DAA Assignment No. : 4
Find out all substrings with same character at start and end in a given string.

Lovely Digra
IIT2017090

Abhishek Vishwakarma
IIT2017091

Anuj Raj
IIT2017092

Abstract—**Our required aim is to design and analyze an optimal algorithm to find out all substrings with same character at the start and end in a given string. For our approach we have used some basic String operations. While designing our problem we calculated the time complexity for the algorithm and plotted graphs of no. of digits of nos. vs time elapsed.**

I. INTRODUCTION

To design and analyse algorithm for finding all sub-strings of a given string whose starting and ending character is same by using basic string operations.

There are two sub parts of the problem which needs separate solution for faster execution.

1. Counting such sub-strings 2. Printing such sub-strings

And it is seen that the approach which is faster in counting the sub-strings is slower in printing them. And the approach which is faster in printing such sub-strings is slower in counting the total occurences of such sub-string.

Now the question arises what is sub-string and how to recognize it in lowercase string?

A sub-string is a contiguous sequence of characters within a string. This is not to be confused with sub-sequence, which is a generalization of sub-string.

II. ALGORITHM DESIGN AND EXPLANATION

For the Counting problem:

Approach 1: Brute Force

We are given a string String, we need to count all sub-strings starting and ending with same character.

In this approach we will use brute force and find all the possible sub-strings. For this we will iterate two loops, where first loop variable(say i) goes from 0 to last character of the input string and the nested j loop goes from i+1 to last character of the input string. Now for each sub-string check if starting and ending character is equal or not. If it is equal update the counter. Finally at the end counter stores the count of total such sub-strings in the given input string.

Approach 2: Optimization with Map

Instead of using brute force we can try another approach. The Key Idea is for any character (say 'a') total sub-strings starting and ending with 'a' is equal to n(n-1)/2 where, n is the frequency of 'a' in the input string.

Generate a map of with distinct character as key and its frequency as its value.This is done with a single loop with O(L). L is length of input string.

Now the total count summation of n(n-1)/2 (n is value, the frequency, of a character stored in the map) over all distinct chararcter in the Map.

For the Printing each sub-strings:

Approach 1: Using Map

It is similar to the Map Approach for counting , but instead of storing frequency of each distinct character in the Map, we store the list of all positions of the character in the input string, in the Map.

Now for a given character total sub-strings starting and ending with the same letter is sub-strings with all combination of these positions taken two at a time. For example, the list for letter 'A' is like A: 1-2-4. All sub-string staring and ending with 'A' will be: substring(1,2) substring(1,4) and substring(2,4). Now, we print all combinations for each distinct character and simulatneously increment the counter(for giving total count).

Approach 2: Without Map

For printing all character storing the list of positions in Map takes more space and extra time for its generation(O(L)).

Printing can be implemented with same Brute force method discussed previously. if there is a match in the starting and ending letter increment the counter as well as print the sub-string.

Algorithm 1 PseudoCode for Approach 2: Counting Problem

```
procedure $subString(stringStr, inta, intb)$
    $len \leftarrow$ String length
    if $b >= len$ then
        $b \leftarrow len$ - 1
    for all $i = a$ to b do
        //Store substring into S
    return S

$ctr \leftarrow 0$
$Str \leftarrow$ input String
$len \leftarrow$ String length
for all $i = 0tolen$ do
map generated storing character,frequency pair
    for all $it = map.begin()$ to EndOfMap do
        $ctr \leftarrow ctr+$ it(value)*((it(value))-1)/2

$print(ctr)$
```

Algorithm 2 PseudoCode for Approach 2: Printing Problem

```
$Str \leftarrow$ Input string
$len \leftarrow$ Str.length()
for all $i = 0$ to len do
    for all $j = i + 1$ to len do
        if $Str[i] == Str[j]$ then
            $print$(Str.subStr(i,j-i+1))
            $ctr \leftarrow$ ctr + 1
```

III. Experimental Study

For the experimental analysis of this algorithm, we will take lowercase string as an input and find it's length and plot a graph of length of string vs execution time. A .dat file is generated containing duration for each corresponding length. For graph refer Fig 1.

IV. Time Complexity and Discussion

Counting Problem

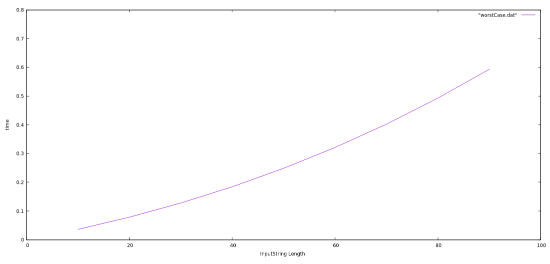

Fig. 1. Graphical Explanation

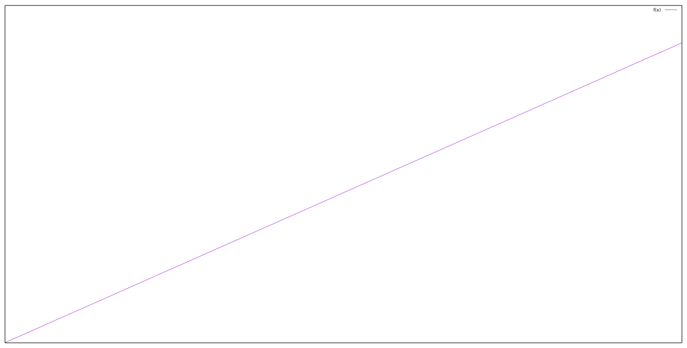

Fig. 2. Graphical Explanation

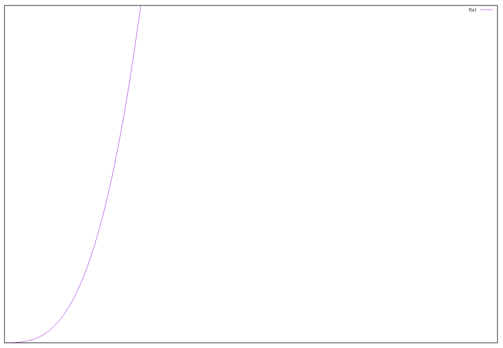

Fig. 3. Graphical Explanation

Approach 1: Brute Force
In the further discussion L is the length of input string. In every case nested loops run with complexity $O(L(L-1)/2)$.

Nature of the graph: The graph Time vs Length of Input Stirng will be Quadratic (See Fig.1) .

Approach 2: With Map
In every case Map is created with complexity $O(L)$, where L is the length of the input string. if the input string all English Alphabet, then result is computed in $O(26)$ in Worst Case. If the Input has only one distinct letter then the result is computed in $\omega(1)$. So in worst case time complexity if $O(L) + O(26)$ Space complexity is same , $O(26)$ in worst case and $\omega(1)$ in best case.

Nature of the graph: The graph Time vs Length of Input Stirng will be Linear (See Fig.2) .

Printing Problem

Approach 2: Without Map
In every case nested loops run with complexity $O(L(L-1)/2)$. Worst case will be every letter in input string is Same. Thus adding the complexity of sub-string function call in every iteration with complexity O(L). Best case is

observed when none of the letter is repeated.
Worst Case time complexity is bounded by $O(L(L-1)2*L)$. $BestCasetimecomplexity : Omega(L(l-1)/2)$.

Nature of the graph: The graph Time vs Length of Input Stirng will be cubic (See Fig.3) .

V. CONCLUSION

We designed an algorithm to find substring having starting and ending character same from a given lowercase string. We also analyzed the algorithm and plotted the graph for Value of length of string vs time elapsed.