Special String Again

Isaac DeJager

March 15, 2023

Click here to view the HackerRank page

1 Problem Description

A string is said to be a special string if either of two conditions are met:

- All of the characters are the same, e.g. "aaa".
- All characters except the middle one are the same, e.g. "aadaa"

A special substring is any substirng of a string which meets one of those criteria. Given a string, determine how many special substrings can be formed form it.

Example
s ="mnonopoo"
s contains the following 12 special substrings: {"m", "n", "o", "o", "p", "o", "o", "non", "ono", "opo", "oo"}.

Function Description
Complete the substrCount function in the editor below.
substrCount has the following parameters:

- int n: the length of string s
- string s: a string

Returns
-int: the number of special substirngs

Input Format
The first line contains an integer, n, the lenght of s.
The second line contains the string s.

Constraints
$1 \leq n \leq 10^5$
Each character of the string is a lowercase English letter, ascii[a - z].

Sample Input 0
5
asasd

Sample Output 0
7

Explanation 0
The special palindromic substrings of s =”asasd” are:
{”a”, ”s”, ”a”, ”s”, ”d”, ”asa”, ”sas”}

Sample Input 1
7
abcbaba

Sample Output 1
10

Explanation 1
The special palindromic substrings of s =”abcbaba” are:
{”a”, ”b”, ”c”, ”b”, ”a”, ”b”, ”a”, ”bcb”, ”bab”, ”aba”}

Sample Input 2
4
aaaa

Sample Output 2
10

Explanation 2
The special palindromic substrings of s =”aaaa” are:
{”a”, ”a”, ”a”, ”a”, ”aa”, ”aa”, ”aa”, ”aaa”, ”aaa”, ”aaaa”}

2 Solution

```
long substrCount(int n, string s) {

    int x, z = 0;
    char c, d;

    //String type 1
    for(int i = 0; i < n; i++) {
        c = s.at(i);
        x = 0;

        while(
            i + x < n &&
            s.at(i + x) == c
        ) {
            x++;
        }

        z += (x * (x + 1) / 2);
        i += x - 1;
    }

    //String type 2
    for(int i = 1; i < n - 1; i++) {
        c = s.at(i);
        d = s.at(i + 1);
        x = 0;

        while(
            c != d &&
            -1 < i - x - 1 &&
            i + x + 1 < n &&
            s.at(i - x - 1) == d &&
            s.at(i + x + 1) == d
        ) {
            x++;
        }

        z += x;
    }

    return z;

}
```

3 Analysis

The program begins by creating for variables. z is an integer that counts the total number of special substrings in s. x is a counter that will be used later. Characters c and d are just char placeholders that are assigned as string s is looped through.

This algorithm breaks the problem into two parts. First, it counts the number of substrings in s that consist of the same character, e.g. "aaa" or "bbbbb". This is coined string type 1. To find the total number of type 1 substrings, a for loop loops through every char in s. At every char, c is set equal to that char and x is set to 0. Then begins a while loop with two qualifiers. The first is to ensure we stay inside the bounds of s by checking to see if $i + x \leq n$ where n is the length of string n. If this fails, the while loop breaks. The second qualifier is to see if s at index $i + x = c$. If it does, that means that character matches c. If both qualifiers are met, x is incremented. Essentially, this while loop takes an integer that starts at 0, and keeps incrementing it as long as the next character is the same as our string char c.

Once the while loop breaks, we count how many substrings there are total and add that to z. For example, if the while loop finds a substring of length 4 like "aaaa", then we need to count how many possible substrings there are in this substring. This is done by a simple summation formula:

$$\sum_{i=1}^{n} i = \frac{n * (n+1)}{2}$$

So for the substring $aaaa$ where $n = 4$, the number of substrings is :

$$\sum_{i=1}^{4} = \frac{4 * (4+1)}{2} = \frac{4 * 5}{2} = \frac{20}{2} = 10$$

After adding that to z we update the value of i by adding $x - 1$ to it. The is because the next character in s that we need to check is at index $x + i$. We started at index i and checked then next x characters. And since the start of each loop we add 1 to i, before the loop ends we add $x - 1$ to i. Finding all substrings of type 1 takes $O(n)$ time since we loop through each character in s once.

Now we start searching for string type 2. These strings are such that all characters are the same except for the character in the middle, e.g. "aabaa" or "bab". To find the number of these substrings we start by entering a for loop that begins at index $i = 1$ and ends at index $i = n - 1$. We set the char c equal to the character at index i and the char d equal to the character at index $i + 1$. Since the for loop runes until $i = n - 2$, setting d equal to the char at index $i + 1$ will not break the bounds of the string.

Then, we set $x = 0$ and, like in string type 1, we enter a while loop. The objective the while loop is to see how many characters on each side of c are equal to d. There are 5 qualifiers in this while loop.

- Check if $c \neq d$. If it does, then we know there are no type 2 substrings where c is the middle character as it would not be unique.
- Check if $-1 < i - x - 1$. This is to ensure we do not try to examine a character outside the lower bound of s
- Check if $i + x + 1 < n$. This is to ensure we do not try to examine a character outside the uper bound of s.
- Check if the character at index $i - x - 1 = d$
- Check if the character at index $i + x + 1 = d$

If all 5 qualifiers are upheld, then we increment x and the loop continues. Essentially, this loop starts by checking the characters adjacent to c and ensures they equal each other and do not equal c. Then each loop iteration checks the next two adjacent characters and keeps looping until we are either outside the bounds of the array or find a character that does not equal d.

Once the while loop breaks, we can easily count the number of substrings by just adding x to z. For example, consider the substring "aaabaaa". The previously described while loop will finish by setting $x = 3$, and we can see that all type 2 substrings are "aba", "aabaa", and "aaabaaa". So x is also the number of type 2 substrings. Once x is added to z, we increment i and repeat.

The time complexity on this step is more complicated than string type 1. For each index from 1 to $n - 2$, we proceed to check x characters on each side, where x could be any number of values. If we consider the most extreme cases, x can be calculated as a function of i. This is because we can only check so many characters before we hit the lower or upper bounds of s.

- For $i = 1$: $x = 1$
- For $i = 2$: $x = 2$
- For $i = 3$: $x = 3$

  ...
- For $i = \frac{n}{2}$: $x = \frac{n}{2}$
- For $i = \frac{n}{2} + 1$: $x = \frac{n}{2} - 1$
- For $i = \frac{n}{2} + 2$: $x = \frac{n}{2} - 2$

  ...
- For $i = n - 4$: $x = 3$
- For $i = n - 3$: $x = 2$
- For $i = n - 2$: $x = 1$

So, the total number of actions can be organized by creating two sums, from 1 to $\frac{n}{2}$ and another that counts back down from $\frac{n}{2} - 1$ to 1. Hence the time complexity looks like:

$$O(\sum_{i=1}^{\frac{n}{2}} i + \sum_{i=1}^{\frac{n}{2}-1} i) = O(\frac{\frac{n}{2}(\frac{n}{2}+1)}{2} + \frac{(\frac{n}{2}-1)(\frac{n}{2}-1+1)}{2})$$

$$= O(\frac{\frac{n}{2}(\frac{n}{2}+1) + (\frac{n}{2}-1)(\frac{n}{2})}{2}) = O(\frac{\frac{n^2}{4} + \frac{n}{2} + \frac{n^2}{4} - \frac{n}{2}}{2}) = O(\frac{n^2}{4}) = O(n^2)$$

Once we count all type 2 substrings, we have now counted all possible special substrings and we return z. The total time complexity is:

$$O(n) + O(n^2) = O(n^2)$$

And the only space complexity are the four variables x, z, c, and d.