Leetcode(python) 树简单题

100相同的树

题目：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

思路：先判断是否为空，然后递归左右子树

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** isSameTree(self, p, q):
10. """
11. :type p: TreeNode
12. :type q: TreeNode
13. :rtype: bool
14. """
15. **if** **not** p **and** **not** q:
16. **return** True
17. **if** p **and** q **and** q.val==p.val:
18. l=self.isSameTree(p.left,q.left)
19. r=self.isSameTree(p.right,q.right)
20. **return** l **and** r
21. **else**:
22. **return** False

101对称二叉树

题目：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

思路：也是考虑到递归来做

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** isSymmetric(self, root):
10. """
11. :type root: TreeNode
12. :rtype: bool
13. """
14. **def** solve(p,q):
15. **if** p==None:
16. **return** q==None
17. **if** q==None:
18. **return** p==None
19. **if** p.val==q.val:
20. **return** solve(p.left,q.right) **and** solve(p.right,q.left)
21. **return** False
23. **if** root==None:
24. **return** True
25. **else**:
26. **return** solve(root.left,root.right)

104二叉树的最大深度

题目：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

思路：继续使用递归

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** maxDepth(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **if** **not** root:
15. **return** 0
16. **else**:
17. l=1+self.maxDepth(root.left)
18. r=1+self.maxDepth(root.right)
19. **return** max(l,r)

107二叉树的层次遍历II

题目：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]

]

思路：利用queue

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** levelOrderBottom(self, root):
10. """
11. :type root: TreeNode
12. :rtype: List[List[int]]
13. """
14. **if** **not** root:
15. **return** []
16. queue=[root]
17. res=[]
18. **while** queue:
19. nodes\_value=[]
20. l=len(queue)
21. **for** i **in** range(l):
22. node=queue.pop(0)
23. **if** node.left:
24. queue.append(node.left)
25. **if** node.right:
26. queue.append(node.right)
27. nodes\_value.append(node.val)
28. res.append(nodes\_value)
29. **return** res[::-1]

108将有序数组转换成二叉搜索树

题目：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

思路：想到了中序遍历，将中位数为根节点，中位数左边为左子树，右边为右子树

代码：

1. #Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** sortedArrayToBST(self, nums):
10. """
11. :type nums: List[int]
12. :rtype: TreeNode
13. """
14. size = len(nums)
15. **if** size == 0:
16. **return** None
17. **if** size == 1:
18. **return** TreeNode(nums[0])
19. size //= 2
20. root = TreeNode(nums[size])
21. root.left = self.sortedArrayToBST(nums[:size])
22. root.right = self.sortedArrayToBST(nums[size + 1:])
23. **return** root

110平衡二叉树

题目：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 。

思路：使用递归，看下左右子树是否高度相差不超过1

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** isBalanced(self, root):
10. """
11. :type root: TreeNode
12. :rtype: bool
13. """
14. **if** root==None:
15. **return** True
16. **elif** abs(self.height(root.left)-self.height(root.right))>1:
17. **return** False
18. **else**:
19. **return** self.isBalanced(root.left) **and** self.isBalanced(root.right)
21. **def** height(self,root):
22. **if** root==None:
23. **return** 0
24. **else**:
25. **return** max(self.height(root.left),self.height(root.right))+1

111二叉树的最小深度

题目：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.

思路：递归有以下几种情况：

1.根节点为空，深度为0

2.只有一个根节点。深度为1

3.左右子树皆不空，则返回1+左右子树中最小的深度。

4.左子树为空，则返回1+右子树深度。这里可能有点难以理解，可以想象成此时只有根节点a,以及其右子树b，此时最小深度为2。

5.右子树为空，则返回1+左子树深度。同上分析。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** minDepth(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **if** root **is** None:
15. **return** 0
16. **if** root.left==None **and** root.right==None:
17. **return** 1
18. **elif** root.left **is** None:
19. **return** 1+self.minDepth(root.right)
20. **elif** root.right **is** None:
21. **return** 1+self.minDepth(root.left)
22. **else**:
23. **return** 1+min(self.minDepth(root.right),self.minDepth(root.left))

112路径总和

题目：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。

思路：

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** hasPathSum(self, root, sum):
10. """
11. :type root: TreeNode
12. :type sum: int
13. :rtype: bool
14. """
15. **if** root **is** None:
16. **return** False
17. **if** sum==root.val **and** root.left **is** None **and** root.right **is** None:
18. **return** True
19. **return** self.hasPathSum(root.left,sum-root.val) **or** self.hasPathSum(root.right,sum-root.val)

226翻转二叉树

题目：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

思路：从图上看是要把每棵树的左右子树互换，所以可以用递归快速解决问题。对于任意一个结点，先将其左子树调用当前函数完成翻转，再将其右子树调用当前函数完成翻转。然后交换左右子树，返回这个结点即可。递归的终点是判断给的结点值是否为空，如果为空说明到达叶结点的子树了，直接返回空就行了。代码中可写的稍微简练些，先记录左子树，然后让右子树完成翻转后赋给左子树，左子树完成翻转后赋给右子树。Python中可直接两个变量一起赋值，无需记录。

代码：（运行速度很慢）

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** invertTree(self, root):
10. """
11. :type root: TreeNode
12. :rtype: TreeNode
13. """
14. **if** root==None:
15. **return** None
16. root.left,root.right=self.invertTree(root.right),self.invertTree(root.left)
17. **return** root

235二叉搜索树的最近公共祖先

题目：

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](https://baike.baidu.com/item/%E6%9C%80%E8%BF%91%E5%85%AC%E5%85%B1%E7%A5%96%E5%85%88/8918834?fr=aladdin)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]

\_\_\_\_\_\_\_6\_\_\_\_\_\_

/ \

\_\_\_2\_\_ \_\_\_8\_\_

/ \ / \

0 \_4 7 9

/ \

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。

**示例 2:**

**输入:** root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4

**输出:** 2

**解释:** 节点 2 和节点 4 的最近公共祖先是 2, 因为根据定义最近公共祖先节点可以为节点本身。

**说明:**

* 所有节点的值都是唯一的。
* p、q 为不同节点且均存在于给定的二叉搜索树中。

思路：由于题目给定了这是一个二叉搜索树，这意味着虽然我们不知道p和q在哪，但现在比较一下root->val,p->val,q->val，我们就能得到其大致方位。那么所有的可能就是p和q分别在root两边或者p和q同时在root的一边。可以发现，当p和q在root结点的左右两边时，此时的root结点即是我们需要的LCA，这也是递归的终点。当p和q同时在root的左边时，我们就要将root递归成root->left，从其左子树上重复进行前面的判断过程。p和q同时在右边同理。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** lowestCommonAncestor(self, root, p, q):
10. """
11. :type root: TreeNode
12. :type p: TreeNode
13. :type q: TreeNode
14. :rtype: TreeNode
15. """
16. **if** p.val <root.val **and** q.val<root.val:
17. **return** self.lowestCommonAncestor(root.left,p,q)
18. **elif** p.val>root.val **and** q.val>root.val:
19. **return** self.lowestCommonAncestor(root.right,p,q)
20. **else**:
21. **return** root

257二叉树的所有路径

题目：给定一个二叉树，返回所有从根节点到叶子节点的路径。

**说明:** 叶子节点是指没有子节点的节点。

**示例:**

**输入:**

1

/ \

2 3

\

5

**输出:** ["1->2->5", "1->3"]

**解释:** 所有根节点到叶子节点的路径为: 1->2->5, 1->3

思路：

1. 建立一个字符串变量path和结果列表result，初始化为空

2. 从根节点开始访问，之后访问其左子树，再访问其右子树

3. 每访问一个节点，将节点的值加入path，例如，访问完根节点后，path = “1 ->”，并将这里的path作为新的变量加入左右子树的遍历函数

4. 递归“触底”的条件：访问的节点为空

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** binaryTreePaths(self, root):
10. """
11. :type root: TreeNode
12. :rtype: List[str]
13. """
14. **def** helper(root,path,res):
15. **if** root.left==None **and** root.right==None:
16. res.append(path+str(root.val))
17. **return**
18. **if** root.left:
19. helper(root.left,path+str(root.val)+'->',res)
20. **if** root.right:
21. helper(root.right,path+str(root.val)+'->',res)
23. **if** root **is** None:
24. **return** []
25. l=[]
26. helper(root,'',l)
27. **return** l

404左叶子之和

题目：计算给定二叉树的所有左叶子之和。

**示例：**

3

/ \

9 20

/ \

15 7

在这个二叉树中，有两个左叶子，分别是 9 和 15，所以返回 24

思路：采用递归的方法

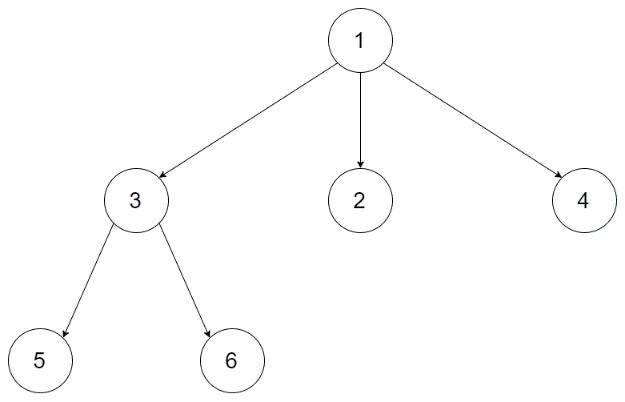
代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** sumOfLeftLeaves(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **if** root==None:
15. **return** 0
16. **if** root.left!=None **and** root.left.left==None **and** root.left.right==None:
17. **return** root.left.val+self.sumOfLeftLeaves(root.right)
18. **return** self.sumOfLeftLeaves(root.left)+self.sumOfLeftLeaves(root.right)

429N叉树的层序遍历

题目：给定一个 N 叉树，返回其节点值的*层序遍历*。 (即从左到右，逐层遍历)。

例如，给定一个 3叉树 :



返回其层序遍历:

[

[1],

[3,2,4],

[5,6]

]

思路：首先得明白，这个N叉树是什么样的数据结构定义的。val是节点的值，children是一个列表，这个列表保存了其所有节点。

层次遍历比较好理解，就是每层的值保存在一个list中，总的再返回一个list即可。

我们知道这个属于先进先出的结构，其实就是用队列就好了。需要注意是每层都在一个list中，所以在进入队列的时候需要保存一下这个节点属于哪个层。这样当遍历它的时候，就能直接放入它那层的list的末尾即可。难点在维护这个层数。

代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, children):
5. self.val = val
6. self.children = children
7. """
8. **class** Solution(object):
9. **def** levelOrder(self, root):
10. """
11. :type root: Node
12. :rtype: List[List[int]]
13. """
14. res=[]
15. que=collections.deque()
16. que.append(root)
17. **while** que:
18. level=[]
19. size=len(que)
20. **for** \_ **in** range(size):
21. node=que.popleft()
22. **if** **not** node:
23. **continue**
24. level.append(node.val)
25. **for** child **in** node.children:
26. que.append(child)
27. **if** level:
28. res.append(level)
29. **return** res

437路径总和III

题目：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。

**示例：**

root = [10,5,-3,3,2,null,11,3,-2,null,1], sum = 8

10

/ \

**5** **-3**

**/** **\** **\**

**3** **2** **11**

/ \ **\**

3 -2 **1**

返回 3。和等于 8 的路径有:

1. 5 -> 3

2. 5 -> 2 -> 1

3. -3 -> 11

思路：递归

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** pathSum(self, root, sum):
10. """
11. :type root: TreeNode
12. :type sum: int
13. :rtype: int
14. """
15. **def** traverse(root, val):
16. **if** **not** root:
17. **return** 0
18. res = (val == root.val)
19. res += traverse(root.left, val - root.val)
20. res += traverse(root.right, val - root.val)
21. **return** res
22. **if** **not** root:
23. **return** 0
24. ans = traverse(root, sum)
25. ans += self.pathSum(root.left, sum)
26. ans += self.pathSum(root.right, sum)
27. **return** ans

501二叉搜索树中的众数

题目：给定一个有相同值的二叉搜索树（BST），找出 BST 中的所有众数（出现频率最高的元素）。

假定 BST 有如下定义：

* 结点左子树中所含结点的值小于等于当前结点的值
* 结点右子树中所含结点的值大于等于当前结点的值
* 左子树和右子树都是二叉搜索树

例如：  
给定 BST [1,null,2,2],

1

\

2

/

2

返回[2].

**提示**：如果众数超过1个，不需考虑输出顺序

**进阶：**你可以不使用额外的空间吗？（假设由递归产生的隐式调用栈的开销不被计算在内）

思路：

1. 代码：# Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** findMode(self, root):
10. """
11. :type root: TreeNode
12. :rtype: List[int]
13. """
14. temp={}
15. ans=[]
16. **if** root==None:
17. **return** ans
18. **def** countNode(r):
19. **if** r.val **in** temp.keys():
20. temp[r.val]+=1
21. **else**:
22. temp[r.val]=1
23. **if** r.left:
24. countNode(r.left)
25. **if** r.right:
26. countNode(r.right)
27. countNode(root)
28. mode=max(temp.values())
29. **for** key,value **in** temp.items():
30. **if** value==mode:
31. ans.append(key)
32. **return** ans

538把二叉搜索树转换为累加树

题目：给定一个二叉搜索树（Binary Search Tree），把它转换成为累加树（Greater Tree)，使得每个节点的值是原来的节点值加上所有大于它的节点值之和。

**例如：**

**输入:** 二叉搜索树:

5

/ \

2 13

**输出:** 转换为累加树:

18

/ \

20 13

思路：用中序遍历得到的list是已经排序好的。最后求和的时候用中序反向遍历，会从最大值开始遍历。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** convertBST(self, root):
10. """
11. :type root: TreeNode
12. :rtype: TreeNode
13. """
14. **if** **not** root:
15. **return** root
16. self.all=[]
17. self.sum=0
18. **def** search(root):
19. **if** root.left:
20. search(root.left)
21. self.all.append(root.val)
22. **if** root.right:
23. search(root.right)
24. **def** change(root):
25. **if** root.right:
26. change(root.right)
27. self.sum+=self.all.pop()
28. root.val=self.sum
29. **if** root.left:
30. change(root.left)
31. search(root)
32. change(root)
33. **return** root

543二叉树的直径

题目：

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。

**示例 :**  
给定二叉树

1

/ \

2 3

/ \

4 5

返回 **3**, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。

**注意：**两结点之间的路径长度是以它们之间边的数目表示。

思路：根节点为root的二叉树的直径 = max(root-left的直径，root->right的直径，root->left的最大深度+root->right的最大深度+1)

代码：

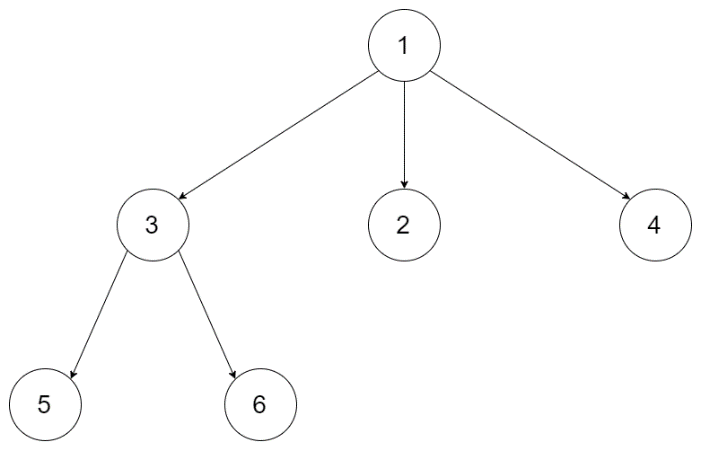
1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** diameterOfBinaryTree(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. self.ans=0
15. **def** dfs(root):
16. **if** **not** root:
17. **return** 0
18. left=dfs(root.left)
19. right=dfs(root.right)
20. self.ans=max(left+right,self.ans)
21. **return** max(left,right)+1
22. dfs(root)
23. **return** self.ans

559N叉树的最大深度

题目：给定一个 N 叉树，找到其最大深度。

最大深度是指从根节点到最远叶子节点的最长路径上的节点总数。

例如，给定一个 3叉树 :



我们应返回其最大深度，3。

**说明:**

1. 树的深度不会超过 1000。
2. 树的节点总不会超过 5000

思路：完全可以转换成一个递归问题。树的高度= 1 + 子树最大高度

代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, children):
5. self.val = val
6. self.children = children
7. """
8. **class** Solution(object):
9. **def** maxDepth(self, root):
10. """
11. :type root: Node
12. :rtype: int
13. """
14. **if** **not** root:
15. **return** 0
16. **if** **not** root.children:
17. **return** 1
18. depth=1+max(self.maxDepth(child) **for** child **in** root.children)
19. **return** depth

563二叉树的坡度

题目：

给定一个二叉树，计算**整个树**的坡度。

一个树的**节点的坡度**定义即为，该节点左子树的结点之和和右子树结点之和的**差的绝对值**。空结点的的坡度是0。

**整个树**的坡度就是其所有节点的坡度之和。

**示例:**

**输入:**

1

/ \

2 3

**输出:** 1

**解释:**

结点的坡度 2 : 0

结点的坡度 3 : 0

结点的坡度 1 : |2-3| = 1

树的坡度 : 0 + 0 + 1 = 1

**注意:**

1. 任何子树的结点的和不会超过32位整数的范围。
2. 坡度的值不会超过32位整数的范围

思路：sum即以当前结点为根节点的所有结点的和，那就是sum\_left + sum\_right + root.value了。当前结点的tilt是左子树结点和与右子树结点和的差，那就是abs(sum\_left - sum\_right)；注意这里返回的tilt是包括其左右子树的tilt的，所以最终就是abs(sum\_left - sum\_right) + tilt\_left + tilt\_right。

再仔细考虑tilt，实际上tilt不必要放在递归结构中返回，直接让tilt作为成员变量，保存累加和就好了，这样也避免和很多语言不支持return多个变量的问题

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** findTilt(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **def** sum\_and\_tilt(root):
15. **if** **not** root:
16. **return** 0,0
17. sum\_left,tilt\_left=sum\_and\_tilt(root.left)
18. sum\_right,tilt\_right=sum\_and\_tilt(root.right)
19. **return** sum\_left+sum\_right+root.val,abs(sum\_left-sum\_right)+tilt\_left+tilt\_right
20. sum\_tree,tilt\_tree=sum\_and\_tilt(root)
21. **return** tilt\_tree

572另一个树的子树

题目：给定两个非空二叉树 **s** 和 **t**，检验 **s** 中是否包含和 **t** 具有相同结构和节点值的子树。**s** 的一个子树包括 **s** 的一个节点和这个节点的所有子孙。**s** 也可以看做它自身的一棵子树。

**示例 1:**  
给定的树 s:

3

/ \

4 5

/ \

1 2

给定的树 t：

4

/ \

1 2

返回 **true**，因为 t 与 s 的一个子树拥有相同的结构和节点值。

**示例 2:**  
给定的树 s：

3

/ \

4 5

/ \

1 2

/

0

给定的树 t：

4

/ \

1 2

返回 **false**。

思路：递归树s的每一个结点，判断以该结点为根的树和树t是否相等，如果相等，为true，如果不等为false

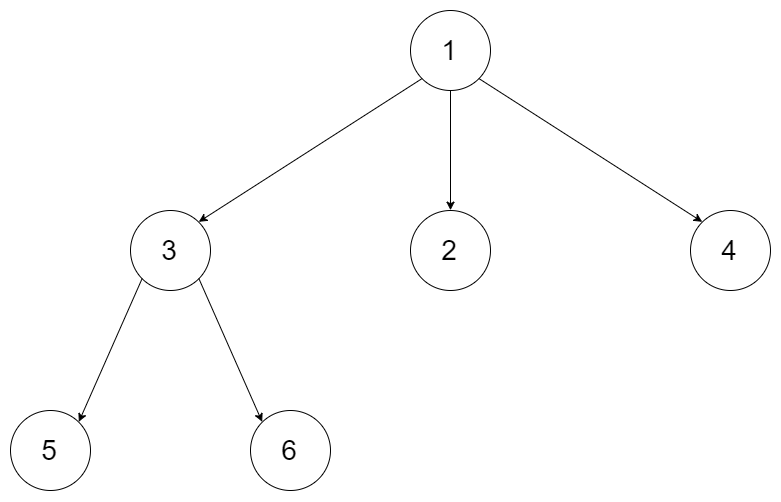
代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** isSame(self,s,t):
10. **if** s==None **and** t==None:
11. **return** True
12. **if** s==None **and** t!=None:
13. **return** False
14. **if** s!=None **and** t==None:
15. **return** False
16. **if** s.val!=t.val:
17. **return** False
18. **return** self.isSame(s.left,t.left) **and** self.isSame(s.right,t.right)
19. **def** isSubtree(self, s, t):
20. """
21. :type s: TreeNode
22. :type t: TreeNode
23. :rtype: bool
24. """
25. **if** s==None **or** t==None:
26. **return** False
27. **if** self.isSame(s,t):
28. **return** True
29. **return** self.isSubtree(s.left,t) **or** self.isSubtree(s.right,t)

589N叉树的前序遍历

题目：给定一个 N 叉树，返回其节点值的*前序遍历*。

例如，给定一个 3叉树 :



返回其前序遍历: [1,3,5,6,2,4]。

**说明:**递归法很简单，你可以使用迭代法完成此题吗?

思路：答案很显然想到用**栈**！！

栈是一种后进先出的数据结构，先把root送入栈中，把它的数值放入结果中，然后需要把它的所有子节点逆序放入栈中。为什么是逆序？因为后进先出啊！我们需要在下一轮遍历它的最左孩子

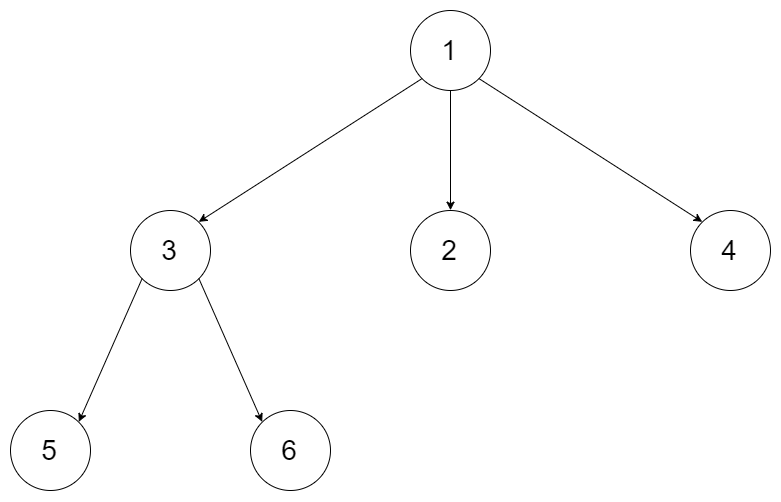
代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, children):
5. self.val = val
6. self.children = children
7. """
8. **class** Solution(object):
9. **def** preorder(self, root):
10. """
11. :type root: Node
12. :rtype: List[int]
13. """
14. **if** **not** root:
15. **return** []
16. stack=[]
17. stack.append(root)
18. res=[]
19. **while** stack:
20. node=stack.pop()
21. res.append(node.val)
22. stack.extend(node.children[::-1])
23. **return** res

590N叉树的后序遍历

题目：给定一个 N 叉树，返回其节点值的*前序遍历*。

例如，给定一个 3叉树 :



返回其后序遍历: [5,6,3,2,4,1]。

**说明:**递归法很简单，你可以使用迭代法完成此题吗?

思路一：递归

代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, children):
5. self.val = val
6. self.children = children
7. """
8. **class** Solution(object):
9. **def** postorder(self, root):
10. """
11. :type root: Node
12. :rtype: List[int]
13. """
14. res = []
15. **if** **not** root:
16. **return** res
17. **for** child **in** root.children:
18. res.extend(self.postorder(child))
19. res.append(root.val)
20. **return** res

思路：，我们同样使用迭代先把根节点放入栈中，然后把它的孩子依次放入，这样我们下次遍历获得的就是它的最后的节点，每层都是如此。我们每层都是根->右孩子->其余孩子的遍历方式，所以最后需要加一个翻转即可得到后序遍历。

代码：

1. """
2. # Definition for a Node.
3. class Node(object):
4. def \_\_init\_\_(self, val, children):
5. self.val = val
6. self.children = children
7. """
8. **class** Solution(object):
9. **def** postorder(self, root):
10. """
11. :type root: Node
12. :rtype: List[int]
13. """
14. **if** **not** root:
15. **return** []
16. stack=[root,]
17. res=[]
18. **while** stack:
19. node=stack.pop()
20. stack.extend(node.children)
21. res.append(node.val)
22. **return** res[::-1]

606根据二叉树创建字符串

题目：你需要采用前序遍历的方式，将一个二叉树转换成一个由括号和整数组成的字符串。

空节点则用一对空括号 "()" 表示。而且你需要省略所有不影响字符串与原始二叉树之间的一对一映射关系的空括号对。

**示例 1:**

**输入:** 二叉树: [1,2,3,4]

1

/ \

2 3

/

4

**输出:** "1(2(4))(3)"

**解释:** 原本将是“1(2(4)())(3())”，

在你省略所有不必要的空括号对之后，

它将是“1(2(4))(3)”。

**示例 2:**

**输入:** 二叉树: [1,2,3,null,4]

1

/ \

2 3

\

4

**输出:** "1(2()(4))(3)"

**解释:** 和第一个示例相似，

除了我们不能省略第一个对括号来中断输入和输出之间的一对一映射关系。

思路：

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** tree2str(self, t):
10. """
11. :type t: TreeNode
12. :rtype: str
13. """
14. **if** **not** t:
15. **return** ""
16. res=""
17. left=self.tree2str(t.left)
18. right=self.tree2str(t.right)
19. **if** left **or** right:
20. res+="(%s)" % left
21. **if** right:
22. res+="(%s)" % right
23. **return** str(t.val)+res

617合并二叉树

题目：给定两个二叉树，想象当你将它们中的一个覆盖到另一个上时，两个二叉树的一些节点便会重叠。

你需要将他们合并为一个新的二叉树。合并的规则是如果两个节点重叠，那么将他们的值相加作为节点合并后的新值，否则**不为**NULL 的节点将直接作为新二叉树的节点。

**示例 1:**

**输入:**

Tree 1 Tree 2

1 2

/ \ / \

3 2 1 3

/ \ \

5 4 7

**输出:**

合并后的树:

3

/ \

4 5

/ \ \

5 4 7

思路：递归遍历的方法

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** mergeTrees(self, t1, t2):
10. """
11. :type t1: TreeNode
12. :type t2: TreeNode
13. :rtype: TreeNode
14. """
15. **if** t1 **and** t2:
16. root=TreeNode(t1.val+t2.val)
17. root.left=self.mergeTrees(t1.left,t2.left)
18. root.right=self.mergeTrees(t1.right,t2.right)
19. **return** root
20. **else**:
21. **return** t1 **or** t2

637二叉树的层平均值

题目：

给定一个非空二叉树, 返回一个由每层节点平均值组成的数组.

**示例 1:**

**输入:**

3

/ \

9 20

/ \

15 7

**输出:** [3, 14.5, 11]

**解释:**

第0层的平均值是 3, 第1层是 14.5, 第2层是 11. 因此返回 [3, 14.5, 11].

**注意：**

1. 节点值的范围在32位有符号整数范围内。

思路：利用队列计算每一层，

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** averageOfLevels(self, root):
10. """
11. :type root: TreeNode
12. :rtype: List[float]
13. """
14. res=[]
15. q=collections.deque()
16. q.append(root)
17. **while** q:
18. sum=0
19. n=len(q)
20. **for** i **in** range(0,n):
21. node=q.popleft()
22. sum+=node.val
23. **if** node.left:
24. q.append(node.left)
25. **if** node.right:
26. q.append(node.right)
27. res.append(1.0\*sum/n)
28. **return** res

653两数之和IV-输入BST

题目：给定一个二叉搜索树和一个目标结果，如果 BST 中存在两个元素且它们的和等于给定的目标结果，则返回 true。

**案例 1:**

**输入:**

5

/ \

3 6

/ \ \

2 4 7

Target = 9

**输出:** True

**案例 2:**

**输入:**

5

/ \

3 6

/ \ \

2 4 7

Target = 28

**输出:** False

思路：凡是题目给出的是二叉搜索树而不是二叉树的时候，一定要以二叉搜索树的性质为切入点，这道题目就用到了二叉搜索树中序遍历的结果为有序数组的性质。得到有序数组后，，采用对撞指针的方式便可高效的得到答案，算法时间复杂度为O(n)

代码有一些案例没有通过

思路二：使用BFS

代码二：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):

11. **def** findTarget(self, root, k):
12. """
13. :type root: TreeNode
14. :type k: int
15. :rtype: bool
16. """
17. **if** root:
18. q=[root]
19. rem=[]
20. **while** q:
21. qtopnode=q.pop(0)
22. **if** k-qtopnode.val **in** rem:
23. **return** True
24. rem.append(qtopnode.val)
25. **if** qtopnode.left:
26. q.append(qtopnode.left)
27. **if** qtopnode.right:
28. q.append(qtopnode.right)
30. **return** False
31. **return** False

669修剪二叉搜索树

题目：给定一个二叉搜索树，同时给定最小边界L 和最大边界 R。通过修剪二叉搜索树，使得所有节点的值在[L, R]中 (R>=L) 。你可能需要改变树的根节点，所以结果应当返回修剪好的二叉搜索树的新的根节点。

**示例 1:**

**输入:**

1

/ \

0 2

L = 1

R = 2

**输出:**

1

\

2

**示例 2:**

**输入:**

3

/ \

0 4

\

2

/

1

L = 1

R = 3

**输出:**

3

/

2

/

1

思路：这一题用递归做，思路：当node.val> R时，此时修剪后的二叉树必然出现在节点的左边。 类似地，当node.val  <L时，修剪的二叉树必然出现在节点右侧。 否则，我们修剪树的两边

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** trimBST(self, root, L, R):
10. """
11. :type root: TreeNode
12. :type L: int
13. :type R: int
14. :rtype: TreeNode
15. """
16. **def** trim(node):
17. **if** **not** node:
18. **return** None
19. **elif** node.val>R:
20. **return** trim(node.left)
21. **elif** node.val<L:
22. **return** trim(node.right)
23. **else**:
24. node.left=trim(node.left)
25. node.right=trim(node.right)
26. **return** node
27. **return** trim(root)

671二叉树中第二小的节点

题目：给定一个非空特殊的二叉树，每个节点都是正数，并且每个节点的子节点数量只能为 2 或 0。如果一个节点有两个子节点的话，那么这个节点的值不大于它的子节点的值。

给出这样的一个二叉树，你需要输出所有节点中的**第二小的值。**如果第二小的值不存在的话，输出 -1 **。**

**示例 1:**

**输入:**

2

/ \

2 5

/ \

5 7

**输出:** 5

**说明:** 最小的值是 2 ，第二小的值是 5 。

**示例 2:**

**输入:**

2

/ \

2 2

**输出:** -1

**说明:** 最小的值是 2, 但是不存在第二小的值。

思路：因为有重复值，所以用一个set来保存BFS遍历得到的所有节点值，最后移除最小值，剩余的set中的最小值既是答案

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** findSecondMinimumValue(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. s=set()
15. queue=[root]
16. **while** queue **and** queue[0]:
17. node=queue.pop(0)
18. s.add(node.val)
19. **if** node.left:
20. queue.append(node.left)
21. **if** node.right:
22. queue.append(node.right)
23. **if** len(s)<2:
24. **return** -1
25. **else**:
26. s.remove(min(s))
27. **return** min(s)

687最长同值路径

题目：给定一个二叉树，找到最长的路径，这个路径中的每个节点具有相同值。 这条路径可以经过也可以不经过根节点。

**注意**：两个节点之间的路径长度由它们之间的边数表示。

**示例 1:**

输入:

5

/ \

4 5

/ \ \

1 1 5

输出:

2

**示例 2:**

输入:

1

/ \

4 5

/ \ \

4 4 5

输出:

2

**注意:** 给定的二叉树不超过10000个结点。 树的高度不超过1000。

思路：首先，求出**以每个节点为根节点的最长路径**，然后从底向上，判断与父亲节点的值是否相同，如果相同，就把当前结点最长的一个分支路径加上1返回给父节点。其中，可以把最长路径保存到一个全局变量中。

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** longestUnivaluePath(self, root):
10. """
11. :type root: TreeNode
12. :rtype: int
13. """
14. **if** **not** root:
15. **return** 0
16. self.res=0
17. self.getpath(root)
18. **return** self.res
19. **def** getpath(self,root):
20. **if** **not** root:
21. **return** 0
22. left=self.getpath(root.left)
23. right=self.getpath(root.right)
24. pl,pr=0,0
25. **if** root.left **and** root.left.val==root.val:
26. pl=1+left
27. **if** root.right **and** root.right.val==root.val:
28. pr=1+right
29. self.res=max(pl+pr,self.res)
30. **return** max(pl,pr)

700二叉搜索树中的搜索

题目：

给定二叉搜索树（BST）的根节点和一个值。 你需要在BST中找到节点值等于给定值的节点。 返回以该节点为根的子树。 如果节点不存在，则返回 NULL。

例如，

给定二叉搜索树:

4

/ \

2 7

/ \

1 3

和值: 2

你应该返回如下子树:

2

/ \

1 3

在上述示例中，如果要找的值是 5，但因为没有节点值为 5，我们应该返回 NULL。

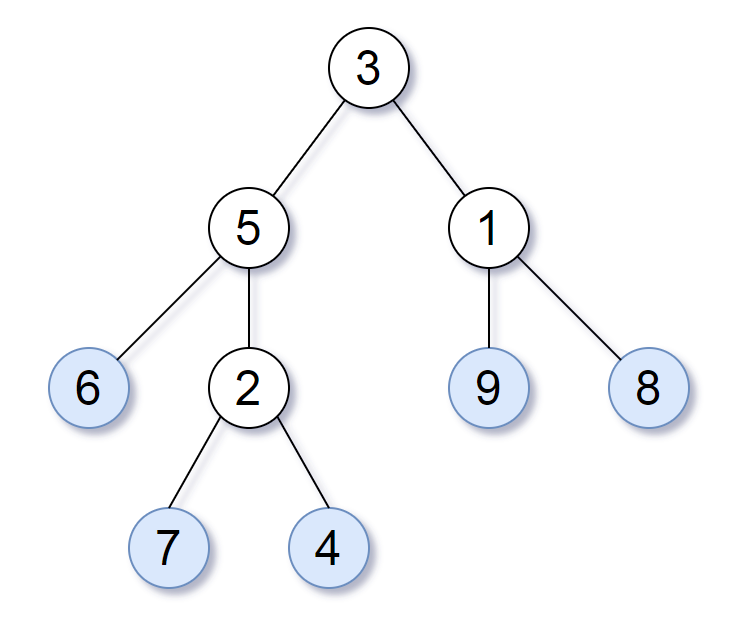
思路：遍历，递归即可

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** searchBST(self, root, val):
10. """
11. :type root: TreeNode
12. :type val: int
13. :rtype: TreeNode
14. """
15. **if**  **not** root:
16. **return** None
17. **elif** root.val==val:
18. **return** root
19. **elif** root.val>val:
20. **return** self.searchBST(root.left,val)
21. **else**:
22. **return** self.searchBST(root.right,val)

872叶子相似的树

题目：请考虑一颗二叉树上所有的叶子，这些叶子的值按从左到右的顺序排列形成一个 *叶值序列*



举个例子，如上图所示，给定一颗叶值序列为 (6, 7, 4, 9, 8) 的树。

如果有两颗二叉树的叶值序列是相同，那么我们就认为它们是 *叶相似*的。

如果给定的两个头结点分别为 root1 和 root2 的树是叶相似的，则返回 true；否则返回 false 。

思路：一棵树从左到右的序列应该使用中序遍历，当中序遍历时，如果节点是叶子节点则放入序列之中。

所以判断两棵树的序列是否相等即可

代码：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** leafSimilar(self, root1, root2):
10. """
11. :type root1: TreeNode
12. :type root2: TreeNode
13. :rtype: bool
14. """
15. leaves1=[]
16. leaves2=[]
17. self.inorder(root1,leaves1)
18. self.inorder(root2,leaves2)
19. **return** leaves1==leaves2
20. **def** inorder(self,root,leaves):
21. **if** **not** root:
22. **return**
23. self.inorder(root.left,leaves)
24. **if** **not** root.left **and** **not** root.right:
25. leaves.append(root.val)
26. self.inorder(root.right,leaves)

897递增顺序查找树

题目：

给定一个树，**按中序遍历**重新排列树，使树中最左边的结点现在是树的根，并且每个结点没有左子结点，只有一个右子结点。

**示例 ：**

**输入：**[5,3,6,2,4,null,8,1,null,null,null,7,9]

5

/ \

3 6

/ \ \

2 4 8

 / / \

1 7 9

**输出：**[1,null,2,null,3,null,4,null,5,null,6,null,7,null,8,null,9]

1

  \

  2

  \

  3

  \

  4

  \

  5

  \

  6

  \

  7

  \

  8

  \

9

思路：使用的方式是最简单的，先中序遍历，得到顺序，然后再连接的方式。

这个做法的问题是用数组保存了整儿个中序遍历的值，然后重建了二叉树，那么空间复杂度挺大的，不是一个好方法。

如果不想使用保存节点的值然后重新构建每个节点的方式，那么有个更简单的方法就是我们在数组里保存节点，然后直接把数组的节点再次构成树就好了。省去了重新构造每个节点的过程。

代码一：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** increasingBST(self, root):
10. """
11. :type root: TreeNode
12. :rtype: TreeNode
13. """
14. res = self.inOrder(root)
15. **if** **not** res:
16. **return**
17. dummy = TreeNode(-1)
18. cur = dummy
19. **for** node **in** res:
20. node.left = node.right = None
21. cur.right = node
22. cur = cur.right
23. **return** dummy.right
25. **def** inOrder(self, root):
26. **if** **not** root:
27. **return** []
28. res = []
29. res.extend(self.inOrder(root.left))
30. res.append(root)
31. res.extend(self.inOrder(root.right))
32. **return** res

思路二：这个做法在上面的基础上再次缩减了空间复杂度，不再需要数组。这种做法中直接在中序遍历的过程中修改每个节点的指向。

修改指向的方式其实比较简单，使用prev指针一直指向了构造出来的这个新树的最右下边的节点，在中序遍历过程中把当前节点的左指针给设置为None，然后把当前节点放到新树的右下角，这样类似于一个越来越长的链表的构建过程。

时间复杂度是O(n)，空间复杂度是O(1).

代码二：

1. # Definition for a binary tree node.
2. # class TreeNode(object):
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
8. **class** Solution(object):
9. **def** increasingBST(self, root):
10. """
11. :type root: TreeNode
12. :rtype: TreeNode
13. """
14. dummy=TreeNode(-1)
15. self.prev=dummy
16. self.inorder(root)
17. **return** dummy.right
19. **def** inorder(self,root):
20. **if** **not** root:
21. **return** None
22. self.inorder(root.left)
23. root.left=None
24. self.prev.right=root
25. self.prev=self.prev.right
26. self.inorder(root.right)