搜索树:

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

题解：动态规划问题,二叉搜索树：左孩子小于父节点的值，右孩子大于父节点的值。设定dp[n]是1到n的二叉搜索树的不同个数。

Dp[i]=dp[左子树节点个数]+dp[右子树节点个数]。

例如，dp[3]=dp[2]\*dp[0]+dp[1]\*dp[1]+dp[0]\*dp[2]

class Solution:

    def numTrees(self, n: int) -> int:

        res = [0 for \_ in range(n+1)]

        res[0] = 1

        for i in range(1, n+1):

            left = i-1

            right = 0

            while left >= 0:

                res[i] += res[left] \* res[right]

                left -= 1

                right += 1

        return res[n]

根据上述情况进行时间复杂度的优化：因为在计算dp[i]时，dp[0]\*dp[i-1]计算两次，所以i为奇数时，子树节点从0到i-1均重复计算2次；i为偶数时，只有i//2的节点数计算1次。

class Solution:

    def numTrees(self, n: int) -> int:

        res = [0 for \_ in range(n+1)]

        res[0] = 1

        for i in range(1, n+1):

            left = 0

            while left < i // 2:

                res[i] += 2 \* res[left] \* res[i - left - 1]

                left += 1

            if (i+1) % 2 == 0:

                res[i] += res[i // 2] \* res[i // 2]

        print(res)

        return res[n]

上面是一个卡特兰数,

98

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。

示例 1:

输入:

2

/ \

1 3

输出: true

示例 2:

输入:

5

/ \

1 4

  / \

  3 6

输出: false

解释: 输入为: [5,1,4,null,null,3,6]。

  根节点的值为 5 ，但是其右子节点值为 4 。

题解：中序遍历二叉树，对生成的序列判断是否递增。

class Solution:

    def isValidBST(self, root: TreeNode) -> bool:

        res = []

        if root is None:

            return True

        self.mid\_print(root, res)

        for i in range(1, len(res)):

            if res[i] <= res[i-1]:

                return False

        return True

    def mid\_print(self, root, res):

        if root is None:

            return

        self.mid\_print(root.left, res)

        res.append(root.val)

        self.mid\_print(root.right, res)

100

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1:

输入: 1 1

/ \ / \

2 3 2 3

[1,2,3], [1,2,3]

输出: true

示例 2:

输入: 1 1

/ \

2 2

[1,2], [1,null,2]

输出: false

示例 3:

输入: 1 1

/ \ / \

2 1 1 2

[1,2,1], [1,1,2]

输出: false

题解：判断两个二叉树是否相同，同时遍历两棵树，存在一点不一样就是不同。

如果两个节点相同，就判断左孩子和右孩子结果的and结果，因为要同时满足；否则就不相同

class Solution:

    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:

        return self.print(p, q)

    def print(self, root1, root2):

        if root1 is None and root2 is None:

            return True

        elif root1 is not None and root2 is not None:

            if root1.val == root2.val:

                return self.print(root1.left, root2.left) and self.print(root1.right, root2.right)

            else:

                return False

        else:

            return False

101

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 [1,2,2,3,4,4,3] 是对称的。

1

/ \

2 2

/ \ / \

3 4 4 3

但是下面这个 [1,2,2,null,3,null,3] 则不是镜像对称的:

1

/ \

2 2

\ \

3 3

说明:

如果你可以运用递归和迭代两种方法解决这个问题，会很加分。

题解：是否对称只需要拿到中序遍历，然后判断这个序列是否是对称的，发现中序可能存在结构的不同。和判断两棵树是否相同类似，将根节点拆分成两个子树，判断子树的对称位置是否相同，只需在上题的基础上拆分左右子树并且修改递归比较节点的位置。

class Solution:

    def isSymmetric(self, root: TreeNode) -> bool:

        if root is None:

            return True

        p = root.left

        q = root.right

        return self.print(p, q)

    def print(self, root1, root2):

        if root1 is None and root2 is None:

            return True

        elif root1 is not None and root2 is not None:

            if root1.val == root2.val:

                return self.print(root1.left, root2.right) and self.print(root1.right, root2.left)

            else:

                return False

        else:

            return False

102

给定一个二叉树，返回其按层次遍历的节点值。 （即逐层地，从左到右访问所有节点）。

例如:

给定二叉树: [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回其层次遍历结果：

[

[3],

[9,20],

[15,7]

]

题解：遍历节点时记录节点的深度，采用任意序遍历，以深度为hash值保存节点值，最终将深度升序排列得到结果。

class Solution:

    def levelOrder(self, root: TreeNode) -> List[List[int]]:

        if root is None:

            return []

        from collections import defaultdict

        d = defaultdict(list)

        depth = 0

        self.print(root, d, depth)

        res = []

        for key in d.keys():

            res.append(d[key])

        return res

    def print(self, root, d, depth):

        if root is None:

            return

        d[depth].append(root.val)

        self.print(root.left, d, depth+1)

        self.print(root.right, d, depth+1)

给定一个整数 n，生成所有由 1 ... n 为节点所组成的二叉搜索树。

示例:

输入: 3

输出:

[

  [1,null,3,2],

  [3,2,null,1],

  [3,1,null,null,2],

  [2,1,3],

  [1,null,2,null,3]

]

解释:

以上的输出对应以下 5 种不同结构的二叉搜索树：

1 3 3 2 1

\ / / / \ \

3 2 1 1 3 2

/ / \ \

2 1 2 3

题解：给定n，那么1-n都可以作为树的根节点，应该遍历生成相应树。在选择i为根节点时，左子树的节点值为1-i-1，右子树的节点值为i+1-n,然后将左子树、右子树和根节点i合并得到完整的树。

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def generateTrees(self, n: int) -> List[TreeNode]:

        def generate\_tree(start, end):

            if start > end:

                return [None, ]

            all\_trees = []

            for i in range(start, end+1):

                left\_trees = generate\_tree(start, i-1)

                right\_trees = generate\_tree(i+1, end)

                for l in left\_trees:

                    for r in right\_trees:

                        node = TreeNode(i)

                        node.left = l

                        node.right = r

                        all\_trees.append(node)

            return all\_trees

        return generate\_tree(1, n) if n else []

思路的主要关键点是：1.统计1-n所有值为根节点的树 2.i为节点的时候，左右子树节点值是固定范围的，所以可以递归完成子树的构建，并且得到的子树一定是符合条件的 3.递归结束条件要用None表示当前节点值不存在，而不是省去，这样体现树的结构。

99恢复二叉搜索树

二叉搜索树中的两个节点被错误地交换。

请在不改变其结构的情况下，恢复这棵树。

示例 1:

输入: [1,3,null,null,2]

  1

  /

 3

  \

  2

输出: [3,1,null,null,2]

  3

  /

 1

  \

  2

示例 2:

输入: [3,1,4,null,null,2]

3

/ \

1 4

  /

  2

输出: [2,1,4,null,null,3]

2

/ \

1 4

  /

 3

进阶:

使用 O(n) 空间复杂度的解法很容易实现。

你能想出一个只使用常数空间的解决方案吗？

题解：

103

给定一个二叉树，返回其节点值的锯齿形层次遍历。（即先从左往右，再从右往左进行下一层遍历，以此类推，层与层之间交替进行）。

例如：

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回锯齿形层次遍历如下：

[

[3],

[20,9],

[15,7]

]

题解：在层次遍历的基础上，偶数深度逆序输出。

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:

        if root is None:

            return []

        from collections import defaultdict

        d = defaultdict(list)

        depth = 0

        self.print(root, d, depth)

        res = []

        for key in d.keys():

            if key % 2 == 1:

                res.append(d[key][::-1])

            else:

                res.append(d[key])

        return res

    def print(self, root, d, depth):

        if root is None:

            return

        d[depth].append(root.val)

        self.print(root.left, d, depth + 1)

        self.print(root.right, d, depth + 1)

104 二叉树的最大深度

给定一个二叉树，找出其最大深度。

二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

说明: 叶子节点是指没有子节点的节点。

示例：

给定二叉树 [3,9,20,null,null,15,7]，

3

/ \

9 20

/ \

15 7

返回它的最大深度 3 。

题解：递归搜索当前节点的左子树和右子树深度，取其中最大的深度为结果。递归终止条件：当前节点为空，返回当前深度为d-1

class Solution:

    def maxDepth(self, root: TreeNode) -> int:

        if root is None:

            return 0

        depth = 1

        res = max(self.print(root.left, depth+1), self.print(root.right, depth+1))

        return res

    def print(self, root, depth):

        if root is None:

            return depth - 1

        return max(self.print(root.left, depth + 1), self.print(root.right, depth + 1))

108 有序数组转换为二叉搜索树

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。

示例:

给定有序数组: [-10,-3,0,5,9],

一个可能的答案是：[0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：

0

/ \

-3 9

/ /

-10 5

题解：初步想法：每次构建树的根节点时选择序列中心的点。例如选择索引为i的点做根节点，则0-i-1索引为左子树，递归构建左子树根节点为i-1//2的位置。i+1到n-1为右子树，(n-i-1)//2+ i位置为右子树的根。

class Solution:

    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:

        if len(nums) == 0:

            return None

        root = TreeNode(nums[len(nums)//2])

        l = self.sortedArrayToBST(nums[:len(nums)//2])

        r = self.sortedArrayToBST(nums[len(nums)//2+1:])

        root.left = l

        root.right = r

        return root

105 从前序和中序遍历序列构造二叉树

根据一棵树的前序遍历与中序遍历构造二叉树。

注意:

你可以假设树中没有重复的元素。

例如，给出

前序遍历 preorder = [3,9,20,15,7]

中序遍历 inorder = [9,3,15,20,7]

返回如下的二叉树：

3

/ \

9 20

/ \

15 7

题解：根据前序和中序遍历构造二叉树，首先前序的首元素是当前树的根节点，然后找到中序中相应元素的位置，则此元素之前的序列为左子树，之后为右子树。然后递归构造。

 if len(preorder) == 0:

            return None

        root = TreeNode(preorder[0])

        index = -1

        for i in range(len(inorder)):

            if inorder[i] == preorder[0]:

                index = i

                break

        left = self.maxProfit(preorder[1:index+1], inorder[:index])

        right = self.maxProfit(preorder[index+1:], inorder[index+1:])

        root.left = left

        root.right = right

        return root

106中序和后序遍历构造二叉树

根据一棵树的中序遍历与后序遍历构造二叉树。

注意:

你可以假设树中没有重复的元素。

例如，给出

中序遍历 inorder = [9,3,15,20,7]

后序遍历 postorder = [9,15,7,20,3]

返回如下的二叉树：

3

/ \

9 20

/ \

15 7

题解：和上题类似，只不过把前序的第一个作为根节点变为后序的最后一个作为根节点。

class Solution:

    def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:

        if len(inorder) == 0:

            return None

        root = TreeNode(postorder[-1])

        index = -1

        for i in range(len(inorder)):

            if inorder[i] == root.val:

                index = i

                break

        left = self.buildTree(inorder[:index], postorder[:index])

        right = self.buildTree(inorder[index+1:], postorder[index:-1])

        root.left = left

        root.right = right

        return root

107 二叉树的层次遍历

给定一个二叉树，返回其节点值自底向上的层次遍历。 （即按从叶子节点所在层到根节点所在的层，逐层从左向右遍历）

例如：

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回其自底向上的层次遍历为：

[

[15,7],

[9,20],

[3]

]

题解：参考层次遍历

class Solution:

    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:

        if root is None:

            return []

        from collections import defaultdict

        d = defaultdict(list)

        depth = 0

        self.print(root, d, depth)

        res = []

        for key in d.keys():

            res.insert(0, d[key])

        return res

    def print(self, root, d, depth):

        if root is None:

            return

        d[depth].append(root.val)

        self.print(root.left, d, depth + 1)

        self.print(root.right, d, depth + 1)

111 二叉树的最小深度

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明: 叶子节点是指没有子节点的节点。

示例:

给定二叉树 [3,9,20,null,null,15,7],

3

/ \

9 20

/ \

15 7

返回它的最小深度  2

题解：参考最大深度。有一种特殊情况是左子树或右子树不存在的情况，这种情况下就要只计算存在的子树的深度。

class Solution:

    def minDepth(self, root: TreeNode) -> int:

        if root is None:

            return 0

        depth = 1

        if root.left is None:

            res = self.print(root.right, depth + 1)

        elif root.right is None:

            res = self.print(root.left, depth + 1)

        else:

            res = min(self.print(root.left, depth + 1), self.print(root.right, depth + 1))

        return res

    def print(self, root, depth):

        if root is None:

            return depth - 1

        if root.left is None:

            res = self.print(root.right, depth + 1)

        elif root.right is None:

            res = self.print(root.left, depth + 1)

        else:

            res = min(self.print(root.left, depth + 1), self.print(root.right, depth + 1))

        return res

112 路径总和

给定一个二叉树和一个目标和，判断该树中是否存在根节点到叶子节点的路径，这条路径上所有节点值相加等于目标和。

说明: 叶子节点是指没有子节点的节点。

示例:

给定如下二叉树，以及目标和 sum = 22，

5

/ \

4 8

/ / \

11 13 4

/ \ \

7 2 1

返回 true, 因为存在目标和为 22 的根节点到叶子节点的路径 5->4->11->2

题解：递归找到叶子节点，然后计算当前和是否为sum。在递归过程中，随着递归的深入，要在传参中更改当前和。

class Solution:

    def hasPathSum(self, root: TreeNode, sum: int) -> bool:

        def has(r, s):

            if r.left is None and r.right is None:

                if s+r.val == sum:

                    return True

                else:

                    return False

            elif r.left is None:

                return has(r.right, s+r.val)

            elif r.right is None:

                return has(r.left, s+r.val)

            else:

                return has(r.left, s+r.val) or has(r.right, s+r.val)

        if root is None:

            return False

        return has(root, 0)

110 平衡二叉树

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:

给定二叉树 [3,9,20,null,null,15,7]

3

/ \

9 20

/ \

15 7

返回 true 。

示例 2:

给定二叉树 [1,2,2,3,3,null,null,4,4]

1

/ \

2 2

/ \

3 3

/ \

4 4

返回 false 。

题解：对当前根节点，递归求出左子树的高度和右子树的高度，判断高度差时候为小于1。递归结束条件，如果当前节点为空，返回深度和True（递归到该叶节点意味着到该节点时满足平衡），然后比较左右子树高度差是否满足条件，并返回和左子树、右子树的平衡判断情况，子树深度表示为左右子树中的最大深度。

class Solution:

    def isBalanced(self, root: TreeNode) -> bool:

        def isB(r, depth):

            if r is None:

                return depth - 1, True

            l\_depth, f1= isB(r.left, depth+1)

            r\_depth, f2 = isB(r.right, depth+1)

            return max(l\_depth, r\_depth), f1 and f2 and abs(l\_depth - r\_depth) <= 1

        if root is None:

            return True

        res, flag = isB(root, 0)

        return flag

113 路径总和

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。

示例:

给定如下二叉树，以及目标和 sum = 22，

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

返回:

[

[5,4,11,2],

[5,8,4,5]

]

题解：找到根节点到叶子节点总和等于给定的所有路径，采用DFS的思想，在递归过程中记录当前路径和和，如果最后的和为目标值，把当前路径加入到结果中，并且将最后一个节点弹出，表示该节点已经搜索过。如果和已经大于目标，表示不可能存在路径，剪枝。

注意：1.python中list.append()不能当作函数参数传递，会报NoneType

2.python中参数传递list、dict都是传值引用，对参数的修改会导致最终结果的修改，可以使用copy（）。

class Solution:

    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:

        res = []

        def dfs(current\_root, current\_sum, current\_path):

            if current\_root.left is None and current\_root.right is None:

                if current\_sum + current\_root.val == sum:

                    current\_path.append(current\_root.val)

                    res.append(current\_path)

                return

            current\_sum += current\_root.val

            current\_path.append(current\_root.val)

            if current\_root.left is not None:

                dfs(current\_root.left, current\_sum, current\_path.copy())

            if current\_root.right is not None:

                dfs(current\_root.right, current\_sum, current\_path.copy())

            current\_sum -= current\_root.val

            current\_path.pop(-1)

        if root is None:

            return []

        path = []

        dfs(root, 0, path)

        return res

114 二叉树展开为链表

给定一个二叉树，原地将它展开为链表。

例如，给定二叉树

1

/ \

2 5

/ \ \

3 4 6

将其展开为：

1

\

2

\

3

\

4

\

5

\

6

题解：空间O(N)的办法：存储二叉树的前序遍历结果，对生成的结果构造链表。

注意：对root的修改，需要删除原有二叉树的左子树，因为结果只需要右子树。

class Solution:

    def flatten(self, root: TreeNode) -> None:

        """

        Do not return anything, modify root in-place instead.

        """

        res = []

        def pre\_order(r):

            if r is None:

                return

            res.append(r.val)

            pre\_order(r.left)

            pre\_order(r.right)

        pre\_order(root)

        # print(res)

        pre\_node = TreeNode(0)

        pre\_node.right = root

        for i in range(1, len(res)):

            root.left = None

            root.right = TreeNode(res[i])

            root = root.right

        root = pre\_node.right

116 填充每个节点的下一个右侧节点指针

给定一个完美二叉树，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

struct Node {

int val;

Node \*left;

Node \*right;

Node \*next;

}

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。。

提示：

你只能使用常量级额外空间。

使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

题解：O(N)空间复杂度做法：找到树的层次遍历，然后对每一层的节点添加next为该层的下一个节点。

class Solution:

def connect(self, root: 'Node') -> 'Node':

from collections import defaultdict

d = defaultdict(list)

depth = 0

self.print(root, d, depth)

# res = []

for key in d.keys():

length = len(d[key])

for i in range(length-1):

d[key][i].next = d[key][i+1]

return root

def print(self, root, d, depth):

if root is None:

return

d[depth].append(root)

self.print(root.left, d, depth + 1)

self.print(root.right, d, depth + 1)

O(1)空间复杂度做法：对于每个节点，当父节点存在时，左孩子的next是父节点的右孩子，右孩子的next是父节点的next的左孩子。递归完成设置，递归结束条件是该节点为叶子节点。但是在leetcode上出现超时错误，不知道为什么，本地同样的测试样例没有问题。

class Solution:

    def connect(self, root: 'Node') -> 'Node':

        def set\_next(parent, child):

            if child is None or parent is None:

                return

            if child.val == parent.left.val:

                child.next = parent.right

            else:

                if parent.next is not None:

                    child.next = parent.next.left

            if child.left is not None:

                set\_next(child, child.left)

                set\_next(child, child.right)

        if root is None:

            return root

        set\_next(root, root.left)

        set\_next(root, root.right)

        return root

199 二叉树的右视图

给定一棵二叉树，想象自己站在它的右侧，按照从顶部到底部的顺序，返回从右侧所能看到的节点值。

示例:

输入: [1,2,3,null,5,null,4]

输出: [1, 3, 4]

解释:

1 <---

/ \

2 3 <---

\ \

5 4 <---

题解：输出每层最右侧节点，最简单的O(N)方法：记录层次遍历的结果然后输出每层的最右侧节点。

class Solution:

    def rightSideView(self, root: TreeNode) -> List[int]:

        if root is None:

            return []

        from collections import defaultdict

        d = defaultdict(list)

        depth = 0

        self.print(root, d, depth)

        res = []

        for key in d.keys():

            res.append(d[key][-1])

        return res

    def print(self, root, d, depth):

        if root is None:

            return

        d[depth].append(root.val)

        self.print(root.left, d, depth + 1)

        self.print(root.right, d, depth + 1)

173 二叉搜索树迭代器

实现一个二叉搜索树迭代器。你将使用二叉搜索树的根节点初始化迭代器。

调用 next() 将返回二叉搜索树中的下一个最小的数。

示例：

BSTIterator iterator = new BSTIterator(root);

iterator.next(); // 返回 3

iterator.next(); // 返回 7

iterator.hasNext(); // 返回 true

iterator.next(); // 返回 9

iterator.hasNext(); // 返回 true

iterator.next(); // 返回 15

iterator.hasNext(); // 返回 true

iterator.next(); // 返回 20

iterator.hasNext(); // 返回 false

提示：

next() 和 hasNext() 操作的时间复杂度是 O(1)，并使用 O(h) 内存，其中 h 是树的高度。

你可以假设 next() 调用总是有效的，也就是说，当调用 next() 时，BST 中至少存在一个下一个最小的数。

题解：先得到中序遍历的结果，然后实现一个指针记录当前的节点位置，根据该指针得到下一个节点的值以及是否还有满足条件的下一个结果。

class BSTIterator:

    def \_\_init\_\_(self, root: TreeNode):

        self.nodes = []

        self.index = -1

        self.inorder(root)

    def inorder(self, root):

        if root is None:

            return

        self.inorder(root.left)

        self.nodes.append(root.val)

        self.inorder(root.right)

    def next(self) -> int:

        """

        @return the next smallest number

        """

        self.index += 1

        return self.nodes[self.index]

    def hasNext(self) -> bool:

        """

        @return whether we have a next smallest number

        """

        if self.index + 1 < len(self.nodes):

            return True

        return False

226 翻转二叉树

翻转一棵二叉树。

示例：

输入：

4

/ \

2 7

/ \ / \

1 3 6 9

输出：

4

/ \

7 2

/ \ / \

9 6 3 1

题解：一个显著的做法：如果当前节点有左孩子和右孩子，那么互换孩子的位置即可。然后递归继续寻找，直到叶子节点停止。递归条件为该节点不为空。

class Solution:

    def invertTree(self, root: TreeNode) -> TreeNode:

        def inverse(r):

            if r.left is None and r.right is None:

                return

            r.left, r.right = r.right, r.left

            if r.left:

                inverse(r.left)

            if r.right:

                inverse(r.right)

        if root is None:

            return None

        inverse(root)

        return root

222完全二叉树的节点个数

给出一个完全二叉树，求出该树的节点个数。

说明：

完全二叉树的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2h 个节点。

示例:

输入:

1

/ \

2 3

/ \ /

4 5 6

输出: 6

题解：O(N)解法，遍历所有节点统计节点个数。

class Solution:

    def countNodes(self, root: TreeNode) -> int:

        if root is None:

            return 0

        from collections import defaultdict

        d = defaultdict(list)

        depth = 0

        self.print(root, d, depth)

        res = 0

        for key in d.keys():

            res += len(d[key])

        return res

    def print(self, root, d, depth):

        if root is None:

            return

        d[depth].append(root.val)

        self.print(root.left, d, depth + 1)

        self.print(root.right, d, depth + 1)

可以通过，但是这样的解法就浪费了题目的一个条件：这是一颗完全二叉树，也就是我只要找到最下一层最右侧节点的位置，就可以找到节点的个数。如何找到最底层最右侧节点的位置呢？首先找到树的深度，因为最左子树一直存在，所以只需向左一直遍历即可得到深度。其次是找到最右节点。在确定二叉树的深度之后，最下层的节点个数范围为1-2^d，d从0开始计算，从而可以使用二分搜索的方法来找到最右节点。为最下层节点编号1-2^d，left, right = 0, 2^d-1，mid = (right + left ) // 2，如果mid存在，则继续向右寻找，如果不存在，则向左寻找。复杂度为O(N2)

class Solution:

    def countNodes(self, root: TreeNode) -> int:

        def get\_depth(r):

            d = 0

            while r.left is not None:

                d += 1

                r = r.left

            return d

#向最深层遍历，如果idx存在，则说明最深层节点至少有idx个节点，遍历方法为从根节点开始，每次向下深入，直到叶子节点，如果idx <= mid=(l+r)//2，说明要找的idx节点在左子树，r=r.left，否则向右,同时令l=m+1保证下一次循环开始不再考虑m的节点。返回节点是否存在

        def find\_deepst(r, idx, d):

            left, right = 0, 2\*\*d-1

            for \_ in range(d):

                mid = left + (right - left) // 2

                if idx <= mid:

                    r = r.left

                    right = mid

                else:

                    r = r.right

                    left = mid+1

            return r is not None

        if root is None:

            return 0

        d = get\_depth(root)

        l, rr = 1, 2\*\*d-1

        while l <= rr:

            m = l + (rr - l) // 2

#如果节点存在，表示至少m左边的节点都是存在的，继续向右查找，否则向左查找。

            if find\_deepst(root, m, d):

                l = m +1

            else:

                rr = m – 1

#最终节点是d-1层的所有节点加上最深层的节点个数。

        return 2\*\*d-1 + l

129 求根到叶子节点数字之和

给定一个二叉树，它的每个结点都存放一个 0-9 的数字，每条从根到叶子节点的路径都代表一个数字。

例如，从根到叶子节点路径 1->2->3 代表数字 123。

计算从根到叶子节点生成的所有数字之和。

说明: 叶子节点是指没有子节点的节点。

示例 1:

输入: [1,2,3]

1

/ \

2 3

输出: 25

解释:

从根到叶子节点路径 1->2 代表数字 12.

从根到叶子节点路径 1->3 代表数字 13.

因此，数字总和 = 12 + 13 = 25.

示例 2:

输入: [4,9,0,5,1]

4

/ \

9 0

 / \

5 1

输出: 1026

解释:

从根到叶子节点路径 4->9->5 代表数字 495.

从根到叶子节点路径 4->9->1 代表数字 491.

从根到叶子节点路径 4->0 代表数字 40.

因此，数字总和 = 495 + 491 + 40 = 1026.

题解：递归做法，每深入一层只需要让当前值\*10+根节点，达到叶子节点时统计和。

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def sumNumbers(self, root: TreeNode) -> int:

        if root is None:

            return 0

        def get\_sum(root, cur\_val):

            if root.left is None and root.right is None:

                return cur\_val\*10 + root.val

            ans = 0

            if root.left is not None:

                ans += get\_sum(root.left, cur\_val\*10+root.val)

            if root.right is not None:

                ans += get\_sum(root.right, cur\_val \* 10 + root.val)

            return ans

            # return get\_sum(root.left, cur\_val\*10+root.val) + get\_sum(root.right, cur\_val\*10+root.val)

        ans = get\_sum(root, 0)

        return ans

144 二叉树的前序遍历

给定一个二叉树，返回它的 前序 遍历。

 示例:

输入: [1,null,2,3]

1

\

2

/

3

输出: [1,2,3]

进阶: 递归算法很简单，你可以通过迭代算法完成吗？

题解：前序遍历：根节点->左子树->右子树。和中序遍历不同的是，中序遍历需要首先找到最左的叶子节点，然后回溯找到根节点（栈返回），再去找根节点的右子树。而前序遍历首先记录根节点（需要从栈中取出根节点），找到最左子节点，并继续找出最左子节点的右子树（如果存在）。需要注意的点是：1.在寻找右子树的过程中，如果当前节点存在左子树，则需要暂停右子树遍历，优先遍历左子树。2.从栈中取出根节点后先找左子树，但是当需要找右子树时需要再次取出根节点，所以需要在转向左子树的时候，将根节点存入栈中以便遍历右子树可以得到根节点。3.根节点的左右子树遍历完成后不再需要遍历，将根节点的左或右置为空，表示当前节点的左右子树已遍历，这样从栈中再次取出根节点时不会重复遍历。

# Definition for a binary tree node.

# class TreeNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.left = None

#         self.right = None

class Solution:

    def preorderTraversal(self, root: TreeNode) -> List[int]:

        res = []

        if root is None:

            return res

        stack = [root, root]

        p = root

        res.append(p.val)

        while stack:

            # res.append(stack[0].val)

            p = stack.pop(-1)

            while p.left is not None:

                stack.append(p.left)

                res.append(p.left.val)

                tmp = p.left

                p.left = None

                p = tmp

            # p.left = None

            while p.right is not None:

                tmp = p.right

                p.right = None

                p = tmp

                res.append(p.val)

                stack.append(p)

                if p.left is not None:

                    stack.append(p)

                    break

            # stack.pop(-1)

            # p.right = None

        return res