链滴

leetcode解题报告-树

作者：Zing

原文链接：https://ld246.com/article/1459760930303

来源网站：链滴

树是最常考察的数据结构。由于树的定义是递归的（树的子树也是树），因此大部分的树的问题，如树的最大深度、判定树是否平衡、共同祖先等，可以由递归进行求解，这也是最简单最直接的方式。

但是，通常面试官都希望你答出非递归解法。如前中后序的非递归解法都是借助栈进行实现（利用栈进后出的特性，结合进出栈的时机）。先序列遍历和中序遍历对栈的操作相同，栈顶左子一直进栈直最左下，弹栈，右子进栈。然后循环操作即可。唯一不同是先序遍历是在进栈前访问，中序遍历是在栈的时候访问。后序遍历，进栈的顺序为根右左，由于栈的特性会使出栈时为左右根。进栈时需标记前节点左右子是否已经进栈了。树的按层次遍历是借助队列先进先出的特性实现的。

对应的树的遍历方法有DFS和BFS，DFS和先序遍历是一致的。BFS和按层次遍历是一致的。

另外，在解决树问题的时候需要充分利用该树的特性，如二叉搜索树，完全二叉树的特性。

##94. Binary Tree Inorder Traversal （中序遍历）

- 难度：Medium
- 题意：

给一个二叉树，返回该树的中序遍历。

- 思路1：

中序遍历即左中右，思路1先使用递归实现，递归是最方便书写且最容易理解的，因为树本身就是递定义的。递归访问左子树，输出根，递归访问右子树。

- 代码1：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    res=[]
    def inorderTraversal(self, root):
        self.res=[]
        self.reduce(root)
        return self.res
    def reduce(self,root):
        if root and root.left:self.reduce(root.left)
        if root:self.res.append(root.val)
        if root and root.right:self.reduce(root.right)
```

- 思路2：

通常面试中，面试官肯定不会想看到上面的解法，一般会让用非递归实现。中序遍历的非递归实现是助栈实现的。将根节点入栈，若栈顶节点存在左子树，则左子树进栈，否则出栈并添加到结果中。

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def inorderTraversal(self, root):
        result = []
        stack = []
        if not root: return result
        stack.append(root)
```

```
        while stack:
            if stack[-1].left:
                stack.append(stack[-1].left)
                stack[-2].left = None
            else:
                node = stack.pop(-1)
                result.append(node.val)
                if node.right:
                    stack.append(node.right)
        return result
```

##96. Unique Binary Search Trees （BST）

- 难度：Medium
- 题意：

给定整数n，求出以1...n作为节点，有多少种结构不同的二叉搜索树

- 思路：

每个节点都可以作为跟，对于每个节点来说，可以构成二叉树的总类型数=左子树总类型数*右子树总型数。因此可以采用dp解决。

- 代码：

```
    class Solution:
        # @param {integer} n
        # @return {integer}
        def numTrees(self, n):
            dp = [1,1]
            for i in range(2,n+1):
                dp.append(0)
                for j in range(0,i):
                    dp[i] += dp[j]*dp[i-j-1]
            return dp[n]
```

##95. Unique Binary Search Trees II （BST）

- 难度：Medium
- 题意：

给定数字n，要求用数字1到n构造出所有不同的二叉搜索树

- 思路：

二叉搜索树，左子树<根<右子树。这也是我们构造的思路。对于(left,right)，我们循环遍历设置i为，则左子树的范围是（left,i-1）,右子树的范围是（i+1,right）。递归构造左子树和右子树，返回所可能的根。

- 代码：

```
    class Solution:
```

```
    # @param {integer} n
    # @return {TreeNode[]}
    def generateTrees(self, n):
        return self.creatTrees(1,n)

    def creatTrees(self,start,end):
        result = []
        if start>end:
            result.append(None)
            return result
        for i in range(start,end+1):
            lefts = self.creatTrees(start,i-1)
            rights = self.creatTrees(i+1,end)
            for left in lefts:
                for right in rights:
                    root = TreeNode(i)
                    root.left = left
                    root.right = right
                    result.append(root)
        return result
```

##98. Validate Binary Search Tree （BST）

- 难度：Medium
- 题意：

给定一个二叉树，检查其是否而二叉搜索树。

- 思路：

二叉搜索树的特点是 左<根<右，因此，对二叉搜索数进行中序遍历，输出的结果即为正序。

- 代码：

```
class Solution:
    def LDR(self, root):
        if root != None:
            if root.left != None:
                self.LDR(root.left)
            self.tree.append(root.val)
            if root.right != None:
                self.LDR(root.right)

    # @param {TreeNode} root
    # @return {boolean}
    def isValidBST(self, root):
        self.tree=[]
        self.LDR(root)
        i=0
        while i<len(self.tree)-1:
            if self.tree[i] >= self.tree[i+1]:
                return False
            i+=1
```

```
        return True
```

##99. Recover Binary Search Tree （BST）

- 难度：Hard
- 题意：

二叉搜索树中有两个节点被调换了位置，在不改变数据结构的基础上还原二叉搜索树。要求在o(n)空复杂度内解决。

- 思路：

这道题有一种巧妙的思路利用了递归中序遍历的递归和回溯过程。这里使用pre来记录被遍历的前一节点。中序遍历的递归方法是，递归访问左子树到最左下端，然后回溯，再对右子进行递归。回溯的候，pre是当前访问节点的左子，再向右递归时，pre是当前右子的根，因此总有pre<root成立，若成立则为被调换的点。

- 思路：

```
class Solution(object):
    def recoverTree(self, root):
        """
        :type root: TreeNode
        :rtype: void Do not return anything, modify root in-place instead.
        """
        self.mis1,self.mis2,self.pre=None,None,None
        self.traval(root)
        if self.mis1 and self.mis2:
            self.mis1.val,self.mis2.val=self.mis2.val,self.mis1.val

    def traval(self,root):
        if not root:return
        if root.left:self.traval(root.left)
        if self.pre and root.val<self.pre.val:
            if not self.mis1:
                self.mis1=self.pre
            self.mis2=root
        self.pre = root
        if root.right:self.traval(root.right)
```

##100. Same Tree （递归）

- 难度：Easy
- 题意：

给定两个二叉树，验证这两个树是否相等。当且仅当这两棵树结构相同且每个节点值相同时，才认为相等。

- 思路：

两颗树相同，当且仅当，根的值相等，且左子树相等，右子树相等。因此，可以用递归方便地求解。

- 代码：

```
class Solution:
    # @param {TreeNode} p
    # @param {TreeNode} q
    # @return {boolean}
    def isSameTree(self, p, q):
        if not p and not q: return True
        elif (p and not q) or (q and not p) or p.val != q.val: return False
        else: return self.isSameTree(p.left,q.left) and self.isSameTree(p.right,q.right)
```

##101. Symmetric Tree （递归）

- 难度：Easy
- 题意：

给定一棵二叉树，检查是否为镜像对称。

- 思路：

一棵树是否为镜像，取决与左子树和右子树是否镜像对称。左子树和右子树是否镜像对称，取决于左树的左子树和右子树的右子树是否镜像对称，且，左子树的右子树和右子树的左子树是否镜像对称。此用递归可以求解。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {boolean}
    def isSymmetric(self, root):
        if not root:return True
        return self.isSame(root.left,root.right)

    def isSame(self, p, q):
        if not p and not q: return True
        elif (p and not q) or (q and not p) or p.val != q.val: return False
        else: return self.isSame(p.left,q.right) and self.isSame(p.right,q.left)
```

##102. Binary Tree Level Order Traversal （按层次遍历）

- 难度:Easy
- 题意：

给定一棵二叉树，返回其按层次遍历的节点值序列。输出将同一层次放在同一列表中。

- 思路：

树的按层次遍历借助队列完成，将根节点入队。若队列中有节点，弹出出队头，并把队头节点的所有节点添加到队尾。由于题目要求把同一层的节点放在同一列表中，因此我们的数据结构是levels[[]],对的操作是，遍历每一层时，创建一个新的newlevel[]，访问levels列表中最后一层的所有节点，并添到每个节点的子节点到newlevel中，最后再把newlevel添加到levels中。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def levelOrder(self, root):
        if not root :return []
        levels = [[root]]
        values = [[root.val]]
        for level in levels:
            newLevel = []
            newValue = []
            for node in level:
                if node.left:
                    newLevel.append(node.left)
                    newValue.append(node.left.val)
                if node.right:
                    newLevel.append(node.right)
                    newValue.append(node.right.val)
            if newLevel:
                levels.append(newLevel)
                values.append(newValue)
        return values
```

##103. Binary Tree Zigzag Level Order Traversal （按层次遍历）

- 难度：Medium
- 题意：

给定一棵二叉树，按照Z字形遍历树的节点。

- 思路：

该道题其实和102. Binary Tree Level Order Traversal是一样的，都是按层次遍历，只是在输出时，z字形输出即可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def zigzagLevelOrder(self, root):
        values = self.levelOrder(root)
        for i in range(1,len(values),2):
            values[i].reverse()
        return values
```

##107. Binary Tree Level Order Traversal II

- 难度：Easy
- 题意：

给定一棵二叉树，要求自底向上按层次遍历。

- 思路：

还是使用队列辅助进行按层次遍历，思路上和普通按层次遍历没有什么区别。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[][]}
    def levelOrderBottom(self, root):
        if not root :return []
        levels = [[root]]
        values = [[root.val]]
        for level in levels:
            newLevel = []
            newValue = []
            for node in level:
                if node.left:
                    newLevel.append(node.left)
                    newValue.append(node.left.val)
                if node.right:
                    newLevel.append(node.right)
                    newValue.append(node.right.val)
            if newLevel:
                levels.append(newLevel)
                values.insert(0,newValue)
        return values
```

##104. Maximum Depth of Binary Tree （最大深度）

- 难度：Medium

- 题意：

给定一棵二叉树，求其最大深度。

- 思路：

最简单的想法，一棵树的最大深度=max(左子树的最大深度，右子树的最大深度)+1。使用递归即可速求解。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer}
    def maxDepth(self, root):
        return 0 if not root else 1+max(self.maxDepth(root.left),self.maxDepth(root.right))
```

- 思路2：

面试官肯定会问，还有非递归的解法吗？这里非递归的解法也非常简单，就是深度优先遍历（先序遍）。深度优先遍历的非递归解法是借助栈来实现的，

1. 跟节点入栈
    2. 循环将栈顶的左子入栈，直到无左子
    3. 弹出栈顶，将其右子入栈
    4. 循环2和3

因此可以看出，栈的深度就是树的深度。

代码这里就不给出了。

##111. Minimum Depth of Binary Tree （最小深度）

- 难度：Easy
- 题意：

求给定二叉树的最小深度

- 思路：

树最小深度 = min(左子最小深度，右子最小深度)+1，递归可求解。当然也可以用按层次遍历求解。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer}
    def minDepth(self, root):
        if not root:return 0
        if not root.left and root.right:return 1+self.minDepth(root.right)
        if not root.right and root.left:return 1+self.minDepth(root.left)
        return 1+min(self.minDepth(root.left),self.minDepth(root.right))
```

##105. Construct Binary Tree from Preorder and Inorder Traversal （先序遍历、中序遍历）

- 难度：Medium
- 题意：

给定一个先序遍历序列和中序遍历序列，要求还原这棵树。

- 思路：

先弄清楚先序遍历是根左右，中序遍历是左根右。因此，根据先序遍历可以将中序遍历划分为左右两，分别代表左子树和右子树。如此递归可以求解。

- 代码：

```
class Solution:
    # @param {integer[]} preorder
    # @param {integer[]} inorder
    # @return {TreeNode},
    def buildTree(self, preorder, inorder):
        if not inorder:return None
```

```
        n = preorder[0]
        i = inorder.index(n)
        preorder.remove(n)
        root = TreeNode(n)
        root.left = self.buildTree(preorder,inorder[:i])
        root.right = self.buildTree(preorder,inorder[i+1:])
        return root
```

##106. Construct Binary Tree from Inorder and Postorder Traversal （中序遍历、后序遍历）

- 难度：Medium
- 题意：

给定一棵树的中序和后序遍历的序列，要求还原这棵树。

- 思路：

中序遍历是左根右，后序遍历是左右根。因此可以根据后序遍历将中序遍历划分为左右两端，分别代左子树和右子树，使用递归可以还原整棵树。

- 代码：

```
class Solution:
    # @param {integer[]} inorder
    # @param {integer[]} postorder
    # @return {TreeNode}
    def buildTree(self, inorder, postorder):
        if not inorder:return None
        n = postorder[-1]
        i = inorder.index(n)
        postorder.remove(n)
        root = TreeNode(n)
        root.right = self.buildTree(inorder[i+1:],postorder)
        root.left = self.buildTree(inorder[:i],postorder)
        return root
```

##144. Binary Tree Preorder Traversal（先序遍历）

- 难度：Medium
- 题意：

给定一棵二叉树，返回其先序遍历序列。

- 思路：

先序遍历的顺序为：根左右。先序遍历和深度优先遍历类似，采用栈实现。若栈的操作顺序是，根出，根右子入栈，根左子入栈。则循环出栈的时候就为根左右的顺序了。

- 代码：

```
class Solution:
    # @param {TreeNode} root
```

```
    # @return {integer[]}
    def preorderTraversal(self, root):
        result=[]
        stack=[]
        if not root:return result
        stack.append(root)
        while stack:
            head = stack.pop(-1)
            result.append(head.val)
            if head.right:
                stack.append(head.right)
            if head.left:
                stack.append(head.left)
        return result
```

##145. Binary Tree Postorder Traversal （后序遍历）

- 难度：Medium
- 题意：

给定一棵二叉树，返回其后序遍历序列。

- 思路：

后序遍历的顺序为：左右根。后序遍历也是借助栈来实现。后序遍历的栈操作顺序为，根入栈。栈顶右子入栈，标记栈顶的右子已经入栈，栈顶的左子入栈，标记栈顶的左子入栈。当栈顶的左右子均以过栈，栈顶弹出，否则，重复刚才的操作。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def postorderTraversal(self, root):
        result=[]
        stack=[]
        if not root:return result
        stack.append(root)
        while stack:
            top = stack[-1]
            if not top.left and not top.right:
                result.append(top.val)
                stack.pop(-1)
            else:
                if top.right:
                    stack.append(top.right)
                    top.right=None
                if top.left:
                    stack.append(top.left)
                    top.left=None
        return result
```

##108. Convert Sorted Array to Binary Search Tree （二叉搜索树）

- 难度：Medium
- 题意：

给定一个正序数组，建立对应的二叉搜索树。

- 思路：

找到中位数作为根节点，左侧为左子树，右侧为右子树。递归建立即可。

- 代码：

```
class Solution:
    # @param {integer[]} nums
    # @return {TreeNode}
    def sortedArrayToBST(self, nums):
        if not nums: return None
        mid = len(nums)//2
        root = TreeNode(nums[mid])
        root.left = self.sortedArrayToBST(nums[:mid])
        root.right = self.sortedArrayToBST(nums[mid+1:])
        return root
```

##110. Balanced Binary Tree （平衡二叉树）

- 难度：Easy
- 题意：

给定一棵二叉树，检查是否为平衡二叉树

- 思路：

一棵树为平衡二叉树的条件是，abs（左子树最大高度-右子树最大高度）<2，且左子树平衡且右子平衡。树最大高度递=（左子树最大高度，右子树最大高度)+1。用递归的方法从定义出发可以解决。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {boolean}
    def isBalanced(self, root):
        if not root:return True
        return abs(self.maxDepth(root.left)-self.maxDepth(root.right))<2 and self.isBalanced(roo
.left) and self.isBalanced(root.right)
    def maxDepth(self, root):
        if not root:return 0
        else:return 1+max(self.maxDepth(root.left),self.maxDepth(root.right))
```

##112. Path Sum（路径和）

- 难度：Easy

- 题意：

给定二叉数组和数字sum，检测是否有从根节点到叶子节点的路径使得路径上节点值的和为sum

- 思路：

直接递归求解即可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @param {integer} sum
    # @return {boolean}
    def hasPathSum(self, root, sum):
        if not root :return False
        sum -= root.val
        if sum==0 and not root.left and not root.right:return True
        if sum!=0 and not root.left and not root.right:return False
        if root.left and not root.right:return self.hasPathSum(root.left,sum)
        if root.right and not root.left:return self.hasPathSum(root.right,sum)
        if root.right and root.left:return self.hasPathSum(root.left,sum) or self.hasPathSum(root.r
ght,sum)
```

##113. Path Sum II（路径和）

- 难度：Medium

- 题意：

给定二叉数组和数字sum，检测是否有从根节点到叶子节点的路径使得路径上节点值的和为sum，并回所有满足要求的路径。

- 思路：

递归的同时把当前路径作为参数传入即可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @param {integer} sum
    # @return {integer[][]}
    def pathSum(self, root, sum):
        self.result = []
        self.reduce(root,sum,[])
        return self.result

    def reduce(self,root,sum,nodes0):
        nodes = nodes0[::]
        if not root:return
        else:
            nodes.append(root.val)
            sum-=root.val
            if sum==0 and not root.left and not root.right:
```

```
            self.result.append(nodes)
        elif sum!=0 and not root.left and not root.right:
            return
        elif root.left and not root.right:
            self.reduce(root.left,sum,nodes)
        elif root.right and not root.left:
            self.reduce(root.right,sum,nodes)
        elif root.right and root.left:
            self.reduce(root.left,sum,nodes)
            self.reduce(root.right,sum,nodes)
```

##124. Binary Tree Maximum Path Sum （路径和）

- 难度：Hard
- 题意：

给定一个二叉树，返回最大路径和。这里的路径指：任意连线两点之间的路径。

- 思路：

对任意节点，经过该节点的最大路径为【以左子为根向下遍历的最大路径】+该节点+【以右子为根下遍历的最大路径】。因此可以用递归解决。用一个全局变量记录最大路径，递归求解当前节点为根下遍历的最大路径。

- 代码：

```
class Solution(object):
    def maxPathSum(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if not root:return 0
        self.maxv = -(2**31)
        self.maxSum(root)
        return self.maxv

    def maxSum(self,root):
        if not root:return 0
        v,lmax,rmax=root.val,0,0
        if root.left:
            lmax=self.maxSum(root.left)
            if lmax>0:
                v+=lmax
        if root.right:
            rmax=self.maxSum(root.right)
            if rmax>0:
                v+=rmax
        self.maxv=max(self.maxv,v)
        return max(root.val,root.val+lmax,root.val+rmax)
```

##129. Sum Root to Leaf Numbers（路径和）

- 难度：Medium
- 题意：

给定一个二叉树只包含数字0-9，每个从根到叶子节点的路径表示一个数字（根表示最高位）。求所从根到叶子节点数字之和。

- 思路：

使用递归的方法，传递父路径代表数字n，则当前路径为n*10+root。递归到叶子节点的时候把数字和即可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer}
    def sumNumbers(self, root):
        self.sum=0
        self.reduce(root,0)
        return self.sum

    def reduce(self,root,n):
        if not root:
            self.sum+=n
        elif not root.left and not root.right:
            n = 10*n+root.val
            self.sum+=n
        else:
            n = 10*n+root.val
            if root.left:self.reduce(root.left,n)
            if root.right:self.reduce(root.right,n)
```

##114. Flatten Binary Tree to Linked List （链表）

- 难度：Medium
- 题意：

给定一棵二叉树，要求就定把树变为一个链表。

- 思路：

求当前节点的左子树的最右下角，并把当前节点右子树接到左子树最右下角，然后把左子树变为右子。循环向右推进即可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {void} Do not return anything, modify root in-place instead.
    def flatten(self, root):
        if not root:return
        while root:
```

```
        if root.left:
            node = root.left
            while node.right:
                node = node.right
            node.right = root.right
            root.right = root.left
            root.left = None
        root = root.right
```

##116. Populating Next Right Pointers in Each Node （按层次遍历）

- 难度：Medium
- 题意：

给定一棵二叉树，对于每个节点如果存在相邻（同层右侧）节点，则填充next指针。假定给定的是一完全二叉树。

- 思路：

按层次遍历，使用tag标记当前层有多少个节点。

- 代码：

```
class Solution:
    # @param root, a tree link node
    # @return nothing
    def connect(self, root):
        nodeList = [root]
        tag = 1
        i = 1
        while len(nodeList):
            treeNode = nodeList.pop(0)
            if  treeNode and treeNode.left:
                nodeList.extend([treeNode.left,treeNode.right])
            if i == tag:
                i = 1
                tag *= 2
            else:
                if len(nodeList):
                    treeNode.next = nodeList[0]
                i += 1
```

##117. Populating Next Right Pointers in Each Node II （按层次遍历）

- 难度：Hard
- 题意：

给定一棵二叉树，对于每个节点如果存在相邻（同层右侧）节点，则填充next指针。给定的树不一定完全二叉树。

- 思路：

根据当前层已经串联起来的next，来生成下一层的next。即根据next横向遍历，判断子节点情况，来联下一层。横向遍历前需要记下，下一层的最左边的节点。

- 代码：

```
class Solution(object):
    def connect(self, root):
        """
        :type root: TreeLinkNode
        :rtype: nothing
        """
        if not root:return
        node=root
        ##自上而下遍历
        while node:
            down = None
            ##横向遍历
            while node:
                ##记录下一层最左边节点
                if not down:
                    if node.left: down = node.left
                    elif node.right: down = node.right
                nex = node.next
                ##跳过无子节点的节点
                while nex:
                    if nex.left or nex.right:
                        break
                    nex=nex.next
                ##串联下一层
                if node.left and node.right:
                    node.left.next = node.right

                if node.left and not node.right and nex:
                    if nex.left:
                        node.left.next=nex.left
                    else:
                        node.left.next=nex.right
                if node.right and nex:
                    if nex.left:
                        node.right.next=nex.left
                    else:
                        node.right.next=nex.right
                node=nex
            node=down
```

##173. Binary Search Tree Iterator （二叉搜索树）

- 难度：Medium
- 题意：

实现二叉搜索树的迭代器。next方法返回下一最小数字。hasNext方法返回是否存在下一最小数字。

- 思路：

二叉搜索其实就是中序遍历。因此借助栈来实现，在初始化时，就将跟节点的所有左子入栈。调用nex方法，返回栈顶，并将栈顶右子树的所有左子入栈。若栈为空，则无next。

- 代码：

```
class BSTIterator:
    # @param root, a binary search tree's root node
    def __init__(self, root):
        self.stack=[]
        while root:
            self.stack.append(root)
            root=root.left

    # @return a boolean, whether we have a next smallest number
    def hasNext(self):
        return len(self.stack)!=0

    # @return an integer, the next smallest number
    def next(self):
        if self.stack:
            root=self.stack.pop(-1)
            rig = root.right
            while rig:
                self.stack.append(rig)
                rig=rig.left
            return root.val
        return None
```

##199. Binary Tree Right Side View （按层次遍历）

- 难度：Medium
- 题意：

给定一个二叉树，想象你从右侧看这棵树，自上而下返回你能看到的数字。

- 思路：

直接按层次遍历便可。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer[]}
    def rightSideView(self, root):
        result=[]
        values = self.levelOrder(root)
        for row in values:
            result.append(row[-1])
        return result
```

```
    def levelOrder(self, root):
        if not root :return []
        levels = [[root]]
        values = [[root.val]]
        for level in levels:
            newLevel = []
            newValue = []
            for node in level:
                if node.left:
                    newLevel.append(node.left)
                    newValue.append(node.left.val)
                if node.right:
                    newLevel.append(node.right)
                    newValue.append(node.right.val)
            if newLevel:
                levels.append(newLevel)
                values.append(newValue)
        return values
```

##222. Count Complete Tree Nodes （完全二叉树）

- 难度：Medium
- 题意：

给定一个完全二叉树，返回节点个数。

- 思路：

对于完全二叉树，关键是找到最底层没有叶子的位置即可。若是满二叉树则可以直接算出节点个数。此可以采用递归的方法查找节点缺失未知。若树最左深度=最右深度，则为满二叉树。否则递归求左和右子数量。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {integer}
    def countNodes(self, root):
        return self.counter(root)

    def leftDeep(self, root):
        i = 0
        node = root
        if not node:
            return 0
        while node.left:
            i += 1
            node = node.left
        return i

    def rightDeep(self, root):
        i = 0
        node = root
```

```
        if not node:
            return 0
        while node.right:
            i += 1
            node = node.right
        return i

    def counter(self, root):
        if not root:
            return 0
        else:
            l = self.leftDeep(root)+1
            r = self.rightDeep(root)+1
            if l == r:
                return 2 ** l - 1
            else:
                return self.counter(root.left) + self.counter(root.right) + 1
```

##226. Invert Binary Tree

- 难度：Easy
- 题意：

左右翻转一棵二叉树。

- 思路：

传说中90%的谷歌工程师没办法在白板完成的题目。思路很简单，递归翻转左右子树。

- 代码：

```
class Solution:
    # @param {TreeNode} root
    # @return {TreeNode}
    def invertTree(self, root):
        if not root:return root
        root.left,root.right=root.right,root.left
        self.invertTree(root.left)
        self.invertTree(root.right)
        return root
```

##230. Kth Smallest Element in a BST （二叉搜索树）

- 难度：Medium
- 题意：

给定一个二叉搜索树，返回其第k小的元素。

- 思路：

借助栈实现二叉搜索树的遍历，弹出第k个为答案。

- 代码：

```
class Solution(object):
    def kthSmallest(self, root, k):
        """
        :type root: TreeNode
        :type k: int
        :rtype: int
        """
        stack=[]
        node = root
        while node:
            stack.append(node)
            node = node.left
        x=1
        while stack and x<=k:
            node = stack.pop()
            x+=1
            right = node.right
            while right:
                stack.append(right)
                right=right.left
        return node.val
```

##235. Lowest Common Ancestor of a Binary Search Tree （共同祖先）

- 难度：Easy
- 题意：

给定一个二叉搜索树，找到给定两个节点的最近公共祖先。

- 思路：

使用递归的方法，若当前节点等于其中一个节点，则该节点必为公共节点。若两个数分布在该节点两，则该节点比为公共节点。若均在左侧，递归左子。若均在右侧，递归右子。

- 代码：

```
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """

        if root==p or root ==q:return root
        elif root.val > max(p.val,q.val):
            return self.lowestCommonAncestor(root.left,p,q)
        elif root.val < min(p.val,q.val):
            return self.lowestCommonAncestor(root.right,p,q)
        else:
            return root
```

##236. Lowest Common Ancestor of a Binary Tree （共同祖先）

- 难度：Medium
- 题意：

给定一个二叉树，找到给定两个节点的最近公共祖先。

- 思路：

该题和235的区别在于，不是二叉搜索树。因此无法通过数字的大小关系，判断数字在哪个子树中。是整体思路还是不便。我们观察发现，其实在递归左子或右子的时候，其实同时也在查找两个给定的。因此也可以判断两个数字在当前节点的哪一侧。而且更巧的是，若在同一侧，递归先找到的那个节就是共同祖先。

- 代码：

```
class Solution(object):
    def lowestCommonAncestor(self, root, p, q):
        """
        :type root: TreeNode
        :type p: TreeNode
        :type q: TreeNode
        :rtype: TreeNode
        """
        if not root: return None
        if root==p or root==q:return root
        left = self.lowestCommonAncestor(root.left,p,q)
        right = self.lowestCommonAncestor(root.right,p,q)
        if left and right:return root
        return left if left else right
```

##257. Binary Tree Paths （路径）

- 难度：Easy
- 题意：

给定一个二叉树，返回所有从根到叶子节点的路径。

- 思路：

递归进行dfs，同时记录路径即可。

- 代码：

```
class Solution(object):
    def binaryTreePaths(self, root):
        """
        :type root: TreeNode
        :rtype: List[str]
        """
        self.result=[]
        self.dfs(root,[])
        return self.result
```

```
    def dfs(self,root,path0):
        path = path0[::]
        if not root :return
        path.append(str(root.val))
        if not root.left and not root.right:
            self.result.append('->'.join(path))
        else:
            if root.left:
                self.dfs(root.left,path)
            if root.right:
                self.dfs(root.right,path)
```

##297. Serialize and Deserialize Binary Tree

- 难度：Hard
- 题意：

要求设计二叉树的序列化和反序列化函数，要求序列化返回一个字符串。反序列化根据这个字符串能新构造出这颗树

- 思路：

其实leetcode中对树的表示就是一种很好的序列化。因此序列化只需要借助队列，按层次遍历，若无节点则用“#”代替。若该树为完全二叉树，则对第i个节点，其左右节点分别为2i+1和2i+2。但是若非完全二叉树，则对于之前出现的每个#，由于#没有左右子节点，会使当前节点的子节点提前2*（当节点之前#个数）

- 代码：

```
class Codec:

    def serialize(self, root):
        """Encodes a tree to a single string.

        :type root: TreeNode
        :rtype: str
        """
        if not root:return None
        q =[root]
        i,endIndex=0,1
        while i<endIndex:
            if q[i]:
                q.append(q[i].left)
                q.append(q[i].right)
                endIndex+=2;
            i+=1
        res=[]
        for node in q:
            if node:
                res.append(str(node.val))
            else:
                res.append("#")
        return ','.join(res)
```

```
def deserialize(self, data):
    """Decodes your encoded data to tree.

    :type data: str
    :rtype: TreeNode
    """
    if not data:return None
    values = data.split(',')
    nodes = [None]*len(values)
    count=0
    for i in range(len(values)):
        if values[i] != '#':
            nodes[i] = TreeNode(int(values[i]))
    for i in range(len(nodes)):
        if not nodes[i]:
            count+=1
        else:
            nodes[i].left=nodes[2*(i-count)+1]
            nodes[i].right=nodes[2*(i-count)+2]
    return nodes[0]
```
