CAPSTONE PROJECT

MAXIMUM NUMBER OF NON-OVERLAPPING SUBSTRINGS

CSA0695- DESIGN ANALYSIS AND ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES

SAVEETHA SCHOOL OF ENGINEERING

SIMATS ENGINEERING

Supervisor

Dr. R. Dhanalakshmi

Done by

DINESH.R (192211430)

MAXIMUM NUMBER OF NON-OVERLAPPING SUBSTRINGS

PROBLEM STATEMENT:

Given a string s of lowercase letters, you need to find the maximum number of non-empty substrings of s that meet the following conditions: The substrings do not overlap, that is for any two substrings s[i..j] and s[x..y], either j < x or i > y is true. A substring that contains a certain character c must also contain all occurrences of c. Find the maximum number of substrings that meet the above conditions. If there are multiple solutions with the same number of substrings, return the one with minimum total length. It can be shown that there exists a unique solution of minimum total length. Notice that you can return the substrings in any order. Example 1: Input: s = "adefaddaccc" Output: ["e","f","ccc"] Explanation: The following are all the possible substrings that meet the conditions: ["adefaddaccc" "adefadda", "ef", "e", "f", "ccc",] If we choose the first string, we cannot choose anything else and we'd get only 1. If we choose "adefadda", we are left with "ccc" which is the only one that doesn't overlap, thus obtaining 2 substrings. Notice also, that it's not optimal to choose "ef" since it can be split into two. Therefore, the optimal way is to choose ["e","f","ccc"] which gives us 3 substrings. No other solution of the same number of substrings exists

The goal is to find the maximum number of non-overlapping substrings from a given string such that for any character 'c' in a substring, all its occurrences in the string are contained within the same substring. Additionally, the solution with the minimum total length of these substrings must be returned. We must ensure that no two substrings overlap in the result.

ABSTRACT:

This project explores an algorithmic problem of finding the maximum number of non-overlapping substrings from a string where each substring contains all occurrences of the characters it holds. The approach balances between maximizing the number of substrings and minimizing their length. We present a step-by-step solution in C, analyze the time and space complexity, and discuss various cases such as best, worst, and average scenarios. Finally, we propose some potential future applications of this problem in real-world situations like substring-based search optimizations.

INTRODUCTION:

In computer science, substrings play a vital role in various applications, such as text processing, pattern matching, and data compression. Given a problem involving a string of lowercase letters, where we need to divide it into maximum non-overlapping substrings, each containing all occurrences of every character within that substring, the challenge becomes both interesting and complex. Such problems are often seen in the realms of data processing, where substring uniqueness or minimal overlap is critical.

To solve this problem optimally, it's essential to ensure that the substrings are not only non-overlapping but also that they contain all occurrences of the characters they enclose. This introduces constraints that must be handled carefully during string traversal and partitioning. Moreover, once these conditions are satisfied, our objective is to select the maximum number of substrings while keeping their total length minimized.

In this project, we break down the algorithm, implement the solution using C programming, and provide an in-depth explanation of the code, complexity analysis, and different case scenarios. Lastly, we will examine potential real-world applications where this method of substring partitioning could be utilized effectively.

CODING:

The following code aims to identify the maximum number of non-overlapping substrings in a given string s. The algorithm works by first identifying the range of each character in the string and then merging these ranges to form the required substrings.

C-programming

```
#include <stdio.h>
#include <string.h>

// Function to find the maximum number of non-overlapping substrings
void findMaxNonOverlappingSubstrings(char *s) {
    int last_occurrence[26] = {0};
    int n = strlen(s);

    // Step 1: Find the last occurrence of each character in the string
    for (int i = 0; i < n; i++) {
        last_occurrence[s[i] - 'a'] = i;
    }

    int start = 0, end = 0;
    for (int i = 0; i < n; i++) {
        // Step 2: Update the end position of the current character's range
        end = last_occurrence[s[i] - 'a'] > end ? last_occurrence[s[i] - 'a'] : end;

        // Step 3: Check if current position matches the end of the range
        if (i == end) {
            // Print the substring from start to end
            for (int j = start; j <= end; j++) {
                printf("%c", s[j]);
            }
            printf("\n");
            start = i + 1;
        }
    }
}
int main() {
    char s[] = "adefaddaccc";
```

```
    printf("The maximum number of non-overlapping substrings are:\n");
    findMaxNonOverlappingSubstrings(s);

    printf("\n\n\nDinesh.R   192211430\n");
    return 0;
}
```

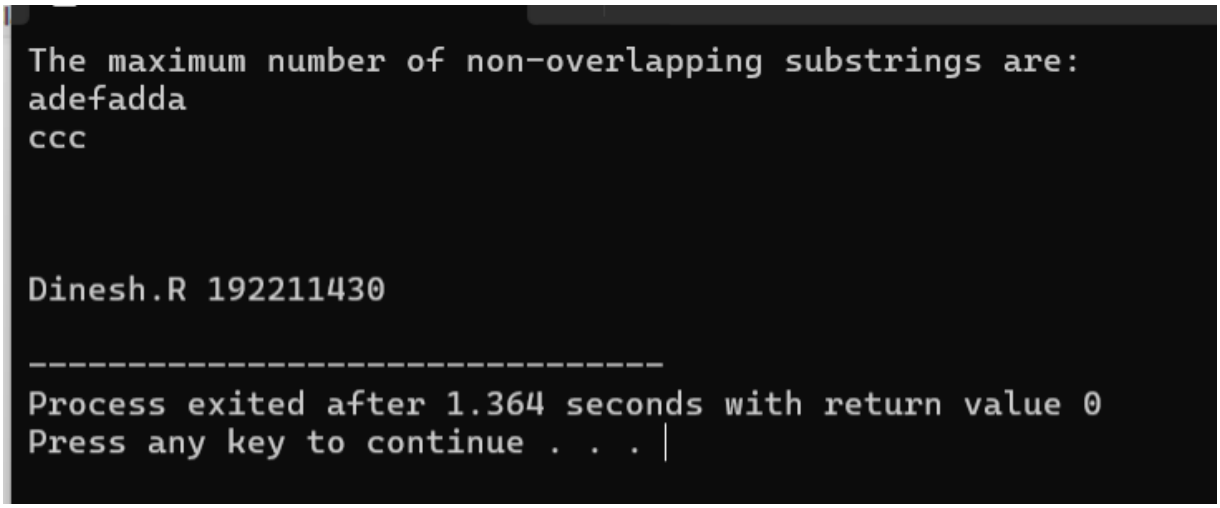
Code Explanation:

Step 1: The code begins by initializing an array last_occurrence[], which stores the index of the last occurrence of each character in the string.

Step 2: The string is then traversed character by character. For each character, its corresponding range (from the first occurrence to the last occurrence) is updated.

Step 3: If the current position i matches the end of the range for the character, a non-overlapping substring is identified and printed. The starting position is updated for the next potential substring.

OUTPUT:



COMPLEXITY ANALYSIS:

Time Complexity: The time complexity is O(n), where n is the length of the string. This is because we make two linear passes over the string—one to calculate the last occurrences of characters and the other to identify the non-overlapping substrings.

BEST CASE:

In the best case, the input string has no repeating characters. In such cases, every character is its own non-overlapping substring, and the algorithm performs at optimal efficiency. The maximum number of substrings will be equal to the length of the string.

WORST CASE:

The worst case occurs when all characters of the string are identical. In this scenario, the entire string becomes one large substring, as all occurrences of the character must be grouped together. Thus, the algorithm would return only one substring, which is the entire input.

AVERAGE CASE:

In the average case, the string contains multiple characters, each appearing a few times in different regions of the string. The algorithm efficiently splits the string into multiple non-overlapping substrings, based on the distribution of characters.

FUTURE SCOPE:

This approach to substring partitioning can be applied in various areas such as:

Text compression algorithms, where minimizing overlap and partitioning text effectively could improve storage efficiency.

Pattern matching and search engines, where identifying and processing non-overlapping regions of text can lead to faster search results.

Natural language processing (NLP), particularly in tokenization and sentence segmentation tasks.

CONCLUSION:

The problem of finding the maximum number of non-overlapping substrings with minimal total length is both interesting and practical. The C program efficiently solves the problem with a linear time complexity, making it suitable for larger inputs. The approach can be expanded and refined to address more complex applications, such as those in search optimization and text processing, providing a broad future scope for further exploration.