剑指 offer 编程 66 题 Python 代码

1 题目描述：在一个二维数组中（每个一维数组的长度相同），每一行都按照从到右递增的顺序排列，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。

解析：由于每行从左到右递增，每列从上到下递增，因此每列的最后一个元素最大；函数可以从每行的最后一个元素 A 开始判断，如果目标 B 大于 A，则从下一行最后一个元素开始判断，如果 B 小于 A，则往前移一列，从新列开始判断

代码如下：

```
class Solution:
    # array  二维列表
    def Find(self, target, array):
        # write code here
        row = len(array)
        col = len(array[0])
        j=0
        i= col-1
        while j < row and i > -1:
            if array[j][i] > target:
                i-=1
            elif array[j][i] < targrt:
                j+=1
            else:
                return 'ture'
            return 'false'
```

2 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。

解析：Python 中提供字符串替换函数 replace()函数，replace()方法的语法：str.replace(old, new[, max])，old 将被替换的子字符串， new 新字符串，用于替换 old 子字符串，max 可选字符串，替换不超过 max 次

代码如下：

```
    class Solution:
        # s  源字符串
        def replaceSpace(self, s):
            return s.replace(' ', '%20')
            # write code here
```

3 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。

解析：将链表的每个值取出来然后存放到一个列表 ArrayList 中，对于 ArrayList 列表可以将其反置(Python 中提供 reverse()方法，用法是 List.reverse()，但该方法不返回值)；此外 Python 中还提供列表的插入函数 insert()函数，用法是 list.insert(index, obj)， index—对象 obj 需要插入的索引位置，同样该方法没有返回值，但会在列表指定位置插入对象。

代码如下：(采用 insert()方法)

```
# class ListNode:
#         def __init__(self, x):
```

```
#             self.val = x
#             self.next = None

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):
        midArrayList = []
        while listNode:
            midArrayList.insert(0, listNode.val)
            listNode = listNode.next
        return midArrayList
        # write code here
```

4 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包含重复的数字。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。

解析：对于前序序列，第一个元素(如果存在)为根节点(1)，由于元素各不相同，则根据中序序列，可以判断出根节点的左子树的中序序列(4, 7,2)和右子树的中序序列(5, 3, 8, 6)，同时可以判断出根节点的左子树的前序序列(2, 4, 7) 和右子树的前序序列(3, 5, 6, 8)。因此可以根据递归重建该二叉树

代码如下：

```
# -*- 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):
        if len(pre) == 0:
            return None
        elif len(pre) == 1:
            return TreeNode(pre[0])
        else:
            treeNode = TreeNode(pre[0])
            treeNode.left = self. reConstructBinaryTree(pre[1:tin.index(pre[0])+1],
                                tin[:tin.index(pre[0])])
            treeNode.right = self. reConstructBinaryTree(pre[tin.index(pre[0])+1:],
                                tin[tin.index(pre[0])+1:])
            return treeNode
        #write code here
```

5 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。

解析：栈 A 用来做入队列， 栈 B 用来做出队列，当栈 B 为空时，栈 A 全部出栈到栈 B，栈

B 再出栈(即出队列)

代码如下：

```
class Solution:
    def __init__(self):
        self.stackA = []
        self.stackB = []
    def push(self, node):
        self.stackA.append(node)
    def pop(self):
        if self.stackB:
            return self.stackB.pop()
        else:
            while self.stackA:
                self.stackB.append(self.stackA.pop())
            return self.stackB.pop()
```

6.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.

解析：由于数组是两段不减的，因此，可以找出数组中第一个最小的值，将该值前面的所有元素移到数组的末尾即可。

代码如下：

```
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        if len(rotateArray) == 0:
            return 0
        else:
            min_value = min(rotateArray)
            min_index = rotateArray.index(min_value)
            newArray = rotateArray[min_index:] + rotateArray[:min_index]
            return newArray[0]
        # write code here
```

7.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

解析：斐波那契数列有一个很重要的性质，后面一项是前两项的和(n>=3),前两项都是 1，很容易想到用递归求解第 n 项的值，但是递归需要消耗更多的内存，因此可以使用动态规划，用两个变量分别保持上次的计算结果和上上次的计算结果

代码如下：

```
class Solution:
    def Fibonacci(self, n):
        a = 0
        b = 1
        if n <= 0:
```

```
            return 0
        else:
            for i in range(n):
                a, b = b, a + b
            return a
        # write code here
```

8. 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)

解析：由于青蛙一次只能跳一个台阶或者跳两个台阶，因此，我们可以考虑青蛙最后通过跳一步或者跳两步到最后的台阶，假设青蛙一步跳到最后的台阶，那么 f(n)和 f(n-1)的跳法是一样多的；假如青蛙两步跳到最后的台阶，那么 f(n)和 f(n-2)的跳法是一样的，故可以知道：f(n)=f(n-1)+f(n-2)的递推公式。

代码如下：

```
class Solution:
    def jumpFloor(self, number):
        a = 0
        b = 1
        if number == 0:
            return 0
        else:
            for i in range(number):
                a, b = b, a + b
            return b
        # write code here
```

9 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。

解析：考虑这样的一种情况，青蛙最后一跳跳到最后台阶，这样的每一跳都是一种独立的方法，因此，青蛙可以从 n-1 台阶跳一次，也可以从 n-2 台阶跳两次，依次类推，则每次跳的方案有 f(n-1), f(n-2), ... f(1)种，此时 f(n) = f(n-1)+ f(n-2) + f(n-3) + ... + f(1)， 可以推出 f(n) = 2f(n-1).

代码如下：

```
class Solution:
    def jumpFloorII(self, number):
        count = 1
        for i in range(number - 1):
            count = count * 2
        return count
        # write code here
```

10 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，总共用多少种方法？

分析：对于矩形的覆盖，可以考虑最后的覆盖是横着还是竖着的小矩形，如果是竖着(占一列)，则有 f(n-1)种覆盖方法，如果是横着(占两行)，则有 f(n-2) 种覆盖方法，因此一共有 f(n) = f(n-1) + f(n-2)种覆盖方法。

代码如下：

```
class Solution:
    def rectCover(self, number):
        if number == 0:
            return 0
        a = 1
        b = 1
        for i in range(number):
            a, b = b, a + b
        return a
        # write code here
```

11 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。

解析：在 Python 中 int 型整数是用 32 位二进制代码存储的，负数按照补码的形式存储的，因此可以将该整数的 32 位二进制代码，一位一位的与 1 做‘&’运算，运算结果的和便是 1 的个数

代码如下：

```
class Solution:
    def NumberOf1(self, n):
        count = 0
        for i in range(32):
            count = count + ((n >> i) & 1) #  ‘+’的运算级比‘&’高，切记加上‘()’
        return count
```

12 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。

解析：Python 中有计算次方的运算符 ’**’，因此直接使用即可。如果不使用’**’运算符，那么需要判断 exponent 正负以及是否为 0， 对于 exponent 的正负，只需要是正的情况，记其结果为 result，负的情况直接 1.0/result 即可计算；此外还需要判断 base 是否为 0

代码如下：

```
class Solution:
    def Power(self, base, exponent):
        result = 1
        if base == 0:
            return 0
        if exponent == 0:
            return 1
        else:
            for i in range(abs(exponent)):
                result = base * result
            if exponent > 0:
                return result
            else:
                return 1.0/result
        # write code here
```

13 题目描述：输入一个整数数组，实现一个函数来调用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。

解析：可以遍历整个数组，将奇数和偶数分别放在不同的列表中，最后，再合并两个列表。

代码如下：

```
class Solution:
    def reOrderArray(self, array):
        odd_numbers = []
        even_numbers = []
        for i in array:
            if i % 2 == 1:
                odd_numbers.append(i)
            else:
                even_numbers.append(i)
        return odd_numbers + even_numbers
        # write code here
```

14 题目描述：输入一个链表， 输出该链表中倒数第 k 个结点。

解析：由于是链表，则可以将链表中的每一个元素保存到列表中，在列表中寻找倒数第 k 个元素。

代码如下:

```
class Solution:
    def FindKthToTail(self, head, k):
        t = []
        while head:
            t.append(head)
            head = head.next
        if k > len(t) or k < 1:
            return None
        return t[-k]
```

15 题目描述：输入一个链表，反转链表后，输出新链表的表头。

解析：可以将链表的每一个结点取出来，插入到新的链表表头，同时要保存原链表的下一个结点。

代码如下：

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    # 返回 ListNode
    last = None
    def ReverseList(self, pHead):
        if pHead == None:
            return pHead
        last = None
        while pHead:
            temp = pHead.next
```

```
                pHead.next = last
                last = pHead
                pHead = temp
        return last
```

16 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。

解析：从头开始比较两个链表元素的大小，将元素小的结点插入到新的链表中，直到一个链表为空。这可以有两种方法，一种是递归实现，一种是非递归

代码如下：

递归：

```
# 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):
        if pHead1 == None:
            return pHead2
        if pHead2 == None:
            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
        # write code here
```

非递归：

```
# 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):
        # write code here
        mergeHead = ListNode(90)
        p = mergeHead
        while pHead1 and pHead2:
            if pHead1.val >= pHead2.val:
```

```
                mergeHead.next = pHead2
                pHead2 = pHead2.next
            else:
                mergeHead.next = pHead1
                pHead1 = pHead1.next
            mergeHead = mergeHead.next
        if pHead1:
            mergeHead.next = pHead1
        elif pHead2:
            mergeHead.next = pHead2
        return p.next
```

17 题目描述：输入两颗二叉树 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)

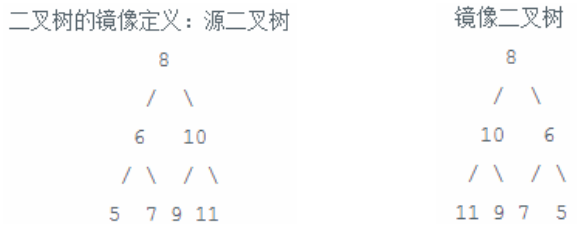
解析：从 A 的判断根节点(约定和 B 的根节点开始判断的 A 结点称为判断根节点)和 B 开始匹配，如果根节点一样，则分别判断左子树和右子树，重复上述过程，如果 B 为空则匹配，如果 A 为空，或者两者的值不相等，则匹配不成功。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def HasSubtree(self, pRoot1, pRoot2):
        if not pRoot1 or not pRoot2:
            return False
        return self.is_subtree(pRoot1, pRoot2) or self.HasSubtree(pRoot1.left, pRoot2) or
self.HasSubtree(pRoot1.right, pRoot2)

    def is_subtree(self, A, B):
        if not B:
            return True
        elif not A or A.val != B.val:
            return False
        else:
            return self.is_subtree(A.left,B.left) and self.is_subtree(A.right, B.right)
        # write code here
```

18 题目描述：操作给的定的二叉树，将其变换为源二叉树的镜像。



解析：对于二叉树的镜像，可以从根节点开始，然后交换左右子树，交换完的左右子树可以看成求新的二叉树镜像，递归的思想。

代码如下：

```
# class TreeNode:
#         def __init__(self, x):
#             self.val = x
#             self.left = None
#             self.right = None
class Solution:
    # 返回镜像树的根节点
    def Mirror(self, root):
        if root == None:
            return root
        else:
            treeNode = root.left
            root.left = root.right
            root.right = treeNode
            self.Mirror(root.left)
            self.Mirror(root.right)
            return root
        # write code here
```

注：对于 else 后面的前三行可以换成 root.left, root.right = root.right, root.left Python 中的交换直接是地址的交换

19 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.

解析：顺时针输出可以看成先输出矩阵的第一行(输出后删除该行)，然后将矩阵逆时针旋转 90 度，又是一个新的矩阵，再输入第一行即可，依次直到矩阵为空为止。

代码如下：

```
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 not matrix :
                break
            matrix = self.turn(matrix)
        return result
    def turn(self,matrix):
        num_r = len(matrix)
        num_c = len(matrix[0])
        newmat = []
        for i in range(num_c):
```

```
            newmat2 = []
            for j in range(num_r):
                newmat2.append(matrix[j][i])
            newmat.append(newmat2)
        newmat.reverse()
        return newmat
```

20 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))

解析：由于要求时间复杂度为 1，因此借助一个辅助栈， 栈顶用来存储栈中的最小元素，当栈压如元素时，判断该元素 A 与辅助栈 栈顶元素 B 的大小，如果 A>B，则将 B 再次压入辅助栈，反之将 A 压入辅助栈；出栈时，两个栈都要删除栈顶元素，这样做的目的是时刻保证两个栈中的元素是一样多的，而且，辅助栈的栈顶时刻是最小元素。

代码如下：

```
class Solution:
    def __init__(self):
        self.stack = []
        self.min_element = []
    def push(self, node):
        self.stack.append(node)
        if not self.min_element:
            self.min_element.append(node)
        else:
            if self.min_element[-1] > node:
                self.min_element.append(node)
            else:
                self.min_element.append(self.min())
        # write code here
    def pop(self):
        if self.stack:
            self.min_element.pop()
            return self.stack.pop()
        return None
        # write code here
    def top(self):
        if self.stack:
            return self.stack[-1]
        return None
        # write code here
    def min(self):
        return self.min_element[-1]
        # write code here
```

21 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都可以为该栈的弹出顺序。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 是该栈序列对应的一个弹出序列，但 4,3,5,1,2 就不可能是该栈序列的

弹出序列。(注意：这两个序列的长度是相等的)

解析：借助一个辅助栈，将栈中的元素一个一个的压入辅助栈，每压入一个判断该元素是否和弹出栈的首元素是否相等，如果相等则删除该元素，同时在弹出栈里面也删除该元素，直到两者元素不相等为止；反之，继续压入。最后只需判断辅助栈是不是空即可。

代码实现：

```
class Solution:
    def IsPopOrder(self, pushV, popV):
        if not pushV or len(pushV) != len(popV):
            return False
        stack = []
        for i in pushV:
            stack.append(i)
            while len(stack) and stack[-1] == popV[0]:
                stack.pop()
                popV.pop(0)
        if len(stack):
            return False
        return True
        # write code here
```

22 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。

解析：二叉树的层次遍历可以采用一个辅助队列，将二叉树的根节点入队列，然后取出根节点，判断根节点有无左右子树，如果有左右子树入队列，依次这样进行，直到队列为空。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回从上到下每个节点值列表，例：[1,2,3]
    def PrintFromTopToBottom(self, root):
        value = []
        queue = [root]
        if not root:
            return []
        while len(queue):
            t= queue.pop(0)
            value.append(t.val)
            if t.left:
                queue.append(t.left)
            if t.right:
                queue.append(t.right)
        return value
        # write code here
```

23 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。(自己注释：二叉查找树它或者是一棵空树，或者是具有下列性质的二叉树： 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；它的左、右子树也分别为二叉排序树)

解析：递归思想： 数组中的最后一个元素是二叉搜索树的根节点，由于左子树都比根节点小，右子树都比根节点大，因此，根节点可以将数组分为左右两部分，每一部分都是一个二叉搜索树的后序遍历，重复上述过程即可。**非递归思想：**非递归也是一个基于递归的思想：左子树一定比右子树小，因此去掉根后，数字分为 left，right 两部分，right 部分的最后一个数字是右子树的根他也比左子树所有值大，因此我们可以每次只看有子树是否符合条件即可，即使到达了左子树左子树也可以看出由左右子树组成的树还想右子树那样处理；对于左子树回到了原问题，对于右子树，左子树的所有值都比右子树的根小可以暂时把他看出右子树的左子树，只需看看右子树的右子树是否符合要求即可。

代码如下：

递归：

```
class Solution:
    def VerifySquenceOfBST(self, sequence):
        sq = sequence
        if not sequence:
            return False
        if len(sequence) == 1:
            return True
        else:
            leftSequence = []
            rightSequence = []
            for i in range(len(sq)):
                if sq[-1] > sq[i]:
                    leftSequence.append(sq[-1])
                else:
                    rightSequence = sq[i:-1]
                    break
            for i in rightSequence:
                if sq[-1] > i:
                    return False
            L = True
            R = True
            if len(leftSequence) > 0:
                L = self.VerifySquenceOfBST(leftSequence)
            if len(rightSequence) >0:
                R = self.VerifySquenceOfBST(rightSequence)
            return  L and R
        # write code here
```

非递归：

```
class Solution:
```

```
    def VerifySquenceOfBST(self, sequence):
        length = len(sequence)
        if length == 0:
            return False
        i = 0
        while length:
            while sequence[i] < sequence[-1]:
                i = i + 1
            while sequence[i] > sequence [-1]:
                i = i + 1
            if i != (length -1):
                return False
            sequence.pop()
            i = 0
            length = length - 1
        return True
        # write code here
```

24 题目描述： 输入一颗二叉树的根结点和一个整数，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:在返回值得 list 中，数组长度大的数组靠前)

解析：先考虑根节点，将整数减去根节点的数值，再考虑左右子树，又是一个新的递归问题。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
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 []
        elif root and not root.left and not root.right and root.val == expectNumber:
            return [[root.val]]
        else:
            res = []
            left = self.FindPath(root.left, expectNumber-root.val)
            right = self.FindPath(root.right, expectNumber-root.val)
            for i in left+right:
                res.append([root.val] + i )
        return res
        # write code here
```

25 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)

解析：将复杂链表的每一个结点复制后插入一个新的链表中，这个过程可以使用递归实现

代码如下：

```
# 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
        p = RandomListNode(pHead.label)
        p.random = pHead.random
        p.next = self.Clone(pHead.next)
        return p
        # write code here
```

26 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，只能调整树中节点指针的指向。

解析：二叉搜索树的中序遍历是一个不减的排序结果，因此可以将二叉树搜索树先中序遍历，，将遍历后的结果用相应的指针连接起来

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Convert(self, pRootOfTree):
        # write code here
        res = self.inorderTraversal(pRootOfTree)
        if len(res) == 0:
            return None
        if len(res) == 1:
            return pRootOfTree
        res[0].left = None
        res[0].right = res[1]
        res[-1].left = res[-2]
        res[-1].right = None
        for i in range(1,len(res)-1):
            res[i].left = res[i-1]
            res[i].right = res[i+1]
        return res[0]
```

```
    def inorderTraversal(self, pRootOfTree):
        if not pRootOfTree:
            return []
        return self.inorderTraversal(pRootOfTree.left) + [pRootOfTree] +
                self.inorderTraversal( pRootOfTree.right)
        # write code here
```

27 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, acd, bac, bca, cab 和 cba。

解析：Python 语言中有内置的全排列函数 permutations()

代码如下：

```
import itertools
class Solution:
    def Permutation(self, ss):
        if not ss:
            return []
        return sorted(list(set(map(''.join, itertools.permutations(ss)))))
        # write code here
```

28 题目描述：数组中有一个数组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}。由于数组 2 在数组中出现了 5 次，超过了数组长度的一半，因此输出 2。如果不存在则输出 0。

解析：遍历列表中的每一个元素，并将元素作为 key，元素出现的次数做为 value，最后统计字典中元素的 value。

代码如下：

```
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        k = len(numbers)
        dic = {}
        for i in numbers:
            if dic.has_key(i):
                dic[i] += 1
            else:
                dic[i] = 1
        for key, value in dic.iteritems():
            if value > k/2:
                return key
        return 0
        # write code here
```

29 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。

解析：考虑先对整数从小到大排序，对排序后的整数取出前 K 个即可。

代码如下：

```
class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        if len(tinput) < k:
```

```
            return []
        tinput.sort()
        return tinput[:k]
        # write code here
```

30 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：在古老的一堆模式识别中，常常需要就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)

解析：对于连续子序列的和最大，可以用动态规划来求，假如用 f(n)表示以第 n 个数结尾的最大连续和，对于每一个结尾的数字都可以找到一个最大的联系和，只需要把最大的那个找出来即可。

代码如下：

```
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        max_value = array[0]
        fn = array[0]
        for i in range(len(array)-1):
            if fn + array[i+1] > array[i+1]:
                fn = fn + array[i+1]
                if fn > max_value:
                    max_value = fn
            else:
                fn = array[i+1]
                if fn > max_value:
                    max_value = array[i+1]
        return max_value
        # write code here
```

31 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，但是对于后面的问题他就没辙了。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)

解析：规律：当计算右数第 i 位包含的 X 的个数时：

1. **取第 i 位左边(高位)的数字，乘以 10i – 1,得到基础值 a。**
2. **取第 i 位数字，计算修正值：**
   1) **如果大于 X，则结果为 a +10i -1**
   2) **如果小于 X，则结果为 a**
   3) **如果等于 X，则取第 i 为右边(低位)数字，设为 b，最后结果为 a+b+1.**

代码如下：

```
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        count = 0
        i = 1
```

```
        tmp = n
        while n :
            div = n // 10 #计算基础值
            value = n % 10
            base = div * i
            if value > 1:
                count = count + base + i
            elif value < 1:
                count = count + base
            else:
                count = count + base + (tmp % i) + 1
            i =i * 10
            n = n / 10
        return count
        # write code here
```

32 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。

解析：由于最后是把数字连接到一起找到最小的值，因此可以按照 ab – ba 这样的规则，将两个数字连接到一起进行比较大小，然后按照这样的方法对数字进行排序，将排序后的数字再连接到一起。

代码如下：

```
class Solution:
    def PrintMinNumbers(self, numbers):
        if not numbers:
            return “”
        lam = lambda x, y: int(str(x)+str(y)) –    int(str(y)+str(x))
        s = sorted(numbers, cmp=lam)
        return int(‘’.join([str(i) for I in s]))
        # write code here
```

33 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。

解析：丑数有一个很重要的性质是，丑数只能由丑数的成绩得到，因此可以从第一个丑数 1 开始，乘以 2、3、5，取其中的最小的数，作为第二个丑数，此时是 2，因子 2 的下一次 index 加 1，因子 3 和 5 的 index 不变，依次类推。

代码如下：

```
class Solution:
    def GetUglyNumber_Solution(self, index):
        if index <= 0:
            return 0
        aList = [1]
        aList2 = 0
        aList3 = 0
```

```
        aList5 = 0
        for i in range(index -1):
            minValue = min(aList[aList2] * 2, aList[aList3] * 3, aList[aList5] * 5)
            aList.append(minValue)
            if minValue == aList[aList2] * 2:
                aList2 += 1
            if minValue == aList[aList3] * 3:
                aList3 += 1
            if minValue == aList[aList5] * 5:
                aList5 += 1
        return aList[-1]
        # write code here
```

34 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)

解析：Python 中 list 有统计元素出现次数的函数 count 以及返回元素索引的函数 index，因此，可以先将字符串中的每一个字符放在列表中，并在列表中找到第一次且仅出现一次的字符。

代码如下：

```
class Solution:
    def FirstNotRepeatingChar(self, s):
        aList = []
        if not s:
            return -1
        for i in s:
            aList.append(i)
        for i in aList:
            if aList.count(i) == 1:
                return aList.index(i)
        return -1
        # write code here
```

35 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

解析：为了减少时间的复杂度可以采用归并排序，归并的过程中，从数组的最后一位开始比较，如果 left 比 right 值的大，则出现一次逆序对，依次这样进行下去。

代码如下：

```
class Solution:
    sum = 0
    def InversePairs(self, data):
        self.MergeSort(data)
        return self.sum % 1000000007
    def Merge(self, left, right): #归并排序
        l, r = 0, 0
        result = []
```

```
        i, j = -1, -1
        while i >= -len(left) and j >= -len(right):
            if left[i] > right[j]:
                self.sum = self.sum + len(right) + j + 1
                i = i - 1
            else:
                j = j - 1
        while l < len(left) and r < len(right):
            if left[l] < right[r]:
                result.append(left[l])
                l += 1
            else:
                result.append(right[r])
                r += 1
        result += right[r:]
        result += left[l:]
        return result
    def MergeSort(self, lists):
        if len(lists) <= 1:
            return lists
        num = len(lists) / 2
        left = self.MergeSort(lists[:num])
        right = self.MergeSort(lists[num:])
        return self.Merge(left, right)
        # write code here
```

36 题目描述：输入两个链表，找出它们的第一个公共节点。

解析：将其中一个链表的值保存在 list 中，之后再遍历第二个链表，检查每个节点的值是不是在 list 中即可。

代码如下：

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def FindFirstCommonNode(self, pHead1, pHead2):
        list1 = []
        node1 = pHead1
        node2 = pHead2
        while node1:
            list1.append(node1.val)
            node1 = node1.next
        while node2:
            if node2.val in list1:
                return node2
```

```
            else:
                node2 =node2.next
        # write code here
```

37 题目描述：统计一个数字在排序数组中出现的次数。

解析：对于有序表的查找可以使用折半查找，由于题目说的全是整数，因此可以查找 k+0.5 以及 k-0.5 的索引，两者的差值便是需要查找元素在数组中出现的次数。此外，Python 中有专门统计某个数字在数组中出现次数的函数 list.count(k)。

代码如下：

```
class Solution:
    def GetNumberOfK(self, data, k):
        return self.binSearch(data, k+0.5) - self.binSearch(data, k - 0.5)
    def binSearch(self, data, k):
        if not data:
            return False
        data_length = len(data)
        left = 0
        right = data_length - 1
        while left <= right:
            mid = (left + right)/2
            if data[mid] > k:
                right = mid -1
            elif data[mid] < k:
                left = mid + 1
        return left
        # write code here
```

38 题目描述：输入一颗二叉树，求该树的深度。从根节点到叶节点依次经过的结点(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。

解析: 树的定义是递归的形式，树的深度可以求左子树和右子树深度的最大值 +1，同样左子树和右子树又是一棵新的树，依次类推，直到叶子结点。

代码实现：

```
# 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
        elif not pRoot.left and not pRoot.right:
            return 1
        else:
            return max(self.TreeDepth(pRoot.left) + 1, self.TreeDepth(pRoot.right) + 1)
        # write code here
```

39 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

解析：平衡二叉树的左右子树的深度差不超过 1，因此，可以判断左子树的深度和右子树的深度，同样左右子树也同样是二叉平衡树。

代码如下：

```
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
        else:
            if abs(self.binTreeDepth(pRoot.left) - self.binTreeDepth(pRoot.right))> 1:
                return False
            return self.IsBalanced_Solution(pRoot.left) and self.IsBalanced_Solution(pRoot.right)
    def binTreeDepth(self, root):
        if not root:
            return 0
        if not root.left and not root.right:
            return 1
        return max(self.binTreeDepth(root.left) + 1, self.binTreeDepth(root.right) + 1)
        # write code here
```

40 题目描述：一个整型数组里的除了两个数字之外，其他的数字都出现了偶数次。请写程序找出这两个只出现一次的数字。

解析：Python 里面有可以统计列表中某个元素出现次数的函数 count。

代码如下：

```
class Solution:
    # 返回[a,b] 其中 ab 是出现一次的两个数字
    def FindNumsAppearOnce(self, array):
        aList= []
        for i in array:
            if array.count(i) == 1:
                aList.append(i)
        return aList
        # write code here
```

41 题目描述：小明很喜欢数学，有一天他在做数学作业时，要求就算出 9-16 的和，他马上就写出了正确的答案是 100.但是他并不满足于此，他在想究竟有多少中连续的正数序列的和为 100(至少包括两个数)。没过多久，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！

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

解析：这类问题完全是需要数学知识来求解的，假设开始数字是 a，一共有 t 个数字的连续和是 S，则有 at + t(t-1)/2 = S, 根据这个式子可以求出 a = (S – t(t-1)/2)/t; 另一方面 t 的最大值从 1 开始到 t 的和是 S，那么可知 t <= [-1 + sqrt(1 + 8S)]/2。

代码如下：

```
import math
class Solution:
    def FindContinuousSequence(self, tsum):
        list1 = []
        conList = []
        maxNumber = (int)((math.sqrt(8 * tsum + 1) -1) / 2)
        while maxNumber >= 2:
            minus = tsum - self.calcContunuousSequenceSum(maxNumber)
            div = minus / maxNumber
            if minus % maxNumber == 0 and    div > 0:
                for i in range(maxNumber):
                    conList.append(div + i)
                list1.append(conList)
            conList = []
            maxNumber -= 1
        return list1
    def calcContunuousSequenceSum(self, n):
        return n * (n-1) / 2
        # write code here
```

42 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。

输出描述：对应每个测试案例，输出两个数，小的先输出。

解析：由于数组是递增有序的且要求的两者和是一定的，因此越是相聚越远的数字乘积越小，可以从数组的前面元素和后面元素一起遍历，找到第一对和为 S 的即可。

代码如下：

```
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        aList = []
        for i in array:
            if array.count(tsum - i) >= 1 and array.index(i) < array.index(tsum - i):
                aList.append(i)
                aList.append(tsum - i)
                break
        return aList
        # write code here
```

43 题目描述：汇编语言汇总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，请你把循环左移 K 位后的序列输出。例如，字符串序列 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！

解析：左移 n 位只需要将前 n 位的字符，移到之前字符串的后面即可

代码如下：

```
class Solution:
    def LeftRotateString(self, s, n):
        if len(s) > n :
            str1 = s[n:]
            str2 = s[:n]
            return str1 + str2
        return s
        # write code here
```

44 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，你能帮助他吗？

解析：对于这样的句子，可以使用 split 函数对句子进行划分，对于划分后的每一个单词，可以使用空格连接起来。

代码如下：

```
class Solution:
    def ReverseSentence(self, s):
        str1 = s.split(' ')
        str1.reverse()
        return " ".join(str1)
        # write code here
```

45 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，想测试自己的手气，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 看做 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(大小王分别看做 2 和 4)，So Lucky！。LL 决定去买体育彩票啦。现在，要求你使用这副牌模拟以上的过程，然后告诉我们 LL 的运气如何，如果牌能够组成顺子就输出 true,否则就输出 false。为了方便起见，你可以认为大小王是 0。

解析：对于能组成顺子的数字，排序后的每一个相邻间隔是 0(我们认为 1,2 的间隔是 0；1，3 的间隔是 1)， 因此，题目给的数组，只需要把非零的数字排序后统计间隔，如果间隔的总和小于非 0 的数字，那么可以组成顺子，反之不可以。

代码如下：

```
class Solution:
    def IsContinuous(self, numbers):
        if not numbers:
            return False
        aList = [i for i in numbers if i > 0]
        aList.sort()
        zero_count = len(numbers) - len(aList)
        interval_total = 0
        for i in range(len(aList) - 1):
            if (aList[i + 1] - aList[i] > 0):
```

```
                interval_total += aList[i +1] -aList[i] - 1
            else:
                return False
        if zero_count >= interval_total:
            return True
        return False
        # write code here
```

46 题目描述：每年六一儿童节，牛客都会准备一些小礼物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，让小朋友围城一个大圈。然后，他随机指定一个数 m，让编号为 0 的小朋友开始报数。每次喊道 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)

解析：该题为约瑟夫环的问题。

代码如下：

```
class Solution:
    def LastRemaining_Solution(self, n, m):
        if not n or not m:
            return -1
        res = [i for i in range(n)]
        i = 0
        while len(res) > 1:
            i = (m - 1 + i)% len(res)
            res.pop(i)
        return res[0]
```

47 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)。

解析：在 Python 里面可以使用求和函数 sum。

代码如下：

```
class Solution:
    def Sum_Solution(self, n):
        aList = [i +1 for i in range(n)]
        return sum(aList)
        # write code here
```

48 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。

解析：此题也是使用 Python 列表里面的 sum 函数

代码如下：

```
class Solution:
    def Add(self, num1, num2):
        list1=[]
        list1.append(num1)
        list1.append(num2)
```

```
        return sum(list1)
```

49 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。

解析：可以使用强制类型转换，如果抛出异常，则返回 0

代码如下：

```
class Solution:
    def StrToInt(self, s):
        try:
            return int(s)
        except Exception as e:
            return 0
        # write code here
```

50 题目描述：在一个长度为 n 的数组里的所有数字都出现在 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。

解析：Python 中的列表有统计元素个数的功能，因此可以直接进行统计元素出现的次数，找到第一个出现的次数大于 1 时，返回即可。

代码如下：

```
class Solution:
    # 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
    def duplicate(self, numbers, duplication):
        for i in numbers:
            if numbers.count(i) > 1:
                duplication.insert(0,i)
                return True
        return False
        # write code here
```

51 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。

解析：连续用同一个运算符操作，可以联合使用 reduce 函数和 lambda 函数

代码如下：

```
class Solution:
    def multiply(self, A):
        B = []
        B.append(reduce(lambda x,y: x*y, A[1:]))
        for i in range(len(A)-2):
            B.append(reduce(lambda x,y: x*y, A[:i+1] + A[i+2:]))
        B.append(reduce(lambda x,y: x*y, A[:(len(A)-1)]))
        return B
        # write code here
```

52 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(包括 0 次)。在本题中，匹配是指字符串(s)的所有字符匹配整个模式(p)。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。

解析：对于模式匹配，可以这样考虑，对于 p，如果下一个字符是*，则会有二种情况: 当前 s 字符和 p 当前字符匹配，当前字符 s 和 p 当前字符不匹配。对于前者，又可以分三种情况，match(s[1:], p[2:])(*前面字符出现一次), match(s, p[2:])(*前面字符出现 0 次),和 match(s[1:], p)(*前面字符出现多次)；对于后者，则只需匹配 match(s, p[2:])。如果模式 p 中的下一次字符不是*，同样也分两种情况，对于第一个字符匹配的，则只需 match(s[1:], p[1:])， 第一个字符不匹配的直接返回 False 即可。

代码如下：

```
class Solution:
    # s, pattern 都是字符串
    def match(self, s, pattern):
        if len(s) == 0 and len(pattern) == 0:
            return True
        elif len(s) > 0 and len(pattern) == 0:
            return False
        elif len(pattern) > 1 and pattern[1] == '*':
            if len(s) > 0 and (s[0] == pattern[0] or pattern[0] == '.'):
                return self.match(s[1:], pattern[2:]) or self.match(s, pattern[2:]) or self.match(s[1:], pattern)
            else:
                return self.match(s, pattern[2:])
        elif len(s) > 0 and (s[0] == pattern[0] or pattern[0] == '.'):
            return self.match(s[1:], pattern[1:])
        return False
        # write code here
```

53 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。

解析：此题的解法和 48 题的解法是一样的

代码如下：

```
class Solution:
    # s 字符串
    def isNumeric(self, s):
        try :
            p = float(s)
            return True
        except :
            return False
        # write code here
```

54 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”。当从该字符流汇总读出前六个字符”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。

解析：由于字符是一个一个的读进来的，因此可以将每次读入的字符连接起来，然后统计每个字符出现的次数。

代码如下：

```
class Solution:
    #  返回对应 char
    def __init__(self):
        self.s=''
        self.dict1={}
    def FirstAppearingOnce(self):
        # write code here
        for i in self.s:
            if self.dict1[i]==1:
                return i
        return '#'
    def Insert(self, char):
        # write code here
        self.s=self.s+char
        if char in self.dict1:
            self.dict1[char]=self.dict1[char]+1
        else:
            self.dict1[char]=1
        # write code here
```

55 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

解析：将链表的结点一个一个的加入列表中，如果存在环，则会出现两次，只需要返回出现两次的结点即可。

代码如下：

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def EntryNodeOfLoop(self, pHead):
        aList = []
        if not pHead:
            return None
        while pHead:
            aList.append(pHead)
            if aList.count(pHead) == 2:
                return pHead
            pHead = pHead.next
        return None
        # write code here
```

56 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

解析：由于链表是递增的，因此可以考虑对链表的取值一一判断，符合条件的保留下来，不符合条件的直接 pass。

代码如下：

```
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def deleteDuplication(self, pHead):
        newListNode = ListNode(0)
        index = newListNode
        while pHead and pHead.next:
            if pHead.val == pHead.next.val:
                while pHead.next and pHead.val == pHead.next.val: #这两个条件不能交换
                    pHead = pHead.next
            else:
                index.next = pHead
                index = index.next
            pHead = pHead.next
        index.next = pHead
        return newListNode.next
        # write code here
```

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

解析：1、有右子树的，那么下个结点就是右子树最左边的点； 2、没有右子树的，也可以分成两类，a)是父节点左孩子，那么父节点就是下一个节点 ； b)是父节点的右孩子找他的父节点的父节点的父节点...直到当前结点是其父节点的左孩子位置。如果没有那么他就是尾节点。

代码如下：

```
# class TreeLinkNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None
class Solution:
    def GetNext(self, pNode):
        if not pNode:
            return pNode
        if pNode.right:#有右孩子
            left1=pNode.right
            while left1.left:
                    left1=left1.left
```

```
            return left1
        p=pNode
        while pNode.next:
            tmp=pNode.next
            if tmp.left==pNode:
                return tmp
            pNode=tmp
        # write code here
```

58 题目描述：请实现一个函数，用来判断一棵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，定于其为对称的。

解析：用递归判断左右子树是不是对称即可。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def sym_Tree(self, root1, root2):
        if not root1 and not root2:
            return True
        if root1 and not root2:
            return False
        if root2 and not root1:
            return False
        if root2.val != root1.val:
            return False
        return self.sym_Tree(root1.left, root2.right) and self.sym_Tree(root1.right, root2.left)
    def isSymmetrical(self, pRoot):
        if not pRoot:
            return True
        result = self.sym_Tree(pRoot, pRoot)
        return result
        # write code here
```

59 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。

解析：可以采用层次遍历的思想，将每一层的节点值保存下来，然后判断杀死从左到右还是从右到左。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
```

```
class Solution:
    def Print(self, pRoot):
        if not pRoot:
            return []
        result = []
        curLevel = [pRoot]
        leftToRight = True
        while curLevel:
            nextLevel = []
            curLevelValue = []
            for i in curLevel:
                curLevelValue.append(i.val)
                if i.left:
                    nextLevel.append(i.left)
                if i.right:
                    nextLevel.append(i.right)
            if leftToRight:
                result.append(curLevelValue)
            else:
                curLevelValue.reverse()
                result.append(curLevelValue)
            curLevel = nextLevel
            leftToRight = not leftToRight
        return result
        # write code here
```

60 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。

解析：层级遍历即可，层次遍历的基础是使用队列。

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回二维列表[[1,2],[4,5]]
    def Print(self, pRoot):
        if not pRoot:
            return []
        curLevel = [pRoot]
        result = []
        while curLevel:
            curLevelValue = []
            nextLevelNode = []
            for i in curLevel:
```

```
                curLevelValue.append(i.val)
                if i.left:
                    nextLevelNode.append(i.left)
                if i.right:
                    nextLevelNode.append(i.right)
            result.append(curLevelValue)
            curLevel = nextLevelNode
        return result
        # write code here
```

61 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

解析：序列化时按照某种遍历方式将树转换成字符串，反序列化时根据已有的字符串建立二叉树

代码如下：

```
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Serialize(self, root):
        # write code here
        if root == None:
            return "#"
        return str(root.val) + "," + self.Serialize(root.left) + "," + self.Serialize(root.right)

    def Deserialize(self, s):
        # write code here
        root, index = self.helper(s.split(","), 0)
        return root

    def helper(self, s, index):
        if s[index] == "#":
            return None, index + 1
        root = TreeNode(int(s[index]))

        index += 1
        root.left, index = self.helper(s, index)
        root.right, index = self.helper(s, index)
        return root, index
```

62 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是 4。

解析：二叉搜索树的中序遍历是递增的序列，可以采用中序遍历二叉树的方法来实现该问题的解。

代码实现：

```
# class TreeNode:
#         def __init__(self, x):
#             self.val = x
#             self.left = None
#             self.right = None
class Solution:
    # 返回对应节点 TreeNode
    def KthNode(self, pRoot, k):
        aList = self.inorderTravel(pRoot)
        if k == 0:
            return None
        if len(aList) < k:
            return None
        else:
            return aList[k-1]
    def inorderTravel(self, pRoot):
        if not pRoot:
            return []
        return self.inorderTravel(pRoot.left) + [pRoot] + self.inorderTravel(pRoot.right)
        # write code here
```

63 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。

解析：这个没啥好说的。

代码实现：

```
class Solution:
    def __init__(self):
        self.data=[]
    def Insert(self, num):
        # write code here
        self.data.append(num)
        self.data.sort()
    def GetMedian(self,data):
        # write code here
        length=len(self.data)
        if length%2==0:
            return (self.data[length/2]+self.data[length/2-1])/2.0
        else:
            return self.data[int(length/2)]
        # write code here
```

64 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]}。

解析：首先要明白滑动窗口的个数是多少，很显然是数组的长度 L 减去窗口的大学 S 然后再加上 1，即：L – S + 1，对于每一个窗口找出最大值即可。

代码如下：

```
class Solution:
    def maxInWindows(self, num, size):
        if len(num) < size or size == 0:
            return []
        maxList = []
        for i in range(len(num) - size + 1):
            maxList.append(max(num[i:(i+size)]))
        return maxList
        # write code here
```

65 题目描述：请设计一个函数，用来判断在一个矩阵汇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 4 X 3 矩阵中包含一条字符串”bcced”的路径，但是矩阵中不包含”abcd”路径，因为字符串的第一个字符 b 占据了矩阵汇总的第一行第二个格子之后，路径不能再次进入该格子。

解析：先搜索第一个匹配的格子，匹配后，搜索该格子的各个方向，直到匹配完所有的路径，否则回溯开始搜索第二个匹配的格子，依次这样进行，直到找到匹配的路径或者是搜索完。(需要注意的是，搜索过该格子后，将该格子用特殊符号标记，防止再次访问该格子，另外 Python 中字符串是不可以修改的数据类型，将其转换成列表，以便修改)

代码实现：

```
class Solution:
    def hasPath(self, matrix, rows, cols, path):
        for i in range(rows):
            for j in range(cols):
                if matrix[i*cols+j] == path[0]:
                    if self.find(list(matrix),rows,cols,path[1:],i,j):
                        return True
        return False
    def find(self,matrix,rows,cols,path,i,j):
        if not path:
            return True
        matrix[i*cols+j]='0'
        if j+1<cols and matrix[i*cols+j+1]==path[0]:
            return self.find(matrix,rows,cols,path[1:],i,j+1)
        elif j-1>=0 and matrix[i*cols+j-1]==path[0]:
            return self.find(matrix,rows,cols,path[1:],i,j-1)
        elif i+1<rows and matrix[(i+1)*cols+j]==path[0]:
            return self.find(matrix,rows,cols,path[1:],i+1,j)
        elif i-1>=0 and matrix[(i-1)*cols+j]==path[0]:
```

```
            return self.find(matrix,rows,cols,path[1:],i-1,j)
        else:
            return False
        # write code here
```

66 题目描述：地上有一个 m 行 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。请问该机器人能到达多少个格子？

解析：对于满足条件的格子，从第一个开始搜索，满足条件了计数器加 1，然后从第一个格子的四个方向开始搜索(如果方向存在)，直到结束；另外对于已经搜索过的格子可以将其进行 1 标记，初始的格子全是 0。

代码如下：

```
class Solution:
    def movingCount(self, threshold, rows, cols):
        matrix = [[0 for _ in range(cols)]for _ in range(rows)]
        def block(r, c):
            return sum(map(int, str(r) + str(c))) > threshold
        def DFS( r, c):
            if not(0 <= r < rows and 0 <= c < cols):
                return 0
            if block(r,c):
                return 0
            if matrix[r][c] == 1:
                return 0
            matrix[r][c] = 1
            return DFS(r-1, c) + DFS(r+1,c) + DFS(r, c-1) + DFS(r, c+1) + 1
        return DFS(0,0)
        # write code here
```