

1. 二维数组中的查找

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     # array 二 维 列 表
4     def Find(self, target, array):
5         if array == []:
6             return False
7         num_row = len(array) # get how many rows the array has
8         num_col = len(array[0])
9         i = num_col - 1
10        j = 0
11        while i >=0 and j<num_row:
12            if array[j][i]>target:
13                i = i - 1
14            elif array[j][i]<target:
15                j = j+1
16            else:
17                return True
18
```

2. 替换空格

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     # s
4     def replaceSpace(self, s):
5         # write code here
6         i = 0
7         n = len(s)
8         ss=[]#用 于 盛 放 转 化 完 的 字 符 串
9         for i in range(n):
10             if s[i].isspace():#判 断 是 否 为 空 格
11                 ss.append('%20')
12             else:
13                 ss.append(s[i])
14             i +=1
15         ss=''.join(ss)#将 列 表 转 成 字 符 串
16         return ss
17
18
```

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。

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 class Solution:
7     # 返回从尾部到头部的列表值序列，例如 [1,2,3]
8     def printListFromTailToHead(self, listNode):
9         # write code here
10        # 注意，Python 中没有那个 == NULL 的写法
11        if listNode == []:
12            return False
13        result = []
14
15        while listNode:
16            result.insert(0, listNode.val) # 这种写法就相当于把 Python 最开始的元
素向后顶了
17            listNode = listNode.next
18        return result
19
```

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     #Need to understand again!
4     def __init__(self, x):
5         self.val = x
6         self.left = None
7         self.right = None
8 class Solution:
9     # 返回构造的TreeNode根节点
10    def reConstructBinaryTree(self, pre, tin):
11        # write code here
12        if not pre and not tin:
13            return None
14        root = TreeNode(pre[0])
15        if set(pre) != set(tin):
16            return None
17        i = tin.index(pre[0])
18        root.left = self.reConstructBinaryTree(pre[1:i+1], tin[:i])
19        root.right = self.reConstructBinaryTree(pre[i+1:], tin[i+1:])
20        return root
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def __init__(self):
4         self.stack1=[]
5         self.stack2=[]
6
7     def push(self, node):
8         # write code here
9         self.stack1.append(node)
10
11    def pop(self):
12        # return xx
13        if len(self.stack1) == 0 and len(self.stack2) == 0:
14            return
15        elif len(self.stack2) == 0:
16            while len(self.stack1) > 0:
17                self.stack2.append(self.stack1.pop())
18        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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def minNumberInRotateArray(self, rotateArray):
4         # write code here
5         # 重新理解下，这道题目非常重要！！！
6         if len(rotateArray) == 0:
7             return 0
8         front = 0
9         rear = len(rotateArray)-1
10        minValue = rotateArray[0]
11        if rotateArray[front] < rotateArray[rear]:
12            return rotateArray[front]
13        else:
14            # 要查找最小值，用二分查找的方法还是最方便的
15            while (rear-front)> 1:
16                mid = (front+rear)//2
17                if rotateArray[mid]>=rotateArray[front]:
18                    front = mid
19                elif rotateArray[mid]<=rotateArray[rear]:
20                    rear = mid
21                elif rotateArray[front] == rotateArray[rear] == rotateArray[mid]:
22                    for i in range(1,len(rotateArray)):
23                        if rotateArray[i]<minValue:
24                            minValue = rotateArray[i]
25                            rear = i
26        minValue = rotateArray[rear]
27        return minValue
28
```

7. 斐波那契数列（动态规划）

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

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def Fibonacci(self, n):
4         # write code here
5         # 这种解法是非递归的解法
6         tempArray = [0,1]
7         #tempArray[0]=0
8         #tempArray[1]=1
9
10        if n<=1:
11            return tempArray[n]
12        if n>=2 and n<=39:
13            for i in range(2,n+1):
14                # 不能用这种方法去赋值，并没有定义空间是多少，c++也类似，碰到
15                # 这种情况只能push_back
16                #tempArray[i]=tempArray[i-1]+tempArray[i-2]
17                tempArray.append(tempArray[i-1]+tempArray[i-2])
18        return tempArray[n]
19
20    # 递归实现算法复杂度太高了递归实现算法复杂度太高了了，是指级别的
21    # 递归实现算法
22    class Solution():
23        def Fibnacci(self,n):
24            if n <= 0:
25                return 0
26            if n == 1:
27                return 1
28            return self.Fibnacci(n-1) + self.Fibnacci(n-2)
```

8. 跳台阶 (动态规划)

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）。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def jumpFloor(self, number):
4         # write code here
5         # 可以理解为分情况的斐波拉契递归的程序
6         # 可以理解为n=1 1种方法； n=2时， 2中方法； 然后递归计算
7         rempArray=[1,2]
8         if number<=2:
9             return number
10        elif number >= 3:
11            for i in range(2,number):
12                rempArray.append(rempArray[i-1]+rempArray[i-2])
13        return rempArray[number-1]
14
```

9. 变态跳台阶 (动态规划)

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def jumpFloorII(self, number):
4         # write code here
5         # for 1, 1 solution; for 2, 2 solution; for 3, 4 solutions
6         # 为什 么 是 两 倍 , 想 不 通 ? ? ?
7         rempArray = [1,2]
8         if number <=2:
9             return number
10        elif number >= 3:
11            for i in range(2,number):
12                #rempArray[i]=rempArray[i-1]+rempArray[i-2]+1
13                rempArray.append(rempArray[i-1]*2)
14        return rempArray[number-1]
```

10. 矩形覆盖 (动态规划)

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 $2 \times n$ 的大矩形，总共有多少种方法？

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def rectCover(self, number):
4         # write code here
5         # n= 1, 1 种 ; n=2, 2 种 ;
6         rempArray=[1,2]
7         if number <=2:
8             return number
9         elif number >=3:
10            for i in range(2,number):
11                rempArray.append(rempArray[i-1]+rempArray[i-2])
12            return rempArray[number-1]
13        ...
14        rempArray=[1,2]
15        if number<=2:
16            return number
17        elif number >= 3:
18            for i in range(2,number):
19                rempArray.append(rempArray[i-1]+rempArray[i-2])
20            return rempArray[number-1]
21        ...
```

11. 二进制中1的个数（位运算）

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def Number0f1(self, n):
4         # 看一下剑指offer书里怎么写的！！！
5         # 每个非零整数n和n-1进行按位与运算，整数n的二进制中，
6         # 最右边的1就会变成0，利用循环，计算经过几次运算二进制变成0
7         # 就有几个1。
8         # 需要重新看一次！！！
9         count = 0
10        if n < 0:
11            n = n & 0xffffffff
12        while n != 0:
13            count += 1
14            n = (n - 1) & n
15        return count
```

12. 数值的整数次方（位运算）

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def Power(self, base, exponent):
4         # write code here
5         return pow(base,exponent)
6     # 这个会不会太简单了，看看别人怎么写的！！
7
8
9
10
11
12
13
14
15
16
17 上面的很简单，没有使用快速幂算法，下面使用一下快速幂算法，快速
18 幂算法参考下面的博客
19 https://blog.csdn.net/hkdgjqr/article/details/5381028
20
21     def fast_power(self, base, exponent):
22         if base == 0:
23             return 0
24         if exponent == 0:
25             return 1
26         e = abs(exponent)
27         tmp = base
28         res = 1
29         while(e > 0):
30             #如果最后一位为1，那么给res乘上这一位的结果
31             if (e & 1 == 1):
32                 res = res * tmp
33             e = e >> 1
34             tmp = tmp * tmp
35         return res if exponent > 0 else 1/res
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def re0rderArray(self, array):
4         # write code here
5         #myarray = list(array)
6         result = []
7         for i in range(len(array)):
8             if array[i]%2 ==1:
9                 result.append(array[i])
10            for j in range(len(array)):
11                if array[j]%2 == 0:
12                    result.append(array[j])
13        return result
14
```

14. 链表汇总倒数第k个节点（链表）

题目描述

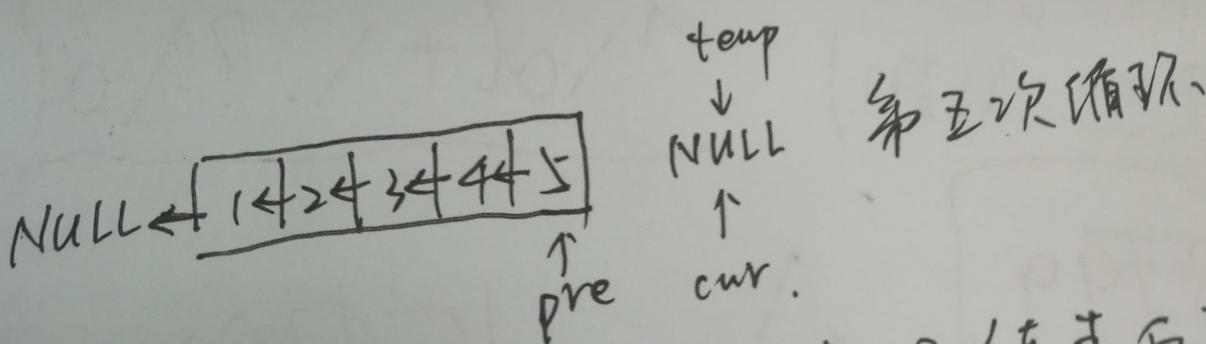
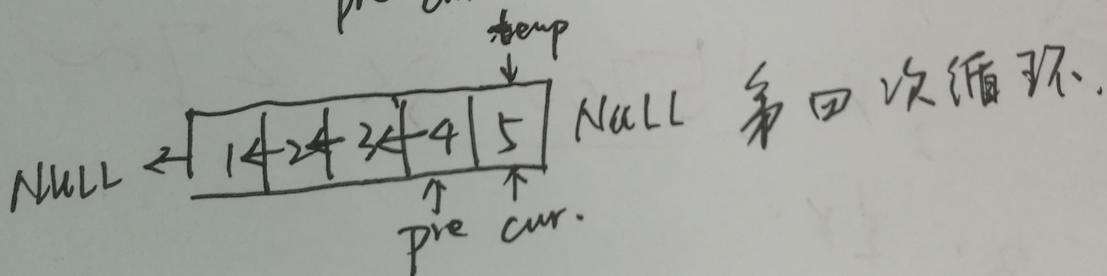
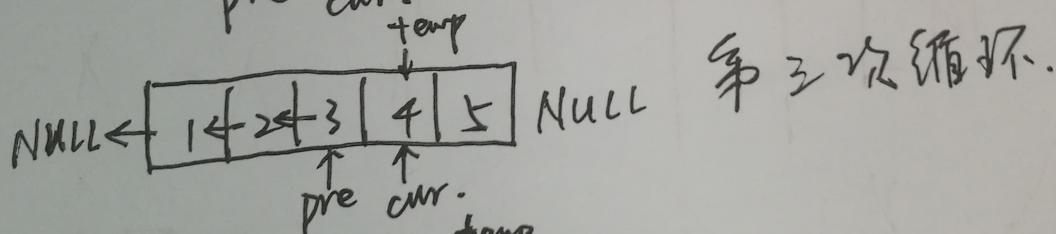
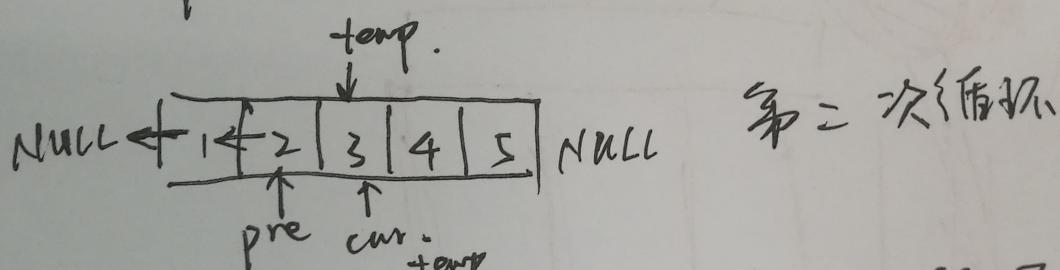
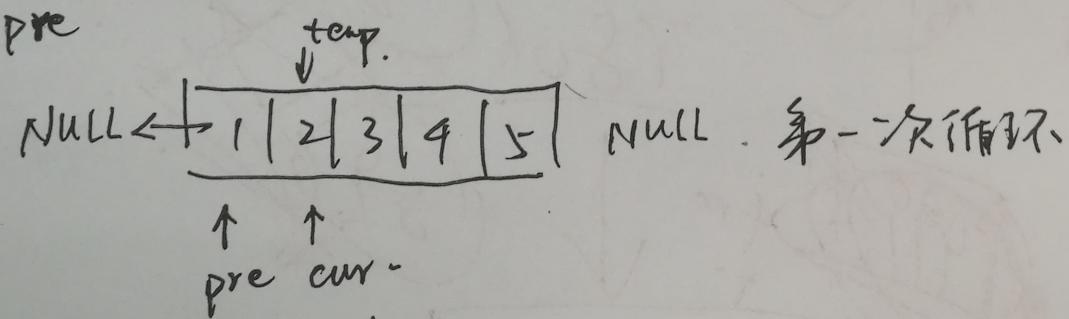
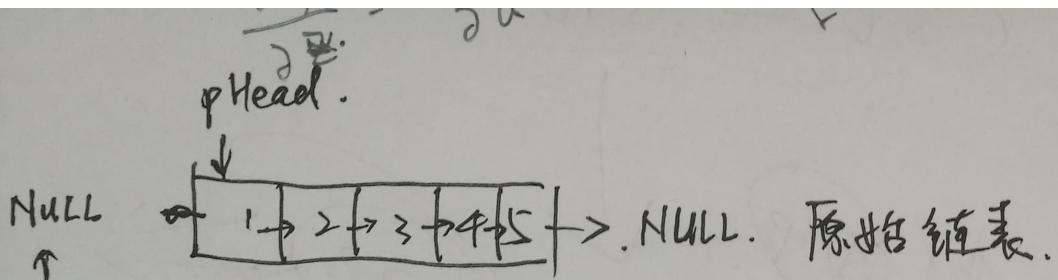
输入一个链表，输出该链表中倒数第k个结点。

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 # Need to repeat again!!! 这种思路可以学习！！！
7 # 设置两个指针指向头节点，第一个指针向前走 k-1 步，走到第 k 个结点，此时，
8 # 第二个指针和第一个指针同时移动，当第一个指针到尾节点的时候，
9 # 第二个指针指向倒数第 k 个结点，注意链表为空，k 为 0，k 大于链表的长度的情况
10 class Solution:
11     def FindKthToTail(self, head, k):
12         # write code here
13         # 链表的倒数第 k 个节点
14         # 设计到倒数第 k 个的问题，总是可以这样两个夹一下！！！
15         if head == None or k <= 0:
16             return None
17
18         pAhead = head # 表示指向头指针
19         pBhead = None # 表示没有指向任何指针
20
21         for i in range(k-1):
22             if pAhead.next != None:
23                 pAhead = pAhead.next
24             else:
25                 return None
26         pBhead = head
27         while pAhead.next != None:
28             pAhead = pAhead.next
29             pBhead = pBhead.next
30
31         return pBhead
```

15. 反转链表（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。



故，输出 pre 即可实现链表反转.

```

1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):

```

```
4     self.val = x
5     self.next = None
6 class Solution:
7     # 返回 ListNode
8     # 这道题目很基础，我还是没写起来，看一下桌面上那个画的示意图
9     ! ! !
10    def ReverseList(self, pHead):
11        # write code here
12        if pHead==None or pHead.next==None:
13            return pHead
14        # pre 和 cur 是两个标记指针，一前一后
15        pre = None # 前面的指针
16        cur = pHead # 后面的指针
17        while cur!=None:
18            tmp = cur.next# 标志 cur后面的指针
19            cur.next = pre # 标志 cur前面的指针
20            pre = cur # 指针向后移动
21            cur = tmp # 指针向后移动
22        return pre
```

16. 合并两个排序的链表 (链表)

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。

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 class Solution:
7     # 返回合并后列表
8     # 重新理解一下这种调用自身的方法！！！---就是递归
9
10    ...
11    1 对2个链表是否为空进行处理
12    2 比较2个链表头结点的值得大小，把较小的一个作为新链表的头结点，继续判断剩余2个链表的头结点
13    的大小，返回结点，与上一个连接起来，然后递归的进行运算。递归判断条件就是其中一个链表为空，返回剩余的链表，与之前的连接起来。
14    ...
15    #这种是非递归的方式
16    def Merge(self, pHead1, pHead2):
17        #write code here
18        phead = ListNode(0)# 创建链表
19        tmp = phead # 新建链表指针
20        while pHead1 and pHead2:
21            if pHead1.val <= pHead2.val:
22                tmp.next = pHead1
23                pHead1 = pHead1.next
24            else:
25                tmp.next = pHead2
26                pHead2 = pHead2.next
27            tmp = tmp.next # 更改tmp游标的位罝，表示现在这才是tmp
28
29            if pHead1 is None:
30                tmp.next = pHead2# 直接把pHead2接到这个指针后面
31            if pHead2 is None:
32                tmp.next = pHead1
33
34            #return tmp
35        return phead.next # 如果不加next，就会出现初始定义的那个0
36
37
```

```
1
2 #这个是递归的版本
3 def Merge(self, pHead1, pHead2):
4     # write code here
5     if pHead1==None:
6         return pHead2
7     if pHead2 == None:
8         return pHead1
9     pMergeHead = None
```

```
10
11
12
13
14
15
16
17
18
19
20
```

```
    if pHead1.val<=pHead2.val:
        pMergeHead = pHead1
        #pHead1=pHead1.next
        pMergeHead.next = self.Merge(pHead1.next,pHead2)
    elif pHead1.val>pHead2.val:
        pMergeHead = pHead2
        #pHead2=pHead2.next
        pMergeHead.next = self.Merge(pHead2.next, pHead1)
    return pMergeHead
```

17. 树的子结构（树）

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)

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 树的题目，很多都用到递归
9     # 回头重点看的一道题目
10    ...
11    思路：对A树DFS，如果B的根节点与A中某个节点值相同，那么以B为树
12    根进行DFS，判断即可
13    时间复杂度为O(n * m)
14    ...
15    def HasSubtree(self, pRoot1, pRoot2):
16        # write code here
17        result = False
18        if pRoot1 != None and pRoot2 != None:
19            if pRoot1.val == pRoot2.val:
20                result = self.DoesTree1haveTree2(pRoot1, pRoot2)
21            if not result:
22                result = self.HasSubtree(pRoot1.left, pRoot2)
23            if not result:
24                result = self.HasSubtree(pRoot1.right, pRoot2)
25        return result
26    # 用于递归判断树的每个节点是否相同
27    # 需要注意的地方是：前两个if语句不可以颠倒顺序
28    # 如果颠倒顺序，会先判断pRoot1是否为None，其实这个时候pRoot2的结点
29    # 已经遍历完成确定相等了，但是返回了False，判断错误
30    def DoesTree1haveTree2(self, pRoot1, pRoot2):
31        if pRoot2 == None:
32            return True
33        if pRoot1 == None:
34            return False
35        if pRoot1.val != pRoot2.val:
36            return False
37        return self.DoesTree1haveTree2(pRoot1.left, pRoot2.left) and self.DoesTree1haveT
ree2(pRoot1.right, pRoot2.right)
```

18. 二叉树的镜像（树）

题目描述

操作给定的二叉树，将其变换为源二叉树的镜像。

输入描述:

二叉树的镜像定义：源二叉树

```
    8
   /   \
  6   10
 / \   / \
5  7  9  11
```

镜像二叉树

```
    8
   /   \
  10   6
 / \   / \
11  9  7  5
```

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 返回镜像树的根节点
9     # 没有太看懂，重新看一遍！！！
10    # 回头重点看！
11    def Mirror(self, root):
12        # write code here
13        if root == None:
14            return
15        if root.left == None and root.right == None:
16            return root
17        temp = root.left
18        root.left = root.right
19        root.right = temp
20        # 递归调用自己的函数
21        self.Mirror(root.left)
22        self.Mirror(root.right)
```

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 4×4 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.

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     # matrix类型为二维列表，需要返回列表
4     # 重新看一遍，理解一遍！！！
5     """
6     直接顺时针去数数字就可以了
7     """
8     def printMatrix(self, matrix):
9         res = []
10        while matrix:
11            res += matrix.pop(0)
12            # pop默认是堆栈方式，pop(0)就是队列方式
13            if matrix and matrix[0]:
14                for row in matrix:
15                    res.append(row.pop(0))
16            if matrix:
17                res += matrix.pop()[:-1]
18            if matrix and matrix[0]:
19                for row in matrix[:-1]:
20                    res.append(row.pop(0))
21        return res
22
```

20. 包含min函数的栈（堆栈）

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 $O(1)$ ）。

```
1 # -*- coding:utf-8 -*-
2 ...
3 用例：
4 ["PSH3","MIN","PSH4","MIN","PSH2","MIN","PSH3","MIN","POP","MIN","POP","MIN","POP","MIN"
5 , "PSH0","MIN"]
6 对应输出应该为：
7
8 3,3,2,2,2,3,3,0
9
10 ...
11 # 这道题目意思不清楚，看一下原来的书上怎么写的
12 class Solution:
13     def __init__(self):
14         self.stack = []
15         self.minStack = [] # 保存最小元素的堆栈
16         # 重新思考，理解一下！！！
17         # 建立一个辅助栈，每次都把最小值压入辅助栈，这样辅助栈的栈顶一直是最小元素。
18         # 当数据栈中，最小值被弹出时，同时弹出辅助栈的栈顶元素。
19     def push(self, node):
20         # write code here
21         # self表示创建的类实例本身
22         # 加了self可以实现跨方法调用
23         # 调用self的时候不必为这个参数赋值
24         self.stack.append(node)
25         if self.minStack == [] or node < self.min():
26             self.minStack.append(node)
27         else:
28             temp = self.min()
29             self.minStack.append(temp)
30
31     def pop(self):
32         # write code here
33         # self.stack.pop(node)
34         if self.stack == None or self.minStack == None:
35             return None
36         self.minStack.pop()
37         self.stack.pop()
38     def top(self):
39         # write code here
40         return self.stack[-1]
41     def min(self):
42         # write code here
43         return self.minStack[-1]
```

21. 栈的压入、弹出序列 (堆栈)

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）

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def IsPopOrder(self, pushV, popV):
4         # write code here
5         # 重新总结与思考！！！
6         # 这种解题的思想特别好
7         if pushV == [] or popV == []:
8             return None
9         # python 中用堆栈都是用 []
10        stack = []
11        for i in pushV:
12            stack.append(i)
13            while len(stack) and stack[-1]==popV[0]:
14                stack.pop()#pop 默认是最最后一个
15                popV.pop(0)
16        if len(stack):
17            return False
18        else:
19            return True
```

22. 从上往下打印二叉树（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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 返回从上到下每个节点值列表，例：[1,2,3]
9     # 重新思考！！！非常精彩！
10    def PrintFromTopToBottom(self, root):
11        # write code here
12        #if root.left == None and root.right == None:
13        #    return root
14        # 这个为什么不行！！
15        if root == None:
16            return []
17        queue = []
18        result = []
19        queue.append(root) # list可以用在树结构里
20        while len(queue) > 0:
21            currentRoot = queue.pop(0)
22            result.append(currentRoot.val)
23            if currentRoot.left:
24                queue.append(currentRoot.left)
25            if currentRoot.right:
26                queue.append(currentRoot.right)
27
28        return result
29
30
31
```

23. 二叉搜索树的后序遍历序列（树）

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def VerifySquenceOfBST(self, sequence):
4         # write code here
5         if len(sequence) == 0:
6             return False
7
8         root = sequence[-1]
9
10        # 在二叉搜索中左子树的结点小于跟结点
11        i = 0
12        for node in sequence[:-1]:
13            if node > root:
14                break
15            i += 1
16
17        # 在二叉搜索中右子树的结点小于跟结点
18        for node in sequence[i:-1]:
19            if node < root:
20                return False
21
22        # 判断左子树是不是二叉搜索树
23        left = True
24        if i > 1:
25            left = self.VerifySquenceOfBST(sequence[:i])
26        right = True
27        if i < len(sequence) - 2 and left:
28            right = self.VerifySquenceOfBST(sequence[i+1:-1])
29        return left and right
30
31
32
```

24. 二叉树中和为某一值的序列（树）

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)

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 返回二维列表，内部每个列表表示找到的路径
9     # 没看懂，回头研究！！
10    def FindPath(self, root, expectNumber):
11        def subFindPath(root):
12            if root:
13                b.append(root.val)
14                if not root.right and not root.left and sum(b) == expectNumber:
15                    a.append(b[:])
16                else:
17                    subFindPath(root.left), subFindPath(root.right)
18                b.pop()
19        a, b = [], []
20        subFindPath(root)
21        return a
22
23
```

25. 复杂链表的复制（链表）

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）

具体分为三步：

(1) 在旧链表中创建新链表，此时不处理新链表的兄弟结点

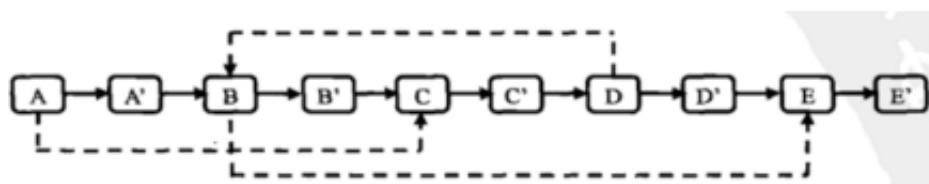


图 4.9 复制复杂链表的第一步

<http://blog.csdn.net/insistGoGo>

(2) 根据旧链表的兄弟结点，初始化新链表的兄弟结点

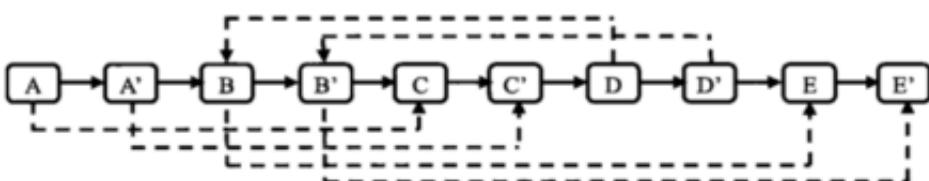


图 4.10 复制复杂链表的第二步

<http://blog.csdn.net/insistGoGo>

(3) 从旧链表中拆分得到新链表

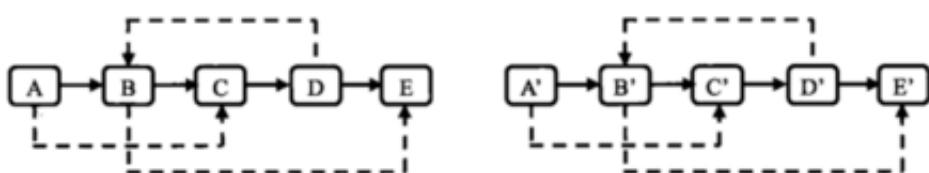


图 4.11 复制复杂链表的第三步

<http://blog.csdn.net/insistGoGo>

```
1 # -*- coding:utf-8 -*-
2 class RandomListNode:
3     def __init__(self, x):
4         self.label = x
5         self.next = None
6         self.random = None
7 class Solution:
8     # 返回 RandomListNode
9     # 重新回头看题目！
10    def Clone(self, pHead):
11        if not pHead:
```

```
12     return None
13
14
15 # first step, N' to N next
16 while dummy:
17     dummysnext = dummy.next
18     copynode = RandomListNode(dummy.label)
19     copynode.next = dummysnext
20     dummy.next = copynode
21     dummy = dummysnext
22
23     dummy = pHead
24
25 # second step, random' to random'
26 while dummy:
27     dummyrandom = dummy.random
28     copynode = dummy.next
29     if dummyrandom:
30         copynode.random = dummyrandom.next
31     dummy = copynode.next
32
33 # third step, split linked list
34 dummy = pHead
35 copyHead = pHead.next
36 while dummy:
37     copyNode = dummy.next
38     dummysnext = copyNode.next
39     dummy.next = dummysnext
40     if dummysnext:
41         copyNode.next = dummysnext.next
42     else:
43         copyNode.next = None
44     dummy = dummysnext
45
46
47 return copyHead
```

26. 二叉搜索树与双向链表（树、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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 基本树的问题都可以分为递归方法和非递归的方法
9     def Convert(self, pRootOfTree):
10        # write code here
11        # 非递归的方式
12        if not pRootOfTree:
13            return None
14
15        p = pRootOfTree
16
17        stack = []
18        resStack = []
19
20        while p or stack:
21            if p:
22                stack.append(p)
23                p = p.left
24            else:
25                node = stack.pop()
26                resStack.append(node)
27                p = node.right
28
29        resP = resStack[0]
30        while resStack:
31            top = resStack.pop(0)
32            if resStack:
33                top.right = resStack[0]
34                resStack[0].left = top
35        return resP
36
37
```

```
1 # 递归的方法：
2 class Solution:
3     def Convert(self, root):
4         if not root:
5             return None
6         if not root.left and not root.right:
7             return root
8
9         # 将左子树构建为双链表，返回链表头
```

```
10    left = self.Convert(root.left)
11    p = left
12
13    # 定位至左子树的最右的一个结点
14    while left and p.right:
15        p = p.right
16
17    # 如果左子树不为空，将当前root加到左子树链表
18    if left:
19        p.right = root
20        root.left = p
21
22    # 将右子树构造为双链表，返回链表头
23    right = self.Convert(root.right)
24    # 如果右子树不为空，将该链表追加到root结点之后
25    if right:
26        right.left = root
27        root.right = right
28
29    return left if left else root
30
31
32
```

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。

输入描述:

输入一个字符串,长度不超过9(可能有字符重复),字符只包括大小写字母。

```
1 # -*- coding:utf-8 -*-
2 from itertools import permutations
3 ...
4 # 这个一开始自己写的, 没有通过
5 class Solution:
6     def Permutation(self, ss):
7         # write code here
8         #ll = []
9
10    if not ss:
11        return []
12    # 这个意思是ss是空的意思吗? ?
13    if len(ss)==1:
14        return list(ss)
15    #charList = list(set(''.join(i) for i in permutations(sorted(ss),int(len(ss)))))
16    charList = list(set(''.join(i) for i in permutations(sorted(ss),int(len(ss))))))
17    return charList
18 ...
19
20 import itertools
21 class Solution:
22     def Permutation(self, ss):
23         # write code here
24         result=[]
25         if not ss:
26             return []
27         else:
28             res=itertools.permutations(ss)
29             for i in res:
30                 if ''.join(i) not in result:
31                     result.append(''.join(i))
32     return result
33
34
35
1
2 class Solution:
3     def Permutation(self, ss):
4         # write code here
5         res = []
6         if len(ss) < 2:
7             return res
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```

```
7      return ss.split()
8  for i in range(len(ss)):
9      for n in map(lambda x: x+ ss[i], self.Permutation(ss[:i]+ss[i+1:])):
10         if n not in res:
11             res.append(n)
12     return sorted(res)
13
```

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。

```
1 # -*- coding:utf-8 -*-
2 from collections import Counter
3 #import collections
4 class Solution:
5     # 典型的 Python 字典计数类型题目
6     def MoreThanHalfNum_Solution(self, numbers):
7         # write code here
8         # 还是用字典的方法去做，可以节约时间复杂度
9         dict = {}
10        for i in numbers:
11            if not dict.has_key(i):
12                dict[i] = 1
13            else:
14                dict[i] = dict[i] + 1
15            if dict[i] > len(numbers)/2:
16                return i
17        return 0
18
```

```
1
2 #方法二，用 collection Counter 函数去做
3 def MoreThanHalfNum_Solution(self, numbers):
4     # write code here
5     from collections import Counter
6     count = Counter(numbers).most_common()
7     if count[0][1] > len(numbers)/2.0:
8         return count[0][0]
9     return 0
10
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def GetLeastNumbers_Solution(self, tinput, k):
4         # write code here
5         #len1 = len(tinput)
6         # 直接用一下排序算法 sort()
7         tinput.sort()
8         result=[]
9         if k > len(tinput):
10             return []
11         for i in range(k):
12             result.append(tinput[i])
13         return result
```

30. 连续子数组的最大和

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)

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def FindGreatestSumOfSubArray(self, array):
4         # write code here
5         # 最大连续子序列的和
6         # 可以用动态规划去做
7         if not array:
8             return False
9         #count = 0
10        cur_sum = 0 # 一个用来存放当前的和
11        max_sum = array[0] # 一个用来存储最大和
12
13        for i in range(len(array)):
14            if cur_sum <= 0:
15                cur_sum = array[i]
16            else:
17                cur_sum += array[i]
18
19            if cur_sum > max_sum:
20                max_sum = cur_sum
21
22        return max_sum
23        ...
24
25        for i in range(len(array)):
26            if array[i]>0:
27                sum1 += array[i]
28                sum1 = max(sum1,maxNum)
29            else:
30                #for j in range
31                maxNum = sum1
32                sum1+=array[i]
33                if sum1 <=0:
34                    sum1 = 0
35        array1 = sorted(array)
36        for j in range(len(array)):
37            if array[i]<0:
38                count++
39            if count == len(array):
40                return array1[0]
41            ...
42
43
```


31. 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)。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def NumberOf1Between1AndN_Solution(self, n):
4         # write code here
5         # 先判断n是几位数,
6         count = 0
7         digitL=len(str(n))
8         if n < 1:
9             return 0
10        # 先将数字转成字符串, 然后用len来计算
11        # 直接用一个循环, 自己对自己处理就好了呀
12        for i in range(1,n+1):
13            while i:
14                if i%10==1:
15                    count+=1
16                i=i/10
17        return count
18
19
20
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def PrintMinNumber(self, numbers):
4         # write code here
5         #digitL = len(str(numbers))
6         # numbers 是一个正整数数组
7         # 字符串类型相互加减也可以互相比较
8         if not numbers:
9             return ''
10        str_num = [str(m) for m in numbers]
11        # 把每个正整数转成字符串
12        for i in range(len(numbers)-1):
13            for j in range(i+1, len(numbers)):
14                if str_num[i]+str_num[j]>str_num[j]+str_num[i]:
15                    str_num[i],str_num[j] = str_num[j],str_num[i]
16        #return str_num
17        return ''.join(str_num)
18    # join 把list类型转换成字符串类型
19
20    ...
21    用例：
22 [3,5,1,4,2]
23
24 对应输出应该为：
25
26 "12345"
27
28 你的输出为：
29
30 "[['1', '2', '3', '4', '5']]"
31      ...
32
33
34
```

33. 丑数 数组判断

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def GetUglyNumber_Solution(self, index):
4         # write code here
5         # 重新看一下，反思一下！
6         if index < 1:
7             return 0
8         if 1<=index < 7:
9             return index
10        res=
11        [2**i*3**j*5**k  for i in range(30)  for j in range(20)    for k in range(15)]
12        res.sort()
13        return res[index-1] if index else 0
14    """
15    return sorted([2**i*3**j*5**k  for i in range(30)  for j in range(20)    for k in range(15)])[index-1] if index else 0
16    """
17
18
```

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) .

```
1 # -*- coding:utf-8 -*-
2 from collections import Counter
3 class Solution:
4     def FirstNotRepeatingChar(self, s):
5         # write code here
6         # 判断输入条件
7         if len(s)<=0 or len(s)>10000:
8             return -1
9         # count 用于统计字符串中某个字符的出现个数
10        # index 为计算字符串中某个字符的位置
11        for i in s:
12            if s.count(i)==1:
13                return s.index(i)
14                break
15
16    ...
17 Python: 开始想用字典,但是字典是无序的,所以无法找到第一个出现次数为1
18 的位置
19 我用的是python内置函数完成的,没有提现到算法思想,
20 看到书中是建一个哈希表,以字母的ascii为下标,出现次数为值,然后再遍
21 历一次找到第一个值为1的位置,用python编写如下:
22     #建立哈希表,字符长度为8的数据类型,共有256种可能,于是创建一个长
23 度为256的列表
24     ls=[0]*256
25     #遍历字符串,下标为ASCII值,值为次数
26     for i in s:
27         ls[ord(i)]+=1
28     #遍历列表,找到出现次数为1的字符并输出位置
29     for j in s:
30         if ls[ord(j)]==1:
31             return s.index(j)
32             break
33
34 ...
```

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 $P \% 1000000007$

输入描述:

题目保证输入的数组中没有相同的数字

数据范围：

对于%50的数据，size<=10^4

对于%75的数据，size<=10^5

对于%100的数据，size<=2*10^5

示例1

输入

复制

1-2-3-4-5-6-7-0

输出

复制

7

```
1 # -*- coding:utf-8 -*-
2 count = 0
3 class Solution:
4     # re 这道题目要再研究下
5     def InversePairs(self, data):
6         global count
7         # global count 可以节省时间吗？？
8         def MergeSort(lists):
9             # 哪函数定义在函数里面和分开来有什么区别呢？？
10            global count
11            if len(lists) <= 1:
12                return lists
13            num = int(len(lists)/2)
14            left = MergeSort(lists[:num])
15            right = MergeSort(lists[num:])
16            r, l=0, 0
17            result=[]
18            while l<len(left) and r<len(right):
19                if left[l] < right[r]:
20                    result.append(left[l])
21                    l += 1
22                else:
23                    result.append(right[r])
24                    r += 1
25            return result
26
27        count = 0
28        MergeSort(data)
29        print(count)
```

```
22
23     else:
24         result.append(right[r])
25         r += 1
26         count += len(left)-l
27     result += right[r:]
28     result += left[l:]
29     return result
30 MergeSort(data)
31     return count%1000000007
```

收藏

笔记 | 纠错

36. 两个链表中的第一个公共节点（链表）

题目描述

输入两个链表，找出它们的第一个公共结点。

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 class Solution:
7     """
8     找出2个链表的长度，然后让长的先走两个链表的长度差，然后再一起走
9     (因为2个链表用公共的尾部)
10    """
11
12    # 这个方法为什么不对呢？不是按照那个思路来的吗？
13    def FindFirstCommonNode(self, pHead1, pHead2):
14        len1 = len2 = 0
15        pa = pHead1
16        pb = pHead2
17        while pa!=None:
18            len1 += 1
19            pa.next = pa
20        while pb!=None:
21            len2 += 1
22            pb.next = pb
23        if len1 > len2:
24            while len1 - len2:
25                pHead1 = pHead1.next
26                len1 -= 1
27        elif len2 > len1:
28            while len2 - len1:
29                pHead2 = pHead2.next
30                len2 -= 1
31        while pHead1 and pHead2:
32            if pHead1 == pHead2:
33                return pHead1
34            pHead1 = pHead1.next
35            pHead2 = pHead2.next
36        return None
37
38
```

```
1
2     #方法二：
3 class Solution:
4     def FindFirstCommonNode(self, pHead1, pHead2):
5         if not pHead1 or not pHead2:
6             return None
7
8         stack1 = []
9         stack2 = []
10
```

```
11 while pHead1:  
12     stack1.append(pHead1)  
13     pHead1 = pHead1.next  
14  
15 while pHead2:  
16     stack2.append(pHead2)  
17     pHead2 = pHead2.next  
18  
19 first = None  
20 while stack1 and stack2:  
21     top1 = stack1.pop()  
22     top2 = stack2.pop()  
23     if top1 is top2:  
24         first = top1  
25     else:  
26         break  
27 return first  
28  
29  
30 ...  
31 class Solution:  
32     def FindFirstCommonNode(self, pHead1, pHead2):  
33         # write code here  
34         # 链表的长度不能直接得出，对吧？  
35         # 这样写时间复杂度太大了也不行！  
36         prHead1 = pHead1  
37         prHead2 = pHead2  
38         while prHead1 != None:  
39             while prHead2 !=None:  
40                 if prHead1.val == prHead2.val:  
41                     return prHead1.val  
42                 else:  
43                     continue  
44             #continue  
45         return None  
46 ...
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def GetNumber0fK(self, data, k):
4         # write code here
5         # 求k一共有多少次，data是排序好的数组
6         # 成功了！
7         data1 = list(data)
8         count=0
9         for i in range(len(data)):
10             if data[i] == k :
11                 count=count+1
12             else :
13                 continue
14         return count
```

38. 二叉树的深度（树）

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 还需要理解一下递归的用法！！！
9     def TreeDepth(self, pRoot):
10        # write code here
11        if pRoot == None:
12            return 0
13        count = max(self.TreeDepth(pRoot.left), self.TreeDepth(pRoot.right))+1
14        return count
15
16
17
```

39. 平衡二叉树（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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     """
9         如果二叉树的每个节点的左子树和右子树的深度不大于1，它就是平衡
10        二叉树。
11        先写一个求深度的函数，再对每一个节点判断，看该节点的左子树的深
12        度和右子树的深度的差是否大于1
13        """
14        # write code here
15        # 平衡二叉树是一棵空树或它的左右两个子树的高度差的绝对值
16        不超过1，
17        # 并且左右两个子树都是一棵平衡二叉树。
18    def IsBalanced_Solution(self, root):
19        if not root:
20            return True
21        if abs(self.maxDepth(root.left) - self.maxDepth(root.right)) > 1:
22            return False
23        return self.IsBalanced_Solution(root.left) and self.IsBalanced_Solution(root.rig
24        ht)
25
26        def maxDepth(self, root):
27            if not root: return 0
28            return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1
```

```
1
2
3 #方法二：自下而上，时间复杂度O(N)
4 class Solution:
5     def IsBalanced_Solution(self, p):
6         return self.dfs(p) != -1
7     def dfs(self, p):
8         if p is None:
9             return 0
10        left = self.dfs(p.left)
11        if left == -1:
12            return -1
13        right = self.dfs(p.right)
14        if right == -1:
15            return -1
16        if abs(left - right) > 1:
17            return -1
18        return max(left, right) + 1
```


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。

示例1

输入

复制

输出

复制

```
1 # -*- coding:utf-8 -*-
2 from collections import Counter
3 class Solution:
4     # 返回 [a,b] 其中ab是出现一次的两个数字
5     def FindNumsAppearOnce(self, array):
6         # write code here
7         #mylist = list(array)
8         #return Counter(mylist).keys() if Counter(mylist).values()==1
9         #需要更熟悉一些lambda函数的用法
10        return list(map(lambda c: c[0],Counter(array).most_common()[-2:]))
11        #为什么最后两个数是这种写法啊？？？为什么不 -2 -1
12
13
14
15
```

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?
Good Luck!

输出描述:

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

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def FindContinuousSequence(self, tsum):
4         # write code here
5         # 找出所有和为S的连续正数序列
6         res=[]
7         for i in range(1,tsum//2+1):
8             sumRes=i
9             for j in range(i+1,tsum//2+2):
10                 sumRes+=j
11                 if sumRes==tsum:
12                     res.append(list(range(i,j+1)))
13                     break
14                 elif sumRes>tsum:
15                     break
16         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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def FindNumbersWithSum(self, array, tsum):
4         # write code here
5         #for i in range(len(array)):
6         i = 0
7         j = len(array)-1
8         result = []
9         while i < j:
10             if array[i]+array[j]<tsum:
11                 i+=1
12             elif array[i]+array[j]>tsum:
13                 j-=1
14             else:
15                 #return [i,j]
16                 result.append(array[i])
17                 result.append(array[j])
18                 return result
19                 #return [array[i],array[j]]
20
21         return []
```

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！

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def LeftRotateString(self, s, n):
4         # write code here
5         # python字符串操作的题目，是全部转化为list做还是什么？---对的
6         ,然后再用join转为字符串
7         return s[n:]+s[:n] # 方法一，太秀了

1 # 方法二：常规方法
2     if not s:
3         return ""
4     l=list(s)
5     m=len(l)
6     a=[]
7     for i in range(m):
8         if i+n<m:
9             a.append(l[i+n])
10    for j in range(n):
11        a.append(l[j])
12    return ''.join(a)
```

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？

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def ReverseSentence(self, s):
4         # write code here
5         # 根据空格的位置做分割
6         # 用空格做分割，把这些单词装到矩阵里面去
7         return " ".join(s.split(" ")[::-1])# 方法一，最简单直接的方法
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
994
995
996
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1095
1096
1096
1097
1098
1098
1099
1099
1100
1100
1101
1102
1103
1104
1105
1106
1107
1108
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1137
1138
1138
1139
1139
1140
1141
1142
1143
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1172
1173
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1202
1203
1204
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1212
1213
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1262
1263
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1302
1303
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1362
1363
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1402
1403
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1462
1463
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1502
1503
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1562
1563
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1602
1603
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1662
1663
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1702
1703
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1735
1736
1736
1737
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1745
1746
1746
1747
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1755
1756
1756
1757
1757
1758
1758
1759
1759
1760
1761
1762
1763
1764
1765
1765
1766
1766
1767
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1775
1776
1776
1777
1777
1778
1778
1779
1779
1780
1781
1782
1783
1784
1785
1785
1786
1786
1787
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1795
1796
1796
1797
1797
1798
1798
1799
1799
1800
1800
1801
1802
1803
1804
1805
1805
1806
1806
1807
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1815
1816
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1825
1826
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1835
1836
1836
1837
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1845
1846
1846
1847
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1855
1856
1856
1857
1857
1858
1858
1859
1859
1860
1861
1862
1863
1864
1865
1865
1866
1866
1867
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1875
1876
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1885
1886
1886
1887
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1895
1896
1896
1897
1897
1898
1898
1899
1899
1900
1900
1901
1902
1903
1904
1905
1905
1906
1906
1907
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1915
1916
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1925
1926
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1935
1936
1936
1937
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1945
1946
1946
1947
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1955
1956
1956
1957
1957
1958
1958
1959
1959
1960
1961
1962
1963
1964
1965
1965
1966
1966
1967
1967
1968
1968
1969
1969
1970
1971
1972
1973
1974
1975
1975
1976
1976
1977
1977
1978
1978
1979
1979
1980
1981
1982
1983
1984
1985
1985
1986
1986
1987
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1994
1995
1995
1996
1996
1997
1997
1998
1998
1999
1999
2000
2000
2001
2002
2003
2004
2005
2005
2006
2006
2007
2007
2008
2008
2009
2009
2010
2011
2012
2013
2014
2015
2015
2016
2016
2017
2017
2018
2018
2019
2019
2020
2021
2022
2023
2024
2025
2025
2026
2026
2027
2027
2028
2028
2029
2029
2030
2031
2032
2033
2034
2035
2035
2036
2036
2037
2037
2038
2038
2039
2039
2040
2041
2042
2043
2044
2045
2045
2046
2046
2047
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2055
2056
2056
2057
2057
2058
2058
2059
2059
2060
2061
2062
2063
2064
2065
2065
2066
2066
2067
2067
2068
2068
2069
2069
2070
2071
2072
2073
2074
2075
2075
2076
2076
2077
2077
2078
2078
2079
2079
2080
2081
2082
2083
2084
2085
2085
2086
2086
2087
2087
2088
2088
2089
2089
2090
2091
2092
2093
2094
2095
2095
2096
2096
2
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def IsContinuous(self, numbers):
4         # write code here
5         ...
6 1、如果输入为空，返回 false
7 2、除了王的任何某个特定数值的牌出现两张或者更多，那么一定凑不齐
8 顺子。
9 思路，先统计王的数量，再把牌排序，如果后面一个数比前面一个数大
10 于1以上，那么中间的差值就必须用王来补了。看王的数量够不够，如果
11 够就返回 true，否则返回 false。
12 ...
13     if not numbers:
14         return False
15     numbers.sort()
16     zeroNum = numbers.count(0)
17     for i, v in enumerate(numbers[:-1]):
18         if v != 0:
19             if numbers[i+1]==v: return False
20             zeroNum = zeroNum - (numbers[i + 1] - v) + 1
21             if zeroNum < 0:
22                 return False
23     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)

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def LastRemaining_Solution(self, n, m):
4         # write code here
5         # 对小朋友进行编号, 从0到n-1
6         # 典型题, 约瑟夫问题, 求递推公式, 每轮的序列中最后出序列的
数都是同一个
7
8         # 方法一, 非递归方法
9         if n < 1:
10             return -1
11
12         con = range(n)
13
14         final = -1
15         start = 0
16         while con:
17             k = (start + m - 1) % n
18             final = con.pop(k)
19             n -= 1
20             start = k
21
22         return final
23
24
```

```
1
2     # 递归问题特别耗内存, 这个问题到一定程度就超出界限了, 所以
谨慎使用递归方法
3     if n < 1:
4         return -1
5     if n == 1:
6         return 0
7     return (self.LastRemaining_Solution(n-1,m)+m)%n
8
9
```

47. 求 $1+2+3+\dots+n$

题目描述

求 $1+2+3+\dots+n$ ，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）。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     #想 想 看 有 没 有 其 他 的 解 法
4     def Sum_Solution(self, n):
5         # write code here
6         # 不 能 使 用 乘 除 法
7         return sum(range(1,n+1))#这 个 是 方 法 一
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
584
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def Add(self, num1, num2):
4         # write code here
5         # 这个是只能用位运算了吗？
6         #return sum([num1,num2])#方法一
7         #方法二，Python对位操作支持不是很友好，主要原因还是因为
8         #python没有无符号又移操作，所以需要越界检查一波～
9         #其他思路和大家是一样的，加法是异或，进位是与<<1
10        while(num2):
11            num1, num2 = (num1^num2) & 0xFFFFFFFF, ((num1&num2)<<1) & 0xFFFFFFFF
12            return num1 if num1<=0x7FFFFFFF else ~(num1^0xFFFFFFFF)
13
14        ...
15
16    1. 两个数异或：相当于每一位相加，而不考虑进位；
17    2. 两个数相与，并左移一位：相当于求得进位；
18    3. 将上述两步的结果相加
19    public int Add(int num1,int num2) {
20        while( num2!=0 ){
21            int sum = num1 ^ num2;
22            int cararry = (num1 & num2) << 1;
23            num1 = sum;
24            num2 = cararry;
25        }
26        return num1;
27    }
28    ...
29
```

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。

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

示例1

输入

复制

```
+2147483647  
1a33
```

输出

复制

```
2147483647  
0
```

```
1 # -*- coding:utf-8 -*-  
2 class Solution:  
3     def StrToInt(self, s):  
4         # write code here  
5         # re 参考了答案  
6         numlist=['0','1','2','3','4','5','6','7','8','9','+','-']  
7         sum=0  
8         label=1# 正负数标记  
9         if s=='':  
10             return 0  
11         for string in s:  
12             if string in numlist:# 如果是合法字符  
13                 if string=='+'.  
14                     label=1  
15                     continue  
16                 elif string=='-':  
17                     label=-1
```

```
18         continue
19     else:
20         sum=sum*10+numlist.index(string)
21         # 把 string 类型 转 换 为 int 类
22     if string not in numlist:#非 合 法 字 符
23         sum=0
24     break#跳 出 循 环
25 return sum*label
```

笔记

收藏

纠错

50. 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。

示例1

输入

复制

输出

复制

```
1 # -*- coding:utf-8 -*-
2 from collections import Counter
3 class Solution:
4     # 这里要特别注意~找到任意重复的一个值并赋值到duplication[0]
5     # 函数返回True/False
6     def duplicate(self, numbers, duplication):
7         # write code here
8         cur = 0
9         while cur < len(numbers):
10             if numbers[cur] == cur:
11                 cur += 1
12                 continue
13
14             if numbers[cur] == numbers[numbers[cur]]:
15                 duplication[0] = numbers[cur]
16                 return True
17
18             # 注意这里不能直接multiple assignment
19             temp = numbers[cur]
20             numbers[cur] = numbers[numbers[cur]]
21             numbers[temp] = temp
22         return False
23
24     ...
25     # 时间复杂度太大了，是O(N2)了
26     def duplicate(self, numbers, duplication):
27         # write code here
28         #duplication = True
29         Myarray = []
30         for i in range(0, len(numbers)-1):
31             for j in range(i+1, len(numbers)):
32                 if numbers[j]==numbers[i]:
33                     duplication[0] = numbers[i]
```

```
34  
35  
36  
37  
38  
39
```

```
    return True  
return False  
'''
```

51. 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。

```
1 # -*- coding:utf-8 -*-
2 import copy
3 class Solution:
4     def multiply(self, A):
5         # write code here
6         # 直接先算A数组的从头到尾的数值
7         # 最后得出sum的乘积的值
8         # 这是时间复杂度小一点的方法一
9         B=[1]*len(A)
10        for i in range(len(A)):
11            C=copy.copy(A)
12            C.pop(i)
13            for j in range(len(C)):
14                B[i]*=C[j]
15        return B
16        ...
17        # 方法二：很容易想到，但是这种方法时间复杂度大了一点
18        B = []
19        for i in range(len(A)):
20            sum = 1
21            for j in range(0,i):
22                sum = sum * A[j]
23            for j in range(i+1,len(A)):
24                sum = sum * A[j]
25            B.append(sum)
26        return B
27        ...
```

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

```
1 # -*- coding:utf-8 -*-
2 import re
3 class Solution:
4     # 好 难 啊， 研究 下 正 则 表 达 式
5     # s, pattern 都 是 字 符 串
6     def match(self, s, pattern):
7         def fullmatch(regex, string, flags=0):
8             return re.match("(?:" + regex + r")\Z", string, flags=flags)
9         return True if fullmatch(pattern,s) else False
10
11
12 ...
13 # -*- coding:utf-8 -*-
14
15 题 目： 请 实 现 一 个 函 数 用 来 匹 配 包 括 ‘.’ 和 ‘*’ 的 正 则 表 达 式 。
16 模 式 中 的 字 符 ‘.’ 表 示 任 意 一 个 字 符 （ 不 包 括 空 字 符 ！ ） ， 而 ‘*’ 表 示 它 前
17 面 的 字 符 可 以 出 现 任 意 次 （ 包 含 0 次 ） 。
18 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"
19 与 模 式 "a.a" 和 "ab*ac*a" 匹 配 ， 但 是 与 "aa.a" 和 "ab*a" 均 不 匹 配
20
21 class Solution:
22     # s, pattern 都 是 字 符 串
23     def match(self, s, pattern):
24         # 如 果 s 与 pattern 都 为 空 ， 则 True
25         if len(s) == 0 and len(pattern) == 0:
26             return True
27         # 如 果 s 不 为 空 ， 而 pattern 为 空 ， 则 False
28         elif len(s) != 0 and len(pattern) == 0:
29             return False
30         # 如 果 s 为 空 ， 而 pattern 不 为 空 ， 则 需 要 判 断
31         elif len(s) == 0 and len(pattern) != 0:
32             # pattern 中 的 第 二 个 字 符 为 * ， 则 pattern 后 移 两 位 继 续 比 较
33             if len(pattern) > 1 and pattern[1] == '*':
34                 return self.match(s, pattern[2:])
35             else:
36                 return False
37         # s 与 pattern 都 不 为 空 的 情 况
38         else:
39             # pattern 的 第 二 个 字 符 为 * 的 情 况
40             if len(pattern) > 1 and pattern[1] == '*':
41                 # s 与 pattern 的 第 一 个 元 素 不 同 ， 则 s 不 变 ， pattern 后 移 两 位 ，
42                 # 相 当 于 pattern 前 两 位 当 成 空
43                 if s[0] != pattern[0] and pattern[0] != '.':
44                     return self.match(s, pattern[2:])
45                 else:
46                     # 如 果 s[0] 与 pattern[0] 相 同 ， 且 pattern[1] 为 * ， 这 个 时 候 有
```

```
44 三种情况
45 匹配后面的          # pattern后移2个，s不变；相当于把pattern前两位当成空
46 配                  # pattern后移2个，s后移1个；相当于pattern前两位与s[0]匹配
47 多位进行匹配，      # pattern不变，s后移1个；相当于pattern前两位，与s中的
48 因为*可以匹配多位
49 return self.match(s, pattern[2:]) or self.match(s[1:], pattern[2:])
50 or self.match(s[1:], pattern)
51 # pattern第二个字符不为*的情况
52 else:
53     if s[0] == pattern[0] or pattern[0] == '.':
54         return self.match(s[1:], pattern[1:])
55     else:
56         return False
57 ...
58 ...
59 解这题需要把题意仔细研究清楚，反正我试了好多次才明白的。
60 首先，考虑特殊情况：
61 1>两个字符串都为空，返回true
62 2>当第一个字符串不空，而第二个字符串空了，返回false（因为这样，就无法
63 匹配成功了，而如果第一个字符串空了，第二个字符串非空，还是可能匹配成
64 功的，比如第二个字符串是"a*a*a*a*"，由于'*'之前的元素可能出现0次，
65 所以有可能匹配成功）
66 之后就开始匹配第一个字符，这里有两种可能：匹配成功或匹配失败
67 。但考虑到pattern
68 下一个字符可能是'*'，这里我们分两种情况讨论：pattern下一个字符
69 为'*'或
70 不为'*'：
71 1>pattern下一个字符不为'*'：这种情况比较简单，直接匹配当前
72 字符。如果
73 匹配成功，继续匹配下一个；如果匹配失败，直接返回false。
74 注意这里的
75 "匹配成功"，除了两个字符相同的情况外，还有一种情况，就是
76 pattern的
77 当前字符为'.'，同时str的当前字符不为'\0'。
78 2>pattern下一个字符为'*'时，稍微复杂一些，因为'*'可以代表0个
79 或多个。
80 这里把这些情况都考虑到：
81 a>当'*'匹配0个字符时，str当前字符不变，pattern当前字符后
82 移两位，           跳过这个'*'符号；
83 前字符             b>当'*'匹配1个或多个时，str当前字符移向下一个，pattern当
84 不变。（这里匹配1个或多个可以看成一种情况，因为：当匹配一个时，由于str移到了下一个字符，而pattern字符不变，就回到了上
85 边的情况；a>当匹配多于一个字符时，相当于从str的下一个字符继续开始匹配）
86 之后再写代码就很简单了。
87 ...
88 ...
89 ...
90 ...
```


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。

```
44     elif s[i] == '.':
45         # 小数点不能出现两次；而且如果已经出现过e了，那么就不能再出现小数点，因为e后面只能是整数
46         if has_point or has_e:
47             return False
48         # 如果是第一次出现小数点，如果前面出现过e，那么还是不能出现小数点
49         else:
50             has_point = True
51             if i > 0 and (s[i-1] == 'e' or s[i-1] == 'E'):
52                 return False
53             else:
54                 # 其他字符必须是'0'到'9'之间的
55                 if s[i] < '0' or s[i] > '9':
56                     return False
57     return True
58
59
60
```

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"。

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     # 返回 对应 char
4     # 还 没 有 看 懂 , 回 头 研 究 下
5     def __init__(self):
6         self.char_list = [-1 for i in range(256)]
7         self.index = 0 # 记录 当 前 字 符 的 个 数 , 可 以 理 解 为 输入 的 字 符 串 中
8         的 下 标
9     def FirstAppearingOnce(self):
10        # write code here
11        # 对 于 字 符 流 的 处 理 是 和 字 符 串 一 样 的 吗 ? ?
12        min_value = 500
13        min_idx = -1
14        for i in range(256):
15            if self.char_list[i] > -1:
16                if self.char_list[i] < min_value:
17                    min_value = self.char_list[i]
18                    min_idx = i
19        if min_idx > -1:
20            return chr(min_idx)
21        else:
22            return '#'
23
24    def Insert(self, char):
25        # write code here
26    # 如 果 是 第 一 出 现 , 则 将 对 应 元 素 的 值 改 为 下 边
27    if self.char_list[ord(char)] == -1:
28        self.char_list[ord(char)] = self.index
29    # 如 果 已 经 出 现 过 两 次 了 , 则 不 修 改
30    elif self.char_list[ord(char)] == -2:
31        pass
32    # 如 果 出 现 过 一 次 , 则 进 行 修 改 , 修 改 为 -2
33    else:
34        self.char_list[ord(char)] = -2
35    self.index += 1
36
```

```
4
5 def __init__(self):
6     self.s=""
7 def FirstAppearingOnce(self):
8
9     res=list(filter(lambda c:self.s.count(c)==1,self.s))
10    return res[0] if res else "#"
11
12 def Insert(self, char):
13
14     self.s+=char
15
16
17
18
```

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。

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 class Solution:
7     def EntryNodeOfLoop(self, pHead):
8         # write code here
9         # 链表中环的处理
10        # 多写几道关于链表的题目！！！
11        slow, fast=pHead, pHead
12        while fast and fast.next:
13            slow=slow.next
14            fast=fast.next.next
15            if slow==fast:
16                slow2=pHead
17                while slow!=slow2:
18                    slow=slow.next
19                    slow2=slow2.next
20                return slow
21
22    #return None
23
24
```

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

```
1 # -*- coding:utf-8 -*-
2 class ListNode:
3     def __init__(self, x):
4         self.val = x
5         self.next = None
6 class Solution:
7     def deleteDuplication(self, pHead):
8         # write code here
9         # 方法一，最常规的做法，必须掌握！！！
10        # 这道题目应该不好用递归去写！
11        if pHead == None or pHead.next == None:
12            return pHead
13        # 新建一个链表
14        # 和那道反转链表用的方法是差不多的
15        new_head = ListNode(-1)
16        new_head.next = pHead
17        pre = new_head
18        p = pHead
19        nex = None
20        while p != None and p.next != None:
21            nex = p.next
22            if p.val == nex.val:
23                while nex != None and nex.val == p.val:
24                    nex = nex.next
25                    pre.next = nex
26                    p = nex
27                else:
28                    pre = p
29                    p = p.next
30        return new_head.next
31
32
```

```
1
2 # 新建一个链表返回
3 def deleteDuplication(self, pHead):
4     res = []
5     while pHead:
6         res.append(pHead.val)
7         pHead = pHead.next
8     res = list(filter(lambda c: res.count(c) == 1, res))
9     dummy = ListNode(0)
10    pre = dummy
11    for i in res:
12        node = ListNode(i)
13        pre.next = node
14        pre = pre.next
```

```
15  
16  
17  
18  
19  
  
    return dummy.next
```

57. 二叉树的下一个结点 (树)

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeLinkNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7         self.next = None
8 class Solution:
9     # 重新看一下题目的解答
10    def GetNext(self, pNode):
11        # write code here
12        if pNode.right:#有右子树
13            p=pNode.right
14            while p.left:
15                p=p.left
16            return p
17        while pNode.next:#无右子树，则找第一个当前节点是父节点左孩子的
18            if(pNode.next.left==pNode):
19                return pNode.next
20            pNode = pNode.next#沿着父节点向上遍历
21        return #到了根节点仍没找到，则返回空
22
23
24
```

节点

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 递归比较左右节点，然后对左右节点的左右分支进一步递归比较
9     def isSymmetrical(self, pRoot):
10         if not pRoot:
11             return True
12         return self.compare(pRoot.left, pRoot.right)
13
14     def compare(self, pRoot1, pRoot2):
15         if not pRoot1 and not pRoot2:
16             return True
17         if not pRoot1 or not pRoot2:
18             return False
19         if pRoot1.val == pRoot2.val:
20             if self.compare(pRoot1.left, pRoot2.right) and self.compare(pRoot1.right, pR
oot2.left):
21                 return True
22             return False
23
24
25
26
```

59. 按之字形顺序打印二叉树 (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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     def Print(self, pRoot):
9         if not pRoot:
10             return []
11         nodeStack=[pRoot]
12         result=[]
13         while nodeStack:
14             res = []
15             nextStack=[]
16             for i in nodeStack:
17                 res.append(i.val)
18                 if i.left:
19                     nextStack.append(i.left)
20                 if i.right:
21                     nextStack.append(i.right)
22             nodeStack=nextStack
23             result.append(res)
24         returnResult=[]
25         for i,v in enumerate(result):
26             if i%2==0:
27                 returnResult.append(v)
28             else:
29                 returnResult.append(v[::-1])
30         return returnResult
31
32
```

60. 把二叉树打印成多行 (树)

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 返回二维列表 [[1,2],[4,5]]
9     # 看的别人的答案
10
11    def Print(self, pRoot):
12        if not pRoot:
13            return []
14        nodeStack = [pRoot]
15        result = []
16        while nodeStack:
17            res = []
18            nextStack = []
19            for i in nodeStack:
20                res.append(i.val)
21                if i.left:
22                    nextStack.append(i.left)
23                if i.right:
24                    nextStack.append(i.right)
25            nodeStack = nextStack
26            result.append(res)
27        return result
28
29
30
31
32        ...
33
34    用例：
35    {8,6,10,5,7,9,11}
36
37    对应输出应该为：
38    [[8],[6,10],[5,7,9,11]]
39        ...
40
```

61. 序列化二叉树（树）

题目描述

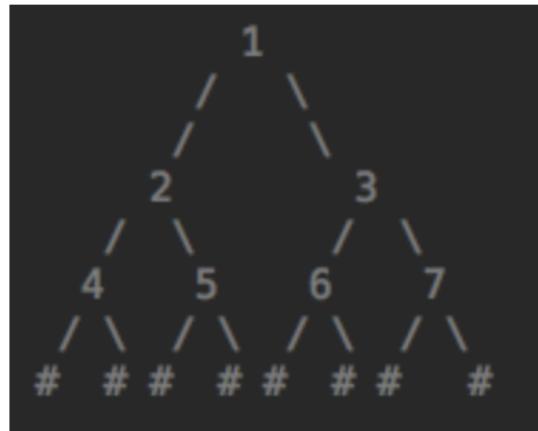
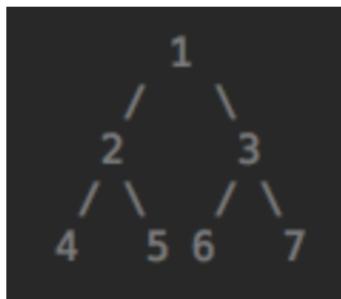
请实现两个函数，分别用来序列化和反序列化二叉树

解题思路

序列化二叉树：把一棵二叉树按照某种遍历方式的结果以某种格式保存为字符串。需要注意的是，序列化二叉树的过程中，如果遇到空节点，需要以某种符号（这里用#）表示。以下图二叉树为例，序列化二叉树时，需要将空节点也存入字符串中。

————>

序列化可以基于先序/中序/后序/按层等遍历方式进行，这里采用先序遍历的方式实现，字符串之间用“，”隔开。代码如下：



————>

```
1 def Serialize(self, root):
2     if not root:
3         return '#'
4     return str(root.val) + ',' + self.Serialize(root.left) + ',' + self.Serialize(root.right)
5 ...
6 反序列化二叉树：根据某种遍历顺序得到的序列化字符串，重构二叉树。
7 具体思路是按前序遍历“根左右”的顺序，根节点位于其左右子节点的前面，即非空（#）的第一个节点是某子树的根节点，左右子节点在该根节点后，以空节点#为分隔符。代码如下：
8 ...
9 def Deserialize(self, s):
10    list = s.split(',')
11    return self.deserializeTree(list)
12 def deserializeTree(self, list):
13    if len(list)<=0:
14        return None
15    val = list.pop(0)
16    root = None
```

```

17     if val != '#':
18         root = TreeNode(int(val))
19         root.left = self.deserializeTree(list)
20         root.right = self.deserializeTree(list)
21     return root
22

1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 # 没有看懂，研究别人的代码！！！
8 # 见收藏夹里的一个CSDN博客
9
10 class Solution:
11     def Serialize(self, root):
12         """前序递归"""
13         ret = []
14         if not root:
15             return '#'
16         ret.append(str(root.val))
17         l = self.Serialize(root.left)
18         ret.append(l)
19         r = self.Serialize(root.right)
20         ret.append(r)
21         return ','.join(ret)
22
23     def Serialize_no_rec(self, root):
24         """前序非递归"""
25         serialize_str = []
26         if not root:
27             return '#'
28         s = []
29         while root or s:
30             while root:
31                 # serialize_str += (str(root.val)+',')
32                 serialize_str.append(str(root.val))
33                 s.append(root.right) # 栈中存放右结点，便于左子树访问完之后回溯
34                 root = root.left
35             # serialize_str += "#,"
36             serialize_str.append('#') # 左结点访问完，用#来标识该结点的空指针
37             root = s.pop() # 依次访问栈中的右子树 # print serialize_str
38         return ','.join(serialize_str)
39
40
41     def Deserialize(self, s):
42         serialize = s.split(',')
43         tree, sp = self.deserialize(serialize, 0)
44         return tree
45
46     def deserialize(self, s, sp):
47         if sp >= len(s) or s[sp] == "#":
48             return None, sp + 1
49         node = TreeNode(int(s[sp]))
50         sp += 1
51         node.left, sp = self.deserialize(s, sp)
52         node.right, sp = self.deserialize(s, sp)

```

```
53  
54    return node, sp  
55
```

62. 二叉搜索树的第k个节点

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。

```
1 # -*- coding:utf-8 -*-
2 class TreeNode:
3     def __init__(self, x):
4         self.val = x
5         self.left = None
6         self.right = None
7 class Solution:
8     # 返回对应节点TreeNode
9     # 这种树的题目非常重要！！！
10    def KthNode(self, pRoot, k):
11        # write code here
12        # 要注意，返回的是节点，而不是节点的值
13        # 树的题目普遍牵涉到递归！
14        self.res=[]
15        self.dfs(pRoot)# 只有用self才能在不同函数中通用
16        if 0<k<=len(self.res):
17            return self.res[k-1]
18        else:
19            return None
20
21    # 中序遍历，输出第k个节点
22    def dfs(self,root):
23        if not root:
24            return
25        # 中序遍历，先打印左节点，后打印中节点，最后打印右节点
26        self.dfs(root.left)
27        self.res.append(root)
28        self.dfs(root.right)
29
30
31
32
```

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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     # 重新看题目，总结！
4     def __init__(self):
5         self.arr=[]
6     def Insert(self, num):
7         self.arr.append(num)
8         self.arr.sort()
9     def GetMedian(self, shabi):
10        length=len(self.arr)
11        if length%2==1:
12            return self.arr[length//2]
13        return (self.arr[length//2]+self.arr[length//2-1])/2.0
14
15    ...
16    用例：如果没有那个shabi的话，就是如下
17 [5,2,3,4,1,6,7,0,8]
18
19 对应输出应该为：
20
21 "5.00 3.50 3.00 3.50 3.00 3.50 4.00 3.50 4.00 "
22
23 你的输出为：
24
25 GetMedian() takes exactly 1 argument (2 given)
26    ...
```

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]}。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def maxInWindows(self, num, size):
4         # write code here
5         # 有没有复杂度低一些的算法???
6         # 解法一，时间复杂度低一些
7         res, i = [], 0
8         while size > 0 and i + size - 1 < len(num):
9             res.append(max(num[i:i + size]))
10            i += 1
11        return res
```

```
1
2     # 方法二：比较好想，时间复杂度高了，也是可行的
3     Mymatrix = []
4     num_len = len(num)
5     B = []
6     if size == 0:
7         return []
8     for i in range(num_len-size+1):
9         test = 0
10        for j in range(size):
11            if num[i+j]>test:
12                test = num[i+j]
13        B.append(test)
14
15    return B
16
```

```
1 # 方法三：
2 queue,res,i = [],[],0
3 while size>0 and i<len(num):
4     if len(queue)>0 and i-size+1 > queue[0]: #若最大值queue[0]位置过期则弹出
5         queue.pop(0)
6     while len(queue)>0 and num[queue[-1]]<num[i]: #每次弹出所有比num[i]的数字
7         queue.pop()
8     queue.append(i)
9     if i>=size-1:
10        res.append(num[queue[0]])
11    i += 1
12 return res
13
```

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 3×4 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。

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     def hasPath(self, board, row, col, word):
4         self.col, self.row = col, row
5         board = [list(board[col * i:col * i + col]) for i in range(row)]
6         for i in range(row):
7             for j in range(col):
8                 if board[i][j] == word[0]:
9                     self.b = False
10                    self.search(board, word[1:], [(i, j)], i, j)
11                    if self.b:
12                        return True
13        return False
14
15    def search(self, board, word, dict, i, j):
16        if word == "":
17            self.b = True
18            return
19        if j != 0 and (i, j - 1) not in dict and board[i][j - 1] == word[0]:
20            self.search(board, word[1:], dict + [(i, j - 1)], i, j - 1)
21        if i != 0 and (i - 1, j) not in dict and board[i - 1][j] == word[0]:
22            self.search(board, word[1:], dict + [(i - 1, j)], i - 1, j)
23        if j != self.col - 1 and (i, j + 1) not in dict and board[i][j + 1] == word[0]:
24            self.search(board, word[1:], dict + [(i, j + 1)], i, j + 1)
25        if i != self.row - 1 and (i + 1, j) not in dict and board[i + 1][j] == word[0]:
26            self.search(board, word[1:], dict + [(i + 1, j)], i + 1, j)
27        ...
28
```

```
1 # 方法二：递归
2 class Solution:
3     def hasPath(self, matrix, rows, cols, path):
4         for i, s in enumerate(matrix):
5             if s==path[0] and self.visit([(i//cols, i%cols)], matrix, rows, cols, path):
6                 return True
7         return False
8
9     def visit(self, ans, matrix, rows, cols, path):
10        if len(ans)==len(path):
11            return True
12        i,j = ans[-1]
13        nex = [(ii,jj) for ii,jj in [(i,j-1),(i,j+1),(i-1,j),(i+1,j)] if 0<= ii <rows and 0<= jj <cols and
```

```
15     (ii,jj) not in ans and
16     matrix[ii*cols +jj]==path[len(ans)])
17 return sum([self.visit(ans+[x], matrix, rows, cols, path) for x in nex])
18 ...
19
20
```

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？

```
1 # -*- coding:utf-8 -*-
2 class Solution:
3     '''def movingCount(self, threshold, rows, cols):
4         # write code here
5         # 这个从大往小看是不是比较简单？
6         # 一道典型的动态规划题目，用DFS|BFS，把大问题分拆成小问题
7         len_row = len(str(rows))#是几位数
8         len_col = len(str(cols))#是几位数
9         ...
10 ...
11     将地图全部置1，遍历能够到达的点，将遍历的点置0并令计数+1.这个思路
12     在找前后左右相连的点很有用，比如leetcode中的海岛个数问题/最大海岛
13     问题都可以用这种方法来求解。
14     ...
15
16     def __init__(self):
17         self.count = 0
18
19     def movingCount(self, threshold, rows, cols):
20         # write code here
21         arr = [[1 for i in range(cols)] for j in range(rows)]
22         self.findway(arr, 0, 0, threshold)
23         return self.count
24
25     def findway(self, arr, i, j, k):
26         if i < 0 or j < 0 or i >= len(arr) or j >= len(arr[0]):
27             return
28         tmpi = list(map(int, list(str(i))))
29         tmpj = list(map(int, list(str(j))))
30         if sum(tmpi) + sum(tmpj) > k or arr[i][j] != 1:
31             return
32         arr[i][j] = 0
33         self.count += 1
34         self.findway(arr, i + 1, j, k)
35         self.findway(arr, i - 1, j, k)
36         self.findway(arr, i, j + 1, k)
37         self.findway(arr, i, j - 1, k)
```