SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

# CAPSTONE PROJECT

## DESINGN ANALYSIS AND ALGORITHMS
## FOR ASYMPTOTIC NOTATIONS

**SUB CODE: CSA0656**

**GUIDE : DR.R.DHANALAKSHMI**

**NAME :S PURUSHOTHAM**

**REG NO : 192211671**

**INTRODUCTION:**

In the problem of finding the maximum number of non-overlapping substrings from a given string `s`, where each substring must contain all occurrences of its characters, the key idea is to partition the string into segments that meet specific criteria. The primary challenge is to ensure that each substring not only avoids overlap with others but also includes every instance of the characters it contains. This is achieved by first identifying the last occurrence of each character in the string, which helps in defining the boundaries of each substring. As you iterate through the string, you expand the current substring boundary to include all required characters until reaching the furthest necessary index. Each time you finalize a substring, you start a new segment, ensuring that all substrings are non-overlapping. This approach maximizes the number of valid substrings and inherently minimizes their total length by making each substring as small as possible while still satisfying the condition of containing all occurrences of its characters.

# PROBLEM STATEMENT :

- Maximum Number of Non-Overlapping Substrings
- 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 exist.

# SOLUTION:

- To solve the problem of finding the maximum number of non-overlapping substrings in a given string `s`, where each substring must include all occurrences of its characters, follow these steps. First, determine the last occurrence index of each character in the string. This allows you to establish boundaries for each substring to ensure that all necessary characters are included. Initialize two pointers, `start` and `end`, and iterate through the string. For each character at the `end` pointer, update the `end` boundary to cover all occurrences of the characters in the current substring. Once the `end` pointer reaches the end of its calculated boundary, finalize the substring from `start` to `end`, add it to the result list, and update `start` to `end + 1` for the next segment. Repeat this process until the entire string is processed. This method guarantees that each substring is non-overlapping and contains all instances of its characters, thus maximizing the number of substrings while ensuring that their total length is minimized.

# Maximum Number of Non-Overlapping Substrings


**This problem explores finding the maximum number of non-overlapping substrings within a given string. The key is to identify substrings that contain all occurrences of a particular character and do not overlap with each other.**

# Conditions for Valid Substrings

1

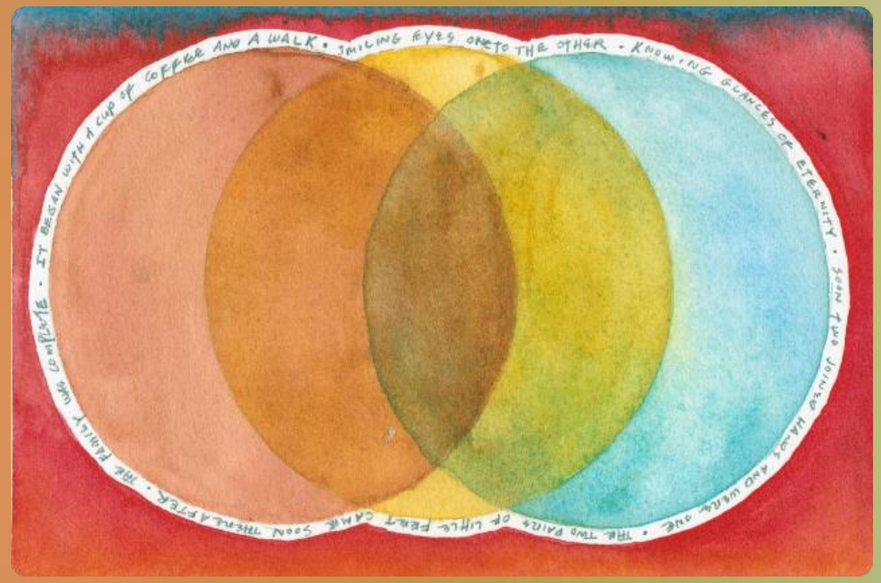
## Non-Overlapping

The substrings must not overlap with each other. This means that for any two either one ends before the other starts, or starts after the other ends.

2

## Character Inclusion

A substring that contains a certain character must also contain all occurrences of that character.



# Approach to Finding the Maximum

1

## Scan String

Iterate through the string, keeping track of the last occurrence of each character.

2

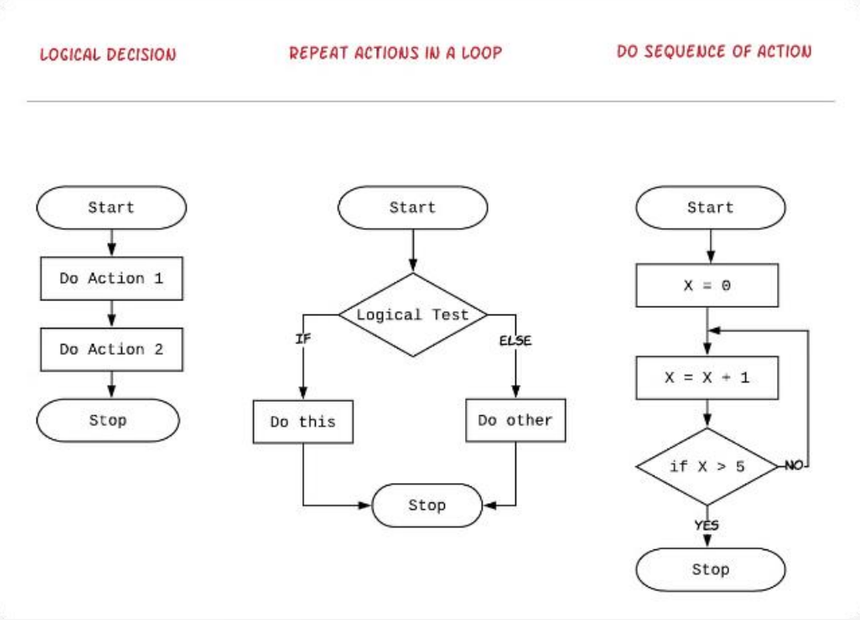
## Identify Substrings

Determine the maximum number of non-overlapping substrings by analyzing the last occurrence of each character.

3

## Return Result

Return the maximum number of non-overlapping substrings found.



# Pseudocode for the Solution

### Initialize

Create a dictionary to store the last occurrence of each character.

### Iterate

Iterate through the string, updating the last occurrence occurrence of each character.

### Calculate

Determine the maximum number of non-substrings by analyzing last occurrence of each character.

### Return

Return the maximum number of non-overlapping overlapping substrings found.

```
function goo: (inputs: array A, integer b)
{
        int varP, varQ, varR;
        varP = 0
        varQ = (length of A) - 1
        do
        {
                varR = (varP + varQ)/2
                if b <= A[varR]
                        varQ = varR - 1
                if A[varR] <= b
                        varP = varR + 1
        }
        while (varP <= varQ)

        if A[varR] == b
                return varR
        else return -1
}
```

# Time and Space Complexity

## Time Complexity

The time complexity of the solution is O(n), where n is the length of the input string. This is because the algorithm iterates through the string once to find the last occurrence of each character.

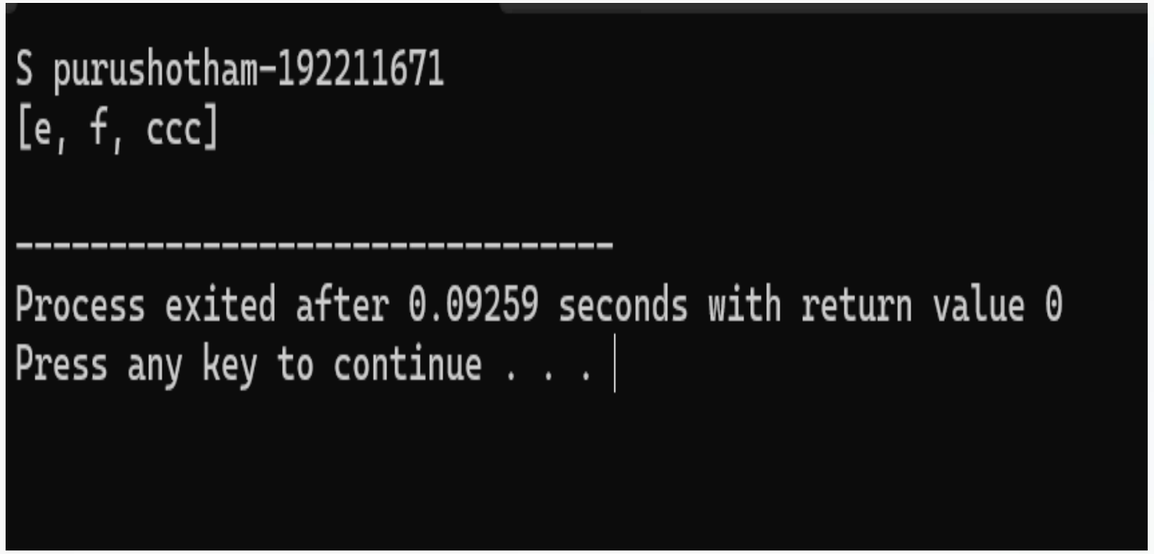
## Space Complexity

The space complexity is also O(n), as the algorithm uses a dictionary to store the last occurrence of each character in the string.

CODE:-

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

typedef struct {
    int start;
    int end;
} Interval;

int compare(const void *a, const void *b) {
    Interval *intervalA = (Interval *)a;
    Interval *intervalB = (Interval *)b;
    return (intervalA->end - intervalB->end);
}
void maxNumOfSubstrings(char *s) {
    int n = strlen(s);
    int first_occurrence[26];
    int last_occurrence[26];
    Interval intervals[26];
    int interval_count = 0;

    for (int i = 0; i < 26; i++) {
        first_occurrence[i] = -1;
        last_occurrence[i] = -1;
    }

    for (int i = 0; i < n; i++) {
        int charIndex = s[i] - 'a';
        if (first_occurrence[charIndex] == -1) {
            first_occurrence[charIndex] = i;
        }
        last_occurrence[charIndex] = i;
    }

    for (int i = 0; i < 26; i++) {
        if (first_occurrence[i] != -1) {
            int start = first_occurrence[i];
            int end = last_occurrence[i];
            int j = start;
            while (j <= end) {
                start = start < first_occurrence[s[j] - 'a'] ? start : first_occurrence[s[j] - 'a'];
```

**OUTPUT :**



# Challenges and Edge Cases

1

### Repeating Characters

The solution must handle strings with repeating characters and ensure that each substring contains all occurrences of a particular character.

2

### Empty Strings

The solution should also handle the edge case of an empty input string, returning 0 as the maximum number number of non-overlapping substrings.

3

### Single Character Strings

For a string consisting of a single character, the should return 1 as the maximum number of non-overlapping substrings.

# Conclusion

1

## Key Takeaways

The problem of finding the maximum number of non-overlapping substrings within a string requires a careful analysis of the last occurrence of each character to identify the optimal substrings.

2

## Applicable Scenarios

This problem can be useful in scenarios where you need to identify and extract the maximum number of non-overlapping substrings, such as in text processing, data analysis, or optimization problems.

3

## Further Exploration

This problem can be extended to explore variations, such as finding the maximum number of non-overlapping substrings with specific length constraints or additional conditions.