<https://leetcode.com/problems/string-compression/>

Given an array of characters chars, compress it using the following algorithm:

Begin with an empty string s. For each group of **consecutive repeating characters** in chars:

* If the group's length is 1, append the character to s.
* Otherwise, append the character followed by the group's length.

The compressed string s **should not be returned separately**, but instead, be stored **in the input character array chars**. Note that group lengths that are 10 or longer will be split into multiple characters in chars.

After you are done **modifying the input array,** return *the new length of the array*.

You must write an algorithm that uses only constant extra space.

**Example 1:**

Input: chars = ["a","a","b","b","c","c","c"]

Output: Return 6, and the first 6 characters of the input array should be: ["a","2","b","2","c","3"]

Explanation: The groups are "aa", "bb", and "ccc". This compresses to "a2b2c3".

**Example 2:**

Input: chars = ["a"]

Output: Return 1, and the first character of the input array should be: ["a"]

Explanation: The only group is "a", which remains uncompressed since it's a single character.

**Example 3:**

Input: chars = ["a","b","b","b","b","b","b","b","b","b","b","b","b"]

Output: Return 4, and the first 4 characters of the input array should be: ["a","b","1","2"].

Explanation: The groups are "a" and "bbbbbbbbbbbb". This compresses to "ab12".

**Constraints:**

* 1 <= chars.length <= 2000
* chars[i] is a lowercase English letter, uppercase English letter, digit, or symbol.

**Attempt 1: 2023-10-21**

**Solution 1: Two Pointers (60 min)**

**Style 1: index 'count' increase before recording current char count number**

class Solution {

public int compress(char[] chars) {

// Test out: chars = 'a'

if(chars.length == 1) {

return 1;

}

int count = 0;

int i = 0;

// 'j == chars.length' is mandatory

// Test case: chars = {'a','b','c'}

for(int j = 1; j <= chars.length; j++) {

// Increase index count to create a placeholder for

// potential current char's count number's first digit

chars[count++] = chars[i];

while(j < chars.length && chars[i] == chars[j]) {

j++;

}

// Need compression

if(j - i > 1) {

// Since we have to modify the input array,

// besides calculating section length, we

// to exactly assign length value digit by

// digit into original input array, easier

// to handle with String

String num = j - i + "";

for(char c : num.toCharArray()) {

//chars[++i] = c;

//count++;

chars[count++] = c;

}

}

// Increase index count to create a placeholder for next char

//count++;

i = j;

}

return count;

}

}

Complexity Analysis

Let n be the length of chars.

Time complexity: O(n). -> 别看for loop里面有个while loop，里面的while loop从来没有回到过初始点，也就是说while loop仅仅继承在for loop的基础坐标上向右边继续，不会形成O(n^2)的情况，而是简单的O(n)

All cells are initially white. We will repaint each white cell blue, and we may repaint some blue cells green. Thus each cell will be repainted at most twice. Since there are nnn cells, the total number of repaintings is O(n).

Space complexity: O(1).

We store only a few integer variables and the string representation of groupLength which takes up O(1) space.

**Style 2: index 'count' increase after recording current char count number**

class Solution {

public int compress(char[] chars) {

// Test out: chars = 'a'

if(chars.length == 1) {

return 1;

}

int count = 0;

int i = 0;

// 'j == chars.length' is mandatory to include the last

// round assignment, if not we will have chance miss

// last char

// Test case: chars = {'a','b','c'}

// if no 'j == chars.length' we will miss 'c'

for(int j = 1; j <= chars.length; j++) {

chars[count] = chars[i];

while(j < chars.length && chars[i] == chars[j]) {

j++;

}

// Need compression

if(j - i > 1) {

// Since we have to modify the input array,

// besides calculating section length, we

// to exactly assign length value digit by

// digit into original input array, easier

// to handle with String

String num = j - i + "";

for(char c : num.toCharArray()) {

chars[++count] = c;

}

}

// Increase index count to create a placeholder for next char

count++;

i = j;

}

return count;

}

}

Complexity Analysis

Let n be the length of chars.

Time complexity: O(n). -> 别看for loop里面有个while loop，里面的while loop从来没有回到过初始点，也就是说while loop仅仅继承在for loop的基础坐标上向右边继续，不会形成O(n^2)的情况，而是简单的O(n)

All cells are initially white. We will repaint each white cell blue, and we may repaint some blue cells green. Thus each cell will be repainted at most twice. Since there are nnn cells, the total number of repaintings is O(n).

Space complexity: O(1).

We store only a few integer variables and the string representation of groupLength which takes up O(1) space.

**Why Time Complexity is O(n) ?**

**Refer to**

<https://leetcode.com/problems/string-compression/editorial/>

In the slideshow above, we compress the array chars = ["c","c","b","a","a","a","a","a","a","a","a","a","a"]. First, we process the group cc, then b, and finally aaaaaaaaaa.

Unprocessed characters are in white cells.

Processed characters that we may overwrite in the future are in blue cells.

Characters that belong to the answer and will not change are in green cells.

When processing a group, we first find its size groupLength and paint its cells blue. Then we append the character of the group to the answer. If groupLength is greater than 111, we also append the string representation of groupLength to the answer. Because the problem wants us to form the answer in place, instead of "appending" to the answer we will overwrite the corresponding blue cells by repainting them green.

White cells will eventually become blue and blue ones may become green. Since the compressed group takes up fewer cells than the uncompressed, the white cell cannot immediately become green.

#### **Complexity Analysis**

Let n be the length of chars.

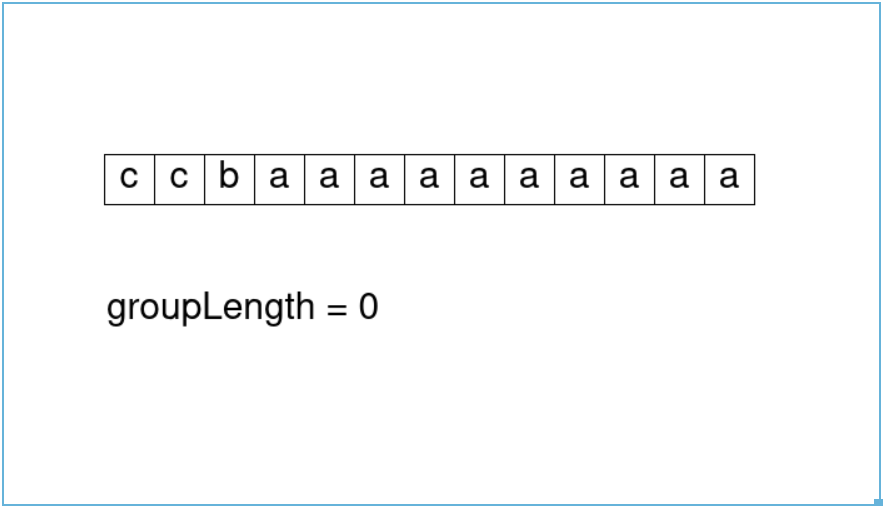
Time complexity: O(n).

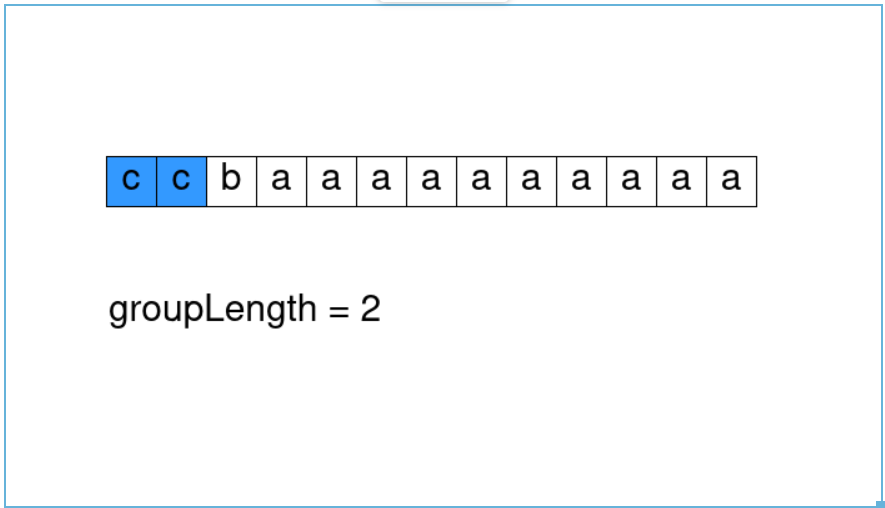
All cells are initially white. We will repaint each white cell blue, and we may repaint some blue cells green. **Thus each cell will be repainted at most twice. Since there are n cells, the total number of repaintings is O(n).**

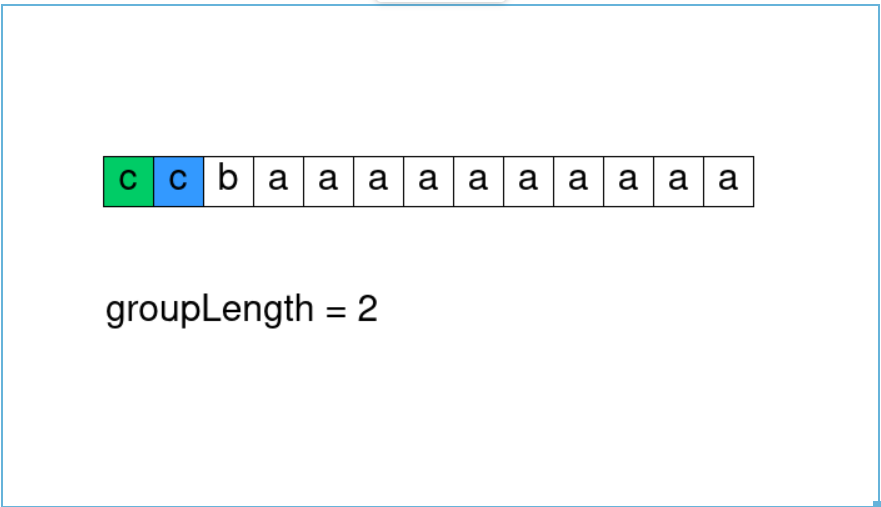
**别看for loop里面有个while loop，里面的while loop从来没有回到过初始点，也就是说while loop仅仅继承在for loop的基础坐标上向右边继续，不会形成O(n^2)的情况，而是简单的O(n)**

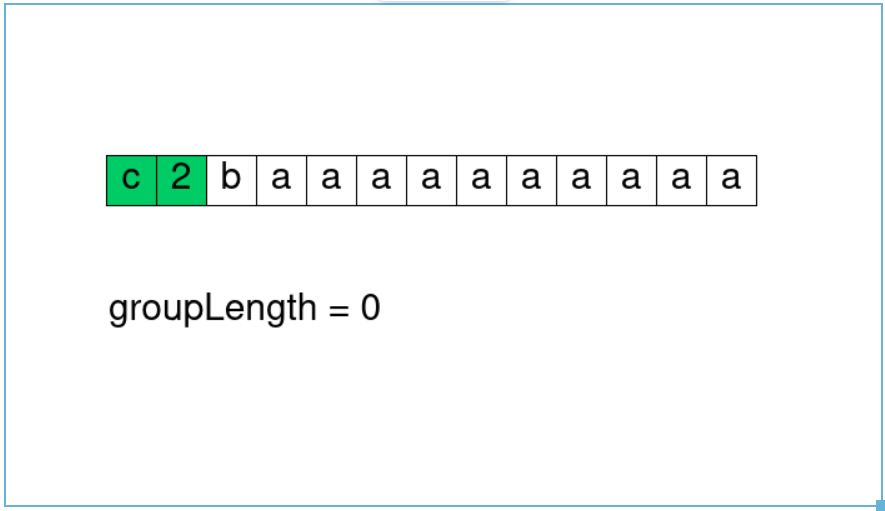
Space complexity: O(1).

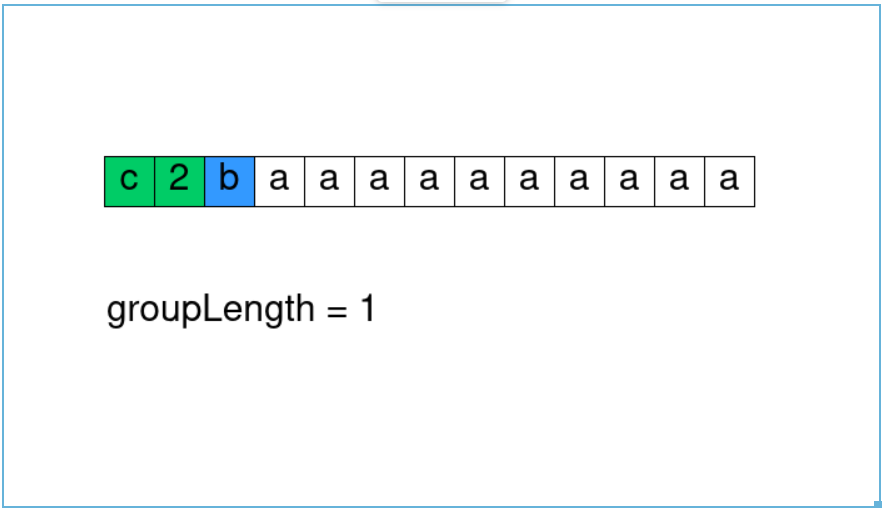
We store only a few integer variables and the string representation of groupLength which takes up O(1)space.

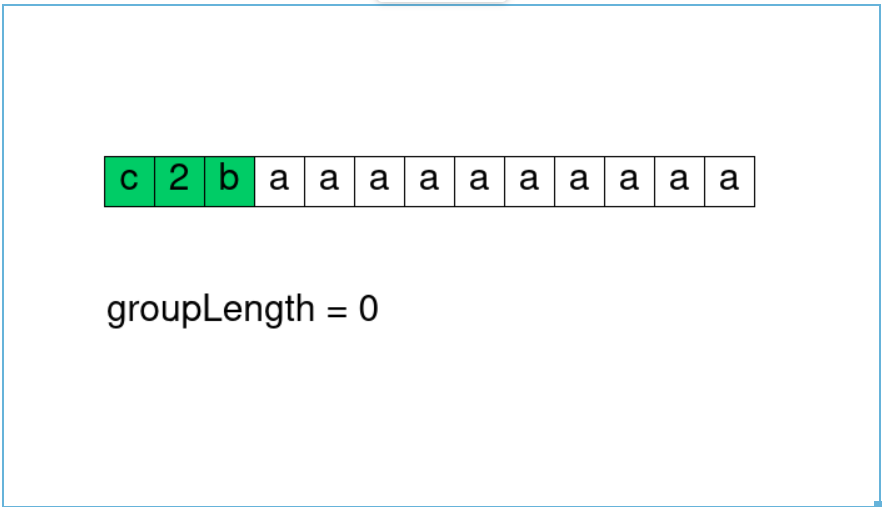


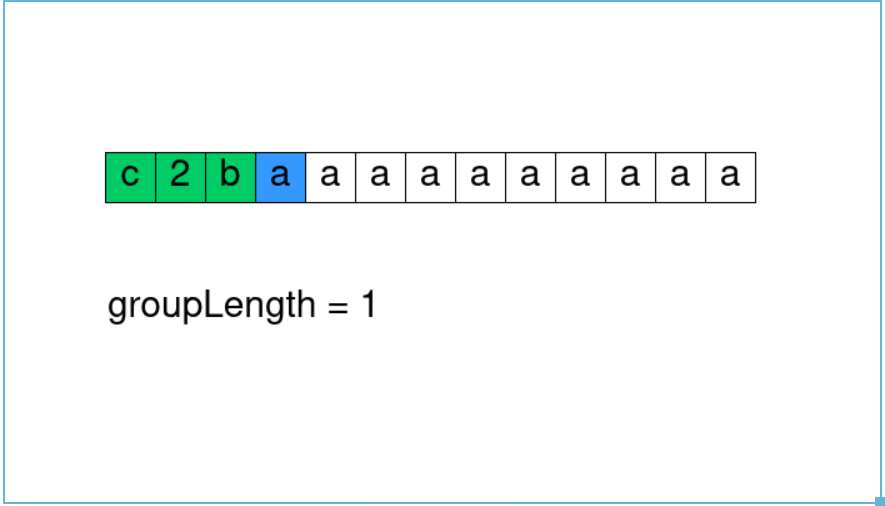


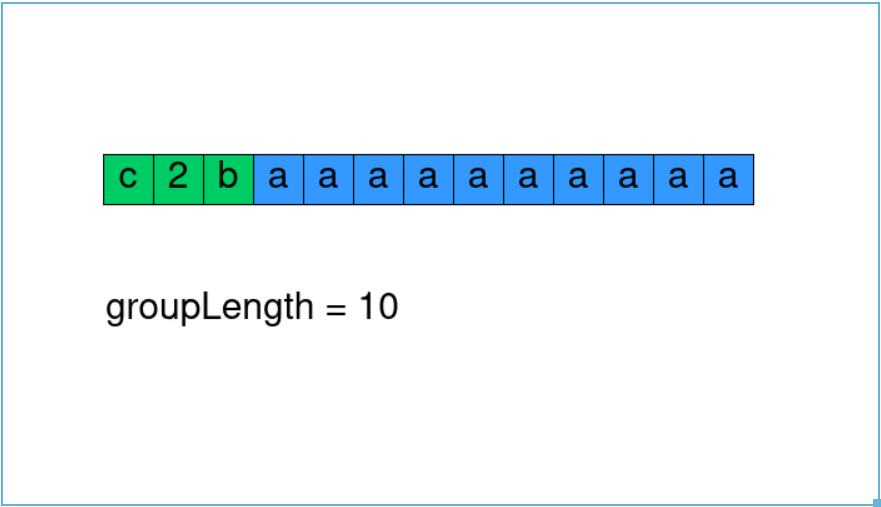


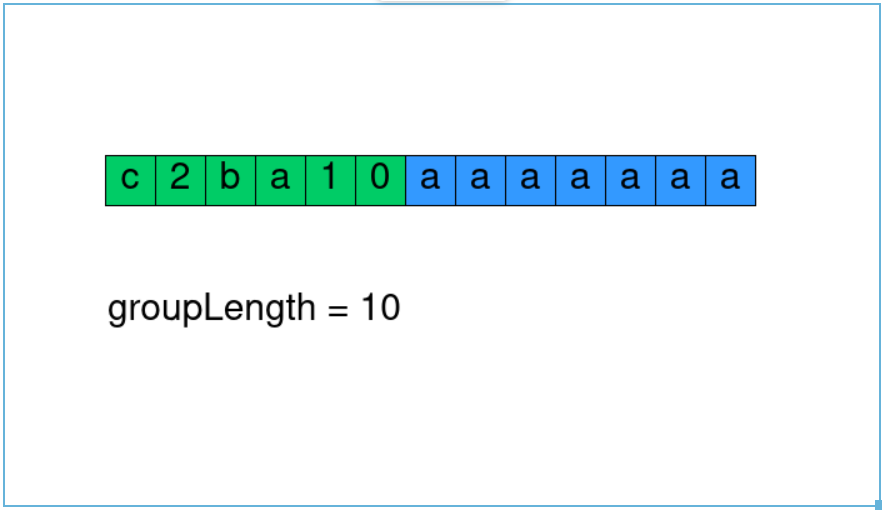












**Refer to**

<https://leetcode.com/problems/string-compression/solutions/92559/simple-easy-to-understand-java-solution/comments/302271>

This problem is similar to [Summary Ranges](https://leetcode.com/problems/summary-ranges/). If you have done that problem, this one wouldn't be difficult.

public int compress(char[] chars) {

int len = 0; // also a pointer to modify array in-place

for (int i = 0; i < chars.length; ) {

chars[len] = chars[i];

int j = i + 1;

while (j < chars.length && chars[j] == chars[i])

j++;

if (j - i > 1) { // need compression

String freq = j - i + "";

for (char c : freq.toCharArray())

chars[++len] = c;

}

len++;

i = j;

}

return len;

}

**Refer to**

<https://grandyang.com/leetcode/443/>

这道题给了我们一个字符串，让我们进行压缩，即相同的字符统计出个数，显示在该字符之后，根据例子分析不难理解题意。这道题要求我们进行in place操作，即不使用额外空间，最后让我们返回修改后的新数组的长度。我们首先想，数组的字符不一定是有序的，如果我们用Map来建立字符和出现次数之间的映射，不管是用HashMap还是TreeMap，一定无法保证原有的顺序。所以不能用Map，而我们有需要统计个数，那么双指针就是不二之选啦。既然双指针，其中一个指针指向重复字符串的第一个，然后另一个指针向后遍历并计数，就能得到重复的个数。我们仔细研究例子3，可以发现，当个数是两位数的时候，比如12，这里是将12拆分成1和2，然后存入数组的。那么比较简便的提取出各个位上的数字的办法就是转为字符串进行遍历。另外，由于我们需要对原数组进行修改，则需要一个指针cur来标记下一个可以修改的位置，那么最终cur的值就是新数组的长度，直接返回即可。

具体来看代码，我们用i和j表示双指针，开始循环后，我们用j来找重复的字符串的个数，用一个while循环，最终j指向的是第一个和i指向字符不同的地方，此时我们需要先将i位置的字符写进chars中，然后我们判断j是否比i正好大一个，因为只有一个字符的话，后面是不用加个数的，所以直接跳过。否则我们将重复个数转为字符串，然后提取出来修改chars数组即可，注意每次需要将i赋值为j，从而开始下一个字符的统计，参见代码如下：

class Solution {

public:

int compress(vector<char>& chars) {

int n = chars.size(), cur = 0;

for (int i = 0, j = 0; i < n; i = j) {

while (j < n && chars[j] == chars[i]) ++j;

chars[cur++] = chars[i];

if (j - i == 1) continue;

for (char c : to\_string(j - i)) chars[cur++] = c;

}

return cur;

}

};