剑指offer

链表

面试题5: 从尾到头打印链表

输入一个链表，从尾到头打印链表每个节点的值。

思路：

方法一：

```
class Solution:
    # 返回从尾部到头部的列表值序列，例如[1,2,3]
    def printListFromTailToHead(self, listNode):
        l = []
        # 直接遍历一遍链表保存结果到list中，再返回倒序的list即可
        while listNode:
            l.append(listNode.val)
            listNode = listNode.next
        return l[::-1]
```

方法二：

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
from collections import deque
class Solution:
    # 返回从尾部到头部的列表值序列，例如[1,2,3]
    def printListFromTailToHead(self, listNode):
        # 输入链表为空返回空[]
        if listNode is None:
            return []
        queue = deque() # 使用deque
        while listNode: # 当listNode不为空时执行循环
            queue.appendleft(listNode.val) # 将当前的listNode结点的value放到队列的最左侧，即可反向输出链表
            listNode =  listNode.next # 将指针指向下一个结点
        return queue
```

面试题15：链表中倒数第k个结点

输入一个链表，输出该链表中倒数第k个结点。

思路：

方法一：个人总结最佳算法，先计算链表的长度，然后计算找到倒数第k个需要几次循环，并判断其中关系。最后用for循环，不断将指针指向下一个节点，即为所求。

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
```

```
    def FindKthToTail(self, head, k):
        len_node = 0
        temp = head
        while temp:
            temp = temp.next
            len_node += 1
        run_times = len_node - k
        if run_times < 0 or k < 0:
            return
        for i in range(run_times):
            head = head.next
        return head
```

方法二：

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def FindKthToTail(self, head, k):
        tmp = head
        count = 0
        # 统计链表中有多少个元素
        while tmp:
            tmp = tmp.next
            count += 1
        if count<k or k<0: # 链表中元素不足k个或者k<0无法得到结果，直接返回
            return
        #一个指针先向前走K步，另一个从头开始。
        #然后两个指针同时向前走，当前一个指针到头时，后一个指针所在位置就是倒数第K个节点的位置
        quick_listnode = head
        slow_listnode = head
        for i in range(0,k):
            quick_listnode = quick_listnode.next
        while quick_listnode:
            quick_listnode = quick_listnode.next
            slow_listnode = slow_listnode.next
        return slow_listnode
```

面试题16：反转链表

输入一个链表，反转链表后，输出链表的所有元素。

思路：定义两个变量，分别保存前指针和后指针。

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    # 返回ListNode
    def ReverseList(self, pHead):
        if not pHead:
```

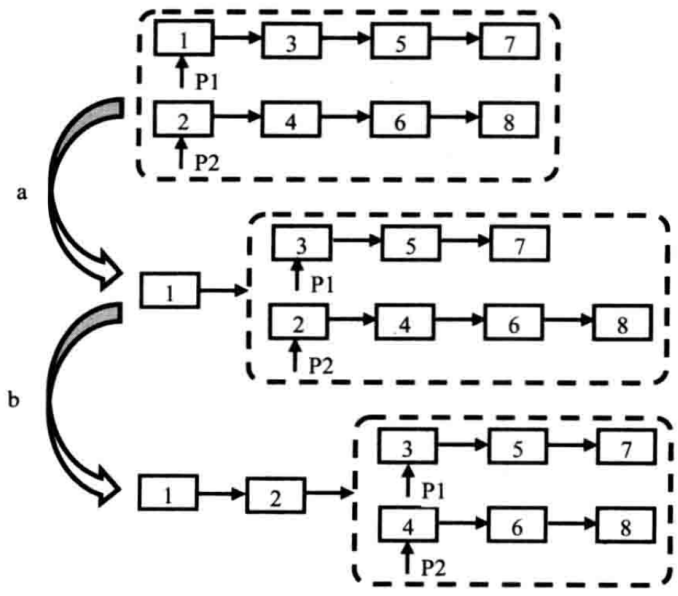
```
        return None
    # 当前节点是pHead，Pre为当前节点的前一节点，Next为当前节点的下一节点
    # 需要pre和next的目的是让当前节点从pre->head->next1->next2变成pre<-head next1->next2
    # 即pre让节点可以反转所指方向，但反转之后如果不用next节点保存next1节点的话，此单链表就此断开了
    # 所以需要用到pre和next两个节点
    # 1->2->3->4->5
    # 1<-2<-3 4->5
    Pre = None
    Next = None
    while pHead:
        Next = pHead.next # 保存当前结点的next指针，方便反转第一次后，在链表断开的情况下，依然找到原来的下一个结点
        pHead.next = Pre # 反转链表，将当前结点的next指针指向前一个结点
        Pre = pHead # 保存当前结点，更新Pre，方便下一次调用
        pHead = Next # 让pHead按原来顺序走到第二个结点
    return Pre
```

面试题17：合并两个排序的链表

输入两个单调递增的链表，输出两个链表合成后的链表，当然我们需要合成后的链表满足单调不减规则。

思路：

首先分析合并两个链表的过程。我们的分析从合并两个链表的头结点开始。链表 1 的头结点的值小于链表 2 的头结点的值，因此链表 1 的头结点将是合并后链表的头结点（如图 3.8（a）所示）。




```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
```

```
#         self.next = None
class Solution:
    # 返回合并后列表
    def Merge(self, pHead1, pHead2):
        merged = None
        # 当pHead1为空时，返回pHead2
        if not pHead1:
            return pHead2
        # 当pHead2为空时，返回pHead1
        if not pHead2:
            return pHead1
        # 第一个链表中的第一个点小于第二个链表第一个点，那么merged第一个点就是pHead1的第一个点
        # 对于他的next，继续执行递归
        if pHead1.val < pHead2.val:
            merged = pHead1
            merged.next = self.Merge(pHead1.next, pHead2)
        # 第一个链表中的第一个点大于第二个链表第一个点，那么merged第一个点就是pHead1的第一个点
        # 对于他的next，继续执行递归
        else:
            merged = pHead2
            merged.next = self.Merge(pHead1, pHead2.next)
        return merged
```

面试题26：复杂链表的复制

输入一个复杂链表（每个节点中有节点值，以及两个指针，一个指向下一个节点，另一个特殊指针指向任意一个节点），返回结果为复制后复杂链表的head。（注意，输出结果中请不要返回参数中的节点引用，否则判题程序会直接返回空）

思路：

```
# -*- coding:utf-8 -*-
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
class Solution:
    # 返回 RandomListNode
    def Clone(self, pHead):
        if not pHead:
            return pHead
        # 开辟一个新结点
        copy = RandomListNode(pHead.label)
        copy.next = pHead.next
        copy.random = pHead.random
        # 递归剩余结点
        copy.next = self.Clone(pHead.next)
        return copy
```

面试题37：两个链表的第一个公共结点

输入两个链表，找出它们的第一个公共结点。

思路：也可以分别将两个链表加入到两个栈中，再依次弹出，因为他们有公共结点，所以后面的结点都是相同的，找到第一个不相同的点即为所求。

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def FindFirstCommonNode(self, pHead1, pHead2):
        # 最优解：O(m+n)，比放到stack里面做，节省了空间
        if not pHead1 or not pHead2:
            return None
        length1 = 0
        length2 = 0
        p1 = pHead1
        p2 = pHead2
        # 分别计算两个链表的长度
        while p1:
            length1 += 1
            p1 = p1.next
        while p2:
            length2 += 1
            p2 = p2.next
        # 根据两个链表的长度，确定长、短链表和它们之间的长度之差
        if length1 >= length2:
            step = length1 - length2
            longList = pHead1
            shortList = pHead2
        else:
            step = length2 - length1
            longList = pHead2
            shortList = pHead1
        # 让长链表先走step步
        for i in range(0,step):
            longList = longList.next
        # 同时遍历两个链表，让他们不断指向next，并判断何时相等，相等时返回任一一个链表即可
        while longList and shortList:
            if longList == shortList:
                return longList
            longList = longList.next
            shortList = shortList.next
        return None
```

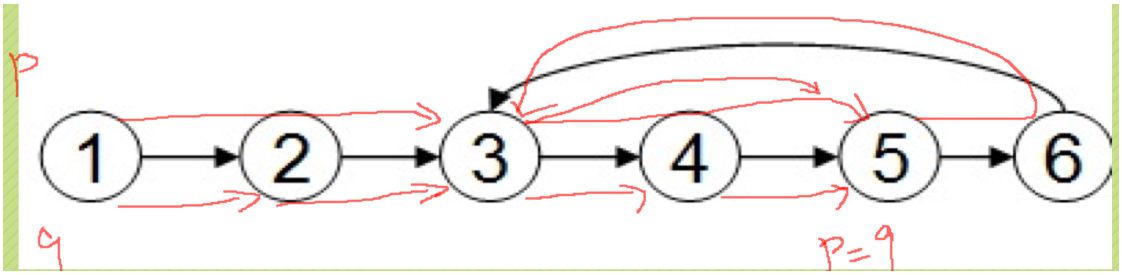
面试题56：链表中环的入口结点

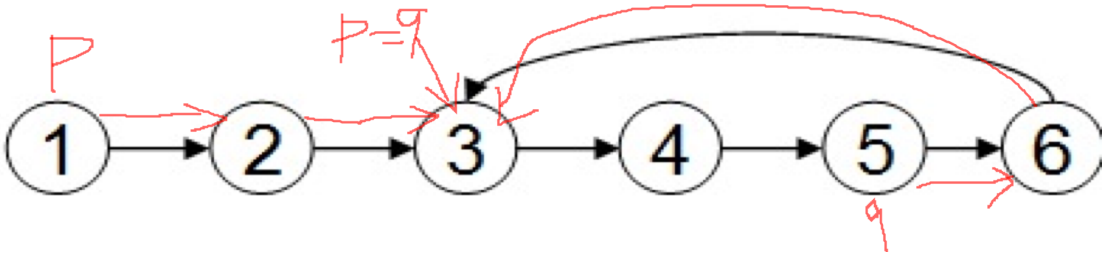
一个链表中包含环，请找出该链表的环的入口结点。

思路：

第一步，找环中相汇点。分别用p1，p2指向链表头部，p1每次走一步，p2每次走二步，直到p1==p2找到在环中的相汇点。

第二步，找环的入口。接上步，当p1==p2时，p2所经过节点数为2x,p1所经过节点数为x,设环中有n个节点,p2比p1多走一圈有2x=n+x; n=x;可以看出p1实际走了一个环的步数，再让p2指向链表头部，p1位置不变，p1,p2每次走一步直到p1==p2;此时p1指向环的入口。






```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def EntryNodeOfLoop(self, pHead):
        if pHead ==None:
            return 1
        if pHead.next==None or pHead.next.next==None:
            return None
        # 使用快慢指针，p每次走两步，q每次走一步
        p = pHead.next.next
        q = pHead.next
        # 第一次循环，直到p和q相遇，p每次走两步，q每次走一步
        while p!=q:
            p = p.next.next
            q = q.next
            if p.next == None or p.next.next ==None:
                return None
        # 第二次循环，直到p和q相遇，让快指针p回到开始的点，p和q每次都走一步
        p = pHead
        while p!=q:
            p = p.next
            q = q.next
        return q
```

面试题57：删除链表中重复的结点

在一个排序的链表中，存在重复的结点，请删除该链表中重复的结点，重复的结点不保留，返回链表头指针。 例如，链表1->2->3->3->4->4->5 处理后为 1->2->5

思路：

解法一：

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def deleteDuplication(self, pHead):
        temp = []
        head = pHead
        # 先将pHead中所有结点的value全部放到temp列表中去
        while head:
            temp.append(head.val)
            head = head.next

        result = ListNode(0) # 创建一个新的指针
        head = result # 让head指向这个指针
        for i in temp:
            if temp.count(i) == 1: # 对于temp中的元素，如果出现次数等于1就添加到head的next指针
                head.next = ListNode(i)
                head = head.next
        return result.next
        # 最后返回result.next，这里用head是因为如果直接操作result最后result会指向最后一个指针无法返回需要的结果
```

解法二：

```
class Solution:
    def deleteDuplication(self, pHead):
        if not pHead:
            return None
        first = ListNode(0) # 生成一个头指针
        last = first
        first.next = pHead
        tmp = pHead
        while tmp and tmp.next:
            if tmp.val == tmp.next.val:
                while tmp.next and tmp.val == tmp.next.val:
                    tmp = tmp.next
            else:
                last.next = tmp # 删除链表中重复的结点
                last = last.next
            tmp = tmp.next
        last.next = tmp
          return first.next
```

二叉树

面试题6: 重建二叉树

输入某二叉树的前序遍历和中序遍历的结果，请重建出该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}，则重建二叉树并返回。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回构造的TreeNode根节点
    def reConstructBinaryTree(self, pre, tin):
        # 判断pre或者tin是否为空，为空就返回None
        if not pre or not tin:
            return None
        # 因为pre为先序遍历，pre中的第一个点为首节点
        root = TreeNode(pre[0])
        # 找到首节点在中序遍历中的索引位置
        pos = tin.index(pre[0])
        # 对左右分别递归  如: [1l2,3,4ll,5,6,7],[3,2,4l1l6,7]
        root.left = self.reConstructBinaryTree(pre[1:pos+1],tin[0:pos])
        root.right = self.reConstructBinaryTree(pre[pos+1:],tin[pos+1:])
        return root
```

面试题18：树的子结构

输入两棵二叉树A，B，判断B是不是A的子结构。（ps：我们约定空树不是任意一个树的子结构）

思路：对于两棵二叉树来说，要判断B是不是A的子结构，首先第一步在树A中查找与B根节点的值一样的节点。 通常对于查找树中某一个节点，我们都是采用递归的方法来遍历整棵树。 第二步就是判断树A中以R为根节点的子树是不是和树B具有相同的结构。 这里同样利用到了递归的方法，如果节点R的值和树的根节点不相同，则以R为根节点的子树和树B肯定不具有相同的节点； 如果它们值是相同的，则递归的判断各自的左右节点的值是不是相同。 递归的终止条件是我们达到了树A或者树B的叶节点。

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def HasSubtree(self, pRoot1, pRoot2):
        # 依题意，如果pRoot2为空，返回False
        if not pRoot1 or not pRoot2:
            return False
        result = False
        # 三个遍历方向
```

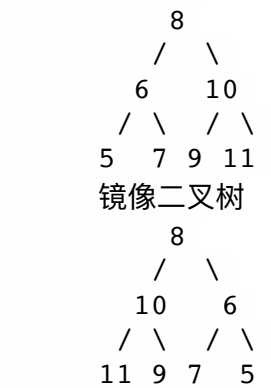
```
        # 如果说当前结点存放的元素相同，则直接往后遍历
        if pRoot1.val == pRoot2.val:
                result = self.IsSubtree(pRoot1, pRoot2)
        # 通过上面if语句的讨论，result还是为False，pRoot1的左子树继续讨论
        if not result:
                result = self.IsSubtree(pRoot1.left, pRoot2)
        # 右子树
        if not result:
                result = self.IsSubtree(pRoot1.right, pRoot2)
        return result

    def IsSubtree(self, p1, p2):
        # p2如果遍历完，则证明pRoot2是pRoot1的子结构，返回True
        if not p2:
                return True
        # 如果说p1遍历完，则证明p2还没有匹配结束，返回False
        if not p1:
                return False
        # 如果说结点元素不相等，则不能匹配
        if p1.val != p2.val:
                return False
        return self.IsSubtree(p1.left, p2.left) and self.IsSubtree(p1.right, p2.right)
```

面试题19：二叉树的镜像

操作给定的二叉树，将其变换为源二叉树的镜像。

二叉树的镜像定义：源二叉树



镜像二叉树

思路：

由题目我们可以知道所谓二叉树的镜像无非就是不断交换根结点的左右子树（注意不是左右孩子，当然说左右孩子也对，但是每次递归的时候真正交换的是两个子树的头结点+整棵子树）。明白了原理，让我们用递归实现吧！

1、我们知道我们要定义一个方法 Mirror ，这个方法的主要功能是实现整棵树的交换，注意是整棵树，而不是某颗子树，为什么这么说，接下来便知道了。

2、首先我们肯定是交换根节点的左右子树嘛！对了，交换完之后，你会发现，对于以根节点左孩子右孩子为根节点的两棵子树，也需要做同样的事情，因此，为了不把代码写得跟论文那么长，我们决定用递归，自己调用自己的功能（想想它自己的功能是什么？无非就是将以参数为头结点的两棵子树交换嘛！），但传入的参数为根节点的左孩子、右孩子了

3、由于是先序遍历，所以呢！我们就先对左子树进行递归，在递归右子树

4、当2,3都执行完成，也就是我们的Mirror函数结束了，你会发现，这个Mirror函数实际操作的是整棵树，但函数体我们很直观理解的代码却只有根节点左右子树的交换的那部分代码（没涉及递归的那部分），其他谁帮我们做了呢？没错，就是递归函数。

还有就是上面强调的Mirror这个大函数是对整棵大树而言的，为什么呢？因为这个函数的方法体，已经包含了帮你处理左右子树的递归函数了！

最后，我不管你是看懂了还是没看懂，我尽力了。对于递归的理解，就是某些相同的功能跟自己实现的功能一模一样时，用递归，但要注意递归的终止条件哦！

```
class Solution:
    # 返回镜像树的根节点
    def Mirror(self, root):
        if not root:
            return
        if root:
            # 用先序遍历从根节点出发
            root.left,root.right = root.right,root.left
            self.Mirror(root.left)
            self.Mirror(root.right)
```

面试题23：从上往下打印二叉树

从上往下打印出二叉树的每个节点，同层节点从左至右打印。

思路：

```
public class Solution {
    public ArrayList<Integer> PrintFromTopToBottom(TreeNode root) {
        ArrayList<Integer> a = new ArrayList<Integer>();
        if(root == null){
            return a;
        }
        Queue <TreeNode> queue= new LinkedList<TreeNode>();
        queue.add(root); //初始化，让首节点进入queue
        while(!queue.isEmpty()){
                int end = queue.size(); //得到queue的size
            for(int start = 0; start<end; start++){ //给定for循环，打印出一行中从开始到结束的所有节点
                TreeNode current = queue.poll(); //取出queue并加入tmp
                a.add(current.val);//不需要tmp直接可以加入到a
                if(current.left!=null){
                    queue.add(current.left);//把下一层节点加入queue中
                }
                if(current.right!=null){
                    queue.add(current.right);//把下一层节点加入queue中 }}}
        return a;
    }
```

}

面试题25：二叉树中和为某一值的路径

输入一颗二叉树和一个整数，打印出二叉树中结点值的和为输入整数的所有路径。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。

思路：

```
class Solution:
    # 返回二维列表，内部每个列表表示找到的路径
    def FindPath(self, root, expectNumber):
        if not root:
            return []
        if not root.left and not root.right and expectNumber == root.val: # 左右子孩子都是空，并且当前值等于expectNumber返回当前结点的值
            return [[root.val]]
        result = []
        left = self.FindPath(root.left, expectNumber - root.val)
        right = self.FindPath(root.right, expectNumber - root.val)
        for i in left+right:
            result.append([root.val]+i)
        return result
```

面试题39：二叉树的深度

输入一棵二叉树，求该树的深度。从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的长度为树的深度。

思路：

方法一：递归

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def TreeDepth(self, pRoot):
        if not pRoot:
            return 0
        left = self.TreeDepth(pRoot.left)
        right = self.TreeDepth(pRoot.right)
        return max(left+1,right+1)
```

方法二：非递归，层序遍历

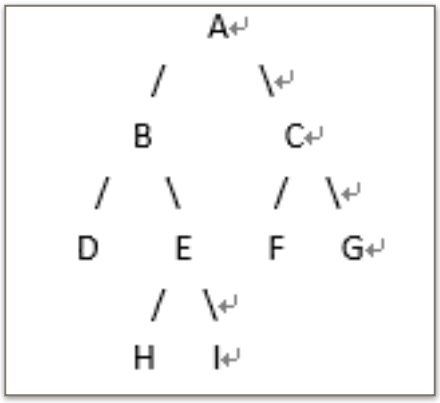
```
from collections import deque
def TreeDepth(self, pRoot):
    if not root:
        return
    depth = 0
    queue = deque([pRoot])
    last =pRoot
    while queue:
        current_node = queue.popleft()
        if current_node.left:
            queue.append(current_node.left)
```

```
        if current_node.right:
            queue.append(current_node.right)
        if current_node == last:
            depth+=1
            if queue:
                last = queue[-1]
    return depth
```

面试题58：二叉树的下一个结点

给定一颗二叉树和其中的一个结点，如何找出中序遍历顺序的下一个结点？树中的结点除了有两个分别指向左右子结点的指针以外，还有一个指向父节点的指针。

思路：



1. 若该节点存在右子树：则下一个节点为右子树最左子节点（如图节点B）
2. 若该节点不存在右子树：这时分两种情况：

2.1 该节点为父节点的左子节点，则下一个节点为其父节点（如图节点D）

2.2 该节点为父节点的右子节点，则沿着父节点向上遍历，知道找到一个节点的父节点的左子节点为该节点，则该节点的父节点下一个节点（如图节点I，沿着父节点一直向上查找找到B（B为其父节点的左子节点），则B的父节点A为下一个节点）。

```
# -*- coding:utf-8 -*-
# class TreeLinkNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None
class Solution:
    def GetNext(self, pNode):
        # 中序遍历：左中右
        if not pNode:
            return None
        # 判断当前结点有没有右子树，若存在右子树，下一个结点就是右子树的最左子结点
        if pNode.right:
            pNode = pNode.right # 进入到右子树的父节点
            while pNode.left: # 不断遍历左结点
                pNode = pNode.left
            return pNode # 返回右子树的最左子结点
        # 没有右子树就向上遍历找到他的父亲结点，如果他是父亲结点的左孩子结点，下一个结点就是父亲结点
```

```
        # 如果不是，就继续向上找寻当前父亲结点w的父亲结点q，并判断q是否为他的父亲结点e的左孩子
        # 如此反复直到找到匹配的情况，找不到就返回None
        while pNode.next:
            if pNode.next.left == pNode:
                return pNode.next
            pNode = pNode.next
        return None
```

面试题58：对称的二叉树

请实现一个函数，用来判断一颗二叉树是不是对称的。注意，如果一个二叉树同此二叉树的镜像是同样的，定义其为对称的。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def isSymmetrical(self, pRoot):
        if not pRoot:
            return True
        return self.judge_helper(pRoot.left,pRoot.right)

    def judge_helper(self,left,right):
        if not left and not right: # 当left和right都为空时，返回True，跳出
            return True
        if not left or not right: # 当left和right有一边为空时，返回False，跳出
            return False
        if left.val == right.val: # 判断left和right的val是否相等，相等就继续递归判断应该对成的两个点是否对称
            return self.judge_helper(left.left,right.right) and self.judge_helper(left.right,right.left)
        return False
```

面试题60：把二叉树打印成多行

从上到下按层打印二叉树，同一层结点从左至右输出。每一层输出一行。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
from collections import deque
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, pRoot):
        if not pRoot:
            return[]
        res = [] # res列表用于记录最终结果
```

```
        tmp = [] # 用于每一层结点的缓存
        last = pRoot # 定义last结点表示下一层的最末结点
        next_level = deque([pRoot]) # 初始化下一行结点的队列，起始位为根结点[pRoot]
        while next_level: # 当下一行有结点时
            current_node = next_level.popleft() # 取出下一行结点最左边的结点
            tmp.append(current_node.val) # 将最左边的值放入tmp
            if current_node.left: # 如果当前结点有左孩子
                next_level.append(current_node.left) # 在队列右侧添加该左孩子结点
            if current_node.right: # 如果当前结点有右孩子
                next_level.append(current_node.right) # 在队列右侧添加该右孩子结点
            if current_node == last: # 若当前结点正好是下一行结点的最后一个位置
                res.append(tmp) # 给结果增添上tmp，也就是当前行的所有结点
                tmp = [] # 清空tmp
                if next_level: # 如果还有下一层
                    last = next_level[-1] # 令last等于末尾项
        return res
```

面试题61：按之字形顺序打印二叉树

请实现一个函数按照之字形打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右至左的顺序打印，第三行按照从左到右的顺序打印，其他行以此类推。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
from collections import deque
class Solution:
    def Print(self, pRoot):
        if not pRoot:
            return []
        result = []
        tmp = []
        last = pRoot #记录每层的最后一个结点，方便层序遍历换行
        left_to_right = True
        next_level_node_LtoR = deque([pRoot]) #初始化下一层所有节点的队列，起初为只有根结点
        while next_level_node_LtoR: #当下一层不为空时
            current_node = next_level_node_LtoR.popleft() #不停从下层队列左边弹出
            tmp.append(current_node.val) #将弹出结点放入tmp中
            if current_node.left:
                next_level_node_LtoR.append(current_node.left)
            if current_node.right:
                next_level_node_LtoR.append(current_node.right)
            if current_node == last: #当运行到最后一个结点，给result赋值当前行的所有元素，也就是
tmp
                if left_to_right:
                    result.append(tmp)
                else: #若该行应该为从右到左，则倒序append
```

```
                result.append(tmp[::-1])
            tmp = [] #清空tmp，以便下一层继续使用
            left_to_right = not left_to_right #调整此项，颠倒下一行的输出顺序
            if next_level_node_LtoR: #更新下一行的last结点，如果下一行已经没有元素就会退出
                last = next_level_node_LtoR[-1]
        return result
```

面试题62：序列化二叉树

请实现两个函数，分别用来序列化和反序列化二叉树

思路：也可以用层序遍历

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    index = -1
    def Serialize(self, root):
        if root == None:
            return '#'
        return str(root.val) + ',' + self.Serialize(root.left) + ',' + self.Serialize(root.right)
        # return形如s = '5,#,#,6'

    def Deserialize(self, s):
        self.index += 1 #每次递归给index+1去到下一个字符
        l = s.split(',')
        if self.index >= len(s):
            return None
        root = None
        if l[self.index] != '#':
            root = TreeNode(int(l[self.index]))
            root.left = self.Deserialize(s) # 对当前结点的左右结点递归
            root.right = self.Deserialize(s)
        return root
```

面试题：平衡二叉树

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def IsBalanced_Solution(self, pRoot):
        if not pRoot:
            return True
        # 对左右两边同时递归
        if abs(self.Tree_depth(pRoot.left) - self.Tree_depth(pRoot.right)) > 1:
            return False
        return self.IsBalanced_Solution(pRoot.left) and self.IsBalanced_Solution(pRoot.right)
```

```
    def Tree_depth(self,pRoot):
        if not pRoot:
            return 0
        # 二叉树的后序遍历
        left = self.Tree_depth(pRoot.left)
        right = self.Tree_depth(pRoot.right)
        return max(left+1,right+1) # 返回当前树深
```

二叉搜索树

面试题24：二叉搜索树的后序遍历

输入一个整数数组，判断该数组是不是某二叉搜索树的后序遍历的结果。如果是则输出Yes,否则输出No。假设输入的数组的任意两个数字都互不相同。

思路：

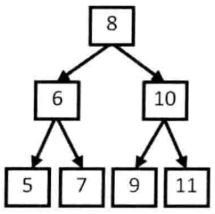


图 4.6　后序遍历序列 5、7、6、9、11、10、8 对应的二叉搜索树

在后序遍历得到的序列中，最后一个数字是树的根结点的值。数组中前面的数字可以分为两部分：第一部分是左子树结点的值，它们都比根结点的值小；第二部分是右子树结点的值，它们都比根结点的值大。

以数组{5, 7, 6, 9, 11, 10, 8}为例，后序遍历结果的最后一个数字 8 就是根结点的值。在这个数组中，前 3 个数字 5、7 和 6 都比 8 小，是值为 8 的结点的左子树结点；后 3 个数字 9、11 和 10 都比 8 大，是值为 8 的结点的右子树结点。

我们接下来用同样的方法确定与数组每一部分对应的子树的结构。这其实就是一个递归的过程。对于序列 5、7、6，最后一个数字 6 是左子树的根结点

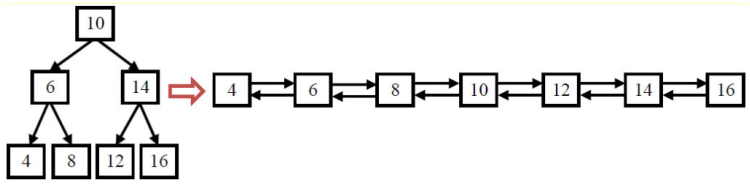
```
# -*- coding:utf-8 -*-
class Solution:
    def VerifySquenceOfBST(self, sequence):
        if not sequence:
            return False
        root = sequence[-1] # 后续遍历中，root是sequence最后一个结点
        d_index = 0
        # 根据二叉搜索树左子树小于root值，找到第一个大于root的index，即为左右子树的划分索引
        for i in range(0,len(sequence)-2):
            d_index +=1 #确定左右子树划分索引
            if sequence[i] > root:
                break
        # 二叉搜索树右子树大于root值
        for j in range(d_index, len(sequence)-2):
            if sequence[j] < root:
```

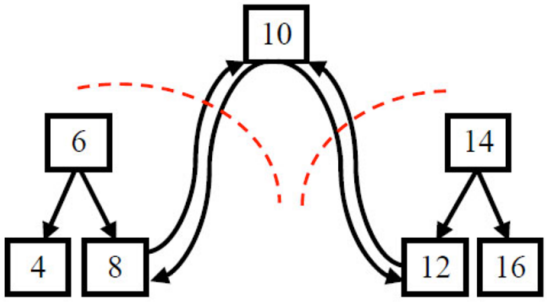
```
            return False
        # 判断左子树是不是二叉搜索树
        left_yes_or_no = True
        left = sequence[:d_index]
        if left:
            left_yes_or_no = self.VerifySquenceOfBST(left)
        # 判断右子树是不是二叉搜索树
        right_yes_or_no = True
        right = sequence[d_index:len(sequence)-1]
        if right:
            right_yes_or_no = self.VerifySquenceOfBST(right)
        return left_yes_or_no and right_yes_or_no
```

面试题27：二叉搜索树与双向链表

输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的双向链表。要求不能创建任何新的结点，只能调整树中结点指针的指向。

思路：





首先，我们知道：在二叉树中，每个结点都有两个指向子结点的指针。在双向链表中，每个结点也有两个指针，它们分别指向前一个结点和后一个结点。

其次，由于要求转换之后的链表是排好序的，我们可以中序遍历树中的每一个结点，这是因为中序遍历算法的特点是按照从小到大的顺序遍历二叉树的每一个结点。

最后，按照中序遍历的顺序，当我们遍历转换到根结点（值为10的结点）时，它的左子树已经转换成一个排序的链表了，并且处在链表中的最后一个结点是当前值最大的结点。我们把值为8的结点和根结点链接起来，此时链表中的最后一个结点就是10了。接着我们去遍历转换右子树，并把根结点和右子树中最小的结点链接起来。

很明显，转换它的左子树和右子树，由于遍历和转换过程是一样的，很自然地想到可以用递归。

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
```

```
#         self.left = None
#         self.right = None
class Solution:
    def Convert(self, pRootOfTree):
        if not pRootOfTree:
            return None
        # 中序遍历
        def In_Order(root):
            if not root:
                return None
            # 访问左子树
            In_Order(root.left)
            # 把左子树指向pre
            root.left = pre[0]
            # 如果pre[0] 不为None的话，则把它指向root
            if pre[0]:
                pre[0].right = root
            # 把pre[0] 指向root
            pre[0] = root
            # 访问右子树
            In_Order(root.right)
            return root
        # pre赋值为列表，因为要在内部函数中修改，如果声明变量的话，会抛出异常
        # 用来记录左子树的最后一个节点
        pre = [None]
        root = In_Order(pRootOfTree)
        # 返回的root是链表的最后一个结点，所以往前遍历，直到链表的头结点再返回
        while root.left:
            root = root.left
        return root
```

面试题63：二叉搜索树的第k个结点

给定一颗二叉搜索树，请找出其中的第k大的结点。例如， 5 / \ 3 7 /\ /\ 2 4 6 8 中，按结点数值大小顺序第三个结点的值为4。

思路：

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回对应节点TreeNode
        #第三个节点是4
        #前序遍历5324768
        #中序遍历2345678
        #后序遍历2436875
        #所以是中序遍历，左根右
    def KthNode(self, pRoot, k):
        global result #设置全局变量
        result = []
```

```
        self.kthNode(pRoot)
        if k<=0 or len(result)<k:
            return None
        else:
            return result[k-1]#返回中序遍历第k个即为第k大的结点

    def kthNode(self, pRoot):
        if not pRoot:
            return None
        self.kthNode(pRoot.left)
        result.append(pRoot)#中序遍历所有结点，加入到result中
        self.kthNode(pRoot.right)
```

数组

面试题3: 二维数组中的查找

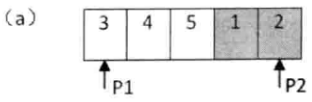
在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # array 二维列表
    def Find(self, target, array):
        if array == [[]]: # 不能用not array，因为[[]]不为空，它是有一个元素为列表，但列表为空的数组
            return False
        x = len(array[0]) - 1
        y = len(array) - 1
        # 从左下角开始遍历
        startX = 0
        startY = y
        # 循环条件，到右上点结束，因为查找方向为向右或者向上，所以横坐标要比不大于array的最大x，纵坐标>=0
        # 可通过画图直观理解何时向上何时向右
        #  1  2  3  4
        #  5  6  7  8
        #  9 10 11 12
        # 13 14 15 16
        while startX <= x and startY >= 0:
            if array[startX][startY] > target:
                startY -= 1
            elif array[startX][startY] < target:
                startX += 1
            else:
                return True
        return False
```

面试题10: 旋转数组的最小数字

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。 输入一个非递减排序的数组的一个旋转，输出旋转数组的最小元素。 例如数组{3,4,5,1,2}为{1,2,3,4,5}的一个旋转，该数组的最小值为1。 NOTE：给出的所有元素都大于0，若数组大小为0，请返回0。



思路：最小值正好位于两段中间，用二分查找即可。

```
# -*- coding:utf-8 -*-
from collections import deque
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        解法一：
        q = deque()
        min = rotateArray[0]
        for i in rotateArray:
            if i <= min:
                min = i
                min_index=rotateArray.index(i)
        for j in rotateArray:
            q.append(j)
            if rotateArray.index(j) >= min_index:
                q.appendleft(j)
        return list(q)

        解法二：  折半查找法
        # 折半查找法
        start = 0
        end = len(rotateArray) - 1
        while start<end:
            mid = (start+end)//2
            # 中间值<最左边值，由于本身是有两个递增数组组成，所以中间值右边的根本不需要考虑，肯定都比他大
            # 所以将end移到mid
            if rotateArray[mid] < rotateArray[start]:
                end = mid
            # 中间值>数组start点值，向右移动start点让他是mid
            elif rotateArray[mid] > rotateArray[start]:
                start = mid
            else:
                return rotateArray[mid+1]
```

面试题14：调整数组顺序使奇数位于偶数前面

输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有的奇数位于数组的前半部分，所有的偶数位于位于数组的后半部分，并保证奇数和奇数，偶数和偶数之间的相对位置不变。

思路：

```
class Solution:
    def reOrderArray(self, array):
        if not array:
            return []
        even = []
        odd = []
        for i in array:
            if i%2 == 1:
                odd.append(i)
            else:
                even.append(i)
        return odd + even
```

面试题29：数组中出现次数超过一半的数字

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如输入一个长度为9的数组{1,2,3,2,2,2,5,4,2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。如果不存在则输出0。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        dict = {}
        for no in numbers:
            if not dict.has_key(no):
                dict[no] = 1
            else:
                dict[no] = dict[no] + 1
            if dict[no] > len(numbers)/2:
                return no
        return 0
```

面试题31：连续子数组的最大和

HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢？例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。你会不会被他忽悠住？(子向量的长度至少是1)

思路：

表 5.2　计算数组{1, -2, 3, 10, -4, 7, 2, -5}中子数组的最大和的过程

步骤	操作	累加的子数组和	最大的子数组和
1	加 1	1	1
2	加-2	-1	1
3	抛弃前面的和-1，加 3	3	3
4	加 10	13	13
5	加-4	9	13
6	加 7	16	16
7	加 2	18	18
8	加-5	13	18

```
# -*- coding:utf-8 -*-
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        if not array:
            return []
        sum = -0xffffff
        result = sum
        for i in array:
```

```
            if sum > 0:
                sum += i
            else:
                sum = i
            result = max(sum,result)
        return result
```

面试题33：把数组排成最小的数

输入一个正整数数组，把数组里所有数字拼接起来排成一个数，打印能拼接出的所有数字中最小的一个。例如输入数组{3，32，321}，则打印出这三个数字能排成的最小数字为321323。

思路： 若 a＋b<b+a a排在在前的规则排序, 如 221 因为 212 < 221 所以 排序后为 212, 实际上sort()方法在不传入参数func的时候 默认cmp为None。调用的是lambda x,y: cmp(x, y),而实际上就是调用cmp函数

```
# -*- coding:utf-8 -*-
class Solution:
    def cmp(self,x,y):
        s1 = str(x) + str(y)
        s2 = str(y) + str(x)
        if s1 <= s2:
            return s1
        if s1 > s2:
            return s2

    def PrintMinNumber(self, numbers):
        if not numbers:
            return ''
        numbers.sort()
        # 定义cmp函数，比较左+右，右+左，判断哪个小，然后返回
        # 不断替换index为0，1的值为cmp的返回值，直至最后数组中仅剩一个元素
        while len(numbers) > 1:
            numbers[0] = self.cmp(numbers[0],numbers[1])
            numbers.remove(numbers[1])
        return numbers[0]
```

面试题33(变形)：把数组排成最大的数

```
# -*- coding:utf-8 -*-
class Solution:
    def cmp(self,x,y):
        s1 = str(x) + str(y)
        s2 = str(y) + str(x)
        if s1 >= s2:
            return s1
        if s1 < s2:
            return s2

    def PrintMinNumber(self, numbers):
        if not numbers:
            return ‘'
       #让原来的数组先倒序
        numbers.sort(reverse = True)
        # 定义cmp函数，比较左+右，右+左，判断哪个大，然后返回
        # 不断替换index为0，1的值为cmp的返回值，直至最后数组中仅剩一个元素
        while len(numbers) > 1:
            numbers[0] = self.cmp(numbers[0],numbers[1])
```

```
            numbers.remove(numbers[1])
        return numbers[0]
```

面试题38：数字在排序数组中出现的次数

统计一个数字在排序数组中出现的次数。

思路：最好用二分查找，只要O(logN)

解法一：

```
# -*- coding:utf-8 -*-
class Solution:
    def GetNumberOfK(self, data, k):
        if not data or k not in data:
            return 0
        # 二分查找分别找k第一次出现的位置和最后一次出现的位置
        # firstk的位置
        left = 0
        right = len(data) - 1
        firstk = 0
        lastk = 0
        while left <= right: # 注意终止条件为left <= right
            mid = (left+right)//2
            if data[mid] < k:
                left = mid + 1
            elif data[mid] > k:
                right = mid - 1
            else:
                if mid == 0:
                    firstk = mid
                    break
                elif data[mid-1]!=k:
                    firstk = mid
                    break
                else:
                    right = mid - 1

        # lastk的位置
        left = 0
        right = len(data) - 1
        while left <= right:
            mid = (left+right)//2
            if data[mid] < k:
                left = mid + 1
            elif data[mid] > k:
                right = mid - 1
            else:
                if mid == len(data) - 1:
                    lastk = mid
                    break
                elif data[mid+1]!=k:
                    lastk = mid
                    break
                else:
                    left = mid + 1
        return lastk - firstk + 1
```

面试题40：数组中只出现一次的数字

一个整型数组里除了两个数字之外，其他的数字都出现了两次。请写程序找出这两个只出现一次的数字。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # 返回[a,b] 其中ab是出现一次的两个数字
    def FindNumsAppearOnce(self, array):
        l = []
        for element in array:
            if element in l:
        # 如果数字在l中已经存在，再次出现的时候就消除他，因为他们都出现两次，所以这一次之后不会再度出现
                l.remove(element)
        # 未出现过就在l中添加
            else:
                l.append(element)
        return l
```

面试题51：数组中重复的数字

在一个长度为n的数组里的所有数字都在0到n-1的范围内。 数组中某些数字是重复的，但不知道有几个数字是重复的。也不知道每个数字重复几次。请找出数组中任意一个重复的数字。 例如，如果输入长度为7的数组{2,3,1,0,2,5,3}，那么对应的输出是第一个重复的数字2。

思路：

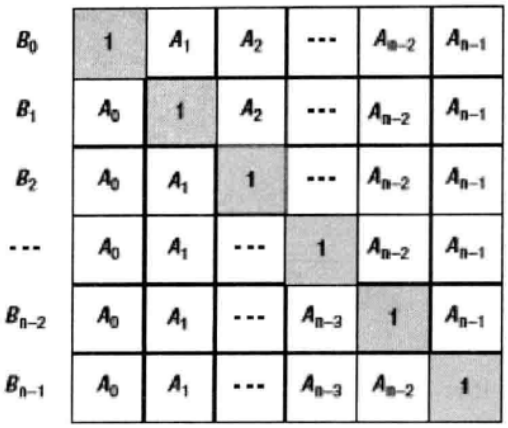
```
# -*- coding:utf-8 -*-
    # 这里要特别注意~找到任意重复的一个值并赋值到duplication[0]
    # 函数返回True/False
class Solution:
    def duplicate(self, numbers, duplication):
        if numbers == None or numbers == []:
            return 0
        l = []
        for i in numbers:
            if i in l:
                duplication[0] = i
                return 1
            else:
                l.append(i)
        return 1
```

面试题52：构建乘积数组

给定一个数组A[0,1,...,n-1],请构建一个数组B[0,1,...,n-1],其中B中的元素B[i]=A[0]*A[1]*…*A[i-1]*A[i+1]*...*A[n-1]。不能使用除法。

思路：下三角用连乘可以很容求得，上三角，从下向上也是连乘。因此我们的思路就很清晰了，先算下三角中的连乘，即我们先算出B[i]中的一部分，然后倒过来按上三角中的分布规律，把另一部分也乘进去。

```
# -*- coding:utf-8 -*-
class Solution:
    def multiply(self, A):
```




```
        if not A:
            return []
        # 计算左三角
        num = len(A)
        B = [None] * num
        B[0] = 1
        for i in range(1, num):
            B[i] = B[i-1] * A[i-1]
        # 计算右三角
        # 自下而上
        # 保留上次的计算结果乘本轮新的数,因为只是后半部分进行累加，所以设置一个tmp，
能够保留上次结果
        tmp = 1
        for i in range(num-2, -1, -1):
            tmp *= A[i+1]
            B[i] *= tmp
        return B
```

栈

面试题7: 用两个栈实现队列

用两个栈来实现一个队列，完成队列的Push和Pop操作。 队列中的元素为int类型。

思路:

```
# -*- coding:utf-8 -*-
class Solution:
    global stack1,stack2
    stack1 = []
    stack2 = []
    # 使用两个stack，stack1用来进栈，stack2是为了出栈时让stack1的所有元素先pop到stack2中，
    # 这样方便让队列的最顶端的元素最先出来，即从stack2中pop出的第一个元素
    # pop出来之后恢复原来的stack1，将stack2别的元素依次pop到stack1中
    def push(self, node):
        stack1.append(node)
```

```
    def pop(self):
        while stack1:
            stack2.append(stack1.pop())
        res = stack2.pop()
        while stack2:
            stack1.append(stack2.pop())
        return res
```

面试题22：栈的压入弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如序列1,2,3,4,5是某栈的压入顺序，序列4，5,3,2,1是该压栈序列对应的一个弹出序列，但4,3,5,1,2就不可能是该压栈序列的弹出序列。（注意：这两个序列的长度是相等的）

思路：借用一个辅助的栈，遍历压栈顺序，先将第一个放入栈中，这里是1，然后判断栈顶元素是不是出栈顺序的第一个元素，这里是4，很显然1≠4，所以我们继续压栈，直到相等以后开始出栈，出栈一个元素，则将出栈顺序向后移动一位，直到不相等，这样循环等压栈顺序遍历完成，如果辅助栈还不为空，说明弹出序列不是该栈的弹出顺序。

举例：

入栈1,2,3,4,5

出栈4,5,3,2,1

首先1入辅助栈，此时栈顶1≠4，继续入栈2

此时栈顶2≠4，继续入栈3

的值。数字 5 比 6 小，是值为 6 的结点的左子结点，而 7 则是它的右子结点。同样，在序列 9、11、10 中，最后一个数字 10 是右子树的根结点，数字 9 比 10 小，是值为 10 的结点的左子结点，而 11 则是它的右子结点。

我们再来分析另一个整数数组{7, 4, 6, 5}。后序遍历的最后一个数是根结点，因此根结点的值是 5。由于第一个数字 7 大于 5，因此在对应的二叉搜索树中，根结点上是没有左子树的，数字 7、4 和 6 都是右子树结点的值。但我们发现在右子树中有一个结点的值是 4，比根结点的值 5 小，这违背了二叉搜索树的定义。因此不存在一棵二叉搜索树，它的后序遍历的结果是 7、4、6、5。

此时栈顶3≠4，继续入栈4

此时栈顶4＝4，出栈4，弹出序列向后一位，此时为5，辅助栈里面是1,2,3

此时栈顶3≠5，继续入栈5 （注意：出栈后要对出栈变化后的栈用while循环保证此时栈顶层第一个节点先跟后续序列进行比较，再压入新的元素）

此时栈顶5=5，出栈5,弹出序列向后一位，此时为3，辅助栈里面是1,2,3

....

依次执行，最后辅助栈为空。如果不为空说明弹出序列不是该栈的弹出顺序。

```
# -*- coding:utf-8 -*-
class Solution:
    def IsPopOrder(self, pushV, popV):
        stack = []
        if not pushV or not popV:
            return False
        for i in pushV:
            stack.append(i)
            while stack and stack[-1]==popV[0] :
```

```
            stack.pop()
            popV.pop(0)
        if stack:
            return False
        return True
```

递归

面试题9: 斐波那契额数列

大家都知道斐波那契数列，现在要求输入一个整数n，请你输出斐波那契数列的第n项。n<=39

思路：

解法一：

```
# -*- coding:utf-8 -*-
class Solution:
    def Fibonacci(self, n):
        if n < 2:
            return n
        else:
            a = [0,1,1]
            for i in range(3,n+1):
                    a.append(a[i-1]+a[i-2])
            return a[n]
```

解法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def Fibonacci(self, n):
        if n <= 0:
            return 0
        if n == 1 or n == 2:
            return 1
        prepre = 1
        pre = 1
        current = 0
        while n > 2:
            current = prepre + pre
            prepre = pre
            pre = current
            n -= 1
        return current
```

面试题：跳台阶，矩形覆盖（**一样的规律，都是这一项，对应前两项的加和**）

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

思路：这一题就是延续斐波那契数列的思路，(注意：斐波那契数列开始是1，1，2不是1，2开始)青蛙跳1级台阶有1种跳法，2级台阶有2种跳法，3级台阶时可以从1级台阶跳上来也可以从2级台阶跳上来，**即等于1级台阶的跳法加2级台阶的跳法因此n级台阶共有n-2级台阶跳法数+n-1级台阶跳法数**，n=1，sum=1,n=2,sum=2,n=3,sum=3;

```
# -*- coding:utf-8 -*-
class Solution:
    def jumpFloor(self, number):
```

```
        if number <= 0:
            return 0
        if number == 1 or number == 2:
            return number
        first = 1
        second = 2
        current = 0
        while number > 2:
            current = first + second
            first = second
            second = current
            # 当number等于3时，减去1，正好只执行了一次，所以以此类推，可以这样操作
            number -= 1
        return current
        # return self.jumpFloor(number - 2) + self.jumpFloor(number -1) 递归
```

面试题：变态跳台阶

一只青蛙一次可以跳上1级台阶，也可以跳上2级……它也可以跳上n级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

思路：每个台阶都有跳与不跳两种情况（除了最后一个台阶），最后一个台阶必须跳。所以共用2^(n-1)中情况。

```
class Solution {
public:
    int jumpFloorII(int number) {
        return pow(2,number-1);
    }
};
```

```
# -*- coding:utf-8 -*-
class Solution:
    def jumpFloorII(self, number):
        # 每个台阶都有两种情况，跳或者不跳，然而最后一个台阶必须要跳
        return 2**(number-1)
```

其他

面试题10: 二进制中1的个数

输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

思路：

```
public class Solution {
    // 减1的实质：把最右的1借走去减，此位1变0，右边其余0变1
    // 再与它原来的值做位与运算，刚刚变0的1还是0，但是更右边变1的0全又变回0
    // 最终n就会变0结束循环，以上每一次循环都会把一个1变0，所以循环多少次就是有几个1
    public int NumberOf1(int n) {
        int count = 0;
        while(n != 0){
            n = n&(n-1);
            count ++;
        }
        return count;
    }
```

}

面试题11: 数值的整数次方

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方。

思路:

```
class Solution:
    def Power(self, base, exponent):
        if base == 0: #底为0，return 0
            return 0
        if base == 1 or exponent == 0: #底为1，或指数为0，都返回1
            return 1
        # 其余情况，先按|base|的exponent次方计算；exponent>0 => return result
        # else: return 1/result
        result = 1
        for i in range(abs(exponent)):
            result *= base
        if exponent < 0:
            return 1/result
        else:
            return result
```

面试题20：顺时针打印矩阵

输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字，例如，如果输入如下矩阵：
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10.

思路:

```
# -*- coding:utf-8 -*-
class Solution:
    # matrix类型为二维列表，需要返回列表
    def printMatrix(self, matrix):
        if not matrix:
            return []
        X = len(matrix[0])
        Y = len(matrix)
        No_of_elements = X * Y
        res = []
        start = 0
        # 循环终止条件为res元素个数 = 矩阵中元素的个数，也可以是X>start*2 and Y>start*2
        while No_of_elements > len(res):
            endX = X - start - 1 # 每一行最右边的坐标
            endY = Y - start - 1 # 每一列最底端的坐标
            for i in range(start,endX+1): # 从左到右遍历第一行，包括最右端结点
                res.append(matrix[start][i])
            if endY > start: # 当行的索引比start索引大时(也就是至少两行)
                for j in range(start+1,endY+1): # 去除之前打印过的最顶部结点，从上到下遍历，包括最底部的结点
                    res.append(matrix[j][endX])
            if endX > start and endY > start:  # 当行和列的索引都比start大时(也就是至少两行两列)
                for m in range(endX-1,start-1,-1): # 去除之前打印过的最底部最右侧结点，从右到左遍历，包括最底部最左侧的结点
                    res.append(matrix[endY][m])
```

```
        if endX > start and endY - start > 1: # 当列索引大于start，并且行索引比start大2(也就是至少三行两列)
            for n in range(endY-1,start,-1): # 从下到上遍历，直至(start，start)下的结点
                res.append(matrix[n][start])
        start += 1 # 更新start
    return res
```

面试题28：字符串的排列

输入一个字符串,按字典序打印出该字符串中字符的所有排列。例如输入字符串abc,则打印出由字符a,b,c所能排列出来的所有字符串abc,acb,bac,bca,cab和cba。

思路：调用itertool.permutations包，生成adc可以组成所有序列的迭代器，用map得到该迭代器并join每个元素，然后让他放入set中，再放入list中(该步可省略)，最后排序。

解法一：

```
# -*- coding:utf-8 -*-
import itertools
class Solution:
    def Permutation(self, ss):
        if not ss:
            return []
        return sorted(list(set(map(''.join,itertools.permutations(ss)))))
```

解法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def Permutation(self, ss):
        if not ss:
            return []
        if len(ss) == 1:
            return [ss]
        ans = []
        for i in range(len(ss)):
            ret = self.Permutation(ss[:i]+ss[i+1:]) # 0: []+bc   1: a+c   2: ab + []
            ans += [ss[i]+s for s in ret]
        return sorted(set(ans))
```

面试题29：最小的K个数

输入n个整数，找出其中最小的K个数。例如输入4,5,1,6,2,7,3,8这8个数字，则最小的4个数字是1,2,3,4,。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        if not tinput or len(tinput) < k:
            return []
        sorted_list = self.heap_sort(tinput)
        return sorted_list[:k]

    def max_heapify(self,heap,heapsize,root):
        left = 2*root+1
        right = left+1
```

```
        larger = root
        if left < heapsize and heap[left] > heap[larger]:
            larger = left
        if right < heapsize and heap[right] > heap[larger]:
            larger = right
        if larger != root:
            heap[larger],heap[root] = heap[root],heap[larger]
            self.max_heapify(heap,heapsize,larger)

    def heap_sort(self,heap):
        heapsize = len(heap)
        for i in range(heapsize//2-1,-1,-1):
            self.max_heapify(heap,heapsize,i)

        for j in range(heapsize-1,-1,-1):
            heap[0],heap[j] = heap[j],heap[0]
            self.max_heapify(heap,j,0)

        return heap
```

面试题32：整数中1出现的次数（从1到n整数）

1~13中包含1的数字有1、10、11、12、13因此共出现6次, 求出任意非负整数区间中1出现的次数。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        # 将1-n全部转换为字符串
        # 只需要统计每个字符串中'1'出现的次数并相加即可
        count = 0
        for i in range(1,n+1):
            for i in str(i):
                if i == '1':
                    count += 1
        return count
```

面试题34：丑数

把只包含因子2、3和5的数称作丑数（Ugly Number）。例如6、8都是丑数，但14不是，因为它包含因子7。 习惯上我们把1当做是第一个丑数。求按从小到大的顺序的第N个丑数。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def GetUglyNumber_Solution(self, index):
        if index == 0:
            return 0
        if index == 1:
            return 1
        t2,t3,t5 = 0,0,0
        UNo_list = [1]
        for i in range(1,index):
            UNo_list.append(min(UNo_list[t2]*2,UNo_list[t3]*3,UNo_list[t5]*5))
            #1: min(ans[0]*2),ans[0]*3,ans[0]*5) => min(2,3,5) => append(2)
            #   t2 = 1, t3 = 0, t5 = 0
```

```
            #2: min(ans[1]*2),ans[0]*3,ans[0]*5) => min(4,3,5) => append(3)
            #   t2 = 1, t3 = 1, t5 = 0 ...
            if UNo_list[i] == UNo_list[t2]*2:
                t2 += 1
            if UNo_list[i] == UNo_list[t3]*3:
                t3 += 1
            if UNo_list[i] == UNo_list[t5]*5:
                t5 += 1
        return UNo_list[index-1]
```

面试题35：第一个只出现一次的字符

在一个字符串(1<=字符串长度<=10000，全部由字母组成)中找到第一个只出现一次的字符,并返回它的位置

思路：

解法一：仅限python3

```
# -*- coding:utf-8 -*-
class Solution:
    def FirstNotRepeatingChar(self, s):
        # write code here
        n = -1
        location = -1
        dict_char = {}
        for i in s:
            n += 1
            if i in dict_char.keys():
                dict_char[i]['value'] += 1
            else:
                dict_char[i] = {}
                dict_char[i]['value'] = 1
                dict_char[i]['index'] = n

        for key in dict_char:
            if dict_char[key]['value'] == 1:
                location =  dict_char[key]['index']
        return location
```

解法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def FirstNotRepeatingChar(self, s):
        dict_char = {}
        # 将字符串中所有元素按出现顺序,添加入字典
        for i in s:
            if i in dict_char.keys():
                dict_char[i] += 1
            else:
                dict_char[i] = 1
        # 遍历字典出现只出现一次的字符就返回他的位置
        for j in range(len(s)):
            if dict_char[s[j]] == 1:
                return j
        return -1
```

面试题41：和为S的两个数字

输入一个递增排序的数组和一个数字S，在数组中查找两个数，是的他们的和正好是S，如果有多对数字的和等于S，输出两个数的乘积最小的。

思路：

方法一：

```
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        couple = [tsum - i for i in array]
        result = [i for i in array if i in couple]
        try:
            return result[0],result[-1]
        except:
            return []
```

方法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        l = []
        # for i, v in enumerate([1,2,3,4,5]):
        # print(i,v)
        # 0 1
        # 1 2
        # 2 3
        # 3 4
        # 4 5
        for i,v in enumerate(array):# 如：当前index为0的数字i = 0, v = 1
            for v_i in array[i+1:]:# v_i in array[1:],[1l2,3,4,5]将1后面的数依次与1相加判断和是否为s，是就append
                if v + v_i == tsum:
                    l.append([v,v_i])
        if l:
            return l[0]
        else:
            return []
```

面试题41：和为S的连续正数序列

小明很喜欢数学,有一天他在做数学作业时,要求计算出9~16的和,他马上就写出了正确答案是100。但是他并不满足于此,他在想究竟有多少种连续的正数序列的和为100(至少包括两个数)。没多久,他就得到另一组连续正数和为100的序列:18,19,20,21,22。现在把问题交给你,你能不能也很快的找出所有和为S的连续正数序列? Good Luck!

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def FindContinuousSequence(self, tsum):
        if tsum < 3: # 因为至少包含两个数，所以最小是1+2 = 3
            return []
        first = 1
        last = 2
        res = []
        mid = (1+tsum)//2 # (1+ 100)/2 = 50
```

```
        sum = first + last
        while first < mid: #因为至少包含两个数所以第一个数要小于mid
            if sum == tsum:
                res.append(range(first,last+1))
                sum += last
                last += 1
            elif sum < tsum: #当当前的和比tsum小，将last指针向后移动一位，增大sum
                sum += last
                last += 1
            else:
                sum -= first #当当前的和比tsum大，将first指针向右移动一位，减小sum
                first += 1
        return res
```

面试题42：翻转单词顺序列

牛客最近来了一个新员工Fish，每天早晨总是会拿着一本英文杂志，写些句子在本子上。同事Cat对Fish写的内容颇感兴趣，有一天他向Fish借来翻看，但却读不懂它的意思。例如，“student. a am I”。后来才意识到，这家伙原来把句子单词的顺序翻转了，正确的句子应该是“I am a student.”。Cat对一一的翻转这些单词顺序可不在行，你能帮助他么?

思路：

```
# -*- coding:utf-8 -*-
from collections import deque
class Solution:
    def ReverseSentence(self, s):
        if s.strip() == "": #单独处理s为" "的情况
            return s
        queue = deque()
        string = ''
        for elements in s.split(' '):
            queue.appendleft(elements) # 将s中所有元素依次加入到队列左边
        for i in queue:
            if not string: # 针对string为空时初始化string直接让他等于第一项
                string = i
            else:
                string = string + ' ' + i # 依次将string相加得到结果
        return string
```

面试题42：左旋转字符串

汇编语言中有一种移位指令叫做循环左移（ROL），现在有个简单的任务，就是用字符串模拟这个指令的运算结果。对于一个给定的字符序列S，请你把其循环左移K位后的序列输出。例如，字符序列S=”abcXYZdef”,要求输出循环左移3位后的结果，即“XYZdefabc”。是不是很简单？OK，搞定它!

思路：

解法一：

```
# -*- coding:utf-8 -*-
class Solution:
    def LeftRotateString(self, s, n):
        if not s:
            return ""
        left = ''
        temp = s
```

```
        # 找出需要左移的字符串为left
        for i in range(0,n):
            if not left:
                left = s[i]
            else:
                left += s[i]
            # 用lstrip去除s左边的字符串
            temp = temp.lstrip(s[i]) # 需要不断给temp赋值，因为lstrip不会更改原来的字符串
        result = temp + left # 去除左移字符串的temp字符串 + 左移字符串(在右侧)
        return result
```

解法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def LeftRotateString(self, s, n):
        a = []
        b = []
        if not s:
            return ""
        else:
            for i in range(n):
                a.append(s[i])
            print(a)
            for j in range(n, len(s)):
                b.append(s[j])
            print(b)
            leftlist = b + a
            return ‘’.join(leftlist)
```

解法三：

```
# -*- coding:utf-8 -*-
class Solution:
    def LeftRotateString(self, s, n):
        return s[n:]+s[:n]
```

面试题44：扑克牌顺子

LL今天心情特别好,因为他去买了一副扑克牌,发现里面居然有2个大王,2个小王(一副牌原本是54张^_^)...他随机从中抽出了5张牌,想测测自己的手气,看看能不能抽到顺子,如果抽到的话,他决定去买体育彩票,嘿嘿！！“红心A,黑桃3,小王,大王,方片5”,“Oh My God!”不是顺子.....LL不高兴了,他想了想,决定大\小 王可以看成任何数字,并且A看作1,J为11,Q为12,K为13。上面的5张牌就可以变成“1,2,3,4,5”(大小王分别看作2和4),“So Lucky!”。LL决定去买体育彩票啦。 现在,要求你使用这幅牌模拟上面的过程,然后告诉我们LL的运气如何。为了方便起见,你可以认为大小王是0。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def IsContinuous(self, numbers):
        if not numbers:
            return False
        numbers.sort()
        number_of_jokers = 0
        total_distance = 0
        # 计算王的个数
```

```
        for i in numbers:
            if i == 0:
                number_of_jokers += 1
        # 因为王是0，sort之后一直在最前面，这部分不需要计算，所以循环开始index可简化为number_of_jokers
        for j in range(number_of_jokers,len(numbers)-1):
            # 用后面一张牌减去之前一张牌，若是顺子则形如5-4=1，这种distance不需要纳入计算，因为本身就是顺子，因此减去1
            next_minus_current = numbers[j+1] - numbers[j]- 1
            #判断是否出现对子，有对子直接返回False
            if next_minus_current == -1:
                return False
            else:
            #统计加和各个牌之间的差值
                total_distance += next_minus_current
        # distance小于等于number_of_jokers就为真
        if total_distance <= number_of_jokers:
            return True
        else:
            return False
```

面试题45：圆圈中最后剩下的数字（约瑟夫环）

每年六一儿童节,牛客都会准备一些小礼物去看望孤儿院的小朋友,今年亦是如此。HF作为牛客的资深元老,自然也准备了一些小游戏。其中,有个游戏是这样的:首先,让小朋友们围成一个大圈。然后,他随机指定一个数m,让编号为0的小朋友开始报数。每次喊到m-1的那个小朋友要出列唱首歌,然后可以在礼品箱中任意的挑选礼物,并且不再回到圈中,从他的下一个小朋友开始,继续0...m-1报数....这样下去....直到剩下最后一个小朋友,可以不用表演,并且拿到牛客名贵的“名侦探柯南”典藏版(名额有限哦!!^_^)。请你试着想下,哪个小朋友会得到这份礼品呢？(注：小朋友的编号是从0到n-1)

思路：

解法一：

```
# -*- coding:utf-8 -*-
class Solution:
    def LastRemaining_Solution(self, n, m):
        if n < 1:
            return -1
        child = range(n) #生成编号
        start = 0 #该删除的编号
        while len(child)>1:
            start = (start + m-1) % len(child)
            child.pop(start)
        return child[0]
```

解法二：

```
# -*- coding:utf-8 -*-
class Solution:
    def LastRemaining_Solution(self, n, m):
        if n<1:
            return -1
        child_no = range(n) #若n=41，child_no为从0-40
        start = 0
        while len(child_no)>1: #循环直至小朋友只剩一人
```

```
            pop_no = (start + m - 1) % n # 下一个pop位置的公式 = (start + m-1) % n
            child_no.pop(pop_no)
            n -= 1 #减小\n = n - 1
            start = pop_no #给start重新赋值，此处注意不要+1因为原位置已经被pop了，所以下一个开始位置即为pop_no
        return child_no[0] #列表仅剩最后一个元素返回即可
```

面试题46：求1+2+3+...+n

求1+2+3+...+n，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

思路：在**Python** 中，and 和 or 执行布尔逻辑演算。但是它们并不返回布尔值，而是返回它们实际进行比较的值之一。（类似C++里面的&&和||的短路求值）

```
# -*- coding:utf-8 -*-
class Solution:
    def Sum_Solution(self, n):
        # 10 and 10 + self.Sum_Solution(9)
        # 9 and 9+ self.Sum_Solution(8)
        # ……
        # 1 and 1 + self.Sum_Solution(0)
        # 0 and 0 + self.Sum_Solution(-1) 只执行and前的，然后退出程序
        return n and n + self.Sum_Solution(n-1)
```

面试题47：不用加减乘除做加法

写一个函数，求两个整数之和，要求在函数体内不得使用+、-、*、/四则运算符号。

思路：

```
public class Solution {
    public int Add(int num1,int num2) {
        if(num2==0)
            return num1;
        return Add(num1^num2, (num1&num2)<<1);
    }
}
```

面试题49：把字符串转换成整数

将一个字符串转换成一个整数，要求不能使用字符串转换整数的库函数。 数值为0或者字符串不是一个合法的数值则返回0

输入描述:

输入一个字符串,包括数字字母符号,可以为空

输出描述:

如果是合法的数值表达则返回该数字，否则返回0

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def StrToInt(self, s):
        if s == None:
            return 0
        dict_number = {'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'0':0}
        dict_addOrMinus ={'+':1,'-':-1}
        posOrNeg = 1 #初始默认为positive 1
```

```
        value = 0
        for i in range(0,len(s)): # 遍历s
            if i == 0 and s[i] in ['+','-']: #若第一个为+，-,就调整正负值
                posOrNeg = dict_addOrMinus[s[i]]
            elif s[i] in dict_number: #判断s[i]是不是数字字典的key
                value = value*10 + dict_number[s[i]] #取出key-s[i]对应的value
            '''
            若s='123'，value迭代过程如下：
            value = 0*10+1=1
            value = 1*10+2=12
            value = 12*10+3=123
            '''
            else:
                return 0 #字符串中间出现非0-9数字就返回0
        return posOrNeg * value #最终的返回值=正负号(value)
```

面试题49(变形)：把字符串转换成浮点数

```
i = '-120.1765'
l = []
for j in i.split('.'):
    l.append(j)
left = l[0]
right = l[1]
int_value = 0
float_value = 0.0
pos_or_neg = 1
dict_number = {'1':1,'2':2,'3':3,'4':4,'5':5,'6':6,'7':7,'8':8,'9':9,'0':0}
for i in range(len(left)):
    if left[i] == '-':
        pos_or_neg = -1
        continue
    int_value = int_value*10 + dict_number[left[i]]
for j in range(1,len(right)+1):
    float_value +=  dict_number[right[j-1]]*(0.1**(j))
print((int_value+float_value)*pos_or_neg)
```

面试题53：正则表达式匹配

请实现一个函数用来匹配包括'.'和'*'的正则表达式。模式中的字符'.'表示任意一个字符，而'*'表示它前面的字符可以出现任意次（包含0次）。 在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab*ac*a"匹配，但是与"aa.a"和"ab*a"均不匹配

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # s, pattern都是字符串
    def match(self, s, pattern):
        import re
        k = re.findall(pattern,s)
        return s in k
```

面试题54：表示数值的字符串

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100","5e2","-123","3.1416"和"-1E-16"都表示数值。 但是”12e","1a3.14","1.2.3","+-5"和"12e+4.3"都不是。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # s字符串
    def isNumeric(self, s):
        try:
            p = float(s)
            return True
        except:
            return False
```

面试题55：字符流中第一个不重复的字符

请实现一个函数用来找出字符流中第一个只出现一次的字符。例如，当从字符流中只读出前两个字符"go"时，第一个只出现一次的字符是"g"。当从该字符流中读出前六个字符“google"时，第一个只出现一次的字符是"l"。

输出描述:

如果当前字符流没有存在出现一次的字符，返回#字符。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # 返回对应char
    def __init__(self):
        self.s = '' # s是为了让原来无序的字典变得有序
        self.stream_dict = {}

    def FirstAppearingOnce(self):
        for key in self.s: # 此处按顺序访问s的每个元素，保证了按顺序访问
            if self.stream_dict[key] == 1:
                return key
        return '#'

    def Insert(self, char):
        self.s += char
        if char in self.stream_dict.keys():
            self.stream_dict[char] += 1
        else:
            self.stream_dict[char] = 1
```

面试题64：数据流中的中位数

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

思路：

```
# -*- coding:utf-8 -*-
```

```
class Solution:
    def __init__(self):
        self.data = []

    def Insert(self, num):
        self.data.append(num)
        self.data.sort()

    def GetMedian(self,data):
        length = len(self.data)
        if length % 2 == 1:
            return self.data[length//2]
        else:
            return (self.data[length//2] + self.data[length//2-1 ])/2.0
```

面试题65：滑动窗口的最大值

给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。例如，如果输入数组{2,3,4,2,6,2,5,1}及滑动窗口的大小3，那么一共存在6个滑动窗口，他们的最大值分别为{4,4,6,6,6,5}； 针对数组{2,3,4,2,6,2,5,1}的滑动窗口有以下6个： {[2,3,4],2,6,2,5,1}, {2,[3,4,2],6,2,5,1}， {2,3,[4,2,6],2,5,1}， {2,3,4,[2,6,2],5,1}， {2,3,4,2,[6,2,5],1}, {2,3,4,2,6,[2,5,1]}。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    def maxInWindows(self, num, size):
        if not num or len(num)<size or not size:
            return []
        result = []
        # 例如：{2,3,4,2,6,2,5,1}，size = 3
        # 依次从2，3，4，2，6，2开始向右滑动三格，找其中的最大值，然后贴加到result列表中
        for i in range(0,len(num) - size+1):
            max = num[i] # 初始化max等于滑动窗口的第一项
            for j in range(i,i+size):
                if num[j] > max:
                    max = num[j]
            result.append(max)
        return result
```

面试题4: 替换空格

请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

思路：

```
# -*- coding:utf-8 -*-
class Solution:
    # s 源字符串
    def replaceSpace(self, s):
        return s.replace(' ‘,'%20')
```

https://www.zybuluo.com/knight/note/493856#剑指offfer编程题python版