## 1. 二维数组的查找

题目描述

在一个二维数组中（每个一维数组的长度相同），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。

```
class Solution:
    # array 二维列表
    def Find(self, target, array):
        len1 = len(array)
        len2 = len(array[0])

        if not array:
            return False
        i = 0
        j = len2-1

        while i<len1 and j>=0:
            if array[i][j]==target:
                return True
                #print(False)
            elif array[i][j]<target:
                i = i+1
            else:
                j = j-1
        return False
```

## 2. 替换空格

题目描述

请实现一个函数，将一个字符串中的每个空格替换成“%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。

```
class Solution:
    # s 源字符串
    def replaceSpace(self, s):
        # write code here
        res = ""
        s = list(s)
        for i in range(len(s)):
            if s[i]!=" ":
                res+=s[i]
            else:
                res+="%20"
        return res
```

## 3. 从尾到头打印链表

题目描述

输入一个链表，按链表值从尾到头的顺序返回一个 ArrayList。

```
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):
        # write code here

        if not listNode:
            return []
        stack = []
        while listNode:
            stack.append(listNode.val)
            listNode = listNode.next
        res = []
        for i in range(len(stack)):
            res.append(stack.pop())
        return res
```

## 4. 重建二叉树

题目描述

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。

```
class Solution:
    # 返回构造的 TreeNode 根节点
    def reConstructBinaryTree(self, pre, tin):
        if len(pre)==0:
            return None
        root_data = TreeNode(pre[0])
        i=tin.index(pre[0])
        root_data.left = self.reConstructBinaryTree(pre[1:1+i],tin[:i])
        root_data.right = self.reConstructBinaryTree(pre[1+i:],tin[i+1:])

        return root_data
```

## 5. 用两个栈实现队列

题目描述

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。

```
class Solution:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []
    def push(self, node):
        self.stack1.append(node)
    def pop(self):
        if not self.stack2:
            if not self.stack1:
                return None
```

```
            while self.stack1:
                self.stack2.append(self.stack1.pop())
            return self.stack2.pop()
        else:
            return self.stack2.pop()
```

## 6. 旋转数组的最小数字

题目描述

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。

```
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        if len(rotateArray)==0:
            return 0
        low = 0
        high = len(rotateArray) - 1
        while low < high:
            mid = int((low + high) / 2)
            if rotateArray[mid] > rotateArray[high]:
                low = mid + 1
            else:
                high = mid
        return rotateArray[high]
```

## 7. 斐波那契数列

题目描述

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（从 0 开始，第 0 项为 0）。 n<=39

```
class Solution:
    def Fibonacci(self, n):
        if n<=0:
            return 0
        if n==1:
            return 1
        one = 0
        two = 1
        for i in range(2,n+1):
            fi = one + two
            one = two
            two = fi
        return fi
```

## 8. 跳台阶

题目描述

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（先后次序不同算不同的结果）。

```
class Solution:
    def jumpFloor(self, number):
        a,b=1,2
        if number==1:
            return a
        if number == 2:
            return b
        for i in range(3,number+1):
            #a,b = b,a+b
            res = a + b
            a = b
            b = res

        return res
```

## 9. 变态跳台阶

题目描述

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。

```
class Solution:
    def jumpFloorII(self, number):
        # write code here
        if number<=1:
            return 1
        a = 1
        b = 1
        for i in range(2,number+1):
            res = 2*b
            b = res

        return res
```

## 10. 矩形覆盖

题目描述

我们可以用 2*1 的小矩形横着或者竖着去覆盖更大的矩形。请问用 n 个 2*1 的小矩形无重叠地覆盖一个 2*n 的大矩形，总共有多少种方法？

```
class Solution:
    def rectCover(self, number):
        # write code here
        if number == 0:
            return 0
        if number == 1:
            return 1
        if number == 2:
            return 2
```

```
        a = 1
        b = 2
        for i in range(3,number+1):
            res = a+b
            a = b
            b = res
        return res
```

## 11. 二进制中 1 的个数

题目描述

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。

```
class Solution:
    def NumberOf1(self, n):
        count = 0
        for i in range(32):
            if n&1:
                count+=1
            n>>=1     #右移
        return count
```

## 12. 数值的整数次方

题目描述

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。

```
class Solution:
    def Power(self, base, exponent):
        return pow(base,exponent)
```

## 13. 调整数组顺序使奇数位于偶数前面

题目描述

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**。

```
class Solution:
    def reOrderArray(self, array):
        odd,even=[],[]
        for i in array:
            if i%2==1:
                odd.append(i)
            else:
                even.append(i)
        return odd+even
```

## 14. 链表中倒数第 k 个结点

题目描述

输入一个链表，输出该链表中倒数第 k 个结点。

```
class Solution:
    def FindKthToTail(self, head, k):
```

```
        if not head:
            return None
        if k ==0:
            return None
        if k == 1:
            while head.next:
                head = head.next
            return head
        pre = head
        for i in range(k-1):
            if  pre.next:
                pre = pre.next
            else:
                return None
        while pre.next:
            pre = pre.next
            head = head.next
        return head
```

## 15. 反转链表

题目描述

输入一个链表，反转链表后，输出新链表的表头。

```
class Solution:
    def ReverseList(self, pHead):
        prev = None
        pNode = pHead
        pReverse = None

        while pNode:
            pNext = pNode.next
            if pNext == None:
                pReverse = pNode
            pNode.next = prev
            prev = pNode
            pNode = pNext
        return  pReverse
```

## 16. 合并两个排序的列表

题目描述

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。

```
class Solution:
    # 返回合并后列表
    def Merge(self, pHead1, pHead2):
        if not pHead1 :
            return pHead2
```

```
        elif not pHead2 :
            return pHead1
        pMergedHead = None
        if pHead1.val <= pHead2.val:
            pMergedHead = pHead1
            pMergedHead.next = self.Merge(pHead1.next,pHead2)
        else:
            pMergedHead = pHead2
            pMergedHead.next = self.Merge(pHead1, pHead2.next)
        return pMergedHead
```

## 17. 树的子结构

题目描述

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）

```
class Solution:
    def SearchSimiler(self, p1, p2):
        if p2 is None:
            return True
        if p1 is None:
            return p1 == p2
        flag = False
        if p1.val == p2.val:
            flag = self.SearchSimiler(p1.left, p2.left) and self.SearchSimiler(p1.right, p2.right)
        if flag == False:
            flag = self.SearchSimiler(p1.left, p2)
        if flag == False:
            flag = self.SearchSimiler(p1.right, p2)
        return flag
    def HasSubtree(self, pRoot1, pRoot2):
        # write code here
        if pRoot1 is None or pRoot2 is None:
            return False
        return self.SearchSimiler(pRoot1, pRoot2)
```

## 18. 二叉树的镜像

题目描述

操作给定的二叉树，将其变换为源二叉树的镜像。

```
class Solution:
    # 返回镜像树的根节点
    def Mirror(self, root):
        # write code here
        if root == None:
            return None
        if not root.left and not root.right:
            return root
```

```
        root.left,root.right = self.Mirror(root.right),self.Mirror(root.left)
        return root
```

## 19. 顺时针打印矩阵

题目描述

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 4 X 4 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.

```
class Solution:
    # matrix 类型为二维列表，需要返回列表
    def printMatrix(self, matrix):
        result = []
        while matrix:
            result += matrix.pop(0)
            if matrix:
                matrix = self.rotate(matrix)
        return result

    def rotate(self, matrix):
        row = len(matrix)
        col = len(matrix[0])
        new_matrix = []
        for i in range(col):
            new_line = []
            for j in range(row):
                new_line.append(matrix[j][col-1-i])
            new_matrix.append(new_line)
        return new_matrix
```

## 20. 包含 min 函数的栈

题目描述

定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的 min 函数（时间复杂度应为 O（1））。

```
class Solution:
    def __init__(self):
        self.stack = []
    def push(self, node):
        # write code here
        self.stack.append(node)
    def pop(self):
        # write code here
        if self.stack != []:
            self.stack.pop(-1)
    def top(self):
        # write code here
        if self.stack != []:
```

```
            return self.stack[-1]
        else:
            return None
    def min(self):
        # write code here
        return min(self.stack)
```

## 21. 栈的压入、弹出序列

题目描述

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可能为该栈的弹出顺序。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）

```
class Solution:
    def IsPopOrder(self, pushV, popV):# 压入 弹出
        stack = []
        while popV:
            if stack and stack[-1] == popV[0]:
                stack.pop()
                popV.pop(0)
            elif pushV:
                stack.append(pushV.pop(0))
            else:
                return False
        return True
```

## 22. 从上往下打印二叉树

题目描述

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。

```
class Solution:
    # 返回从上到下每个节点值列表，例：[1,2,3]
    def PrintFromTopToBottom(self, root):
        # write code here
        res = []
        result = []
        if not root:
            return result
        res.append(root)
        while res:
            current_root = res.pop(0)
            result.append(current_root.val)
            if current_root.left:
                res.append(current_root.left)
            if current_root.right:
                res.append(current_root.right)
        return result
```

## 23. 二叉搜索树的后序遍历系列

题目描述

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。

```
class Solution:
    def VerifySquenceOfBST(self, sequence):
        # write code here
        if not sequence:
            return False
        root = sequence[-1]
        i = 0
        while i<len(sequence)-1:
            if sequence[i]>root:
                break
            i +=1
        j = i
        while j<len(sequence)-1:
            if sequence[j]<root:
                return False
            j+=1
        Left = True
        Righe = True

        if i>0:
            Left = self.VerifySquenceOfBST(sequence[:i])
        if i<len(sequence)-1:
            Righe = self.VerifySquenceOfBST(sequence[i:len(sequence)-1])
        return Left and Righe
```

## 24. 二叉树中和为某一值的路径

题目描述

输入一颗二叉树的跟节点和一个整数，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。(注意：在返回值的 list 中，数组长度大的数组靠前)

```
class Solution:
    def FindPath(self, root, expectNumber):
        ans=[]
        if root==None:
            return ans
        def iterpath(root,expectNumber,dir=[]):
            if expectNumber>root.val:
                dir.append(root.val)
                if root.left!=None:
                    iterpath(root.left,expectNumber-root.val,dir)
                if root.right!=None:
```

```
                iterpath(root.right,expectNumber-root.val,dir)
            elif expectNumber==root.val:
                dir.append(root.val)
                if root.right==None and root.left==None:
                    #如果节点的值与期望值相同，则判断节点是否为叶子结点，如果是叶子结点则是符合条件的路径
                    tmp=dir[:]
                    ans.append(tmp)
            else:
                dir.append(0)
            dir.pop()
        iterpath(root,expectNumber)
        return ans
```

## 25. 复杂链表的复制

题目描述

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）

```
class Solution:
    # 返回 RandomListNode
    def Clone(self, head):
        # write code here
        if head == None:
            return head
        node_dict = {}
        node_dict[head] = RandomListNode(head.label)
        tmp = head
        while(head):
            random = head.random
            nexthd = head.next
            if random !=None:
                if random not in node_dict:
                    node_dict[random] = RandomListNode(random.label)
                node_dict[head].random = node_dict[random]
            if nexthd !=None:
                if nexthd not in node_dict:
                    node_dict[nexthd] = RandomListNode(nexthd.label)
                node_dict[head].next = node_dict[nexthd]
            head = head.next
        return node_dict[tmp]
```

## 26. 二叉搜索树与双向链表

题目描述

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。

```
class Solution:
    def Convert(self, pRootOfTree):
        # write code here
        if pRootOfTree == None:
            return pRootOfTree
        if pRootOfTree.left == None and pRootOfTree.right == None:
            return pRootOfTree
        left = self.Convert(pRootOfTree.left)
        p = left
        if left:
            while(p.right):
                p = p.right
            p.right = pRootOfTree
            pRootOfTree.left = p
        right = self.Convert(pRootOfTree.right)
        if right:
            pRootOfTree.right = right
            right.left = pRootOfTree
        return left if left else pRootOfTree
```

## 27. 字符串的排列

题目描述

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。

```
class Solution:
    def Permutation(self, ss):
        # write code here
        import itertools
        if ss == "":
            return []
        result = []
        res= list(itertools.permutations(list(ss),len(ss)))
        for s in res:
            s = ''.join(s)
            if s not in result:
                result.append(s)
        return result
```

## 28. 数组中出现次数超过一半的数字

题目描述

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。

```
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        # write code here
```

```
        len1 = len(numbers)
        if len1==0:
            return 0
        elif len1>=1:
        # 遍历每个元素，并记录次数；若与前一个元素相同，则次数加 1，否则次数减 1
            res = numbers[0] # 初始值
            count = 1 # 次数
            for i in range(1,len1):
                if count == 0:
                    # 更新 result 的值为当前元素，并置次数为 1
                    res = numbers[i]
                    count = 1
                elif numbers[i] == res:
                     count += 1 # 相同则加 1
                elif numbers[i] != res:
                     count -= 1 # # 不同则加 1
                             # 判断 res 是否符合条件，即出现次数大于数组长度的一半
            counts = 0
            for j in range(len1):
                if numbers[j] == res:
                    counts += 1
            if counts>len1//2: # python3 整除为//，python2 为/
                return res
            else:
                return 0
```

## 29. 最小的 k 个数

题目描述

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,。

```
class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        # write code here
        if not tinput:
            return None
        if k<=0 or k>len(tinput):
            return None
        key = tinput[k]
        def quick_sort(res,left,right):
            key = res[left]
            while left<right:
                while left<right    and res[right]>=key:
                    right-=1
                res[left] = res[right]
                while left<right    and res[left]<=key:
```

```
                left+=1
            res[right]=res[left]
        res[left] = key
        return left
    start= 0
    end = len(tinput)-1
    pos = quick_sort(tinput,start,end)
    while pos!=k:
        if pos>k-1:
            end = pos-1
        else:
            start    = pos +1
        pos = quick_sort(tinput,start,end)
    return tinput[0:k]

class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        # write code here
        if not tinput:
            return []
        if k<=0 or k>len(tinput):
            return []
        return sorted(tinput)[0:k]
```

## 30. 连续子数组最大和

题目描述

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)。给一个数组，返回它的最大连续子序列的和，你会不会被他忽悠住？(子向量的长度至少是 1)

```
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        # write code here
        n = len(array)
        dp = [i for i in array]
        for i in range(1,n):
            dp[i] = max(dp[i-1]+array[i],array[i])
        return max(dp)
```

## 31. 整数中 1 出现的次数（从 1 到 n 整数中 1 出现的次数）

题目描述

求出 1~13 的整数中 1 出现的次数,并算出 100~1300 的整数中 1 出现的次数？为此他特别数了一下 1~13 中包含 1 的数字有 1、10、11、12、13 因此共出现 6 次,但是对于后面问题他就没辙了。ACMer 希望你们帮帮他,并把问题更加普遍化,可以很快的求出任意非负整数区间中 1 出现的次数（从 1 到 n 中 1 出现的次数）。

```
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        count = 0
        while(n>0):
            string = str(n)
            array = list(string)
            for i in array:
                if i=='1':
                    count+=1
            n=n-1
        return count
```

## 32. 把数组排成最小的数

题目描述

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。

```
class Solution:
    def theMax(self, str1, str2):
        return str1 if str1+str2 > str2+str1 else str2
    def PrintMinNumber(self, numbers):
        string = [str(num) for num in numbers]
        flag = True
        # 冒泡排序的次数
        count = len(string) - 1
        while flag    and    count > 0:
            flag = False
            for i in range(len(string)-1):
                if self.theMax(string[i], string[i+1]) == string[i]:
                    string[i],string[i+1] = string[i+1] ,string[i]
                    flag = True
            count -= 1
        string = ''.join(string)
        return string
```

## 33. 丑数

题目描述

把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。例如 6、8 都是丑数，但 14 不是，因为它包含质因子 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。

```
class Solution:
    def GetUglyNumber_Solution(self, index):
        # 1 2 3 5
        if index <= 0:
            return 0
        if index == 1:
            return 1
```

```
        res = [1]
        t2 = t3 = t5 = 0
        nextIdx = 1
        while nextIdx < index:
            minNum = min(res[t2] * 2, res[t3] * 3, res[t5] * 5)
            res.append(minNum)
            if res[t2] * 2 <= minNum:
                t2 += 1
            if res[t3] * 3 <= minNum:
                t3 += 1
            if res[t5] * 5 <= minNum:
                t5 += 1
            nextIdx += 1
        return res[nextIdx - 1]
```

## 34. 第一个只出现一次的字符

题目描述

在一个字符串(0<=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, 如果没有则返回 -1（需要区分大小写）.

```
class Solution:
    def FirstNotRepeatingChar(self, s):
        # write code here
        dict = {}
        for i in s:
            if i in dict:
                dict[i] = dict[i]+1
            else:
                dict[i] = 1
        for i in s:
            if dict[i] == 1:
                return s.index(i)
        return -1
```

## 35. 数组中的逆序对

题目描述

在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组,求出这个数组中的逆序对的总数P。并将P对1000000007取模的结果输出。即输出P%1000000007

超时：

```
class Solution:
    def InversePairs(self, data):
        sortData = sorted(data)
        count = 0
        for i in sortData:
            pos = data.index(i)
            count += pos
```

```
            data.pop(pos)
        return count
```

## 36. 两个链表的第一个公共结点

题目描述

输入两个链表，找出它们的第一个公共结点。

解法：

第一种就是把全部结点分别压入两个栈，利用栈的特性 LIFO，然后同时 pop 出栈，一开始两边的元素肯定是相同的，当遇到不同的元素时，肯定已经遇到了最后一个节点，那就 break

第二种就是分别从链表的头结点开始遍历，当两条链表有长度差时，先让长链表走他们的差值，当走到还剩下和短链表一样长时，两个链表同时遍历，这样就能找到第一个公共结点了

```
class Solution:
    def FindFirstCommonNode(self, pHead1, pHead2):
        stack1 = []
        stack2 = []
        while pHead1:
            stack1.append(pHead1)
            pHead1 = pHead1.next
        while pHead2:
            stack2.append(pHead2)
            pHead2 = pHead2.next
        node = None
        while stack1 and stack2 and stack1[-1] is stack2[-1]:
            node = stack1.pop()
            stack2.pop()
        return node
```

## 37. 数字在排序数组中出现的次数

题目描述

统计一个数字在排序数组中出现的次数。

```
class Solution:
    def GetNumberOfK(self, data, k):
        return data.count(k)
class Solution:
    def GetNumberOfK(self, data, k):
        if len(data) == 0:
            return 0
        else:
            count, mid = 0, len(data) // 2
            if k > data[mid]:
                for i in range(mid + 1, len(data)):
                    if data[i] == k:
                        count += 1
                return count
            elif k < data[mid]:
                for i in range(mid - 1, -1, -1):
```

```
                if data[i] == k:
                    count += 1
            return count
        else:
            count += 1
            for i in range(mid - 1, -1, -1):
                if data[i] == k:
                    count += 1
                else:
                    break
            for i in range(mid + 1, len(data)):
                if data[i] == k:
                    count += 1
                else:
                    break
            return count
```

## 38. 二叉树的深度

题目描述

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。

```
class Solution:
    def TreeDepth(self, pRoot):
        if not pRoot:
            return 0
        if not pRoot.left and not pRoot.right:
            return 1
        return 1+max(self.TreeDepth( pRoot.left),self.TreeDepth( pRoot.right))
```

## 39. 平衡二叉树

题目描述

输入一棵二叉树，判断该二叉树是否是平衡二叉树。

```
class Solution:
    def TreeDepth(self, pRoot):
        if not pRoot:
            return 0
        if not pRoot.left and not pRoot.right:
            return 1
        return 1+max(self.TreeDepth( pRoot.left),self.TreeDepth( pRoot.right))
    def IsBalanced_Solution(self, pRoot):
        if not pRoot:
            return True
        if not pRoot.left and not pRoot.right:
            return True
        if abs(self.TreeDepth(pRoot.left) - self.TreeDepth( pRoot.right))<=1:
            return True
```

```
        else:
            return False
```

## 40. 数组中只出现一次的数字

题目描述

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。

思考：用 hash；或者位运算

首先利用 0 ^ a = a; a^a = 0 的性质

两个不相等的元素在位级表示上必定会有一位存在不同，

将数组的所有元素异或得到的结果为不存在重复的两个元素**异或**的结果，

据异或的结果 1 所在的最低位，把数字分成两半，每一半里都还有一个出现一次的数据和其他成对出现的数据,

问题就转化为了两个独立的子问题：数组中只有一个数出现一次，其他数都出现了 2 次，找出这个数字。

```
class Solution:
    def FindNumsAppearOnce(self, array):
        ans,a1,a2,flag= 0,0,0,1
        for num in array:
            ans = ans ^ num
        while(ans):
            if ans%2 == 0:
                ans = ans >>1
                flag = flag <<1
            else:
                break
        for num in array:
            if num & flag:
                a1 = a1 ^ num
            else:
                a2 = a2 ^ num
        return a1,a2
```

## 41. 和为 S 的连续正数序列

题目描述

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?

输出描述:

输出所有和为 S 的连续正数序列。序列内按照从小至大的顺序，序列间按照开始数字从小到大的顺序

思路：类似于操作系统里的滑动窗口,如果和等于 tsum 则将现在窗口内序列加入 res,如果小于 tsum,增加 quick,如果大于 tsum,增加 low.

```
class Solution:
    def FindContinuousSequence(self, tsum):
```

```
        low = 1
        quick = 2
        res = []
        while quick > low:
            esum = sum(range(low, quick+1))
            if esum == tsum:
                res.append(list(range(low, quick+1)))
                low += 1
            elif esum < tsum:
                quick += 1
            else:
                low += 1
        return res
```

## 42. 和为 S 的两个数字

题目描述

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，使得他们的和正好是 S，如果有多对数字的和等于 S，输出两个数的乘积最小的。

```
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        memorys= {}
        ret = []
        for num in array:
            if tsum-num in memorys:
                if ret == []:
                    ret = [tsum-num,num]
                elif ret and ret[0]*ret[1]>(tsum-num)*num:
                    ret = [tsum-num,num]
            else:
                memorys[num] = 1
        return ret
```

## 43. 左旋转字符串

题目描述

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！

```
class Solution:
    def LeftRotateString(self, s, n):
        # write code here
        length = len(s)
        if length == 0:
            return ""
        n = n%length
        return s[n:length]+s[0:n]
```

## 44. 翻转单词顺序列

题目描述

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？

```
class Solution:
    def ReverseSentence(self, s):
        ret = s.split(" ")
        ret.reverse()
        return " ".join(ret)
```

## 45. 扑克牌顺子

题目描述

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， 如果牌能组成顺子就输出 true，否则就输出 false。为了方便起见,你可以认为大小王是 0。

```
class Solution:
    def IsContinuous(self, numbers):
        if not numbers:
            return False
        numbers.sort()
        zeros = numbers.count(0)
        for i, v in enumerate(numbers[:-1]):
            if v!=0:
                if numbers[i+1]==v:
                    return False
                zeros -= (numbers[i+1]-numbers[i]-1)
                if zeros<0:
                    return False
        return True
```

## 46. 孩子们的游戏（圆圈中最后剩下的数）

题目描述

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)

```
class Solution:
    def LastRemaining_Solution(self, n, m):
        if n<1:
            return -1
        final,start = -1,0
        cnt = [i for i in range(n)]
        while cnt:
            k = (start+m-1)%n
            final = cnt.pop(k)
            n-=1
            start = k
        return final
```

## 47. 求 1+2+3+…+n

题目描述

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：递归

```
class Solution:
    def Sum_Solution(self, n):
        if n == 1:
            return 1
        return n+self.Sum_Solution(n-1)
```

## 48. 不用加减乘除做加法

题目描述

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：不计进位的和为 a^b，进位就是 a&b

a+b = a^b + (a&b)<<1;

Java 解法：

```
public class Solution {
    public int Add(int a,int b) {
        int unit = a ^ b;
        int carry_bit = a & b;
        while(carry_bit != 0) {
            int temp_a = unit;
            int temp_b = carry_bit << 1;
            unit = temp_a ^ temp_b;
            carry_bit = temp_a & temp_b;
        }
        return unit;
    }
}
```

## 49. 把字符串转换为整数

题目描述

将一个字符串转换成一个整数(实现 Integer.valueOf(string)的功能，但是 string 不符合数字要

求时返回 0)，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。

```
class Solution:
    def StrToInt(self, s):
        flag = True
        pos = 1
        ret = None
        if s==":
            return 0
        for i in s:
            if i=='+' or i=='-':
                if flag:
                    pos = -1 if i=='-' else 1
                    flag = False
                else:
                    return 0
            elif i>='0' and i<='9':
                flag = False
                if ret == None:
                    ret = int(i)
                else:
                    ret = ret*10+int(i)
            else:
                return 0
        return pos*ret if ret else 0
```

## 50. 构建乘积数组

题目描述

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。

```
class Solution:
    def multiply(self, A):
        size = len(A)
        B = [1]*size
        for i in range(1,size):
            B[i] = B[i-1]*A[i-1]
        tmp = 1
        for i in range(size-2,-1,-1):
            tmp = tmp*A[i+1]
            B[i] = B[i]*tmp
        return B
```

## 51. 数组中重复的数字

题目描述

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。

```
class Solution:
    def duplicate(self, numbers, duplication):
        dup = {} #dict()
        for num in numbers:
            if num not in dup:
                dup[num] = True
            else:
                duplication[0]=num
                return True
        return False
```

## 52. 正则表达式匹配

题目描述

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

```
class Solution:
    def __init__(self):
        self.dic = {}
    def match(self, s, p):
        if (s,p) in self.dic:
            return self.dic[(s,p)]
        if p == '':
            return s==''
        if len(p)==1 or p[1]!='*':
            self.dic[(s[1:],p[1:])] = self.match(s[1:],p[1:])
            return len(s)>0 and (p[0]=='.' or p[0]==s[0]) and self.dic[(s[1:],p[1:])]
        while(len(s) and (p[0]=='.' or p[0]==s[0])):
            self.dic[(s,p[2:])] = self.match(s,p[2:])
            if self.match(s[:],p[2:]):
                return True
            s = s[1:]
        self.dic[(s,p[2:])] = self.match(s,p[2:])
        return self.dic[(s,p[2:])]
```

## 53. 表示数值的字符串

题目描述

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。

```
class Solution:
    def isNumeric(self, s):
        INVALID=0; SPACE=1; SIGN=2; DIGIT=3; DOT=4; EXPONENT=5;
```

```
        transitionTable=[[-1,  0,  3,  1,  2, -1],      #0 no input or just spaces
                         [-1,  8, -1,  1,  4,  5],      #1 input is digits
                         [-1, -1, -1,  4, -1, -1],      #2 no digits in front just Dot
                         [-1, -1, -1,  1,  2, -1],      #3 sign
                         [-1,  8, -1,  4, -1,  5],      #4 digits and dot in front
                         [-1, -1,  6,  7, -1, -1],      #5 input 'e' or 'E'
                         [-1, -1, -1,  7, -1, -1],      #6 after 'e' input sign
                         [-1,  8, -1,  7, -1, -1],      #7 after 'e' input digits
                         [-1,  8, -1, -1, -1, -1]]      #8 after valid input input space
        state=0; i=0
        while i<len(s):
            inputtype = INVALID
            if s[i]==' ': inputtype=SPACE
            elif s[i]=='-' or s[i]=='+': inputtype=SIGN
            elif s[i] in '0123456789': inputtype=DIGIT
            elif s[i]=='.': inputtype=DOT
            elif s[i]=='e' or s[i]=='E': inputtype=EXPONENT
            state=transitionTable[state][inputtype]
            if state==-1: return False
            else: i+=1
        return state == 1 or state == 4 or state == 7 or state == 8
```

## 54. 字符流中第一个不重复的字符

题目描述

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"。

```
class Solution:
    def __init__(self):
        self.memory = dict()
        self.queue = []
    def FirstAppearingOnce(self):
        while len(self.queue) and self.memory[self.queue[0]]>1:
            self.queue.pop(0)
        return self.queue[0] if len(self.queue) else '#'
    def Insert(self, char):
        if char not in self.memory:
            self.memory[char]=0
        self.memory[char]+=1
        if self.memory[char]==1:
            self.queue.append(char)
```

## 55. 链表中环的入口结点

题目描述

给一个链表，若其中包含环，请找出该链表的环的入口结点，否则，输出 null。

思路：判断是否存在环用 fast 和 slow 两个指针，从 head 开始，一个走一步，一个走两步，如果最终到达同一个结点，则说明存在环。

```
class Solution:
    def EntryNodeOfLoop(self, pHead):
        if not pHead or not pHead.next:
            return None
        fast = slow = pHead
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                fast = pHead
                while fast != slow:
                    fast = fast.next
                    slow = slow.next
                return fast
        return None
```

## 56. 删除链表中重复的结点

题目描述

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

```
class Solution:
    def deleteDuplication(self, pHead):
        first = ListNode(-1)
        first.next = pHead
        curr = pHead
        last = first
        while curr and curr.next:
            if curr.val != curr.next.val:
                curr = curr.next
                last = last.next
            else:
                val = curr.val
                while curr and curr.val == val:
                    curr = curr.next
                last.next = curr
        return first.next
```

## 57. 二叉树的下一个结点

题目描述

给定一个二叉树和其中的一个结点，请找出中序遍历顺序的下一个结点并且返回。注意，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

思路：中序遍历的顺序为 LVR

则有以下三种情况：

a. 如果该结点存在右子结点，那么该结点的下一个结点是右子结点树上最左子结点

b. 如果该结点不存在右子结点，且它是它父结点的左子结点，那么该结点的下一个结点是它的父结点
c. 如果该结点既不存在右子结点，且也不是它父结点的左子结点，则需要一路向祖先结点搜索，直到找到一个结点，该结点是其父亲结点的左子结点。如果这样的结点存在，那么该结点的父亲结点就是我们要找的下一个结点。

```
class Solution:
    def GetNext(self, pNode):
        if pNode == None:
            return None
        if pNode.right:
            tmp = pNode.right
            while(tmp.left):
                tmp = tmp.left
            return tmp
        p = pNode.next
        while(p and p.right==pNode):
            pNode = p
            p = p.next
        return p
```

## 58. 对称的二叉树

题目描述

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。

```
class Solution:
    def Symmetrical(self,Lnode,Rnode):
        if Lnode == None and Rnode == None:
            return True
        if Lnode and Rnode:
            return Lnode.val == Rnode.val and self.Symmetrical(Lnode.right,Rnode.left) and self.Symmetrical(Lnode.left,Rnode.right)
        else:
            return False
    def isSymmetrical(self, pRoot):
        if pRoot == None:
            return True
        return self.Symmetrical(pRoot.left,pRoot.right)
```

## 59. 按之字形顺序打印二叉树

题目描述

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。

```
class Solution:
    def Print(self, pRoot):
        if pRoot == None:
            return []
```

```
        stack = [pRoot]
        step = 1
        ret = []
        while(stack):
            tmpstack = []
            tmp = []
            for node in stack:
                tmp+=[node.val]
                if node.left:
                    tmpstack.append(node.left)
                if node.right:
                    tmpstack.append(node.right)
            if step%2==0:
                tmp.reverse()
            ret.append(tmp)
            step += 1
            stack = tmpstack[:]
        return ret
```

## 60. 把二叉树打印成多行

题目描述

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。

```
class Solution:
    def Print(self, pRoot):
        if pRoot == None:
            return []
        stack = [pRoot]
        ret = []
        while(stack):
            tmpstack = []
            tmp = []
            for node in stack:
                tmp.append(node.val)
                if node.left:
                    tmpstack.append(node.left)
                if node.right:
                    tmpstack.append(node.right)
            ret.append(tmp[:])
            stack = tmpstack[:]
        return ret
```

## 61. 序列化二叉树

题目描述

请实现两个函数，分别用来序列化和反序列化二叉树

```
class Solution:
    def Serialize(self, root):
```

```
        def doit(node):
            if node:
                vals.append(str(node.val))
                doit(node.left)
                doit(node.right)
            else:
                vals.append('#')
        vals = []
        doit(root)
        return ' '.join(vals)
    def Deserialize(self, s):
        def doit():
            val = next(vals)
            if val == '#':
                return None
            node = TreeNode(int(val))
            node.left = doit()
            node.right = doit()
            return node
        vals = iter(s.split())
        return doit()
```

## 62. 二叉搜索树的第 k 个结点

题目描述

给定一棵二叉搜索树，请找出其中的第 k 小的结点。例如，（5，3，7，2，4，6，8）中，按结点数值大小顺序第三小结点的值为 4。

```
class Solution:
    def KthNode(self, pRoot, k):
        stack = []
        node = pRoot
        while node:
            stack.append(node)
            node = node.left
        cnt = 1
        while(stack and cnt<=k):
            node = stack.pop()
            right = node.right
            while right:
                stack.append(right)
                right = right.left
            cnt+=1
        if node and k==cnt-1:
            return node
        return None
```

## 63. 数据流中的中位数

题目描述
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。我们使用 Insert()方法读取数据流，使用 GetMedian()方法获取当前读取数据的中位数。

```
class Solution:
    def __init__(self):
        self.nums = []
    def Insert(self, num):
        self.nums.append(num)
    def GetMedian(self, fuck):
        self.nums.sort()
        if len(self.nums) % 2 == 1:
            return self.nums[(len(self.nums) - 1) / 2]
        else:
            return (self.nums[len(self.nums) / 2] + self.nums[len(self.nums) / 2 - 1]) / 2.0
```

## 64. 滑动窗口的最大值

题目描述
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：假设当前窗口起始位置为 start,结束位置为 end，我们要构造一个 stack，使得 stack[0]为区间[start,end]的最大值

```
class Solution:
    def maxInWindows(self, num, size):
        if size == 0:
            return []
        ret = []
        stack = []
        for pos in range(len(num)):
            while (stack and stack[-1][0] < num[pos]):
                stack.pop()
            stack.append((num[pos], pos))
            if pos>=size-1:
                while(stack and stack[0][1]<=pos-size):
                    stack.pop(0)
                ret.append(stack[0][0])
        return ret
```

## 65. 矩阵中的路径

题目描述
请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动

一个格子。如果一条路径经过了矩阵中的某一个格子，则之后不能再次进入这个格子。 例如 a b c e s f c s a d e e 这样的 3 X 4 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符 b 占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

```
class Solution:
    def hasPath(self, matrix, rows, cols, path):
        def dfs(memories,r,c,s):
            if s=='':
                return True
            dx = [-1,1,0,0]
            dy = [0,0,-1,1]
            for k in range(4):
                x = dx[k] + r
                y = dy[k] + c
                if x >= 0 and x < rows and y >= 0 and y < cols and memories[x][y] and
matrix[x*cols+y]==s[0]:
                    memories[x][y]=False
                    if dfs(memories[:], x, y,s[1:]):
                        return True
                    memories[x][y]=True
            return False

        if path == '':
            return True
        memories = [[True for c in range(cols)] for r in range(rows)]
        for r in range(rows):
            for c in range(cols):
                if matrix[r*cols+c] == path[0]:
                    memories[r][c] = False
                    if dfs(memories[:],r,c,path[1:]):
                        return True
                    memories[r][c] = True
        return False
```

## 66. 机器人的运动范围

题目描述

地上有一个 m 行和 n 列的方格。一个机器人从坐标 0,0 的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于 k 的格子。 例如，当 k 为 18 时，机器人能够进入方格（35,37），因为 3+5+3+7 = 18。但是，它不能进入方格（35,38），因为 3+5+3+8 = 19。请问该机器人能够达到多少个格子？

思路：dfs+记忆化搜索

```
class Solution:
    def movingCount(self, threshold, rows, cols):
        memories = set()
        def dfs(i,j):
```

```
        def judge(i,j):
            return sum(map(int, list(str(i)))) + sum(map(int, list(str(j)))) <= threshold
        if not judge(i,j) or (i,j) in memories:
            return
        memories.add((i,j))
        if i != rows - 1:
            dfs(i + 1, j)
        if j != cols - 1:
            dfs(i, j + 1)
    dfs(0,0)
    return len(memories)
```