剑指Offer快速回顾

剑指Offer快速回顾

- 前言
- 题解
  - 标准有序链表实例
  - 标准二叉（搜索）树实例
  - 3. 数组中重复的数字
  - 4. 二维数组中的查找
  - 5. 替换空格
  - 6. 从尾到头打印链表
  - 7. 重建二叉树
  - 9. 用两个栈实现队列
  - 10. 斐波那契数列
  - 11. 旋转数组的最小数字
  - 12. 矩阵中的路径
  - 13. 机器人的运动范围
  - 14. 剪绳子
  - 15. 二进制中1的个数
  - 16. 数值的整数次方
  - 17. 打印从1到最大的n位数
  - 18. 删除链表的结点
  - 19. 正则表达式匹配
  - 20. 表示数值的字符串
  - 21. 调整数组顺序使奇数位于偶数前面
  - 22. 链表中倒数第K个结点
  - 24. 反转链表
  - 25. 合并两个排序的链表
  - 26. 树的子结构
  - 27. 二叉树的镜像
  - 28. 对称的二叉树
  - 29. 顺时针打印矩阵

30. 包含min函数的栈
31. 栈的压入、弹出序列
32. 从上到下打印二叉树
33. 二叉搜索树的后序遍历序列
34. 二叉树中和为某一值的路径
35. 复杂链表的复制
36. 二叉搜索树与双向链表
37. 序列化二叉树
38. 字符串的排列
39. 数组中出现次数超过一半的数字
40. 最小的k个数
41. 数据流中的中位数
42. 连续子数组的最大和
43. 1～n整数中1出现的次数
44. 回文链表
44. 数字序列中某一位的数字
45. 把数组排成最小的数
46. 把数字翻译成字符串
47. 礼物的最大价值
48. 最长不含重复字符的子字符串
49. 丑数
50. 第一个只出现一次的字符
51. 数组中的逆序对
52. 两个链表的第一个公共节点
53. 在排序数组中查找数字
54. 二叉搜索树的第k大节点
55. 二叉树的深度
55-II. 平衡二叉树
56. 数组中数字出现的次数
57. 和为s的两个数字
57-II. 和为s的连续正数序列
58. 翻转单词顺序
59. 滑动窗口的最大值
60. n个骰子的点数
61. 扑克牌中的顺子
62. 圆圈中最后剩下的数字

63. 股票的最大利润
66. 构建乘积数组
67. 把字符串转换成整数
68-I. 二叉搜索树的最近公共祖先
68-II. 二叉树的最近公共祖先

前言

这次回顾是在刷了两遍剑指Offer的基础上，在2021.02.23日收到字节面试通知以后匆匆完成的，大概花了三天时间，把剑指Offer中大部分的题目重新做完，对于一些比较基础的题目则选择性跳过了～

今天是2021.03.01，我已经完成了字节的四轮面试，正在等待最后的结果，希望offer顺利拿到！（面试做了三个算法题，两个是剑指Offer原题）

题解

标准有序链表实例

```
class ListNode:
    def __init__(self, val):
        self.val = val
        self.next = None

root = ListNode(1)
head = root
for i in range(2, 6):
    head.next = ListNode(i)
    head = head.next
# [1, 2, 3, 4, 5]
```

标准二叉（搜索）树实例

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

root = TreeNode(6)
root.left, root.right = TreeNode(2), TreeNode(8)
root.left.left, root.left.right = TreeNode(0), TreeNode(4)
root.right.left, root.right.right = TreeNode(7), TreeNode(9)
root.left.right.left, root.left.right.right = TreeNode(3), TreeNode(5)
# in_order: [0, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. 数组中重复的数字

- 在一个长度为 n 的数组 nums 里的所有数字都在 0～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重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字
- 测试用例

```
nums = [2, 3, 1, 0, 2, 5, 3] # return 2
```

```
# 常规解法
# 时间复杂度O(N) 空间复杂度O(N)
# 顺序遍历 + hash存储
```

```
# 特殊解法：一个萝卜一个坑
# 时间复杂度O(N) 空间复杂度O(1)
# 根据题设条件，只要判断当前索引所存变量是不是与索引相等，如果相等进入下一索引，不等则与以当前值为索引的值进行交换。如果发现两者值相等，则找到重复数字
```

4. 二维数组中的查找

- 在一个 n * m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，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
- 测试用例

```
matrix = [
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
target = 5 # return True
target = 20 # return False
```

```
# 特殊解法：从右上角向外搜索
```

5. 替换空格

- 请实现一个函数，把字符串 `s` 中的每个空格替换成"%20"
- 测试用例

```
s = "We are happy." # return "We%20are%20happy."
```

```
# replace函数实现
class Solution:
    def replaceSpace(self, s: str) -> str:
        return s.replace(' ', '%20')
```

```
# join + split
class Solution:
    def replaceSpace(self, s: str) -> str:
        return '%20'.join(s.split(' '))
```

6. 从尾到头打印链表

- 输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）
- 测试用例

```
head = [1,3,2] return [2, 3, 1]
```

```
# 辅助栈法 遍历链表入栈，逆序输出
```

```
# 递归法
```

7. 重建二叉树

- 输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。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
- 测试用例

```
前序遍历 preorder = [3,9,20,15,7]
中序遍历 inorder = [9,3,15,20,7]
```

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

class Solution:
```

```
        def buildTree(self, preorder, inorder):

            def dfs(pre, ino):
                if not pre: return None
                head = TreeNode(pre[0])
                idx = ino.index(pre[0])
                head.left = dfs(pre[1:idx+1], ino[:idx])
                head.right = dfs(pre[idx+1:], ino[idx+1:])
                return head

            return dfs(preorder, inorder)
```

9. 用两个栈实现队列

- 用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1)

```
# 第一个栈用于存放入队元素
# 当进行出队操作时，利用第二个栈，将第一个栈所有元素全部pop进第二个栈，再从第二个栈pop顶部元素，如果没有元素返回-1
class CQueue:
    def __init__(self):

    def appendTail(self, value: int) -> None:

    def deleteHead(self) -> int:
```

10. 斐波那契数列

- 写一个函数，输入 `n` ，求斐波那契（Fibonacci）数列的第 `n` 项（即 `F(N)`）。斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1
- 测试用例

```
n = 5 # return 5
```

```
# 递归法 当n稍大即报错
```

```
# 循环法
```

11. 旋转数组的最小数字

- 把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个递增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
- 测试用例

```
nums = [3,4,5,1,2] # return 1
```

```
# 经典题
# 二分搜索的变种
# 一定要注意当数组中大部分数都一样的时候如何解决(mid == right: right -= 1)
class Solution:
    def minArray(self, nums) -> int:
        left, right = 0, len(nums) - 1
        while left < right:
            mid = (left + right) // 2
            if nums[mid] < nums[right]:
                right = mid
            elif nums[mid] > nums[right]:
                left = mid + 1
            else: right -= 1
        return nums[left]
```

12. 矩阵中的路径

- 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格开始，每一步可以在矩阵中向左、右、上、下移动一格。如果一条路径经过了矩阵的某一格，那么该路径不能再次进入该格子
- 测试用例

```
board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]]
word = "ABCCED" # return True
```

```
# 经典题
# 标准的dfs案例，需要有一定基础
```

13. 机器人的运动范围

- 地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也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。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子?
- 测试用例

```
m = 2, n = 3, k = 1 # return 3
m = 3, n = 1, k = 0 # return 1
```

```
# dfs与bfs皆可，仔细分析后不是很难
class Solution:
    def movingCount(self, m, n, k) -> int:
        self.aset = set()
        def check_sum(num):
            pass
        def dfs(i, j):
            pass
```

14. 剪绳子

- 给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]*k[1]*...*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18
- 测试用例

```
n = 2 # return 1
n = 10 # return 36
```

```
# 特殊题型
# 这是一道数学题，涉及数论知识，尽可能拆成3，不足补2
```

15. 二进制中1的个数

- 请实现一个函数，输入一个整数（以二进制串形式），输出该数二进制表示中 1 的个数。例如，把 9 表示成二进制是 1001，有 2 位是 1。因此，如果输入 9，则该函数输出 2

```
# 涉及位运算
```

16. 数值的整数次方

- 实现函数double Power(double base, int exponent)，求base的exponent次方。不得使用库函数，同时不需要考虑大数问题
- 测试用例

```
base, exp = 2.000, 10 # return 1024.0000
base, exp = 2.100, 3 # return 9.2610
base, exp = 2.000, -2 # return 0.25
```

```
# 利用栈结构实现二分递增
# 稍微有点复杂
```

17. 打印从1到最大的n位数

- 输入数字 `n`，按顺序打印出从 1 到最大的 n 位十进制数。比如输入 3，则打印出 1、2、3 一直到最大的 3 位数 999
- 测试用例

```
n = 1 # return [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
# 此题在python环境下没有难点（涉及大数问题）
```

18. 删除链表的结点

- 给定单向链表的头指针和一个要删除的节点的值，定义一个函数删除该节点。返回删除后的链表的头节点
- 测试用例

```
head, val = [4,5,1,9], 5 # return [4, 1, 9]
```

```
# 经典的链表操作（这里一并给出）
class Solution:
    # 链表删除指定元素
    def deleteNode(self, head, val):
        if head.val == val: return head.next
        cur, pre = head.next, head
        while cur.val != val:
            pre = cur
            cur = cur.next
        pre.next = cur.next
        return head
```

19. 正则表达式匹配

- 请实现一个函数用来匹配包含 `.` 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（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"abaca"匹配，但与"aa.a"和"ab*a"均不匹配
- 测试用例

```
s, p = 'aa', 'a'
s, p = 'aa', 'a*'
s, p = 'ab', '.*'
s, p = 'aab', 'c*a*b'
s, p = 'mississippi', 'mis*is*p*.'
```

```
# 私以为此题为剑指Offer最难题，已经给我做出心里阴影了......
# 动态规划题
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
```

20. 表示数值的字符串

- 请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。例如，字符串"+100"、"5e2"、"-123"、"3.1416"、"-1E-16"、"0123"都表示数值，但"12e"、"1a3.14"、"1.2.3"、"+-5"及"12e+5.4"都不是

```
# 字符串解析题型
# 这种题目确实非常繁琐，比较麻烦，需要做大量的分类讨论，有时间补上吧
```

21. 调整数组顺序使奇数位于偶数前面

- 输入一个整数数组，实现一个函数来调整该数组中数字的顺序，使得所有奇数位于数组的前半部分，所有偶数位于数组的后半部分
- 测试用例

```
nums = [1,2,3,4] # return [3, 1, 2, 4]
```

```
# 标准双指针题型
# 熟悉快排的话，简直不要太简单
class Solution:
    def exchange(self, nums):
        left, right = 0, len(nums) - 1
        while left < right:
            while left < right and nums[right] % 2 == 0: right -= 1
            while left < right and nums[left] % 2 == 1: left += 1
            nums[left], nums[right] = nums[right], nums[left]
```

22. 链表中倒数第K个结点

- 输入一个链表，输出该链表中倒数第k个节点。为了符合大多数人的习惯，本题从1开始计数，即链表的尾节点是倒数第1个节点
- 测试用例

```
给定一个链表: 1->2->3->4->5, 和 k = 2
```

```
# 双指针法 一目了然
```

24. 反转链表

- 定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点
- 测试用例

```
1->2->3->4->5->NULL # 5->4->3->2->1->NULL
```

```
# 经典链表题 这里只给出原地反转解法，需要一点点逻辑
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        if not head or not head.next: return head
        post, cur, pre = head.next.next, head.next, head
        pre.next = None
        while post:
            cur.next = pre
            pre = cur
            cur = post
            post = post.next
        cur.next = pre
        return cur
```

25. 合并两个排序的链表

- 输入两个递增排序的链表，合并这两个链表并使新链表中的节点仍然是递增排序的
- 测试用例

```
输入：1->2->4, 1->3->4
输出：1->1->2->3->4->4
```

```
# 常规解法 按传统有序数组合并的解法进行，思路复杂了
class Solution:
    def mergeTwoLists(self, l1, l2):
        if not l1: return l2
        if not l2: return l1
        if l1.val < l2.val:
            l1, l2 = l2, l1

        node = l2
        while l1:
            while node:
                if node.next:
                    if node.val <= l1.val <= node.next.val:
```

```
                        tmp = node.next
                        node.next = ListNode(l1.val)
                        node.next.next = tmp
                        break
                else:
                    node.next = l1
                    return l2
                node = node.next
            l1 = l1.next
        return l2
```

```
# 链表合并有链表的特有解法
# 利用链表属性解题
class Solution:
    def mergeTwoLists(self, l1, l2):
        dum = cur = ListNode(0)
        while l1 and l2:
            if l1.val < l2.val:
                cur.next, l1 = l1, l1.next
            else:
                cur.next, l2 = l2, l2.next
            cur = cur.next
        cur.next = l1 if l1 else l2
        return dum.next
```


26. 树的子结构

- 输入两棵二叉树A和B，判断B是不是A的子结构(约定空树不是任意一个树的子结构)。B是A的子结构， 即 A中有出现和B相同的结构和节点值

```
# 二叉树题目
# 这一题如果准备不充分，有一定难度
# 双层深度优先搜索，第一层前序遍历A，找到与B的根结点值相同的结点进入第二层dfs
class Solution:
    def isSubStructure(self, A, B):
```

```
        if not A or not B: return False
        self.res = False

        def pre_order(node):
            if self.res: return
            if node.val == B.val: self.res = check(node, B)
            if node.left: pre_order(node.left)
            if node.right: pre_order(node.right)

        def check(n1, n2):
            if not n2: return True
            elif n1 and n2:
                if n1.val == n2.val:
                    left = check(n1.left, n2.left)
                    right = check(n1.right, n2.right)
                    return (left and right)
            return False

        pre_order(A)
        return self.res
```


27. 二叉树的镜像

- 请完成一个函数，输入一个二叉树，该函数输出它的镜像

```
# 比较简单，只需要从根结点开始置换叶子节点顺序即可
class Solution:
    def mirrorTree(self, root: TreeNode) -> TreeNode:
        def dfs(node):
            node.left, node.right = node.right, node.left
            if node.left: dfs(node.left)
            if node.right: dfs(node.right)
        if not root: return root
        dfs(root)
        return root
```

28. 对称的二叉树

- 请实现一个函数，用来判断一棵二叉树是不是对称的。如果一棵二叉树和它的镜像一样，那么它是对称的

```
# 独创特殊解法
class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        def dfs(n1, n2):
            if not n1 and not n2: return True
            elif n1 and n2 and n1.val == n2.val:
                left = dfs(n1.left, n2.right)
                right = dfs(n1.right, n2.left)
                return (left and right)
            return False
        return dfs(root, root)
```

```
# 前两遍做的时候采取层次遍历，也能实现，略复杂
```

29. 顺时针打印矩阵

- 输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字
- 测试用例

```
输入：matrix = [[1,2,3],[4,5,6],[7,8,9]]
输出：[1,2,3,6,9,8,7,4,5]

输入：matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
输出：[1,2,3,4,8,12,11,10,9,5,6,7]
```

```
# 模拟法 思路略复杂
```

30. 包含min函数的栈

- 定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)

```python
# 经典题
# 单调栈的实现原理
class MinStack:

    def __init__(self):
        self.list = []
        self.stack = []

    def push(self, x: int) -> None:
        self.list.append(x)
        if not self.stack or self.stack[-1] >= x:
            self.stack.append(x)

    def pop(self) -> None:
        x = self.list.pop()
        if x == self.stack[-1]:
            self.stack.pop()

    def top(self) -> int:
        return self.list[-1]

    def min(self) -> int:
        return self.stack[-1]
```


31. 栈的压入、弹出序列

- 输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，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

```
# 基础题目 暂时略过
```

32. 从上到下打印二叉树

- 从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印

```
# 老本行了 略过
```

33. 二叉搜索树的后序遍历序列

- 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结果。如果是则返回 `true`，否则返回 `false`。假设输入的数组的任意两个数字都互不相同
- **测试用例**

```
输入: [1,6,3,2,5]
输出: false

输入: [1,3,2,6,5]
输出: true
```

```
# 逐层递归，找当前结点的左右子列表，判断是否合理
class Solution:
    def verifyPostorder(self, nums):
        self.res = True

        def recur(i, j):
            if i >= j or not self.res: return
            idx, val = i, nums[j]
            while idx < j:
                if nums[idx] > val:
                    break
                idx += 1
            for k in range(idx, j):
                if nums[k] < val:
                    self.res = False
```

```
                    break
                recur(i, idx-1)
                recur(idx, j-1)

            recur(0, len(nums) - 1)
            return self.res
```

```
# Krahets大佬的解法
class Solution:
    def verifyPostorder(self, postorder: [int]) -> bool:
        def recur(i, j):
            if i >= j: return True
            p = i
            while postorder[p] < postorder[j]: p += 1
            m = p
            while postorder[p] > postorder[j]: p += 1
            return p == j and recur(i, m - 1) and recur(m, j - 1)

        return recur(0, len(postorder) - 1)
```

34. 二叉树中和为某一值的路径

- 输入一棵二叉树和一个整数，打印出二叉树中节点值的和为输入整数的所有路径。从树的根节点开始往下一直到叶节点所经过的节点形成一条路径

```
# 二叉树常规题
# 比较简单，dfs即可
```

35. 复杂链表的复制

- 请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null

```
# 这一题是前两刷留下的问题，一直没解决，不太好理解
# 涉及深拷贝问题
```

```
    # 遍历两次，第一次只建立一个字典，存储原结点和新结点的一一对应关系，第二次根据字典构造新建结点的next和random指针（注意使用get方法，否则指向None会报错）
    # 字节跳动面试题
    class Solution:
        def copyRandomList(self, head):
            if not head: return
            node = head
            dic = {}
            while node:
                dic[node] = Node(node.val)
                node = node.next
            node = head
            while node:
                dic[node].next = dic.get(node.next)
                dic[node].random = dic.get(node.random)
                node = node.next
            return dic[head]
```

36. 二叉搜索树与双向链表

- 输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向

```
    # 此题应该是剑指Offer系列中最难的二叉树题目
    # 1. 中序遍历各结点（有序的）
    # 2. 构建前驱结点self.pre，当走到最左侧时为其赋值，self.pre会不停变化
    # 3. 构建头结点self.head，也是最左侧时赋值，self.head固定不动
    # 4. 由底向上，每次构造当前结点与前躯结点的双向指针
    # 5. 递归完成后，更新self.head与self.pre的关系
    class Solution:
        def treeToDoublyList(self, root):
            if not root: return
            self.pre, self.head = None, None

            def dfs(node):
                if not node: return
```

```
            dfs(node.left)
            if not self.pre:
                self.pre = node
                self.head = node
            else:
                self.pre.right = node
                node.left = self.pre
            self.pre = node
            dfs(node.right)

        dfs(root)
        self.head.left, self.pre.right = self.pre, self.head
        return self.head
```

37. 序列化二叉树

- 请实现两个函数，分别用来序列化和反序列化二叉树

```
# 二叉树常规题
# 说白了就是把一个二叉树转成层次遍历生成的列表，以及把列表还原为二叉树
```

38. 字符串的排列

- 输入一个字符串，打印出该字符串中字符的所有排列
- 测试用例

```
输入: s = "abc"
输出: ["abc","acb","bac","bca","cab","cba"]
```

```
# 排列问题 dfs常规题
class Solution:
    def permutation(self, s: str):

        def dfs(s, cur_str):
            if not s:
```

```
                self.res.add(cur_str)
                return
            repeat = set()
            for i in range(len(s)):
                if s[i] not in repeat:
                    repeat.add(s[i])
                    dfs(s[:i] + s[i+1:], cur_str + s[i])

        self.res = set()
        dfs(s, '')
        return list(self.res)
```

39. 数组中出现次数超过一半的数字

- 数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。你可以假设数组是非空的，并且给定的数组总是存在多数元素
- 测试用例

```
输入：[1, 2, 3, 2, 2, 2, 5, 4, 2]
输出：2
```

```
# 特殊解法：排序后去中位数
# 这是一个想法超越算法的解法，只要想到，排序后直接获得结果，缺点是算法复杂度还是高了，还有更快的
```

```
# 特殊解法：投票法
```

40. 最小的k个数

- 输入整数数组 `arr` ，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
- 测试用例

```
输入：arr = [3,2,1], k = 2
输出：[1,2] 或者 [2,1]

输入：arr = [0,1,2,1], k = 1
输出：[0]
```

```python
# 经典题
# 面试经典题型，面试出的题目之所以精髓，就精髓在：每个人都能做出来，但从不同人的解法中能看出差距
# 基于快排的解法
class Solution:
    def getLeastNumbers(self, nums, k):
    self.res = []

        def fast_sort(left, right):
            if left > right or self.res: return
            i, j = left, right
            while i < j:
                while i < j and nums[j] >= nums[left]: j -= 1
                while i < j and nums[i] <= nums[left]: i += 1
                nums[i], nums[j] = nums[j], nums[i]
            nums[i], nums[left] = nums[left], nums[i]
            if i == k: self.res = nums[:k]
            elif i < k: fast_sort(i+1, right)
            else: fast_sort(left, i-1)

        if len(nums) == k: return nums
        fast_sort(0, len(nums) - 1)
        return self.res
```


41. 数据流中的中位数

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

```python
# 堆排序问题
# 这题如果利用最大堆，需要导入heapq库 Python只提供了最小堆的实现，最大堆可以通过添加负号间接实现
# 然而，不能期待利用heappush函数构造出一个有序的列表，用heappush只能保证列表索引为0的值为当前列表的最小值
from heapq import *
class MedianFinder:

    def __init__(self):
        self.s1 = []
        self.s2 = []

    def addNum(self, num: int) -> None:
        if len(self.s1) == len(self.s2):
            heappush(self.s2, -num)
            heappush(self.s1, -heappop(self.s2))
        else:
            heappush(self.s1, num)
            heappush(self.s2, -heappop(self.s1))
    def findMedian(self) -> float:
        if len(self.s1) == len(self.s2):
            return (self.s1[0] - self.s2[0]) / 2.0
        return self.s1[0]
```

```python
# 显然，当不熟悉库函数又不会自己写出堆函数时，二分插入是最佳策略
# 同样构建两个列表，各存一半，用二分插入构造两个有序列表
class MedianFinder:

    def __init__(self):
        self.s1 = []
        self.s2 = []

    def addNum(self, num: int) -> None:
        if len(self.s1) == len(self.s2):
            self.binary_insert(self.s2, num)
```

```
            self.binary_insert(self.s1, self.s2.pop(0))
        else:
            self.binary_insert(self.s1, num)
            self.binary_insert(self.s2, self.s1.pop())
        # print(self.s1, self.s2)

    def findMedian(self) -> float:
        if len(self.s1) == len(self.s2):
            return (self.s1[-1] + self.s2[0]) / 2.0
        return self.s1[-1]

    def binary_insert(self, nums, k):
        if not nums:
            nums.append(k)
            return
        left, right = 0, len(nums) - 1
        while left <= right:
            if k <= nums[left]:
                nums.insert(left, k)
                break
            elif k >= nums[right]:
                nums.insert(right+1, k)
                break
            else:
                mid = (left + right) // 2
                if nums[mid] < k:
                    left = mid + 1
                elif nums[mid] > k:
                    right = mid - 1
                else:
                    nums.insert(mid, k)
                    break
```

42. 连续子数组的最大和

- 输入一个整型数组，数组中的一个或连续多个整数组成一个子数组。求所有子数组的和的最大值
- 测试用例

```
输入: nums = [-2,1,-3,4,-1,2,1,-5,4]
输出: 6
解释: 连续子数组 [4,-1,2,1] 的和最大，为 6。
```

```
# 最简单的动态规划（dp），一维数组原地更新即可
```

43. 1～n整数中1出现的次数

- 输入一个整数 `n` ，求1～n这n个整数的十进制表示中1出现的次数。例如，输入12，1～12这些整数中包含1 的数字有1、10、11和12，1一共出现了5次
- 测试用例

```
输入: n = 12   输出: 5
输入: n = 13   输出: 6
```

```
# 比较难的逻辑推理题
# 1. 以1204为例，十位上值为0，high=12，low=4，十位出现1的数字范围是0010～1119，把十位遮住，数字范围是000～119，共计120种情况（high * digit）
# 2. 以1214为例，十位数上值为1，high=12，low4，十位出现1的数字范围是0010～1214，把十位遮住，数字范围是000～124，共计125种情况（high * digit + low + 1）
# 3. 以1234为例，十位数上值为3，high=12，low=4，十位数出现1的数字范围是0010～1219，把十位遮住，数字范围是000～129，共计130种情况（high + 1）* digit
# 考虑清楚 代码非常简单
class Solution:
    def countDigitOne(self, n: int) -> int:
        low, digit = 0, 1
        res = 0
        while n != 0:
```

```
            high = n // 10
            cur = n % 10
            if cur == 0: res += digit * high
            elif cur == 1: res += digit * high + low + 1
            else: res += (high + 1) * digit

            n //= 10
            low += digit * cur
            digit *= 10
        return res
```

44. 回文链表

- 请判断一个链表是否为回文链表
- 测试用例

```
输入: 1->2  # 输出: false
输入: 1->2->2->1  # 输出: true
```

```
# 这一题的难点在于如何在O(1)的空间复杂度条件下解决问题
# 将后半部分链表反转然后进行比较
# 恢复链表
class ListNode:
    def __init__(self, val):
        self.val = val
        self.next = None

root = ListNode(1)
head = root
for i in range(2, 6):
    head.next = ListNode(i)
    head = head.next

class Solution:
    def isPalindrome(self, head):
```

```
        def reverse(head):
            if not head or not head.next: return head
            pre, cur, post = head, head.next, head.next.next
            pre.next = None
            while post:
                cur.next = pre
                pre = cur
                cur = post
                post = post.next
            cur.next = pre
            return cur

        if not head or not head.next: return True
        p1, p2 = head, head.next
        left, right = head, None
        while p2 and p2.next:
            p2 = p2.next.next
            p1 = p1.next
        right = p1.next
        right = reverse(right)
        while right:
            if right.val != left.val:
                return False
            right = right.next
            left = left.next

        return True
```


44. 数字序列中某一位的数字

- 数字以0123456789101112131415...的格式序列化到一个字符序列中。在这个序列中，第5位（从下标0开始计数）是5，第13位是1，第19位是4，等等。写一个函数，求任意第n位对应的数字
- 测试用例

```
输入：n = 3  # 输出：3
输入：n = 11  # 输出：0
```

```
# 逻辑推理问题 先找规律 没有那么复杂
```

45. 把数组排成最小的数

- 输入一个非负整数数组，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
- 测试用例

```
输入：[3,30,34,5,9]
输出："3033459"
```

```
# 经典题
# 这种题目，第一次做，没有头绪，把数字转字符串后硬排
# 实际上转成字符串后 x + y > y + x，就表示x > y
# 理解上述思路后用快排即可
```

46. 把数字翻译成字符串

- 给定一个数字，我们按照如下规则把它翻译为字符串：0 翻译成 “a”，1 翻译成 “b”，……，11 翻译成 “l”，……，25 翻译成 “z”。一个数字可能有多个翻译。请编程实现一个函数，用来计算一个数字有多少种不同的翻译方法
- 测试用例

```
输入：12258
输出：5
解释：12258有5种不同的翻译，分别是"bccfi"，"bwfi"，"bczi"，"mcfi"和"mzi"
```

```
# 深度优先策略解决
# 构建一个函数用于抽取当前位下的两位数
```

47. 礼物的最大价值

- 在一个 m*n 的棋盘的每一格都放有一个礼物，每个礼物都有一定的价值（价值大于 0）。你可以从棋盘的左上角开始拿格子里的礼物，并每次向右或者向下移动一格、直到到达棋盘的右下角。给定一个棋盘及其上面的礼物的价值，请计算你最多能拿到多少价值的礼物?
- 测试用例

```
输入:
[
  [1,3,1],
  [1,5,1],
  [4,2,1]
]
输出: 12
解释: 路径 1→3→5→2→1 可以拿到最多价值的礼物
```

```
# 标准动态规划问题 原地修改就行 这不是小学二年级的题
```

48. 最长不含重复字符的子字符串

- 请从字符串中找出一个最长的不包含重复字符的子字符串，计算该最长子字符串的长度
- 测试用例

```
输入: "abcabcbb"
输出: 3
解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。
```

```
# 面试高频题
# 双指针
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if not s: return 0
        res, dic = 0, []
        left, right = 0, 0
```

```
        for idx, k in enumerate(s):
            right += 1
            if k not in dic:
                res = max(res, right - left)
            else:
                index = dic.index(k)
                left += index + 1
                dic = dic[index+1:]
            dic.append(k)

        return res
```

```
# 双指针+哈希
# 来自Krahets大佬
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        dic, res, i = {}, 0, -1
        for j in range(len(s)):
            if s[j] in dic:
                i = max(dic[s[j]], i) # 更新左指针 i
            dic[s[j]] = j # 哈希表记录
            res = max(res, j - i) # 更新结果
        return res
```


49. 丑数

- 我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数
- 测试用例

```
输入: n = 10
输出: 12
解释: 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 是前 10 个丑数
```

```
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        res = [1]
        l1, l2, l3 = 0, 0, 0
        while len(res) != n:
            cur_max = min(res[l1] * 2, res[l2] * 3, res[l3] * 5)
            if res[l1] * 2 == cur_max: l1 += 1
            if res[l2] * 3 == cur_max: l2 += 1
            if res[l3] * 5 == cur_max: l3 += 1
            res.append(cur_max)
        return res[-1]
```

50. 第一个只出现一次的字符

- 在字符串 s 中找出第一个只出现一次的字符。如果没有，返回一个单空格。 s 只包含小写字母

```
# 有序哈希表（python3.6以后的字典默认有序）
# 或者遍历字符串两次也可以
```

51. 数组中的逆序对

- 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
- 测试用例

```
输入：[7,5,6,4]
输出：5
```

```
# 类归并排序解法
# 这一题之前如果没有做过 很难下手
class Solution:
    def reversePairs(self, nums) -> int:
        self.res = 0
```

```
        def merge(l1, l2):
            if not l2: return l1
            idx = 0
            for num in l1:
                while idx < len(l2) and num > l2[idx]:
                    idx += 1
                self.res += idx
            return sorted(l1 + l2)

        def merge_sort(nums):
            if len(nums) < 2: return nums
            left = merge_sort(nums[:len(nums)//2])
            right = merge_sort(nums[len(nums)//2:])
            return merge(left, right)

        merge_sort(nums)
        return self.res
```

52. 两个链表的第一个公共节点

- 输入两个链表，找出它们的第一个公共节点
- 测试用例

```
输入：intersectVal = 8, listA = [4,1,8,4,5], listB = [5,0,1,8,4,5], skipA = 2, skipB = 3
输出：Reference of the node with value = 8
输入解释：相交节点的值为 8 （注意，如果两个列表相交则不能为 0）。从各自的表头开始算起，链表 A 为 [4,1,8,4,5]，链表 B 为 [5,0,1,8,4,5]。在 A 中，相交节点前有 2 个节点；在 B 中，相交节点前有 3 个节点。
```

```
# 浪漫相遇问题 略
```

53. 在排序数组中查找数字

- 统计一个数字在排序数组中出现的次数
- 测试用例

```
输入: nums = [5,7,7,8,8,10], target = 8
输出: 2
```

```
# 二分搜索算法
class Solution:
    def search(self, nums, target: int) -> int:

        def binary_insert(nums, k):
            left, right = 0, len(nums) - 1
            while left < right:
                if nums[left] >= k: return left
                if nums[right] < k: return right + 1
                mid = (left + right) // 2
                if nums[mid] >= k:
                    right = mid - 1
                elif nums[mid] < k:
                    left = mid + 1
            return left if nums[left] >= k else left + 1
        if not nums: return 0
        return binary_insert(nums, target+1) - binary_insert(nums,
target)
```

54. 二叉搜索树的第k大节点

- 给定一棵二叉搜索树，请找出其中第k大的节点

```
# 中序遍历倒序即可
```

55. 二叉树的深度

- 输入一棵二叉树的根节点，求该树的深度。从根节点到叶节点依次经过的节点（含根、叶节点）形成树的一条路径，最长路径的长度为树的深度

```
# 二叉树经典题型
# 传值型的深度优先
```

55-II. 平衡二叉树

- 输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树

```
# 传值型深度优先
```

56. 数组中数字出现的次数

- 在一个数组 `nums` 中除一个数字只出现一次之外，其他数字都出现了三次。请找出那个只出现一次的数字

```
# 略吧 这个如果是考察位运算 我认输
```

57. 和为s的两个数字

- 输入一个递增排序的数组和一个数字s，在数组中查找两个数，使得它们的和正好是s。如果有多对数字的和等于s，则输出任意一对即可

```
# 双指针
```

57-II. 和为s的连续正数序列

- 输入一个正整数 `target` ，输出所有和为 `target` 的连续正整数序列（至少含有两个数）。序列内的数字由小到大排列，不同序列按照首个数字从小到大排列
- 测试用例

```
输入: target = 15
输出: [[1,2,3,4,5],[4,5,6],[7,8]]
```

```
# 常规解法: 枚举 + 暴力
```

```
# 双指针法 左右指针均只增不减
```

58. 翻转单词顺序

- 输入一个英文句子，翻转句子中单词的顺序，但单词内字符的顺序不变。为简单起见，标点符号和普通字母一样处理。例如输入字符串"I am a student. "，则输出"student. a am I"
- 测试用例

```
输入: "the sky is blue"
输出: "blue is sky the"

输入: "  hello world!  "
输出: "world! hello"

输入: "a good   example"
输出: "example good a"
```

```
class Solution:
    def reverseWords(self, s: str) -> str:
        alist = s.strip().split(' ')
        blist = []
        for k in alist:
            if k: blist.append(k)
        blist.reverse()
        return ' '.join(blist)
```

59. 滑动窗口的最大值

- 给定一个数组 `nums` 和滑动窗口的大小 `k`，请找出所有滑动窗口里的最大值
- 测试用例

```
输入: nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出: [3,3,5,5,6,7]
```

```python
# 单调栈问题 这题实际并不好做
# 面试经典题
# 一定要注意，这里实际上存的是递减的单调栈
class Solution:
    def maxSlidingWindow(self, nums, k):
        res, stack = [], []
        if not nums: return res

        for i in range(k):
            while stack and stack[-1] < nums[i]:
                stack.pop()
            stack.append(nums[i])

        for i in range(k, len(nums)):
            res.append(stack[0])
            if nums[i-k] == stack[0]: stack.pop(0)
            while stack and stack[-1] < nums[i]:
                stack.pop()
            stack.append(nums[i])
        res.append(stack[0])
        return res
```


60. n个骰子的点数

- 把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率
- 测试用例

```
输入: 1
输出: [0.16667,0.16667,0.16667,0.16667,0.16667,0.16667]
```

```python
# 这一题官方都是动态规划解法，暂时没有细看
class Solution:
    def dicesProbability(self, n: int):
        dic = {1: 1/6, 2: 1/6, 3: 1/6, 4: 1/6, 5: 1/6, 6: 1/6}
        for i in range(1, n):
            new_dic = {}
            for key, val in dic.items():
                for i in range(1, 7):
                    if key + i in new_dic:
                        new_dic[key+i] += val * 1/6
                    else:
                        new_dic[key+i] = val * 1/6
            dic = new_dic
        res = []
        for key, val in dic.items():
            res.append(val)
        return res
```


61. 扑克牌中的顺子

- 从扑克牌中随机抽5张牌，判断是不是一个顺子，即这5张牌是不是连续的。2～10为数字本身，A为1，J为11，Q为12，K为13，而大、小王为 0 ，可以看成任意数字。A 不能视为 14
- 测试用例

```
输入: [1,2,3,4,5]
输出: True

输入: [0,0,1,2,5]
输出: True
```

```python
# 常规解法：排序 + 遍历
class Solution:
    def isStraight(self, nums) -> bool:
        nums.sort()
        cnt = 0
        for idx, n in enumerate(nums):
            if n == 0: cnt += 1
            else: break
        pre = nums[cnt]
        i = cnt + 1
        while i < len(nums):
            if nums[i] == pre + 1:
                i += 1
                pre += 1
            else:
                if cnt != 0:
                    cnt -= 1
                    pre += 1
                else: return False

        return True
```

```python
# 给出充分条件：
# 1. 所有牌无重复（0除外）
# 2. 五张牌中，max - min < 5
class Solution:
    def isStraight(self, nums) -> bool:
        alist = set()
        for n in nums:
            if n == 0: continue
            else:
                if n in alist: return False
                else: alist.add(n)
        return max(alist) - min(alist) < 5
```

62. 圆圈中最后剩下的数字

- 0,1,⋯,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字（删除后从下一个数字开始计数）。求出这个圆圈里剩下的最后一个数字。例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3
- 测试用例

```
输入: n = 5, m = 3
输出: 3

输入: n = 10, m = 17
输出: 2
```

```
# 传统解法: 构建循环链表模拟实现
```

```
# 数学解法 公式递推
class Solution:
    def lastRemaining(self, n: int, m: int) -> int:
        res = 0
        for i in range(2, n+1):
            res = (res + m) % i
        return res
```


63. 股票的最大利润

- 假设把某股票的价格按照时间先后顺序存储在数组中，请问买卖该股票一次可能获得的最大利润是多少?
- 测试用例

输入：[7,1,5,3,6,4]
输出：5
解释：在第 2 天（股票价格 = 1）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，最大利润 = 6-1 = 5 。
注意利润不能是 7-1 = 6，因为卖出价格需要大于买入价格。

```python
# 如果只能购买一次，构造递减单调栈即可
class Solution:
    def maxProfit(self, nums):
        if not nums: return 0
        res = 0
        stack = [nums[0]]
        for n in nums:
            if stack[-1] >= n:
                stack.append(n)
            else:
                res = max(res, n - stack[-1])
        return res
```

```python
# 如果可以购买多次 单调栈都不用 太简单了～
class Solution:
    def maxProfit(self, nums):
        if not nums: return 0
        res = 0
        pre = nums[0]
        for n in nums:
            if n > pre:
                res += n - pre
            pre = n
        return res
```

66. 构建乘积数组

- 给定一个数组 A[0,1,...,n-1]，请构建一个数组 B[0,1,...,n-1]，其中 B[i] 的值是数组 A 中除了下标 i 以外的元素的积, 即 B[i]=A[0]×A[1]×...×A[i-1]×A[i+1]×...×A[n-1]。不能使用除法
- 测试用例

```
输入: [1,2,3,4,5]
输出: [120,60,40,30,24]
```

```python
# 动态规划的思路
# 实际上不用想的太复杂，简单来说，为了减少不必要的计算，就用空间来换时间，利用两个数组，分别存储正向累乘和逆向累乘的结果，然后对位相乘即可得到每一个结果
class Solution:
    def constructArr(self, a):
        if not a: return []
        l1, l2 = [1], [1]
        for i in range(len(a) - 1):
            l1.append(l1[-1] * a[i])
        for i in range(len(a) - 1, 0, -1):
            l2.append(l2[-1] * a[i])
        res = []
        for i in range(len(l1)):
            res.append(l1[i] * l2[-i-1])
        return res
```

67. 把字符串转换成整数

- 写一个函数 StrToInt，实现把字符串转换成整数这个功能。不能使用 atoi 或者其他类似的库函数
- 测试用例

```
输入: "42" 输出: 42
输入: "   -42" 输出: -42
输入: "42 with words" 输出: 42
输入: "words and 42" 输出: 0
输入: "-1123123123123123123" 输出: -2 ** 31
```

68-I. 二叉搜索树的最近公共祖先

- 给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先
- 测试用例

```
输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出: 6
解释: 节点 2 和节点 8 的最近公共祖先是 6。
```

```python
# 二叉搜索树找最近公共祖先是有简单方法的
class Solution:
    def lowestCommonAncestor(self, root, p, q):
        if not root: return
        self.res = None
        def recur(node):
            if self.res or not node: return
            if node.val == p.val or node.val == q.val:
                self.res = node
                return
            elif node.val < p.val and node.val < q.val:
                recur(node.right)
            elif node.val > p.val and node.val > q.val:
                recur(node.left)
            else:
                self.res = node
                return

        recur(root)
        return self.res
```

68-II. 二叉树的最近公共祖先

- 给定一个二叉树, 找到该树中两个指定节点的最近公共祖先
- 测试用例

```
输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
```

```python
# 二叉树的最近公共祖先需要用到recur函数的返回值了
class Solution:
    def lowestCommonAncestor(self, root, p, q):
        self.res = None

        def recur(node):
            if not node or self.res: return False
            cur = True if node.val == p.val or node.val == q.val
else False
            left = recur(node.left)
            right = recur(node.right)
            if (left and right) or (left and cur) or (right and
cur):
                self.res = node
                return False
            elif not left and not right and not cur: return False
            else: return True

        recur(root)
        return self.res
```