牛客在线编程试题详解

剑指offer-Python篇

牛客资料库出品
nowcoder.com

目录

剑指 offer 题目题解：Python 篇

在线编程地址：https://www.nowcoder.com/ta/coding-interviews?from=EDjob

1.不用加减乘除作加法

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def Add(self, num1, num2):
        return sum([num1, num2])
```

2、丑数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def GetUglyNumber_Solution(self, index):
        if index <= 0:
            return 0
        uglylist = [1]
        twoidx = 0
        threeidx = 0
        fiveidx = 0
        for i in range(index-1):
            new = min(uglylist[twoidx]*2, uglylist[threeidx]*3,
uglylist[fiveidx]*5)
            uglylist.append(new)
            if new%2 == 0:
                twoidx += 1
            if new%3 == 0:
                threeidx += 1
            if new%5 == 0:
                fiveidx += 1
        return uglylist[-1]
```

3、两个链表的第一个公共节点

代码展示

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def FindFirstCommonNode(self, pHead1, pHead2):
        if not pHead1 or not pHead2:
            return None
        p1, p2 = pHead1, pHead2
        while p1 != p2:
            p1 = pHead2 if not p1 else p1.next
            p2 = pHead1 if not p2 else p2.next
        return p1
```

4、二位数组中的查找

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    # array 二维列表
    def Find(self, target, array):
        # write code here
        flag = 'false'
        for i in range(len(array)):
            if target in array[i]:
                flag = 'true'
                break
        return flag
while True:
    try:
        s = Solution()
        L = list(eval(raw_input()))
        target = L[0]
        array = L[1]
```

```
        res = s.Find(target, array)
        print(res)
    except:
        Break
```

5、二叉搜索树与双向链表

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Convert(self, pRootOfTree):
        if not pRootOfTree:
            return pRootOfTree
        if not pRootOfTree.left and not pRootOfTree.right:
            return pRootOfTree
        self.Convert(pRootOfTree.left)
        left = pRootOfTree.left
        if left:
            while left.right:
                left = left.right
            pRootOfTree.left, left.right = left, pRootOfTree
        self.Convert(pRootOfTree.right)
        right = pRootOfTree.right
        if right:
            while right.left:
                right = right.left
            pRootOfTree.right, right.left = right, pRootOfTree
        while pRootOfTree.left:
            pRootOfTree = pRootOfTree.left
        return pRootOfTree
```

6、二叉搜索树的后序遍历序列

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def VerifySquenceOfBST(self, sequence):
        if len(sequence) == 0:
            return False
        if len(sequence) == 1:
            return True
        i = 0
        root = sequence[-1]
        while sequence[i] < root:
            i += 1
        j = i
        for k in range(j, len(sequence)-1):
            if sequence[k] < root:
                return False
        left, right = True, True
        if j > 0:
            left = self.VerifySquenceOfBST(sequence[:j])
        if len(sequence) - j > 1:
            right = self.VerifySquenceOfBST(sequence[j:-1])
        return left and right
```

7、二叉搜索树的第 k 个结点

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    # 返回对应节点 TreeNode
    def KthNode(self, pRoot, k):
        self.idx = 0
```

```
        self.k = k
        self.target = None
        self.inorder(pRoot)
        return self.target

    def inorder(self, root):
        if not root:
            return
        self.inorder(root.left)
        self.idx += 1
        if self.idx == self.k:
            self.target = root
            return self.target
        self.inorder(root.right)
```

8、二叉树中和为某一值的路径

代码展示

```
# -*- coding:utf-8 -*-
# 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 []
        if root and not root.left and not root.right and root.val ==
expectNumber:
            return [[root.val]]
        res = []
        left = self.FindPath(root.left, expectNumber - root.val)
        right = self.FindPath(root.right, expectNumber - root.val)
        for val in left+right:
            res.append([root.val] + val)
        return res
```

9、二叉树的下一个结点

代码展示

```
# -*- coding:utf-8 -*-
# class TreeLinkNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
#         self.next = None
class Solution:
    def GetNext(self, pNode):
        if not pNode:
            return None
        if pNode.right:
            pr = pNode.right
            while pr.left:
                pr = pr.left
            return pr
        while pNode.next:
            pp = pNode.next
            if pp.left == pNode:
                return pp
            pNode = pp
```

10、二叉树的深度

代码展示

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

```
        else:
            left = self.TreeDepth(pRoot.left) + 1
            right = self.TreeDepth(pRoot.right) + 1
            return max(left, right)
```

11、二叉树的镜像

代码展示

```
# -*- coding:utf-8 -*-
# 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:
            root.left, root.right = root.right, root.left
            self.Mirror(root.left)
            self.Mirror(root.right)
```

12、二进制中 1 的个数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def NumberOf1(self, n):
        return sum([(n >> i & 1) for i in range(32)])
```

13、从上往下打印二叉树

代码展示

```
# -*- coding:utf-8 -*-
# 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):
        if not root:
            return []
        res, nodes = [], [root]
        while nodes:
            node = nodes.pop(0)
            res.append(node.val)
            if node.left:
                nodes.append(node.left)
            if node.right:
                nodes.append(node.right)
        return res
```

14、从尾到头打印链表

代码展示

```
# -*- coding:utf-8 -*-
# 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):
        # write code here
        cur = listNode
```

```
        prev = None
        while cur:
            curnext = cur.next
            cur.next = prev
            prev = cur
            cur = curnext
        res = []
        while prev:
            res.append(prev.val)
            prev = prev.next
        return res
```

15、删除链表中重复的结点

代码展示

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def deleteDuplication(self, pHead):
        if not pHead:
            return pHead
        pl = []
        while pHead:
            pl.append(pHead.val)
            pHead = pHead.next
        npl = []
        for p in pl:
            if pl.count(p) == 1:
                npl.append(p)
        head = ListNode(0)
        h = head
        for p in npl:
            h.next = ListNode(p)
            h = h.next
        return head.next
```

16、包含 min 函数的栈

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def __init__(self):
        self.stack = []
        self.curmin = float('inf')

    def push(self, node):
        self.curmin = min(self.curmin, node)
        self.stack.append([node, self.curmin])

    def pop(self):
        if not self.stack:
            return None
        node = self.stack[-1]
        self.stack.remove(node)
        return node[0]

    def top(self):
        if not self.stack:
            return None
        return self.stack[-1][0]

    def min(self):
        if not self.stack:
            return None
        return self.stack[-1][1]
```

17、反转链表

代码展示

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

```
class Solution:
    # 返回 ListNode
    def ReverseList(self, pHead):
        if not pHead or not pHead.next:
            return pHead
        pre = None
        h = pHead
        while h:
            nex = h.next
            h.next = pre
            pre = h
            h = nex
        return pre
```

18、变态跳台阶

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def jumpFloorII(self, number):
        if number < 3:
            return number
        return 2 * self.jumpFloorII(number - 1)
```

19、合并两个排序的链表

代码展示

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    # 返回合并后列表
    def Merge(self, pHead1, pHead2):
        head = ListNode(0)
```

```
        p = head
        while pHead1 and pHead2:
            if pHead1.val <= pHead2.val:
                p.next = pHead1
                pHead1 = pHead1.next
            else:
                p.next = pHead2
                pHead2 = pHead2.next
            p = p.next
        if pHead1:
            p.next = pHead1
        if pHead2:
            p.next = pHead2
        return head.next
```

20、和为 S 的两个数字

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def FindNumbersWithSum(self, array, tsum):
        if not isinstance(array, list):
            return []
        ls = []
        for i, v in enumerate(array):
            for v1 in array[i:]:
                if v + v1 == tsum:
                    ls.append([v, v1])
        return ls[0] if ls else []
```

21、和为 S 的连续整数序列

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def FindContinuousSequence(self, tsum):
```

```
        if tsum < 1:
            return 0
        res = []
        for i in range(1, tsum-1):
            cursum = i
            cur = [i]
            for j in range(i+1, tsum):
                cursum += j
                cur.append(j)
                if cursum == tsum:
                    res.append(cur)
                    break
                elif cursum > tsum:
                    break
        return res
```

22、复杂链表的复制

代码展示

```
# -*- coding:utf-8 -*-
# class RandomListNode:
#     def __init__(self, x):
#         self.label = x
#         self.next = None
#         self.random = None
import collections
class Solution:
    # 返回 RandomListNode
    def Clone(self, pHead):
        dic = collections.defaultdict(lambda :RandomListNode(0))
        head = pHead
        dic[None] = None
        while head:
            dic[head].label = head.label
            dic[head].next = dic[head.next]
            dic[head].random = dic[head.random]
            head = head.next
        return dic[pHead]
```

23、字符串的排列

代码展示

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

24、字符流中第一个不重复的字符

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    # 返回对应 char
    def __init__(self):
        self.s = ''

    def FirstAppearingOnce(self):
        for ch in self.s:
            if self.s.count(ch) == 1:
                return ch
        return '#'

    def Insert(self, char):
        self.s = self.s + char
```

25、孩子们的游戏（圆圈中最后剩下的数）

代码展示

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

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

26、对称的二叉树

代码展示

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

27、左旋转字符串

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def LeftRotateString(self, s, n):
        if n > len(s) or n == 0:
            return s
        return s[n:]+s[:n]
```

28、平衡二叉树

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def IsBalanced_Solution(self, pRoot):
        if not pRoot:
            return True
        if abs(self.depth(pRoot.left) - self.depth(pRoot.right)) > 1:
            return False
        return self.IsBalanced_Solution(pRoot.left) and
self.IsBalanced_Solution(pRoot.right)

    def depth(self, root):
        if not root:
            return 0
        left = self.depth(root.left) + 1
        right = self.depth(root.right) + 1
        return max(left, right)
```

29、序列化二叉树

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    flag = -1
    def Serialize(self, root):
        if not root:
            return '#'
        return str(root.val) + ',' + self.Serialize(root.left) + ',' +
self.Serialize(root.right)

    def Deserialize(self, s):
        self.flag += 1
        l = s.split(',')
        if self.flag >= len(s):
            return None
        root = None
        if l[self.flag] != '#':
            root = TreeNode(int(l[self.flag]))
            root.left = self.Deserialize(s)
            root.right = self.Deserialize(s)
        return root
```

30、扑克牌顺子

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def IsContinuous(self, numbers):
        if not numbers:
            return False
        numbers.sort()
```

```
        nzero = numbers.count(0)
        if len(set(numbers[nzero:])) != len(numbers[nzero:]):
            return False
        if numbers[-1] - numbers[nzero] == len(numbers)-1 or
numbers[nzero] + nzero >= numbers[-1]:
            return True
        return False
```

31、把二叉树打印成多行

代码展示

```
# -*- coding:utf-8 -*-
# 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):
        stack = [pRoot]
        res = []
        while pRoot and stack:
            temp = [root.val for root in stack]
            res.append(temp)
            stack = [kid for root in stack for kid in (root.left,
root.right) if kid]
        return res
```

32、把字符串转换成整数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def StrToInt(self, s):
        if not s:
```

```
            return 0
        sign = '+'
        if s[0] in ['+', '-']:
            sign = s[0]
            s = s[1:]
        res = 0
        for ch in s:
            if not ch.isdigit():
                return 0
            res = res*10 + int(ch)
        return -res if sign == '-' else res
```

33、把数组排成最小的数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def PrintMinNumber(self, numbers):
        if not numbers:
            return ''
        lmd = lambda x,y:int(str(x)+str(y)) - int(str(y)+str(x))
        res = sorted(numbers, cmp=lmd)
        return ''.join([str(i) for i in res])
```

34、按之字形顺序打印二叉树

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def Print(self, pRoot):
        stack = [pRoot]
```

```
        level = 1
        res = []
        while stack and pRoot:
            temp = [root.val for root in stack]
            if level % 2 == 0:
                res.append(temp[::-1])
            else:
                res.append(temp)
            stack = [kid for root in stack for kid in (root.left,
root.right) if kid]
            level += 1
        return res
```

35、数值的整数次方

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def Power(self, base, exponent):
        if exponent == 0:
            return 1
        if exponent < 0:
            base = 1/base
            exponent = abs(exponent)
        res = 1
        for i in range(exponent):
            res *= base
        return res
```

36、数字在排序数组中出现的次数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def GetNumberOfK(self, data, k):
        return data.count(k)
```

37、数据流中的中位数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def __init__(self):
        self.stack = []

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

    def GetMedian(self, stack):
        n = len(self.stack)
        if n % 2 != 0:
            return self.stack[int(n/2)]
        else:
            return (self.stack[int(n/2)-1]+self.stack[int(n/2)])/2.0
```

38、数组中出现次数超过一半的数字

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def MoreThanHalfNum_Solution(self, numbers):
        numbers = sorted(numbers)
        length = len(numbers)
        mid = int(length / 2)
        if numbers.count(numbers[mid]) > mid:
            return numbers[mid]
        else:
            return 0
```

39、数组中只出现一次的数字

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    # 返回[a,b] 其中ab是出现一次的两个数字
    def FindNumsAppearOnce(self, array):
        if not array:
            return []
        res = []
        for n in array:
            if array.count(n) == 1:
                res.append(n)
                if len(res) == 2:
                    break
        return res
```

40、数组中的逆序对

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def InversePairs(self, data):
        count = 0
        copy = []
        for n in data:
            copy.append(n)
        copy.sort()
        for i in range(len(copy)):
            count += data.index(copy[i])
            data.remove(copy[i])
        return count%1000000007
```

41、数组中重复的数字

代码展示

```
# -*- coding:utf-8 -*-
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):
        if not numbers:
            return False
        for n in numbers:
            if numbers.count(n) > 1:
                duplication[0] = n
                return True
        return False
```

42、整数中 1 出现的次数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def NumberOf1Between1AndN_Solution(self, n):
        res = 0
        i = 1
        while i <= n:
            a = n/i
            b = n%i
            res += (a+8)/10*i + (a%10==1)*(b+1)
            i *= 10
        return res
```

43、旋转数组的最小数字

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        if len(rotateArray) == 0:
            return 0
        for i in range(1, len(rotateArray)):
            if rotateArray[i] < rotateArray[i-1]:
                return rotateArray[i]
        return rotateArray[0]
```

利用 python 投机取巧

```
# -*- coding:utf-8 -*-
class Solution:
    def minNumberInRotateArray(self, rotateArray):
        if len(rotateArray) == 0:
            return 0
        return min(rotateArray)
```

44、替换空格

代码展示

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

45、最小的 k 个数

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def GetLeastNumbers_Solution(self, tinput, k):
        tinput = sorted(tinput)
        if k > len(tinput):
            return []
        return tinput[:k]
```

46、机器人的运动范围

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def __init__(self):
        self.val = {}

    def movingCount(self, threshold, rows, cols):
        return self.moving(threshold, rows, cols, 0, 0)

    def moving(self, threshold, rows, cols, i, j):
        if i/10 + i%10 + j/10 + j%10 > threshold:
            return 0
        if i >= rows or j >= cols or i < 0 or j < 0:
            return 0
        if (i, j) in self.val:
            return 0
        self.val[(i, j)] = 1
        return 1 + self.moving(threshold, rows, cols, i+1, j) +
self.moving(threshold, rows, cols, i-1, j) + self.moving(threshold, rows,
cols, i, j+1) + self.moving(threshold, rows, cols, i, j-1)
```

47、构建乘积数组

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def multiply(self, A):
        if not A:
            return []
        B = []
        cur = 1
        for i in range(len(A)):
            for j in range(len(A)):
                if i != j:
                    cur *= A[j]
            B.append(cur)
            cur = 1
        return B
```

48、栈的压入、弹出序列

代码展示

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

49、树的子结构

代码展示

```
# -*- coding:utf-8 -*-
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None
class Solution:
    def HasSubtree(self, pRoot1, pRoot2):
        if not pRoot1 or not pRoot2:
            return False
        return self.isSubtree(pRoot1, pRoot2) or
self.isSubtree(pRoot1.left, pRoot2) or self.isSubtree(pRoot1.right,
pRoot2)

    def isSubtree(self, p1, p2):
        if p2 == None:
            return True
        if p1 == None or p1.val != p2.val:
            return False
        return self.isSubtree(p1.left, p2.left) and
self.isSubtree(p1.right, p2.right)
```

50、正则表达式匹配

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    # s, pattern 都是字符串
    def match(self, s, pattern):
        m, n = len(s), len(pattern)
        match = [[False for j in range(n+1)] for i in range(m+1)]
        match[0][0] = True
        for j in range(n+1):
            if j > 1 and pattern[j-1] == '*':
                match[0][j] = match[0][j-2]
```

```
            for i in range(1, m+1):
                for j in range(1, n+1):
                    if s[i-1] == pattern[j-1] or pattern[j-1] == '.':
                        match[i][j] = match[i-1][j-1]
                    elif j > 1 and pattern[j-1] == '*':
                        if s[i-1] == pattern[j-2] or pattern[j-2] ==
'.':
                            match[i][j] = match[i-1][j] or
match[i][j-2]
                        else:
                            match[i][j] = match[i][j-2]
        return match[m][n]
```

51、求 1+2+3+...+n

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def Sum_Solution(self, n):
        res = list(range(1, n+1))
        return sum(res)
```

52、滑动窗口的最大值

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def maxInWindows(self, num, size):
        if size == len(num):
            return [max(num)]
        if size < 1 or size > len(num):
            return []
        res = []
        for i in range(len(num)-size+1):
            res.append(max(num[i:i+size]))
        return res
```

53、用两个栈实现队列

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def __init__(self):
        self.stack1 = []
        self.stack2 = []

    def push(self, node):
        return self.stack1.append(node)

    def pop(self):
        # return xx
        if self.stack2 == []:
            while self.stack1:
                self.stack2.append(self.stack1.pop())
        return self.stack2.pop()
```

54、矩形覆盖

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def rectCover(self, number):
        if number < 3:
            return number
        f1, f2 = 1, 2
        for i in range(3, number+1):
            f1, f2 = f2, f1+f2
        return f2
```

55、矩阵中的路径

代码展示

```
# -*- coding:utf-8 -*-
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.findPath(list(matrix), rows, cols, path[1:],
i, j):
                        return True
        return False

    def findPath(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.findPath(matrix, rows, cols, path[1:], i, j+1)
        elif j-1 >= 0 and matrix[i*cols+j-1] == path[0]:
            return self.findPath(matrix, rows, cols, path[1:], i, j-1)
        elif i+1 < rows and matrix[(i+1)*cols+j] == path[0]:
            return self.findPath(matrix, rows, cols, path[1:], i+1, j)
        elif i-1 >= 0 and matrix[(i-1)*cols+j] == path[0]:
            return self.findPath(matrix, rows, cols, path[1:], i-1, j)
        else:
            return False
```

56、第一个只出现一次的字符

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def FirstNotRepeatingChar(self, s):
        if len(s) == 0:
            return -1
        for i, c in enumerate(s):
```

```
            if s.count(c) == 1:
                return i
        return -1
```

57、翻转单词顺序列

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def ReverseSentence(self, s):
        if len(s) == 0:
            return s
        ls = s.split(' ')
        ls.reverse()
        return ' '.join(ls)
```

58、菲波那切数列

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def Fibonacci(self, n):
        if n < 2:
            return n
        f0, f1 = 0, 1
        for i in range(2, n+1):
            f0, f1 = f1, f0+f1
        return f1
```

59、表示数值的字符串

代码展示

```
# -*- coding:utf-8 -*-
import re
class Solution:
    # s 字符串
    def isNumeric(self, s):
        return
re.match(r"^[\+\-]?[0-9]*(\.[0-9]*)?([eE][\+\-]?[0-9]+)?$", s)
```

60、调整数组顺序使奇数位于偶数前面

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def reOrderArray(self, array):
        if len(array) < 2:
            return array
        odd, even = [], []
        for num in array:
            if num % 2 == 0:
                odd.append(num)
            else:
                even.append(num)
        return even + odd


# -*- coding:utf-8 -*-
from collections import deque
class Solution:
    def reOrderArray(self, array):
        res = deque()
        length = len(array)
        for i in range(length):
            if array[i] % 2 == 0:
                res.append(array[i])
            if array[length - i - 1] % 2 != 0:
```

```
                res.appendleft(array[length - i - 1])
        return res
```

61、跳台阶

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def jumpFloor(self, number):
        if number < 3:
            return number
        f1, f2 = 1, 2
        for i in range(3, number+1):
            f1, f2 = f2, f1+f2
        return f2
```

62、连续子数组的最大和

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    def FindGreatestSumOfSubArray(self, array):
        if len(array) < 2:
            return len(array)
        res, curmax = -float('inf'), -float('inf')
        for num in array:
            curmax = max(curmax+num, num)
            res = max(res, curmax)
        return res
```

63、重建二叉树

代码展示

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

64、链表中倒数第 k 个点

代码展示

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

class Solution:
    def FindKthToTail(self, head, k):
        linklist = []
        while head:
```

```
            linklist.append(head)
            head = head.next
        length = len(linklist)
        if k > length or k < 1:
            return None
        return linklist[-k]
```

65、链表中环的入口结点

代码展示

```
# -*- coding:utf-8 -*-
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
class Solution:
    def EntryNodeOfLoop(self, pHead):
        if not pHead or not pHead.next or not pHead.next.next:
            return None
        fast, slow = pHead.next.next, pHead.next
        while slow != fast:
            if not fast.next or not fast.next.next:
                return None
            slow = slow.next
            fast = fast.next.next
        fast = pHead
        while fast != slow:
            slow = slow.next
            fast = fast.next
        return fast
```

66、顺时针打印矩阵

代码展示

```
# -*- coding:utf-8 -*-
class Solution:
    # matrix 类型为二维列表，需要返回列表
    def printMatrix(self, matrix):
        res = []
        while matrix:
            res += matrix.pop(0)
            if not matrix:
                break
            matrix = self.turn(matrix)
        return res

    def turn(self, matrix):
        m, n = len(matrix), len(matrix[0])
        tp = []
        for j in range(n):
            t = []
            for i in range(m):
                t.append(matrix[i][j])
            tp.append(t)
        tp.reverse()
        return tp
```

感谢牛友聪小白（牛客 id：1877463）提供此部分题解！