剑指offer（python 版）41-44

41和为S的连续正数序列

题目：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

思路：

解题思路

S除奇数时，如果余数为0时，存在这样的序列；

S除偶数时，如果余数为偶数的一半，则存在这样的序列；

需要确保序列的最左端（最小数）值大于0.

铺开来说：

1）由于我们要找的是和为S的连续正数序列，因此这个序列是个公差为1的等差数列，而这个序列的中间值代表了平均值的大小。假设序列长度为n，那么这个序列的中间值可以通过（S/n）得到，知道序列的中间值和长度，也就不难求出这段序列了。

2）满足条件的n分两种情况：

n为奇数时，序列中间的数正好是序列的平均值，所以条件为：(n & 1) == 1 && sum % n == 0；

n为偶数时，序列中间两个数的平均值是序列的平均值，而这个平均值的小数部分为0.5，所以条件为：(sum % n) \* 2 == n.

3）由题可知n >= 2，那么n的最大值是多少呢？我们完全可以将n从2到S全部遍历一次，但是大部分遍历是不必要的。为了让n尽可能大，我们让序列从1开始，

根据等差数列的求和公式：S = (1 + n) \* n / 2，知

代码：

初始版：

1. **def** FindContinuousSequence(self, tsum):
2. div = 2
3. res, soleArray = [],[]
4. **while** tsum//div >= div//2:
5. **if** (div%2)==0 **and** (tsum%div)==(div//2):
6. start = tsum//div - div//2 + 1
7. **while** start <= tsum//div + div//2:
8. soleArray.append(start)
9. start += 1
10. res.append(soleArray)
11. **elif** (div%2)==1 **and** tsum%div==0 **and** tsum//div > div//2:
12. start = tsum//div - div//2
13. **while** start <= tsum//div + div//2:
14. soleArray.append(start)
15. start += 1
16. res.append(soleArray)
17. div += 1
18. soleArray = []
19. **return** sorted(res)

改进版：

1. # -\*- coding:utf-8 -\*-
2. **import** math
3. **class** Solution:
4. **def** FindContinuousSequence(self, tsum):
5. # write code here
6. res=[]
7. **for** div **in** range(2,int(math.sqrt(tsum\*2))+1):
8. **if** (div%2==0 **and** tsum%div==div/2) **or** (div%2==1 **and** tsum%div==0):
9. start=tsum//div-div//2+1 **if** div%2==0 **else** tsum//div-div//2
10. res.append(list(range(start,tsum//div+div//2+1)))
11. **return** sorted(res)

42和为S的两个数字

题目：输入一个递增排序的数组和一个数字S，在数组中查找两个数，使得他们的和正好是S，如果有多对数字的和等于S，输出两个数的乘积最小的。

思路：注意题目给出的是个有序的数组，那么可以考虑类似二分的思路。但其实不是二分，而是左右指针向中间移动。在移动的过程中由于只移动一步，且是根据大小调节的指针，所以不会“错过”要查找的target。

另外注意题目中要求的点，第一，小的数字在前面，这个没问题。第二，题目中要求当存在多个组合的时候，返回其中的乘积最小的一个。我们从两头向中间的移动过程中找到的第一组一定是乘积最小的。原因如下：

我们把两个数字想成矩形的两条边，根据中学的知识，当两条边越接近，面积越大（乘积越大）。由于从两头向中间进行查找的，找到的第一个组合一定是边差距最大的，所以乘积最小

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** FindNumbersWithSum(self, array, tsum):
4. # write code here
5. **if** **not** array:
6. **return** []
7. left,right=0,len(array)-1
8. **while** left<right:
9. sum=array[left]+array[right]
10. **if** sum>tsum:
11. right-=1
12. **elif** sum<tsum:
13. left+=1
14. **else**:
15. **return** [array[left],array[right]]
16. **return** []

43左旋转字符串

题目：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

思路：把str转换成list，先把0～n - 1翻转，然后把n ~ len() - 1翻转，最后再翻转。，直接join列表就变成str了

例如：

s = ‘abcdefg’ n=2

1.把字符串从[0:n-1]翻转,s = ‘bacdefg’ 是前后交换位置翻转

2.把翻转一次的字符串再次翻转[n:len(s)-1]，s = ‘bagfedc’

3.把翻转两次的字符串再次整体翻转[0:len(s)-1], s = ‘cdegfab’ over

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** LeftRotateString(self, s, n):
4. # write code here
5. **if** **not** s:
6. **return** s
7. s=list(s)
8. self.reverse(s,0,n-1)
9. self.reverse(s,n,len(s)-1)
10. self.reverse(s,0,len(s)-1)
11. **return** ''.join(s)
12. **def** reverse(self,s,l,r):
13. **while** l<r:
14. s[l],s[r]=s[r],s[l]
15. l+=1
16. r-=1
17. **return** s

44翻转单词顺序列

题目：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

思路一：按空格切分为数组，翻转数组，再用空格连接（算法题中一般不这样用，要再回顾一下切片的知识）

代码一：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** ReverseSentence(self, s):
4. # write code here
5. **if** s **is** None **or** len(s) == 0:
6. **return** s
7. **return** " ".join(s.split(' ')[::-1]

思路二：按空格切分为数组，依次入栈，再出栈(用空格连接)

代码二：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** ReverseSentence(self, s):
4. # write code here
5. **if** s **is** None **or** len(s) == 0:
6. **return** s
7. stack = []
8. **for** i **in** s.split(' '):
9. stack.append(i)
10. ans = ""
11. **while** len(stack) > 0:
12. ans += stack.pop() + " "
13. ans = ans[:-1]
14. **return** ans

思路三：两次翻转，先将字符串全部原地翻转。使用双指针，分别对应前后两个空格的位置，翻转两指针之间的子字符串，即单词。

代码三：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** ReverseSentence(self, s):
4. # write code here
5. **if** s **is** None **or** len(s) == 0 **or** " " **not** **in** s:
6. **return** s
7. # 第一次翻转
8. s = s[::-1]
9. l = 0
10. r = s.index(" ")
11. ans = ""
12. **while** r < len(s):
13. # 第二次局部翻转
14. ans += s[l: r][::-1] + " "
15. l = r + 1
16. r += 1
17. **while** r < len(s) **and** s[r] != " ":
18. r += 1
19. ans += s[l: r][::-1]
20. **return** ans