Maximize Partitions in a String

Given a string **s** of lowercase English alphabets, your task is to return the **maximum** number of substrings formed, after possible **partitions** (probably zero) of **s** such that **no two** substrings have a **common character**.

Examples:

Input: s = "acbbcc"
Output: 2
Explanation: "a" and "cbbcc" are two substrings that do not share any characters between them.

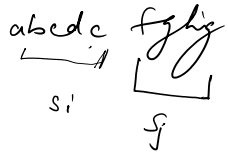
Input: s = "ababcbacadefegdehijhklij"
Output: 3
Explanation: Partitioning at the index 8 and at 15 produces three substrings: “ababcbaca”, “defegde”, and “hijhklij” such that none of them have a common character. So, the maximum number of substrings formed is 3.

Input: s = "aaa"
Output: 1
Explanation: Since the string consists of same characters, no further partition can be performed. Hence, the number of substring (here the whole string is considered as the substring) is 1.

Constraints:
$1 \leq s.size() \leq 10^5$
'a' ≤ s[i] ≤ 'z'

eg: I/p: s = "acbbcc"
O/p: 2

abcdc fghig



[Naive Approach] Using Recursion – Exponential Time

We generate every possible partitioning of the string & then check each partition to ensure that no 2 substrings share a common character. We can generate all partitioning by considering two choices at every point, we make a partition & do not make. The partition having the maximum count of valid substrings among all the generated partitions is the answer.

[Expected Approach 1) Using Greedy Approach and array – O(n) Time & O(1) Space

The idea is to first initialize an array of last of size 26 (to represent all lowercase English letters) with -1, which is used to store the last occurrence index of each character in the string. Then, traverse the string in reverse order to record the last position of each character, ensuring that each character's final appearance is noted.

After storing the last occurrences, initialize

two variables :

- cnt to count the no. of partitions
- a to track the farthest last occurrence of any character.

Iterate through the string again, updating a to reflect the maximum last occurrence index encountered so far. If a matches the current index (i), it means the substring can be closed at this position, forming a valid partition. The partition count (cnt) is then incremented.

This process continues until the end of the string is reached, return cnt (max. no. of partitions).

```
C++ (g++ 5.4)                Start Timer
1  // } Driver Code Ends
7
8  class Solution {
9    public:
10     int maxPartitions(string &s) {
11         // code here
12         // Vector to store the last occurrence of
13         // each character in the string
14         vector<int> last(26, -1);
15
16         // Traverse the string in reverse to record
17         // the last position of each character
18         for(int i=s.size()-1;i>=0;i--){
19             if(last[s[i]-'a'] == -1)last[s[i]-'a']=i;
20         }
21
22         // Count the number of partitions
23         int cnt=0;
24
25         // To track the farthest last occurrence seen
26         int a=-1;
27
28         for(int i=0;i<s.size();i++){
29             a=max(last[s[i]-'a'], a);
30
31             // If the current index matches the
32             // farthest occurrence, form a partition
33             if(a==i)cnt++;
34         }
35         return cnt;
36     }
37 };
38
```

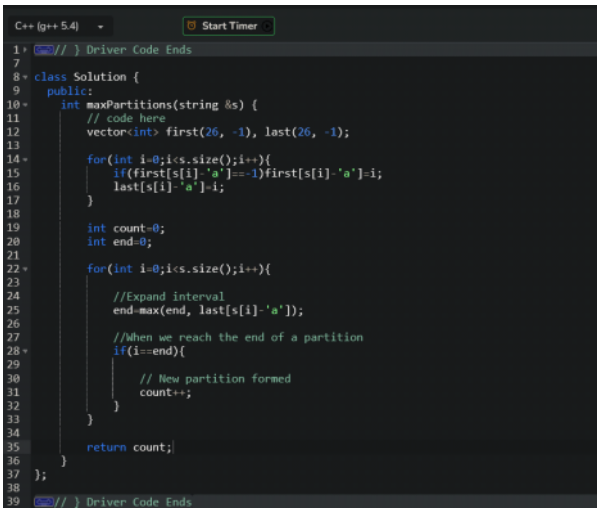
[Expected Approach 2] By merging intervals - O(n) Time & O(1) Space

The main idea is to track first & last occurrence of character & treat them as intervals (first occ, last occ).

Once we have these intervals, we need to merge overlapping ones. If two intervals overlap, it means that the characters in those intervals must be part of the same substring. The merging process ensures that we are not splitting a character across multiple partitions.

After merging all overlapping intervals, the

no of separate intervals left gives the desired result.



```
C++ (g++ 5.4)   Start Timer
// } Driver Code Ends

class Solution {
  public:
    int maxPartitions(string &s) {
        // code here
        vector<int> first(26, -1), last(26, -1);

        for(int i=0;i<s.size();i++){
            if(first[s[i]-'a']==-1)first[s[i]-'a']=i;
            last[s[i]-'a']=i;
        }

        int count=0;
        int end=0;

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

            //Expand interval
            end=max(end, last[s[i]-'a']);

            //When we reach the end of a partition
            if(i==end){

                // New partition formed
                count++;
            }
        }

        return count;
    }
};

// } Driver Code Ends
```