刷题笔记

Zhengjv

2021 年 5 月 5 日

目录

1 两数之和

给定一个整数数组 nums 和一个整数目标值 target，请你在该数组中找出和为目标值的那两个整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现

你可以按任意顺序返回答案。

示例 1:
输入: nums = [2,7,11,15], target = 9
输出: [0,1]
解释: 因为 nums[0] + nums[1] == 9 ，返回 [0, 1] 。
示例 2:
输入: nums = [3,2,4], target = 6
输出: [1,2]
示例 3:
输入: nums = [3,3], target = 6
输出: [0,1]

1.1 暴力解法

1.1.1 思路

看到题目首先想到暴力解法，利用两层循环遍历数组

1.1.2 代码

```
class Solution(object):
    def twoSum(self, nums, target):
        lis = []
        for i in range(len(nums)):
    for j in range(len(nums)):
        if i==j:
                countinue
            if nums[i]+nums[j]==target:
            return [i,j]
```

1.2 优化解法

1.2.1 思路

由于只有一种结果，每拿出一个数在后面剩下的数组中查找 target-当前 num

若找到，将其下表标分别返回 1 即可。

1.2.2 代码

```
class Solution(object):
    def twoSum(self, nums, target):
        lis = []
        for id, num in enumerate(nums):
            if (target - num) in nums[id+1:]:
                lis.append(id)
                lis.append(nums[id+1:].index(target - num)+id+1)
                break
        return lis
```

1.3 哈希查找

1.3.1 思路

利用 python 字典构建哈希函数，实现复杂度为 O（n）的查找 1

1.3.2 代码

```
class Solution:
    def twoSum(self, nums, target):
        hashmap = {}
        for index, num in enumerate(nums):
            another_num = target - num
            if another_num in hashmap:
                return [hashmap[another_num], index]
            hashmap[num] = index
        return None
```

2 两数相加

给你两个非空的链表，表示两个非负的整数。它们每位数字都是按照逆序的方式存储的，并且每个节点只能存储一位数字。

请你将两个数相加，并以相同形式返回一个表示和的链表。

你可以假设除了数字 0 之外，这两个数都不会以 0 开头。

示例 1：

输入：l1 = [2,4,3], l2 = [5,6,4]、

输出：[7,0,8]

解释：342 + 465 = 807.

示例 2：

输入：l1 = [0], l2 = [0]

输出：[0]

示例 3：

输入：l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]

输出：[8,9,9,9,0,0,0,1]

2.1 解决方法

2.1.1 思路

对于两个链表分别进行遍历，将其数字 val 提取出来，然后对于不同位乘不同倍数加起来得整型 sum。

再将其两个 sum 加起来，得整型的和，利用 str(sum) 方法转换为字符串。

再切片为逆序，将其添加在新的链表中即可。

2.1.2 代码

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def addTwoNumbers(self, l1, l2):
        p1=l1
        p2=l2
        ex=1
        sum=0
```

```
        while p1 is not None:
            sum=sum+p1.val*ex
            ex=ex*10
            p1=p1.next
        ex=1
        while p2 is not None:
            sum=sum+p2.val*ex
            ex=ex*10
            p2=p2.next
        ssum=str(sum)
        re=ListNode(0)
        p3=re
        for s in ssum[::-1]:
            p3.next=ListNode(int(s))
            p3=p3.next
        return re.next
```

3 无重复最长字符串

给定一个字符串，请你找出其中不含有重复字符的最长子串的长度。
示例 1:
输入: s = "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。
示例 2:
输入: s = "bbbbb"
输出: 1
解释: 因为无重复字符的最长子串是"b"，所以其长度为 1。
示例 3:
输入: s = "pwwkew"
输出: 3
解释: 因为无重复字符的最长子串是"wke"，所以其长度为 3。

请注意，你的答案必须是子串的长度，”pwke” 是一个子序列，不是子串。

示例 4:

输入: s = ””

输出: 0

3.1 解决方法

3.1.1 思路

想到做两数之和时候的哈希方法 1，于是将字符串遍历，建立字典用于存放哈希键值对，键为字符，val 为当前的 index

建立一个列表 1 用于存放相同字符之间的距离，每次经过一个字符便将其用字典存起来，若遍历时候检查到此字符已经出现过，

则将其与上一次出现时的距离计算出来，添加入列表之中。最后遍历完后只需要返回列表的最大值即可，实现了 O(N) 的复杂度。

3.1.2 代码

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        hashmap={}
        lis=[0]
        for idx,c in enumerate(s):
            if c in hashmap.keys():
                lis.append(idx-hashmap[c])
            hashmap[c] = idx
        return max(lis)
```

4 寻找两个正序数组的中位数

给定两个大小分别为 m 和 n 的正序 (从小到大) 数组 nums1 和 nums2。请你找出并返回这两个正序数组的中位数。

示例 1:

输入: nums1 = [1,3], nums2 = [2]

输出：2.00000
解释：合并数组 = [1,2,3] ，中位数 2
示例 2：
输入：nums1 = [1,2], nums2 = [3,4]
输出：2.50000
解释：合并数组 = [1,2,3,4] ，中位数 (2 + 3) / 2 = 2.5
示例 3：
输入：nums1 = [0,0], nums2 = [0,0]
输出：0.00000
示例 4：
输入：nums1 = [], nums2 = [1]
输出：1.00000
示例 5：
输入：nums1 = [2], nums2 = []
输出：2.00000

4.1 解决方法

4.1.1 思路

自然而然，先将两个数组合并，然后排序，然后分奇数偶数讨论，计算出中位数

4.1.2 代码

```
class Solution(object):
    def findMedianSortedArrays(self, nums1, nums2):
        """
        :type nums1: List[int]
        :type nums2: List[int]
        :rtype: float
        """
        nums1.extend(nums2)
        nums1.sort()
```

```
        if len(nums1)//2==0:
            return (nums1[int(len(nums1)/2)]
                                        +nums1[int(len(nums1)/2-1)])/2
        else:
            return nums1[int((len(nums1)-1)/2)]
```

4.2 优化方法

4.2.1 思路

由于做了复杂度的要求，自然而然，想到利用一些其他复杂度较低的排序算法，

可以采用插入排序，将较短的数组向较长的数组中插入

可实现降低复杂度，也可以采用二分方法，利用二路归并排序进行排序。

由于数组本身局部有序，所以采用快速排序的方法更为合适。

4.2.2 代码

```
class Solution:
    def findMedianSortedArrays(self, nums1, nums2):
        def getKthElement(k):

            index1, index2 = 0, 0
            while True:
                # 特殊情况
                if index1 == m:
                    return nums2[index2 + k - 1]
                if index2 == n:
                    return nums1[index1 + k - 1]
                if k == 1:
                    return min(nums1[index1], nums2[index2])

                # 正常情况
                newIndex1 = min(index1 + k // 2 - 1, m - 1)
```

```
            newIndex2 = min(index2 + k // 2 - 1, n - 1)
            pivot1, pivot2 = nums1[newIndex1], nums2[newIndex2]
            if pivot1 <= pivot2:
                k -= newIndex1 - index1 + 1
                index1 = newIndex1 + 1
            else:
                k -= newIndex2 - index2 + 1
                index2 = newIndex2 + 1

        m, n = len(nums1), len(nums2)
        totalLength = m + n
        if totalLength % 2 == 1:
            return getKthElement((totalLength + 1) // 2)
        else:
            return (getKthElement(totalLength // 2)
                + getKthElement(totalLength // 2 + 1)) / 2
```

5 寻找最长回文

给你一个字符串 s，找到 s 中最长的回文子串。
示例 1:
输入: s = "babad"
输出: "bab"
解释: "aba" 同样是符合题意的答案。
示例 2:
输入: s = "cbbd"
输出: "bb"
示例 3:
输入: s = "a"
输出: "a"
示例 4:
输入: s = "ac"
输出: "a"

5.1 暴力方法

5.1.1 思路

找出当前字符串长度大于 2 的所有子串，
然后判断其是否为回文。依次从长串向短串判断，即可得出最长回文字串

5.1.2 代码

```
class Solution(object):
    def longestPalindrome(self, s):
        max_length = len(s)
        if (max_length == 0):
            return ""
        while (max_length >= 1):
            for i in range(len(s)):
                right = i + max_length - 1
                if (right < len(s)):
                    if (i == right):
                        newstr = s[i]
                    else:
                        newstr = s[i:right+1]
                    if (self.IsPalindrome(newstr)):
                        return newstr
                else:
                    break
            max_length -= 1
        return None
    def IsPalindrome(self,str):  # 用来判断一个字符串是否为回文子串
        length = len(str)
        left = 0
        right = length - 1

        while (left < right):
```

```
            if (str[left] == str[right]):
                left += 1
                right -= 1
            else:
                return False
        return True
```

5.2 动态规划方法

5.2.1 思路

首先是求最优问题，首先想到动态规划三要素：最优子结构、边界、状态转移方程

其中，

状态转移方程：f(i,j)：s[i]==s[j] i-j<=1

f(i,j)：s[i]==s[j] and f(i+1,j-1) j-i>1

其中：f(i,j) 表示 s[i:j+1] 是否回文串

当 j-i<=1 时，如果 s[i] == s[j] 则表示 s[i:j] 为回文串，及 f(i,j) = true，否则 f(i,j) = false。

当 j-i > 1 时，则判断 s[i]、s[j] 是否相等以及 f(i+1, j-1) 是否为 true，即 s[i+1:j-1] 是否为回文串，如果为真，则 f(i,j) = true

所以就需要一个 n*n 的二维矩阵用于存储 f(i,j) 的值，其中 j in range(0, k)，i in range(0, j+1)。

5.2.2 代码

```
class Solution(object):
    def longestPalindrome(self,s):
        k=len(s)
        matrix=[[0 for i in range(k)] for j in range(k)]
        longestString=""
        longestLen=0
        for j in range(0,k):
            for i in range(0,j+1):
                if j-i<=1:
```

```
                if s[i]==s[j]:
                    matrix[i][j]=1
                    if longestLen<j-i+1:
                        longestString=s[i:j+1]
                        longestLen=j-i+1
            else:
                if s[j]==s[i] and matrix[i+1][j-1]:
                    matrix[i][j]=1
                    if longestLen<j-i+1:
                        longestString=s[i:j+1]
                        longestLen=j-i+1
        return longestString
```

6 Z 字形变换

将一个给定字符串 s 根据给定的行数 numRows ，以从上往下、从左到右进行 Z 字形排列。

比如输入字符串为"PAYPALISHIRING" 行数为 3 时，排列如下：

P A H N

A P L S I I G

Y I R

之后，你的输出需要从左往右逐行读取，产生出一个新的字符串，比如："PAHNAPLSIIGYIR"。

请你实现这个将字符串进行指定行数变换的函数：

string convert(string s, int numRows);

示例 1：

输入：s = "PAYPALISHIRING", numRows = 3

输出："PAHNAPLSIIGYIR"

示例 2：

输入：s = "PAYPALISHIRING", numRows = 4

输出："PINALSIGYAHRPI"

解释：

P I N

A L S I G
Y A H R
P I
示例 3:
输入: s = "A", numRows = 1
输出: "A"

6.1 解决方法

6.1.1 思路

很容易想到利用二维矩阵来存字符串，由于 z 字形的特殊之处，可以逐列进行储存，

然后可以轻松地确定字符位于 Z 字形图案中的哪一行。

从左到右迭代 s，将每个字符添加到合适的行。先将 z 字形矩阵的长宽算出来，

将没有字符的地方用“0”代替，遍历 s 添加进矩阵中。

最后用 replace 方法剔除即可。

6.1.2 代码

```
class Solution(object):
    def convert(self, s, numRows):
        """
        :type s: str
        :type numRows: int
        :rtype: str
        """
        t=numRows*3-2
        num=len(s)//t+1


        row0=row=[]
        list=[]
        for i in range(numRows):
```

```
            row0.append("#")
        k=0

        for i in range(num):
            for j in range(numRows):
                for e in range(numRows):
                    if j==0 or j==numRows:
                        row[e]=s[k]
                        k=k+1
                    else:
                        row[numRows-j]=s[k]
                        k=k+1
                list.append(row)
                row=row0

        s2=[]
        for j in range(numRows):
            for i in range(len(list)):
                s2.append(list[j][i])
        return str(s2).replace("#","")
```

7 整数反转

给你一个 32 位的有符号整数 x ，返回将 x 中的数字部分反转后的结果。

如果反转后整数超过 32 位的有符号整数的范围 $[-231, 231 - 1]$ ，就返回 0。

假设环境不允许存储 64 位整数（有符号或无符号）。

示例 1:

输入: x = 123

输出: 321

示例 2:

输入: x = -123

输出: -321
示例 3:
输入: x = 120
输出: 21
示例 4:
输入: x = 0
输出: 0

7.1 解决方法

7.1.1 思路

转换为字符串操作，最后再转换为整形拼接回去。

7.1.2 代码

```
class Solution(object):
    def reverse(self, x):
        """
        :type x: int
        :rtype: int
        """
        ex=1
        t=0
        sum=0
        if x<0:
            x=-x
            t=1
        s=str(x)
        for i in s:
            sum=sum+int(i)*ex
            ex=ex*10
        if t==0:
            return sum
        else:
```

```
        return -sum
```

7.2 官方题解

7.2.1 思路

考虑大数的数学方法

7.2.2 代码

```
class Solution:
    def reverse(self, x: int) -> int:
        INT_MIN, INT_MAX = -2**31, 2**31 - 1

        rev = 0
        while x != 0:
            # INT_MIN 也是一个负数，不能写成 rev < INT_MIN // 10
            if rev < INT_MIN // 10 + 1 or rev > INT_MAX // 10:
                return 0
            digit = x % 10
            # Python3 的取模运算在 x 为负数时
                        也会返回 [0, 9) 以内的结果，
                        因此这里需要进行特殊判断
            if x < 0 and digit > 0:
                digit -= 10

            # 同理，Python3 的整数除法在
                 x 为负数时会向下（更小的负数）取整，
                        因此不能写成 x //= 10
            x = (x - digit) // 10
            rev = rev * 10 + digit

        return rev
```