学习目标

字符串本身不是一种数据结构，但是由于其本身的特殊性，可以产生很多特殊的算法题。另外，字符串在工程里也有非常广泛的应用，因此一直都是算法考察的重点问题之一，我们有必要认真研究一下相关的问题。另外，python提供了很多内置方法，参考地址https://docs.python.org/zh-cn/3/library/stdtypes.html#string-methods，我们要熟悉这些方法是如何用的，并且要注意思考自己如何实现这些方法。

学习的过程中要注意以下几个问题：

1.字符串的特性以python如何管理字符串的

2.了解python提供的内置处理方法，并且能够自己给出实现思路。

3.认真研究《2.1.2》中字符串转整数的问题。

4.掌握其他常见问题的解决思路和实现方法。

1.python中的字符串

1.1. 字符串介绍

字符串（String）是字符序列，或者说是一串字符。字符只是一个符号,例如，英语具有26个字符。Python 不支持单字符类型，单字符在 Python 中也是作为一个字符串使用。

Python非常简单，并没有专门分出一个char（Character）类型（搞过C/Java的同学都熟悉）。通过将字符括在单引号或双引号中来创建字符串，如下：

```
varS = 'Hello World!'
```

Python中可以使用三引号，但通常用于表示多行字符串和文档字符串，三引号字符串可以扩展多行，例如;

```
if __name__ == '__main__':
    my_str = """Hello, welcome to
          the world of Python"""
    print my_str
```

输出结果为：

```
Hello, welcome to
    the world of Python
```

与Java等一样，python中字符串的索引也是从0开始的，如下图所示：

与其他语言不同的是，Python允许负索引，-1 为从末尾的开始位置。

1.2 python2和3里的字符串类型

在python2中，字符串一般有两种类型，unicode和str，（python3中是Unicode类型）。

str类型，字节码类型，根据某种编码把字符串转成对应的字节，一个字符根据不同的编码规则对应不同的字节数。

常见的编码有两种，一个是GBK中文编码，一个字符对应两个字节。更通用一点的是unicode类型，是用unicode编码的字符串，一个字符对应两个字节，能够对目前全世界的语言进行编码，因此适用面更广一些。

声明一个字符串的时候，如果直接赋值字符串，则类型为str，会按照开头的encoding来编码成对应的字节。如果赋值的时候在字符串前面加个u，类型则为unicode，如下所示：

```
# coding=utf-8
s1 = "字节串"
print(type(s1)) #输出  <type 'str'>，按照开头的encoding来编码成相应的字节
print(len(s1)) #输出9，因为按utf8编码，一个汉字占3个字节，3个字就占9个字节

s2 = u"万国码"
print(type(s2)) #输出  <type 'unicode'>，用unicode编码，2个字节1个字符
print(len(s2)) #输出3，unicode用字符个数来算长度，从这个角度上看，unicode才是真正意义上的字符串类型
```

再举个例子，比如要从一个文件中找出所有后两位是'字符'的词语，进行如下的判断：

```
# coding=utf-8
s = '中文字符'
s[-2:] == '字符'
```

返回false，本以为相等但在python2中是不相等的,这里的字符是用开头的encoding声明解释的，我开头用的是utf8，汉字占3个字节，所以"字符"占了6个字节），而s[-2:]取的是最后两个"双字节"，所以不相同。

```
s = u'中文字符'
s[-2:] == u'字符'
# 加u强制转换成unicode
```

返回true，这也是为什么说unicode是真正意义上的字符串类型。因为使用的是unicode，"字符"占的是两个"双字节"，一个"双字节"一个字。

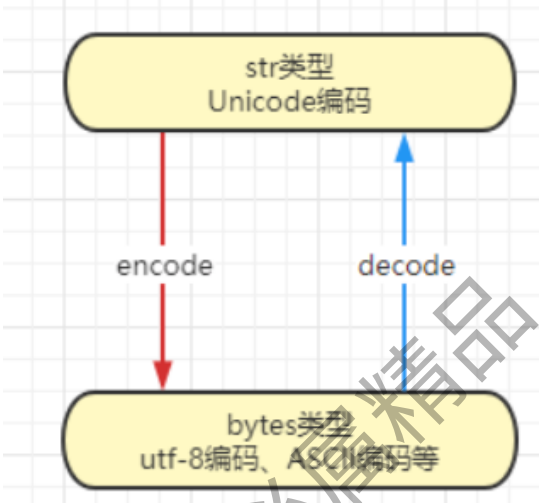
对于经常处理中文字符串的人，**统一用unicode**（加u强制转换成unicode）就可以避免这个坑了。

虽然有些字符串处理函数用str也可以，应该是函数里面帮你处理了编码问题。

而在python3中，**字符串是以Unicode编码的**。

如果要在网络上传输，或者保存到磁盘，就需要把str变为以字节为单位的bytes。python3中对bytes类型的数据用带b前缀的单引号或者双引号表示。

以Unicode表示的str通过encode()方法可以编码为指定的bytes。反过来，从网络或磁盘上读取的字节流，即bytes，要把bytes变为str，通过decode()方法。



```
>>> "ABC".encode("utf-8")
b'ABC'
>>> "中文".encode("utf-8")
b'\xe4\xb8\xad\xe6\x96\x87'
>>> '中文'.encode('ascii')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1: ordinal not
in range(128)
```

纯英文的str可以用ASCII编码为bytes，内容是一样的，含有中文的str可以用UTF-8编码为bytes。含有中文的str无法用ASCII编码，因为中文编码的范围超过了ASCII编码的范围，Python会报错。因为python3中字符串类型是Unicode编码的，所以不需要先decode成Unicode，直接encode成指定编码的bytes。

1.3 Python字符串操作

Python定义了很多功能强大的字符串运算符，常用的有如下几个：

在下表示例中，变量 a 值为字符串 "Hello"，b 变量值为 "Python"：

操作符	描述	实例
+	字符串连接(也称为拼接)	a + b 输出结果：HelloPython
*	重复输出字符串	a*2 输出结果：HelloHello
[]	通过索引获取字符串中字符	a[1] 输出结果 **e**
[:]	截取字符串中的一部分，遵循**左闭右开**原则，str[0:2] 是不包含第 3 个字符的。	a[1:4] 输出结果 ell
in	成员运算符 - 如果字符串中包含给定的字符返回 True	'H' in a 输出结果 True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 True	'M' not in a 输出结果 True

上面的例子都比较简单，我们重点看一下字符串裁剪的[:]操作。

使用裁剪操作，可以很方便的是可以从一个字符串中返回特定的部分，例如：

正向索引：获取从位置 0 到位置 2（不包括）的字符：

```
a = 'HelloWorld'
print(a[0:2])  #He
```

也可以反向索引：获取从位置 5 到位置 1 的字符，从字符串末尾开始计数：

```
a = 'HelloWorld'
print(a[-5:-2]) #Wor
```

切片操作在我们处理字符串时经常能让代码变得特别骚，甚至一行就搞定一个算法，所以必须掌握。

再看一下in操作，如需检查字符串中是否存在特定短语或字符，我们可以使用 `in` 或 `not in` 关键字。

```
txt = "Welcome to python"
x = "pyt" in txt
print(x) #True
```

除此之外，python还内置了很多功能强大的操作，详细请参考官方文档：https://docs.python.org/zh-cn/3/library/stdtypes.html#string-methods。这里列举了大量的读者在看的时候务必思考一个问题：如果让你实现该方法，该如何做呢，，例如split()等等。

2. 字符串常见算法题

从上面可以看到字符串与数组有很多相似之处，比如使用 `名称[下标]` 来得到一个字符，很多字符串的题本质就是数组的题，不过我们需要先将字符串转换成数组， 处理完之后再转成字符串。

字符串最经典的算法问题是子串匹配问题， 但是这个算法有些难度，我们后面单独看，这里我们先研究一些关于字符串的常见算法问题。

2.1 转换的问题

字符串里存放的可以是字母，可以是数字，也可以是特殊字符，字母又可以大写和小写，这就导致字符串有一类常见的转换的题目，这些题目无非就是这几种类型的相互转换。但是在转换过程中需要处理几种特殊情况：例如首先就是转之前先判断当前元素能不能转。如果是字符串转数字，则要考虑当前元素是不是数字。转完之后会不会溢出等。这些问题本身不复杂，但是必须考虑周全 ，如果考虑不周，就是面试时的扣分点了。我们看几个题目：

2.1.1 转换成小写字母

LeetCode709. 给你一个字符串 `s` ，将该字符串中的大写字母转换成相同的小写字母，返回新的字符串。

```
示例1：
输入：s = "Hello"
输出："hello"

示例2：
输入：s = "here"
输出："here"

示例3：
输入：s = "LOVELY"
输出："lovely"
```

我们知道每个字母都是有确定的 `ASCII` 的，因此我们可以根据 码表操作字符串即可。常见ASCII范围是：
a-z：97-122
A-Z：65-90
0-9：48-57

这个题可以先遍历整个字符串，然后对每一位字符进行判断，如果str[i]的值在A-Z之间，则需要在原来的基础上ASCII码加上32即可转换成对应小写：

```
class ToLowerCase(object):
    def toLowerCase(self, s):
        list1 = []
        for i in s:
            if ord('A') <= ord(i) <= ord('Z'):
                list1.append(chr(ord(i)+32))
            else:
                list1.append(chr(ord(i)))
        return ''.join(list1)
```

这个也可以采用流式写法，这个只在python3才支持：

```
class Solution:
    def toLowerCase(self, s: str) -> str:
        return "".join(chr(asc | 32) if 65 <= (asc := ord(ch)) <= 90 else ch for ch in
s)
```

2.1.2 字符串转换整数 (atoi)

LeetCode8. 本题的题目要求比较长，看原文：

```
请你来实现一个 myAtoi(string s) 函数，使其能将字符串转换成一个 32 位有符号整数（类似 C/C++ 中的
atoi 函数）。
函数 myAtoi(string s) 的算法如下：

* 读入字符串并丢弃无用的前导空格
* 检查下一个字符（假设还未到字符末尾）为正还是负号，读取该字符（如果有）。 确定最终结果是负数还是正数。
如果两者都不存在，则假定结果为正。
* 读入下一个字符，直到到达下一个非数字字符或到达输入的结尾。字符串的其余部分将被忽略。
* 将前面步骤读入的这些数字转换为整数（即，"123" -> 123， "0032" -> 32）。如果没有读入数字，则整数为
0 。必要时更改符号（从步骤 2 开始）。
* 如果整数超过 32 位有符号整数范围 [−2^31， 2^31 − 1] ，需要截断这个整数，使其保持在这个范围内。具体
来说，小于 −2^31 的整数应该被固定为 −2^31 ，大于 2^31 − 1 的整数应该被固定为 2^31 − 1 。
* 返回整数作为最终结果。

**注意：**
- 本题中的空白字符只包括空格字符 `' '` 。
- 除前导空格或数字后的其余字符串外，**请勿忽略** 任何其他字符。
```

示例：

```
示例1：
输入：s = "42"
输出：42
解释：加粗的字符串为已经读入的字符，插入符号是当前读取的字符。
第 1 步："42"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："42"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
         ^
第 3 步："42"（读入 "42"）
           ^
解析得到整数 42 。
由于 "42" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 42 。
```

```
示例2：
输入：s = "   -42"
输出：-42
解释：
第 1 步："   -42"（读入前导空格，但忽视掉）
            ^
第 2 步："   -42"（读入 '-' 字符，所以结果应该是负数）
             ^
第 3 步："   -42"（读入 "42"）
               ^
解析得到整数 -42 。
由于 "-42" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 -42 。
```

```
示例3：
输入：s = "4193 with words"
输出：4193
解释：
第 1 步："4193 with words"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："4193 with words"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
         ^
第 3 步："4193 with words"（读入 "4193"；由于下一个字符不是一个数字，所以读入停止）
             ^
解析得到整数 4193 。
由于 "4193" 在范围 [-2^31, 2^31 - 1] 内，最终结果为 4193 。
```

```
示例4：
输入：s = "words and 987"
输出：0
解释：
第 1 步："words and 987"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："words and 987"（当前没有读入字符，因为这里不存在 '-' 或者 '+'）
         ^
第 3 步："words and 987"（由于当前字符 'w' 不是一个数字，所以读入停止）
         ^
解析得到整数 0 ，因为没有读入任何数字。
由于 0 在范围 [-2^31, 2^31 - 1] 内，最终结果为 0 。
```

```
示例5：
输入：s = "-91283472332"
输出：-2147483648
解释：
第 1 步："-91283472332"（当前没有读入字符，因为没有前导空格）
         ^
第 2 步："-91283472332"（读入 '-' 字符，所以结果应该是负数）
          ^
第 3 步："-91283472332"（读入 "91283472332"）
                     ^
解析得到整数 -91283472332 。
由于 -91283472332 小于范围 [-2^31, 2^31 - 1] 的下界，最终结果被截断为 -2^31 = -2147483648 。
```

该题的要求很长，给的示例也很多，但是真正在面试的时候，这些要求是我们自己应该知道的，写代码也必须考虑的问题。面试官不会告诉你这些，如果你的代码考虑不周，他可能会提醒，如果提醒超过三次，你的代码就废了。

这个题最好不要用高级的特性，就是最基本的方式写。这里没有考察算法的知识，更多是开发中对数据的处理（如「参数校验」等）。如果面试中遇到，应先仔细阅读题目文字说明，认真细致的分析可能存在的情况，有疑问及时和面试官确认，千万不要阴沟里翻船。

在这里我罗列几个要点：

- 根据示例 1，需要去掉前导空格；
- 根据示例 2，需要判断第 1 个字符为 + 和 - 的情况，因此，可以设计一个变量 sign，初始化的时候为 1，如果遇到 - ，将 sign 修正为 -1；
- 判断是否是数字，可以使用字符的 ASCII 码数值进行比较，即 '0' <= c <= '9',如果0在最前面，则应该将其去掉；
- 根据示例 3 和示例 4 ，在遇到第 1 个不是数字的字符的情况下，转换停止，退出循环；
- 根据示例 5，如果转换以后的数字超过了 int 类型的范围，需要截取。这里不能将结果 res 变量设计为 long 类型，注意：由于输入的字符串转换以后也有可能超过 long 类型，因此需要在循环内部就判断是否越界，只要越界就退出循环，这样也可以减少不必要的计算；
- 由于涉及下标访问，因此全程需要考虑数组下标是否越界的情况。

特别注意：

1、由于题目中说「环境只能保存 32 位整数」，因此这里在每一轮循环之前先要检查，具体细节请见编码。

2、Java 、Python 和 C++ 字符串的设计都是不可变的，即使用 trim() 会产生新的变量，因此我们尽量不使用库函数，使用一个变量 index 去做遍历，这样遍历完成以后就得到转换以后的数值。

```
class MyAtoi:
    def myAtoi(self, s):
        if s.isspace() or s == "":
            return 0
        s = s.strip(" ")
        # print(s)
        mid = s.split()
        #print(mid)
        char = 1
        sum1 = ""
        result = 0
        count = 0
        if s[0].isalpha():
            return 0
        for i in mid:
            if i[0].isalpha():
                continue
            elif i[0] =="-":
                target = i
                break
            elif i[0] == "+":
                target = i
                break
            elif i[0].isdigit():
                target = i
                break
            elif i[0]==".":
                return 0
        if target[0].isdigit():
            for t in target:
                if t.isdigit():
                    sum1 += t
                else:
                    break
        if target[0] == "-":
            char = 0
            for t in target[1:]:
                if t.isdigit():
                    sum1 += t
                else:
                    break
        if target[0] == "+":
            for t in target[1:]:
                if t.isdigit():
                    sum1 += t
                else:
```

```
                break
        if sum1 == "":
            return 0
        # #print(char)
        if char==1:
            result = int(sum1)
        else:
            result = 0-int(sum1)
        if result>= 2**(31)-1:
            result = 2**(31)-1
        elif result<= -2**(31):
            result = -2**(31)
        return result
```

上面这个实现确实能解决问题，但是太长了，我们可以使用jdk自带的库函数来简化部分操作，例如这里的去掉前导空格，我们就可以使用trim。

2.1.3 表示数值的字符串

【剑指Offer】53 这个题与上面的题看似一致，但是要求其实有很大差异，先看题意：

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）。

数值（按顺序）可以分成以下几个部分：

- 若干空格
- 一个 小数 或者 整数
- （可选）一个 'e' 或 'E' ，后面跟着一个 整数
- 若干空格

小数（按顺序）可以分成以下几个部分：

- （可选）一个符号字符（'+' 或 '-'）
- 下述格式之一：
    - 至少一位数字，后面跟着一个点 '.'
    - 至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字
    - 一个点 '.' ，后面跟着至少一位数字

整数（按顺序）可以分成以下几个部分：

- （可选）一个符号字符（'+' 或 '-'）
- 至少一位数字

["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]
部分非数值列举如下：

["12e", "e12","1e3.14", "1.2.3", "+-5", "12e+5.4"]

```
示例1：
输入：s = "0"
输出：true
```

```
示例2:
输入: s = "e"
输出: false

示例3:
输入: s = "."
输出: false

示例4
输入: s = "    .1  "
输出: true
```

这个问题还是有点挑战的，一种方式是通过自动机来做，也就是更复杂的迭代。这个我们也到高级部分再分析，这里看一种常规的方法：

首先想到的是判断否false而不是判断是true，毕竟有这么多条件满足才能判断true，但是只要有一个条件不满足就可以判断false，最后代码的效率也还可以：

本题重点在于考虑到所有的情况如何来处理，表示数值的字符串遵循共同的模式：`A[.[B]][e|EC]`或者`.B[e|EC]`。

以上模式的含义是：A为数值的整数部分，B为跟在小数点之后的小数部分，C为跟在e或者E之后的指数部分。其中，A部分可以没有，比如小数.123代表0.123。如果一个数没有整数部分，那么小数部分必须有。

具体说来，A和C（也就是整数部分和指数部分）都是可能以"+"、"-"开头或者没有符号的数字串，B是数字序列，但前面不能有符号。

我们可以通过**顺序扫描字符串**来判断是否符合上述模式，首先尽可能多的扫描数字序列（开头可能有正负号），如果遇到小数点，那么扫描小数部分，遇到e或者E，则开始扫描指数部分。

除了顺序扫描以外，判断一个字符串是否满足某个模式，我们很容易想到的一个办法是**使用正则表达式**，以下给出这两种方法代码实现。

由于这个问题本身已经比较复杂了，所以我们就不必再考虑前导0和前导空格的情况了。以下是python3的写法：

```
class IsNumber:
    def isNumber(self, s: str) -> bool:
        def isNum(s: str) -> bool:
            return ord(s) >= ord('0') and ord(s) <= ord('9')

        def atLeastOneNumber(s: str) -> bool:
            if len(s) == 0:
                return False
            for each in s:
                if not isNum(each):
                    return False
            return True

        def isInt(s: str) -> bool:
```

```
            if not s:
                return False
            i = 0
            if s[i] == '+' or s[i] == '-':
                i += 1
            return atLeastOneNumber(s[i:])

        def isFloat(s: str) -> bool:
            if not s:
                return False
            i = 0
            if s[i] == '+' or s[i] == '-':
                i += 1
            if s.find('.') == -1:
                return False
            l = s[i:].split('.')
            if len(l) > 2:
                return False
            if l[0] == '':
                return atLeastOneNumber(l[1])
            if l[1] == '':
                return atLeastOneNumber(l[0])
            return atLeastOneNumber(l[0]) and atLeastOneNumber(l[1])

        s = s.strip()
        if s.find('e') > -1:
            l = s.split('e')
            if len(l) > 2:
                return False
            return (isFloat(l[0]) or isInt(l[0])) and isInt(l[1])
        elif s.find('E') > -1:
            l = s.split('E')
            if len(l) > 2:
                return False
            return (isFloat(l[0]) or isInt(l[0])) and isInt(l[1])
        else:
            return isFloat(s) or isInt(s)
```

2.2 反转的问题

我们知道反转是链表的一个重要考点，反转同样是字符串的重要问题。常见问题也就是在LeetCode中列举的相关题目：

【1】LeetCode344. 反转字符串：编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 `s` 的形式给出。

【2】LeetCode541. K个一组反转：给定一个字符串 `s` 和一个整数 `k`，从字符串开头算起，每计数至 `2k` 个字符，就反转这 `2k` 字符中的前 `k` 个字符。

【3】LeetCode.917. 仅仅反转字母：·给定一个字符串 `s`，返回 “反转后的” 字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。

【4】LeetCode151. 反转字符串里的单词：给你一个字符串 `s` ，逐个反转字符串中的所有 **单词** 。

【5】LeetCode.557. 反转字符串中的单词 III：给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

这几个题目你是否发现前三道就是要么反转字符，要么反转里面的单词。针对字符的反转又可以变换条件造出多问题。我们就从基本问题出发，各个击破。

2.2.1 反转字符串

LeetCode344. 题目要求：编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 s 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

```
示例1：
输入：s = ["h","e","l","l","o"]
输出：["o","l","l","e","h"]

示例2：
输入：s = ["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]
```

这是最基本的反转题，也是最简单的问题，使用双指针方法最直接。具体做法是:

对于长度为 N 的待被反转的字符数组，我们可以观察反转前后下标的变化，假设反转前字符数组为 s[0] s[1] s[2] ... s[N - 1]，那么反转后字符数组为 s[N - 1] s[N - 2] ... s[0]。比较反转前后下标变化很容易得出 s[i] 的字符与 s[N - 1 - i] 的字符发生了交换的规律，因此我们可以得出如下双指针的解法：

- 将 left 指向字符数组首元素，right 指向字符数组尾元素。
- 当 left < right：
    - 交换 s[left] 和 s[right]；
    - left 指针右移一位，即 left = left + 1；
    - right 指针左移一位，即 right = right - 1。
- 当 left >= right，反转结束，返回字符数组即可。

  具体再实现的时候有很多种写法，可以直接循环实现，也可以用递归，先递归到中间 ，然后再逐步反转，递归方式实现如下：

```
class ReverseString:
    def reverseString(self, s):
        self.recursion(s, 0, len(s) - 1)

    def recursion(self, s, left, right):
        if left >= right:  # 2.直至头尾指针相等时返回回去
            return
        # 即从内向外开始交换元素 继而达到反转字符串效果
        self.recursion(s, left + 1, right - 1)
        s[left], s[right] = s[right], s[left]  # 交换
```

如果充分利用python提供的库函数，处理该问题将变得非常容易，例如一行搞定：

```
    def reverseString(self, s):
        s[:] = s[::-1]
```

如果面试时遇到，建议还是用第一种方式比较好，因为如果面试官感觉算法太简单，可能会再给你出一道，假如你正好不会，那就歇菜了😄😄😄😄😄。

2.2.2 K个一组反转

LeetCode541 这个题，我感觉有点没事找事，先看一下要求：

给定一个字符串 s 和一个整数 k，从字符串开头算起，每计数至 2k 个字符，就反转这 2k 字符中的前 k 个字符。

- 如果剩余字符少于 k 个，则将剩余字符全部反转。
- 如果剩余字符小于 2k 但大于或等于 k 个，则反转前 k 个字符，其余字符保持原样。

```
示例1:
输入: s = "abcd efg", k = 2
输出: "bacdfeg"
示例2:
输入: s = "abcd", k = 2
输出: "bacd"
```

我们直接按题意进行模拟就可以：反转每个下标从 2k的倍数开始的，长度为 k的子串。若该子串长度不足 k，则反转整个子串。

```
class ReverseStr:
    def reverseStr(self, s, k):
        t = list(s)
        for i in range(0, len(t), 2 * k):
            t[i: i + k] = reversed(t[i: i + k])
        return "".join(t)
```

2.2.3 拓展 仅仅反转字母

LeetCode.917 这个题有点难度，我们来看一下：

给定一个字符串 `S`，返回 “反转后的” 字符串，其中不是字母的字符都保留在原地，而所有字母的位置发生反转。

```
示例1：
输入："ab-cd"
输出："dc-ba"

示例2：
输入："a-bC-dEf-ghIj"
输出："j-Ih-gfE-dCba"

示例3：
输入："Test1ng-Leet=code-Q!"
输出："Qedo1ct-eeLg=ntse-T!"
```

这里第一眼感觉不是特别复杂，同样从两头向中间即可，但问题是"-"不是均匀的有些划分的段长，有的短，这就增加了处理的难度。

方法1：使用栈

将 s 中的所有字母单独存入栈中，所以出栈等价于对字母反序操作。（或者，可以用数组存储字母并反序数组。）

然后，遍历 s 的所有字符，如果是字母我们就选择栈顶元素输出。

```
class ReverseOnlyLetters(object):
    def reverseOnlyLetters(self, S):
        letters = []
        N = len(S)
        for i, s in enumerate(S):
            if s.isalpha():
                letters.append(s)
        res = ""
        for i, s in enumerate(S):
            if s.isalpha():
                res += letters.pop()
            else:
                res += s
        return res
```

方法2：拓展 双指针

一个接一个输出 s 的所有字符。当遇到一个字母时，我们希望找到逆序遍历字符串的下一个字母。

所以我们这么做：维护一个指针 j 从后往前遍历字符串，当需要字母时就使用它。

```
class reverseOnlyLetters:
    def reverseOnlyLetters(self, s):
```

```
        ans = list(s)
        left, right = 0, len(ans) - 1
        while True:
            while left < right and not ans[left].isalpha():  # 判断左边是否扫描到字母
                left += 1
            while right > left and not ans[right].isalpha():  # 判断右边是否扫描到字母
                right -= 1
            if left >= right:
                break
            ans[left], ans[right] = ans[right], ans[left]
            left += 1
            right -= 1
        return ''.join(ans)
```

2.2.4 . 反转字符串里的单词

LeetCode151 :给你一个字符串 `s` ，逐个反转字符串中的所有 **单词** 。

单词 是由非空格字符组成的字符串。s 中使用至少一个空格将字符串中的 单词 分隔开。

请你返回一个反转 s 中单词顺序并用单个空格相连的字符串。

说明：

- 输入字符串 s 可以在前面、后面或者单词间包含多余的空格。
- 反转后单词间应当仅用一个空格分隔。
- 反转后的字符串中不应包含额外的空格。

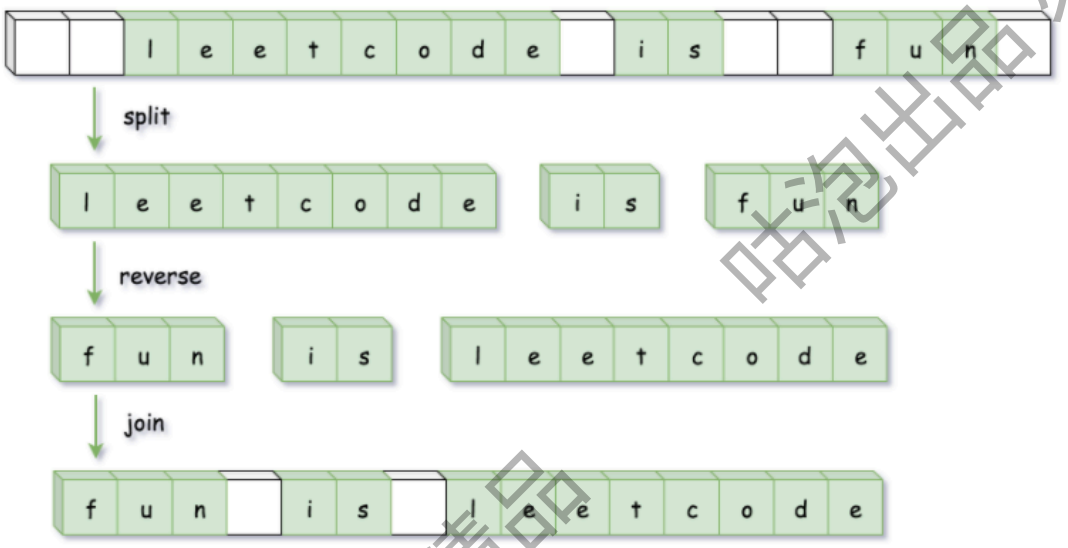
```
示例1：
输入：s = "the sky is blue"
输出："blue is sky the"

示例2：
输入：s = "hello world"

输出："world hello"
解释：输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
```

这个题也经常出现在很多面试题中，我记得曾经见过有个题是这样出的，要你按照同样的方式反转“ I love youzan”。这个题的关键在于如何处理单词。很多语言提供了相关的特性，因此我们可以首先使用语言的特性来实现：

很多语言对字符串提供了 split（拆分），reverse（反转）和 join（连接）等方法，因此我们可以简单的调用内置的 API 完成操作：

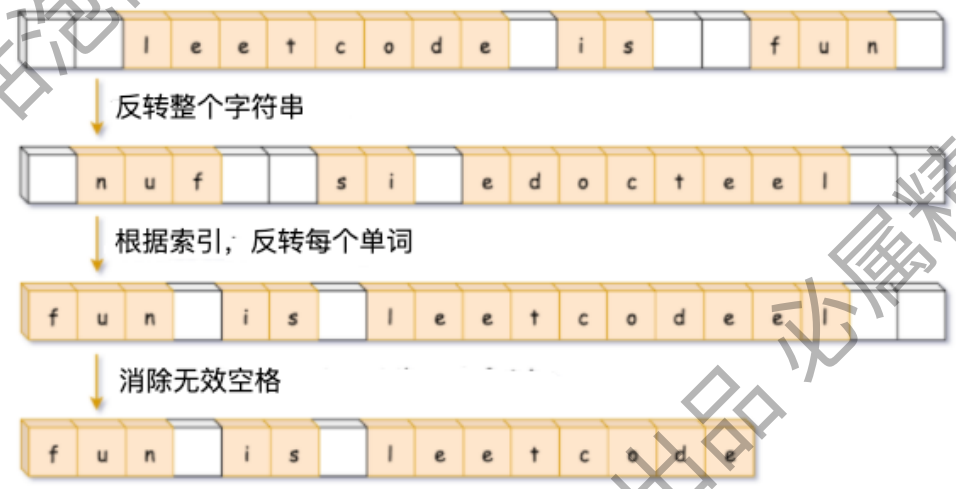
- 使用 split 将字符串按空格分割成字符串数组；
- 使用 reverse 将字符串数组进行反转；
- 使用 join 方法将字符串数组拼成一个字符串。

如图：




```
class ReverseWords:
    def reverseWords(self, s):
        return " ".join(reversed(s.split()))
```

如果我们要自行编写实现函数，对于字符串不可变的语言，例如java中的String，首先得把字符串转化成其他可变的数据结构，同时还需要在转化的过程中去除空格。



实现方法：

```
class TrimSpaces:
    def trim_spaces(self, s):
        left, right = 0, len(s) - 1
        # 去掉字符串开头的空白字符
        while left <= right and s[left] == ' ':
            left += 1

        # 去掉字符串末尾的空白字符
        while left <= right and s[right] == ' ':
            right -= 1
```

```
        # 将字符串间多余的空白字符去除
        output = []
        while left <= right:
            if s[left] != ' ':
                output.append(s[left])
            elif output[-1] != ' ':
                output.append(s[left])
            left += 1

        return output

    def reverse(self, l, left, right) :
        while left < right:
            l[left], l[right] = l[right], l[left]
            left, right = left + 1, right - 1

    def reverse_each_word(self, l) :
        n = len(l)
        start = end = 0

        while start < n:
            # 循环至单词的末尾
            while end < n and l[end] != ' ':
                end += 1
            # 翻转单词
            self.reverse(l, start, end - 1)
            # 更新start，去找下一个单词
            start = end + 1
            end += 1

    def reverseWords(self, s) :
        l = self.trim_spaces(s)
        # 翻转字符串
        self.reverse(l, 0, len(l) - 1)
        # 翻转每个单词
        self.reverse_each_word(l)
        return ''.join(l)
```

2.2.5 反转字符串中的单词 III

LeetCode557. 给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

```
示例：
输入："Let's take LeetCode contest"
输出："s'teL ekat edoCteeL tsetnoc"
```

提示在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

上一个题是将单词本身不变，单词之间的关系反转。而本题就是单词反转，单词之间的关系不变。

分析

这个题在Java或者C语言中，要花不少的代码来处理单词的问题，然而在python里却简单的过分，就是因为其强大的库函数。

因为在 `Python` 中字符串是不可变，因此遍历字符串交换每个单词内字符位置的方法不太可行，但是利用 `Python` 切片的便利，可以写出更优雅的实现方式。

第一种思路：将字符串分割成单词列表 然后把每个单词反转切片。代码如下：

```
class ReverseWords(object):
    def reverseWords(self, s):
        return " ".join(word[::-1] for word in s.split(" "))
```

第二种思路：利用两次切片，先反转单词列表 再反转字符串，以字符串 `“I love queen”` 为例：

s.split(" ") 将字符串分割成单词列表:

```
['I', 'love', 'queen']
```

s.split(" ")[::-1] 将单词列表反转:

```
['queen','love', 'I']
```

" ".join(s.split(" ")[::-1]) 将单词列表转换为字符串，以空格分隔:

```
"queen love I"
```

最后，**" ".join(s.split(" ")[::-1])[::-1]** 将字符串反转：

```
”I evol  neeuq“
```

如果用代码实现，也可以简单到一行搞定：

```
class Solution(object):
    def reverseWords(self, s):
        return " ".join(s.split(" ")[::-1])[::-1]
```

第三种思路，就是第二种方式的调整，可以先反转字符串，在反转单词列表。

还是上面的例子，

s[::-1] 反转字符串：

```
“neeuq evol I”
```

s[::-1].split(" ") 将字符串分割成单词列表：

```
['neeuq', 'evol', 'I']
```

s[::-1].split(" ")[::-1] 将单词列表反转：

```
['I', 'evol', 'neeuq']
```

" ".join(s[::-1].split(" ")[::-1]) 将单词列表转换为字符串，以空格分隔:

```
“I evol neeuq”
```

实现代码：

```
class ReverseWords(object):
    def reverseWords(self, s):
         return " ".join(s[::-1].split(" ")[::-1])
```

2.3 验证回文串

LeetCode125.给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。
说明：本题中，我们将空字符串定义为有效的回文串。

```
示例1:
输入: "A man, a plan, a canal: Panama"
输出: true
解释: "amanaplanacanalpanama" 是回文串

示例2:
输入: "race a car"
输出: false
解释: "raceacar" 不是回文串
```

回文问题在链表中是重点，在字符串中同样是个重点。当初我去美团面试第一轮技术面的第一个算法题就是让写判断字符串回文的问题。这个本身还是比较简单的，只要先转换成字符数组，然后使用双指针方法从两头到中间比较就行了。也许是过于简单了吧，面试时经常被加餐，例如LeetCode里的两道题。 一个是普通的验证回文串，第二个是找最长的子回文串。第二个问题需要动态规划等技术，有点难度，我们到高级算法里再看，这里先看一下基本的。

这个题我们可以有多种思路，最简单的方法是对字符串 s 进行一次遍历，并将其中的字母和数字字符进行保留，放在另一个字符串 sgood 中。这样我们只需要判断 sgood 是否是一个普通的回文串即可。

判断的方法有两种。第一种是使用语言中的字符串反转 API 得到 sgood 的逆序字符串 sgood_rev，只要这两个字符串相同，那么sgood 就是回文串。

```
class IsPalindrome:
    def isPalindrome(self, s) :
        sgood = "".join(ch.lower() for ch in s if ch.isalnum())
        return sgood == sgood[::-1]
```

第二种是使用双指针。初始时，左右指针分别指向 sgood 的两侧，随后我们不断地将这两个指针相向移动，每次移动一步，并判断这两个指针指向的字符是否相同。当这两个指针相遇时，就说明 sgood 时回文串。

```
class IsPalindrome:
    def isPalindrome(self, s) :
        sgood = "".join(ch.lower() for ch in s if ch.isalnum())
        n = len(sgood)
        left, right = 0, n - 1

        while left < right:
            if sgood[left] != sgood[right]:
                return False
            left, right = left + 1, right - 1
        return True
```

2.4 简单搜索问题

我们为什么叫简单搜索问题呢？因为字符串的有些搜索问题非常复杂，需要dp或者更高级的算法，例如字符匹配等等，因此这里我们先看几个简单的情况。

2.4.1 字符串中的第一个唯一字符

LeetCode387. 给定一个字符串，找到它的第一个不重复的字符，并返回它的索引。如果不存在，则返回 -1。

```
示例：
s = "leetcode"
返回 0
s = "loveleetcode"
返回 2
```

提示：你可以假定该字符串只包含小写字母。

我们可以对字符串进行两次遍历，在第一次遍历时，我们使用哈希映射统计出字符串中每个字符出现的次数。在第二次遍历时，我们只要遍历到了一个只出现一次的字符，那么就返回它的索引，否则在遍历结束后返回 -1。

```
class FirstUniqChar:
    def firstUniqChar(self, s) :
        frequency = collections.Counter(s)
        for i, ch in enumerate(s):
            if frequency[ch] == 1:
                return i
        return -1
```

2.4.2 最后一个单词的长度

LeetCode58. 给你一个字符串 s，由若干单词组成，单词前后用一些空格字符隔开。返回字符串中最后一个单词的长度。

单词 是指仅由字母组成、不包含任何空格字符的最大子字符串。

```
示例1:
输入: s = "Hello World"
输出: 5

示例2
输入: s = "   fly me   to   the moon  "
输出: 4

输入: s = "luffy is still joyboy"
输出: 6
```

这个题还是比较简单的，反向遍历。题目要求得到字符串中最后一个单词的长度，可以反向遍历字符串，寻找最后一个单词并计算其长度。

由于字符串中至少存在一个单词，因此字符串中一定有字母。首先找到字符串中的最后一个字母，该字母即为最后一个单词的最后一个字母。

从最后一个字母开始继续反向遍历字符串，直到遇到空格或者到达字符串的起始位置。遍历到的每个字母都是最后一个单词中的字母，因此遍历到的字母数量即为最后一个单词的长度。

```
class Solution:
    def lengthOfLastWord(self, s) :
        last = [x for x in s.split(' ') if x]
        if not last:
            return 0
        return len(last[-1])
```

2.5 旋转和重排

2.5.1 左旋转字符串

[剑指Offer】58字符串的左旋转操作是把字符串前面的若干个字符转移到字符串的尾部。请定义一个函数实现字符串左旋转操作的功能。比如，输入字符串"abcdefg"和数字2，该函数将返回左旋转两位得到的结果"cdefgab"。

```
示例1:
输入: s = "abcdefg", k = 2
输出: "cdefgab"

示例2:
输入: s = "lrloseumgh", k = 6
输出: "umghlrlose"
```

本题有多种方式处理的，最直观的方式是前面剪贴下来的若干字符和后面的字符保存到两个数组里，然后再按照要求合并就行了。这个在go、JavaScript、python等语言中有切片的操作，可以非常方便地处理，java中虽然没有切片，但是可以通过子串来实现相同的功能。

```
class Solution:
    def reverseLeftWords(self, s, n) :
        return s[n:] + s[:n]
```

第二种方式是通过append来实现拼接，先将第k个之后的元素添加进来，然后再将前k个添加进来，代码如下：

```
class Solution:
    def reverseLeftWords(self, s: str, n: int) -> str:
        res = []
        for i in range(n, len(s)):
            res.append(s[i])
        for i in range(n):
            res.append(s[i])
        return ''.join(res)
```

2.5.2 判定是否互为字符重排

```
给定两个字符串 `s1` 和 `s2`，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。
示例1:
输入: s1 = "abcadfhg", s2 = "bcafdagh"
输出: true

示例2:
输入: s1 = "abc", s2 = "bad"
输出: false
```

这个题第一眼看，感觉是个排列组合的题目，然后如果使用排列的算法来处理，难度会非常大，而且效果还不一定好。用简单的方式就能解决。

第一种方法：将两个字符串全部从小到大或者从大到小排列，然后再逐个位置比较，这时候不管两个原始字符串是什么，都可以判断出来。但是在python里可以忽略大小写，直接统计两个字符串中字母数是否一样就搞定了：

```
class CheckPermutation:
    def checkPermutation(self, s1, s2):
        return Counter(s1) == Counter(s2)
```

这种问题使用Counter()就太简单了，如果要自己实现该如何进行呢？这里提供两种思路：

第二种方法：使用Hash，注意这里我们不能简单的存是否已经存在，因为字符可能在某个串里重复存在例如"abac"。我们可以记录出现的次数，如果一个字符串经过重新排列后，能够变成另外一个字符串，那么它们的每个不同字符的出现次数是相同的。如果出现次数不同，那么表示两个字符串不能够经过重新排列得到。

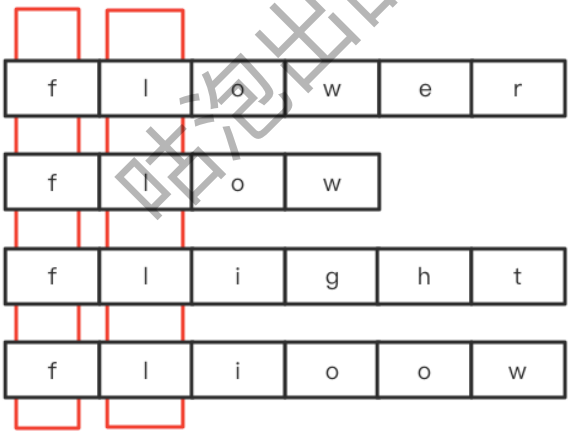
第三种方法，就是不管原始字符串有多长，是什么，基本元素都是26个英文字母，只少不多，那么我们可以换个思维：为两个字符串分别建立两个大小为26的字母表数组，每个位置是对应的字母出现的次数。最后统计一下两个数组的字母数和每个字母出现的次数就可以了。这种方法其实也是文本搜索引擎的基本思想，例如elasticSearch等，在文本搜索里有个专门的名字，叫“倒排索引”。

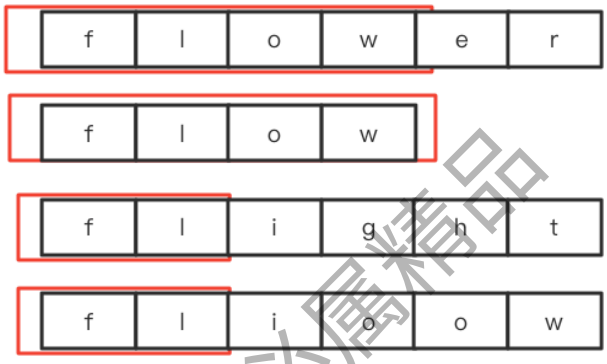
2.6 最长公共前缀

这是一道经典的字符串问题，先看题目要求：

```
编写一个函数来查找字符串数组中的最长公共前缀。如果不存在公共前缀，返回空字符串 ""。
示例1：
输入：strs = ["flower","flow","flight"]
输出："fl"
示例2：
输入：strs = ["dog","racecar","car"]
输出：""
解释：输入不存在公共前缀。
```

要解答这个问题，我们需要先看一下公共前缀的分布有什么特点，如下图：





可以看到，第一种方式，我们可以竖着比较，如左图所示，每前进一个位置就比较各个串，看是不是都是相等的，只要在某一轮遇到一个不相等的，那么就结束。

第二种方式，还可以横着比较，先比较前两个找到公共前缀fix1，然后再和第三个比较公共前缀得到fix2，我们可以确定fix2一定不会比fix1更长，然后和第四个比较，得到fix4，一直到最后一个fixn。每次得到的fix都不会比前面的长，最后比较完了还剩下的就是需要找的前缀了。

看到这里你是否有种似曾相识的感觉，我们前面合并K个数组或者K个链表不也是类似的思路吗？是的，就是类似的思路。

第三种方式， 我们是否可以对第二种进行优化一下，借鉴归并的思想，先两两一组找fix，然后将找到的fix再两两归并呢？当然可以了，这就是归并的方式。

先看第一种的实现方法，竖着比较。纵向扫描时，从前往后遍历所有字符串的每一列，比较相同列上的字符是否相同，如果相同则继续对下一列进行比较，如果不相同则当前列不再属于公共前缀，当前列之前的部分为最长公共前缀。

```
class Solution:
```

```
    def longestCommonPrefix(self, strs):
        if not strs:
            return ""

        prefix, count = strs[0], len(strs)
        for i in range(1, count):
            prefix = self.lcp(prefix, strs[i])
            if not prefix:
                break

        return prefix

    def lcp(self, str1, str2):
        length, index = min(len(str1), len(str2)), 0
        while index < length and str1[index] == str2[index]:
            index += 1
        return str1[:index]
```

第二种是横着依次比较，依次遍历字符串数组中的每个字符串，对于每个遍历到的字符串，更新最长公共前缀（其实就是看是否要缩短，一定不会变长），当遍历完所有的字符串以后，即可得到字符串数组中的最长公共前缀。如果在尚未遍历完所有的字符串时，最长公共前缀已经是空串，则最长公共前缀一定是空串，因此不需要继续遍历剩下的字符串，直接返回空串即可。

```
class LongestCommonPrefix:
    def longestCommonPrefix(self, strs):
        if not strs:
            return ""

        length, count = len(strs[0]), len(strs)
        for i in range(length):
            c = strs[0][i]
            if any(i == len(strs[j]) or strs[j][i] != c for j in range(1, count)):
                return strs[0][:i]

        return strs[0]
```

2.7 字符串压缩问题

LeetCode443 这个题也是出现频率很高的题目，经常在面经中看到。实现起来略有难度，我们一起看一下。

给你一个字符数组 chars ，请使用下述算法压缩：

从一个空字符串 s 开始。对于 chars 中的每组 连续重复字符：

- 如果这一组长度为 1 ，则将字符追加到 s 中。
- 否则，需要向 s 追加字符，后跟这一组的长度。
  压缩后得到的字符串 s 不应该直接返回 ，需要转储到字符数组 chars 中。需要注意的是，如果组长度为 10 或 10 以上，则在 chars 数组中会被拆分为多个字符。

请在 修改完输入数组后 ， 返回该数组的新长度。你必须设计并实现一个只使用常量额外空间的算法来解决此问题。

```
示例1：
输入：chars = ["a","a","b","b","c","c","c"]
输出：返回 6 ，输入数组的前 6 个字符应该是：["a","2","b","2","c","3"]
解释：
"aa" 被 "a2" 替代。"bb" 被 "b2" 替代。"ccc" 被 "c3" 替代。

示例2：
输入：chars = ["a"]
输出：返回 1 ，输入数组的前 1 个字符应该是：["a"]
解释：
没有任何字符串被替代。

示例3：
输入：chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]
输出：返回 4 ，输入数组的前 4 个字符应该是：["a","b","1","2"]。
解释：
由于字符 "a" 不重复，所以不会被压缩。"bbbbbbbbbbbb" 被 “b12” 替代。
注意每个数字在数组中都有它自己的位置。
```

这个题貌似采用双指针策略来处理就行，但是再分析发现三个指针才够:read,write和记录重复长度的num。

我们可以使用两个指针分别标志我们在字符串中读和写的位置，还要一个指针left用来标记重复字段的开始位置。read指针不断向前读取，每次当读指针 read 移动到某一段连续相同子串的最右侧，我们就在写指针 write 处依次写入该子串对应的字符和子串长度即可。

当读指针read 位于字符串的末尾，或读指针read 指向的字符不同于下一个字符时，我们就认为读指针read 位于某一段连续相同子串的最右侧。该子串对应的字符即为读指针 read 指向的字符串。我们使用变量 left 记录该子串的最左侧的位置，这样子串长度即为 read−left+1。

这里还有一个问题，就是长度可能超过10，因此还要实现将数字转化为字符串写入到原字符串的功能。这里我们采用短除法将子串长度倒序写入原字符串中，然后再将其反转即可。

```python
class Compress:
    def compress(self, chars) :
        def reverse(left: int, right: int) -> None:
            while left < right:
                chars[left], chars[right] = chars[right], chars[left]
                left += 1
                right -= 1

        n = len(chars)
        write = left = 0
        for read in range(n):
            if read == n - 1 or chars[read] != chars[read + 1]:
                chars[write] = chars[read]
                write += 1
```

```
            num = read - left + 1
            if num > 1:
                anchor = write
                while num > 0:
                    chars[write] = str(num % 10)
                    write += 1
                    num //= 10
                reverse(anchor, write - 1)
            left = read + 1
        return write
```

3.大厂算法实战

本章我们介绍了很多字符串的基本问题，字符串是算法考察中的重要组成部分。本文列举的题目还是比较简单的，但这是我们研究高级问题的基础。后面我们会继续补充大厂考察过的字符串问题。