*“Nothing succeeds like success and nothing fails like failure”*

Dear reader, welcome to the new problem based on Strings. The problem name is *‘****String Compression****’*’.

***Problem Statement***

Given a string, compress it into two following ways:

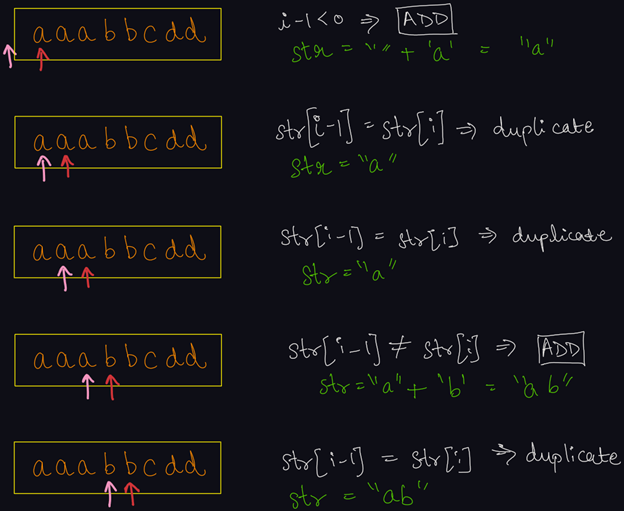
1. The string should be compressed such that the consecutive duplicates of characters are replaced with a single character. Example) For “aaabbccdee”, the compressed string will be “abcde”
2. The string should be compressed such that consecutive duplicates of characters are replaced with the character and followed by the number of consecutive duplicates. However, if there is only one consecutive duplicate of a character, we need not append 1 to it. Example) For "aaabbccdee", the compressed string will be "a3b2c2de2".

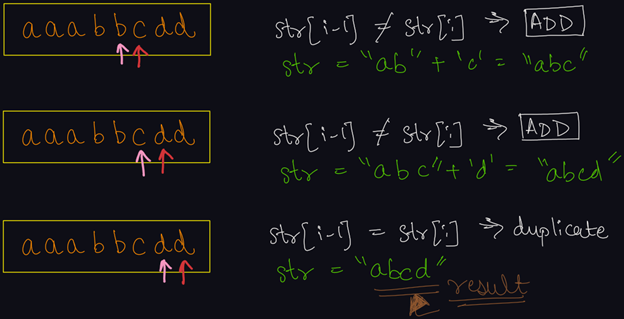
Important Links: [Problem Link](https://www.pepcoding.com/resources/online-java-foundation/string,-string-builder-and-arraylist/string-compression-official/ojquestion), [Solution Video](https://www.youtube.com/watch?v=6W1MDK7nPZs)

***Deducing Algorithm: Time - O(n) & Space - O(n)***

First, let us see how we can compress the string in the first way (with no frequency count).

* As you can see, we have to replace all consecutive duplicates with a single character, hence we will take the first character in each consecutive duplicate substring, and ignore other duplicates.
* We will initialize an empty output string *str (or ans)* = “”, and then run a loop through our input string.
* We will add the first character of the input string to our output string. (Because there is no previous character to compare it with).
* Now, for each character in the input string, we will see if it is the same as the previous character (in input string) or not.
* If it is the same, then it means it is a consecutive duplicate, and since we have already added it’s first occurrence in the output string, we will skip the current character.
* Else, since it is not the same as the previous character, we will add the current character to our output string.

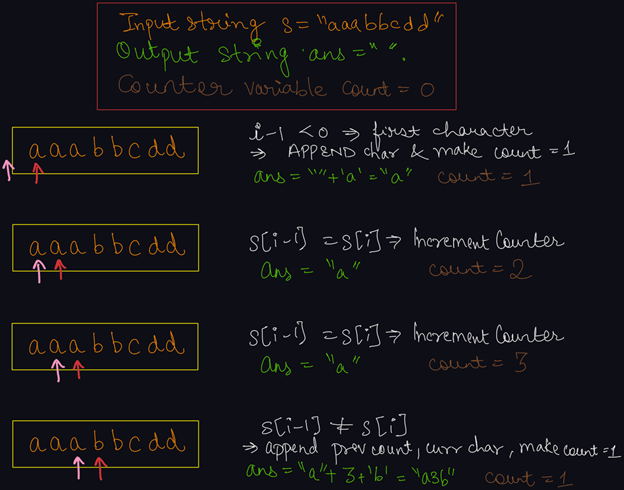


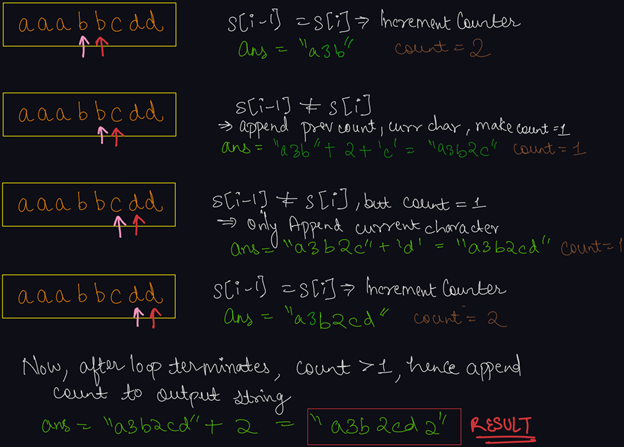


Now, we will learn how to compress the string in the second way (with frequency count).

The algorithm will be similar to the one discussed before. We will have to add the first occurrence of each of the consecutive duplicates. But, in addition to this, we need to append the count of that character (consecutively only, not for the entire string).

* Along with the output string *str (or ans)*, we need to maintain an integer counter variable set as 0.
* We will loop through the input string, as we did before.
* For the first character (0th index) in the input string, we will simply add it to the output string and increment counter variable to 1.
* Now, (from index = 1 onwards), we will check if the current character is the same as the previous character.
* If it is the same, then we will only increment the *counter* variable by 1, and not append the character to the output string.
* Else,
  + We will append the count of ***previous*** consecutive duplicates, (which is stored in the counter variable), to the output string.
  + Note: We will append the value of counter variable (count of previous consecutive duplicate), only if the value of counter > 1, as we need not append 1 for only one consecutive duplicate.
  + Also, we will add the current character to the output string.
  + For current character, also make counter = 1.
* Now, when the loop terminates, there is a small condition left.
  + What if the counter variable != 1, i.e. there were consecutive duplicates in the end?
  + Since, we write the counter of *previous* consecutive duplicate and not the count of *current* consecutive duplicate, thus the count of last consecutive duplicate characters is remaining to be appended.
  + Hence, after the loop terminates, we will check if counter > 1, we will append the value of counter to the output string.





***Implementation (Java)***

Please try to code this without taking help of the video solution. After trying to implement it yourself, have a look at the solution code to check whether the code you wrote was correct and optimal or not .

import java.io.\*;

import java.util.\*;

public class Main {

public static String compression1(String s){

String ans = "";

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

while(i < s.length() - 1 && s.charAt(i) == s.charAt(i + 1)){

i++;

}

ans += s.charAt(i);

}

return ans;

}

public static String compression2(String s){

String ans = "";

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

int count = 1;

while(i < s.length() - 1

&& s.charAt(i) == s.charAt(i + 1)){

count++;

i++;

}

ans += s.charAt(i);

if(count > 1){

ans += count;

}

}

return ans;

}

public static void main(String[] args) {

Scanner scn = new Scanner(System.in);

String s = scn.next();

System.out.println(compression1(s));

System.out.println(compression2(s));

}

}

This code is written and explained by our team in [this video](https://www.youtube.com/watch?v=6W1MDK7nPZs) from *[1:40, 7:05]*. Please refer to it if you are stuck somewhere.

***Time & Space Complexity Analysis***

**Time Complexity** - In both ways, we are traversing the input string once, hence the time complexity will be O(n).

**Space Complexity** - If we take the space of the output string, then the solution is taking O(n) space. Although, if we ignore the space taken by the output, (which we generally do), the solution is said to take O(1) auxiliary space.

I hope you enjoyed solving the problem with me. Dear reader, we will see you in the next topic: ‘*String Interning & Immutability*’. **Happy Coding**!

Contributor : Archit Aggarwal