<https://leetcode.com/problems/count-number-of-homogenous-substrings/description/>

Given a string s, return the number of **homogenous** substrings of s. Since the answer may be too large, return it **modulo** 109 + 7.

A string is **homogenous** if all the characters of the string are the same.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** s = "abbcccaa"

**Output:** 13

**Explanation:** The homogenous substrings are listed as below:

"a" appears 3 times."aa" appears 1 time.

"b" appears 2 times."bb" appears 1 time.

"c" appears 3 times."cc" appears 2 times.

"ccc" appears 1 time.3 + 1 + 2 + 1 + 3 + 2 + 1 = 13.

**Example 2:**

**Input:** s = "xy"

**Output:** 2

**Explanation:** The homogenous substrings are "x" and "y".

**Example 3:**

**Input:** s = "zzzzz"

**Output:** 15

**Constraints:**

1 <= s.length <= 10^5

s consists of lowercase letters.

**Attempt 1: 2025-01-05**

**Solution 1: Math (10 min, exactly same strategy as** [L1513.Number of Substrings With Only 1s (Ref.L1759,L2062)](note://WEB88cee6b65c48df54b2a36e150a94a79b)**)**

**Style 1: Start with index = 1 as actual prev**

class Solution {

    public int countHomogenous(String s) {

        int MOD = (int) (1e9 + 7);

        int total\_count = 1;

        int cur\_section\_count = 1;

        char prev = s.charAt(0);

        for(int i = 1; i < s.length(); i++) {

            char cur = s.charAt(i);

            if(cur == prev) {

                cur\_section\_count++;

            } else {

                cur\_section\_count = 1;

                prev = cur;

            }

            total\_count = (total\_count + cur\_section\_count) % MOD;

        }

        return total\_count;

    }

}

**Style 2: Start with index = 0 and dummy prev**

class Solution {

    public int countHomogenous(String s) {

        int MOD = (int) (1e9 + 7);

        int total\_count = 0;

        int cur\_section\_count = 0;

        char prev = '\*';

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

            char cur = s.charAt(i);

            if(cur == prev) {

                cur\_section\_count++;

            } else {

                cur\_section\_count = 1;

                prev = cur;

            }

            total\_count = (total\_count + cur\_section\_count) % MOD;

        }

        return total\_count;

    }

}

**Refer to**

<https://leetcode.com/problems/count-number-of-homogenous-substrings/solutions/1064530/java-c-python-straight-forward/>

**Explanation**

cur is the previous character in type integer, count the number of continuous same character.

We iterate the whole string character by character, if it's same as the previous, we increment the count, otherwise we set it to 1.

There are count characters to start with, ending at this current character, in order to construct a homogenous string.

So increment our result res = (res + count) % mod.

**Complexity**

Time O(n)

Space O(1)

public int countHomogenous(String s) {

int res = 0, cur = 0, count = 0, mod = 1\_000\_000\_007;

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

count = s.charAt(i) == cur ? count + 1 : 1;

cur = s.charAt(i);

res = (res + count) % mod;

}

return res;

}

**Refer to**

<https://algo.monster/liteproblems/1759>

**Problem Description**

The task is to find the number of homogenous substrings in a given string s. A string is considered homogenous if all characters in the string are identical. A substring refers to a consecutive sequence of characters within a string. The answer should be returned modulo 10^9 + 7 to prevent overflow issues due to potentially large numbers.

For example, if s = "aaa", then the homogenous substrings are "a", "a", "a", "aa", "aa", and "aaa", which totals up to 6.

**Intuition**

To solve this problem efficiently, we can utilize a two-pointer technique.

We iterate over the string using two pointers, i and j.

The first pointer i marks the start of a potential homogenous substring, while j scans ahead to find where this substring ends (i.e., where a character different from s[i] is encountered).

For each character position i, we find the longest stretch of the same character by incrementally increasing j as long as s[j] is equal to s[i].

The length of the homogenous substring starting at i is (j - i). For each such substring, we calculate the number of homogenous substrings that can be made, which is given by the formula (1 + cnt) \* cnt / 2, where cnt is the length of the homogenous substring.

Why this formula? Consider a homogenous string of length n. We can make n single-character substrings, n-1 substrings of length 2, n-2 of length 3, and so on, down to 1 substring of length n. This forms an arithmetic sequence that sums to n\*(n+1)/2.

The answer is incremented by this count for each homogenous stretch we find.

We use the modulo operation to keep our calculations within the prescribed limit to avoid integer overflow.

The first pointer i is then set to j to start searching for the next homogenous substring.

This approach optimizes the process by minimizing the number of times we traverse the string, leading to an efficient solution.

**Solution Approach**

The implementation of the solution uses a **two-pointer technique** along with basic arithmetic calculations to find the number of homogenous substrings. Here is the walkthrough of the code:

The function countHomogenous starts by initializing the variable mod to 10\*\*9 + 7 for modulo operations to prevent overflow.

Two variables are declared, i being the start pointer (initialized at index 0) and n being the length of the input string s.

We also initialize a variable ans to store the cumulative number of homogenous substrings found.

The solution enters a loop that continues until the start pointer i has reached the end of the string (i < n):

A second pointer j is set to start at the same position as i. This will be used to find the end of the current homogenous substring.

A while loop is used to move j forward as long as s[j] is the same as s[i]. When s[j] is different from s[i], it means we have found the end of the homogenous substring.

After the while loop, we now have a substring from index i to j - 1 that is homogenous. The length of this substring is cnt = j - i.

To find the number of homogenous substrings within this section, we use the arithmetic series sum formula (1 + cnt) \* cnt / 2, where cnt is the length of the homogenous substring.

The result is added to ans, which keeps the running total of homogenous substrings. Every time a new count is added, we perform a modulo operation to make sure ans doesn't exceed 10^9 + 7.

Finally, we move the start pointer i to the position where j ended, as everything before j is already part of a homogenous substring we've counted.

The use of the two-pointer technique efficiently reduces the time complexity since each character in the string is checked only once. By only considering stretches of identical characters and using the arithmetic series sum formula, we avoid having to individually count each possible substring. This is what makes the algorithm efficient.

The function ends by returning the total count ans as the resulting number of homogenous substrings.

**Solution Implementation**

class Solution {

private static final int MOD = (int) 1e9 + 7;

public int countHomogenous(String s) {

// Length of the input string

int length = s.length();

// Variable to hold the total count of homogenous substrings

long totalHomogenousSubstrings = 0;

// Loop through the string characters

for (int startIndex = 0, endIndex = 0; startIndex < length; startIndex = endIndex) {

// Set the end index to the current start index

endIndex = startIndex;

// Extend the end index while the end character is the same as the start character

while (endIndex < length && s.charAt(endIndex) == s.charAt(startIndex)) {

endIndex++;

}

// Calculate the length of the homogeneous substring

int homogeneousLength = endIndex - startIndex;

// Use the formula for sum of first n natural numbers to calculate the number of substrings

totalHomogenousSubstrings += (long) (1 + homogeneousLength) \* homogeneousLength / 2;

// Apply modulo operation to prevent overflow

totalHomogenousSubstrings %= MOD;

}

// Cast the result to int before returning, since the final output must be an integer

return (int) totalHomogenousSubstrings;

}

}

**Time and Space Complexity**

The time complexity of the given code is O(n), where n is the length of the string s. This is because each character in the string is checked exactly once to form homogenous substrings (characters that are the same and contiguous). The inner while loop runs only once for each homogenous substring, and since it only moves j to the end of a homogenous substring, the iterations of the inner loop throughout the entire run of the algorithm sum up to O(n).

The space complexity of the code is O(1). This is because the algorithm uses a constant number of additional variables (mod, i, n, ans, j, cnt) which do not scale with the input size - they use a constant amount of space regardless of the length of the string s.

**Refer to**

[L1513.Number of Substrings With Only 1s (Ref.L1759,L2062)](note://WEB88cee6b65c48df54b2a36e150a94a79b)