Longest substring where all the characters appear at least K times | Set 3

Difficulty Level : Hard • Last Updated : 03 Jun, 2022

Given a string **str** and an integer **K**, the task is to find the length of the longest substring **S** such that every character in **S** appears at least **K** times.

Examples:

Input: *str = "aabbba", K = 3*
Output: *6*
Explanation: *In substring "aabbba", each character repeats at least k times and its length is 6.*

Input: *str = "ababacb", K = 3*
Output: *0*
Explanation: *There is no substring where each character repeats at least k times.*

Naive Approach: The simplest approach to solve the given problem is discussed in Set 1.

Time Complexity: *O(N^2)*

Auxiliary Space: *O(26)*

Divide and Conquer Approach: The divide and conquer approach for the given problem is discussed in the Set 2.

Time Complexity: *O(N*log N)*

Auxiliary Space: *O(26)*

Efficient Approach: The above two approaches can be optimized further by using Sliding Window technique. Follow the steps below to solve the problem:

- Store the number of unique characters in the string **str** in a variable, say **unique**.
- Initialize an array **freq[]** of size **26** with **{0}** and store the frequency of each character in this array.
- Iterate over the range **[1, unique]** using the variable **curr_unique**. In each iteration, **curr_unique** is the maximum number of unique characters that must be present in the window.
  - Reinitialize the array **freq[]** with **{0}** to store the frequency of each character in this window.
  - Initialize **start** and **end** as **0**, to store the starting and the ending point of the window respectively.
  - Use two variables **cnt**, for storing the number of unique characters and **countK**, for storing the number of characters with at least **K** repeating characters in the current window.
  - Now, iterate a loop while **end < N**, and perform the following:
    - If the value of **cnt** is less than or equals to **curr_unique,** then

character from **start** and decrementing its frequency by **1** in **freq[]**.
- At every step, update the values of **cnt** and **countK**.
- If the value of **cnt** is same as **curr_unique** and each character occurs **at least K times**, then update the overall maximum length and store it in **ans**.

- After completing the above steps, print the value of **ans** as the result.

Below is the implementation of the above approach:

C++

```
// C++ program for the above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the length of
// the longest substring
int longestSubstring(string s, int k)
{
    // Store the required answer
    int ans = 0;

    // Create a frequency map of the
    // characters of the string
    int freq[26] = { 0 };

    // Store the length of the string
    int n = s.size();

    // Traverse the string, s
    for (int i = 0; i < n; i++)

        // Increment the frequency of
        // the current character by 1
        freq[s[i] - 'a']++;
```

```
// characters in string
for (int i = 0; i < 26; i++)
    if (freq[i] != 0)
        unique++;

// Iterate in range [1, unique]
for (int curr_unique = 1;
     curr_unique <= unique;
     curr_unique++) {

    // Initialize frequency of all
    // characters as 0
    memset(freq, 0, sizeof(freq));

    // Stores the start and the
    // end of the window
    int start = 0, end = 0;

    // Stores the current number of
    // unique characters and characters
    // occurring atleast K times
    int cnt = 0, count_k = 0;

    while (end < n) {
        if (cnt <= curr_unique) {
            int ind = s[end] - 'a';

            // New unique character
            if (freq[ind] == 0)
                cnt++;

            freq[ind]++;

            // New character which
            // occurs atleast k times
            if (freq[ind] == k)
                count_k++;

            // Expand window by
            // incrementing end by 1
            end++;
```

```
                // Check if this character
                // is present atleast k times
                if (freq[ind] == k)
                    count_k--;

                freq[ind]--;

                // Check if this character
                // is unique
                if (freq[ind] == 0)
                    cnt--;

                // Shrink the window by
                // incrementing start by 1
                start++;
            }

            // If there are curr_unique
            // characters and each character
            // is atleast k times
            if (cnt == curr_unique
                && count_k == curr_unique)

                // Update the overall
                // maximum length
                ans = max(ans, end - start);
        }
    }

    // return the answer
    return ans;
}

// Driver Code
int main()
{
    string S = "aabbba";
    int K = 3;
    cout << longestSubstring(S, K) << endl;

    return 0;
```

```java
// Java program for the above approach
import java.util.*;
class GFG
{

// Function to find the length of
// the longest subString
static void longestSubString(char[] s, int k)
{

    // Store the required answer
    int ans = 0;

    // Create a frequency map of the
    // characters of the String
    int freq[] = new int[26];

    // Store the length of the String
    int n = s.length;

    // Traverse the String, s
    for (int i = 0; i < n; i++)

        // Increment the frequency of
        // the current character by 1
        freq[s[i] - 'a']++;

    // Stores count of unique characters
    int unique = 0;

    // Find the number of unique
    // characters in String
    for (int i = 0; i < 26; i++)
        if (freq[i] != 0)
            unique++;

    // Iterate in range [1, unique]
    for (int curr_unique = 1;
         curr_unique <= unique;
         curr_unique++)
```

```
Arrays.fill(freq, 0);

// Stores the start and the
// end of the window
int start = 0, end = 0;

// Stores the current number of
// unique characters and characters
// occurring atleast K times
int cnt = 0, count_k = 0;
while (end < n)
{
    if (cnt <= curr_unique)
    {
        int ind = s[end] - 'a';

        // New unique character
        if (freq[ind] == 0)
            cnt++;
        freq[ind]++;

        // New character which
        // occurs atleast k times
        if (freq[ind] == k)
            count_k++;

        // Expand window by
        // incrementing end by 1
        end++;
    }
    else
    {
        int ind = s[start] - 'a';

        // Check if this character
        // is present atleast k times
        if (freq[ind] == k)
            count_k--;
        freq[ind]--;

        // Check if this character
```

```
                // Shrink the window by
                // incrementing start by 1
                start++;
            }

            // If there are curr_unique
            // characters and each character
            // is atleast k times
            if (cnt == curr_unique
                && count_k == curr_unique)

                // Update the overall
                // maximum length
                ans = Math.max(ans, end - start);
        }
    }

    // Print the answer
    System.out.print(ans);
}

// Driver Code
public static void main(String[] args)
{
    String S = "aabbba";
    int K = 3;
    longestSubString(S.toCharArray(), K);
}
}

// This code is contributed by 29AjayKumar
```

Python3

```
# Python3 program for the above approach

# Function to find the length of
# the longest substring
def longestSubstring(s, k) :

    # Store the required answer
```

```
freq = [0]*26

# Store the length of the string
n = len(s)

# Traverse the string, s
for i in range(n) :

    # Increment the frequency of
    # the current character by 1
    freq[ord(s[i]) - ord('a')] += 1

# Stores count of unique characters
unique = 0

# Find the number of unique
# characters in string
for i in range(26) :
    if (freq[i] != 0) :
        unique += 1

# Iterate in range [1, unique]
for curr_unique in range(1, unique + 1) :

    # Initialize frequency of all
    # characters as 0
    Freq = [0]*26

    # Stores the start and the
    # end of the window
    start, end = 0, 0

    # Stores the current number of
    # unique characters and characters
    # occurring atleast K times
    cnt, count_k = 0, 0

    while (end < n) :
        if (cnt <= curr_unique) :
            ind = ord(s[end]) - ord('a')
```

```
            Freq[ind] += 1

            # New character which
            # occurs atleast k times
            if (Freq[ind] == k) :
                count_k += 1

            # Expand window by
            # incrementing end by 1
            end += 1

        else :
            ind = ord(s[start]) - ord('a')

            # Check if this character
            # is present atleast k times
            if (Freq[ind] == k) :
                count_k -= 1

            Freq[ind] -= 1

            # Check if this character
            # is unique
            if (Freq[ind] == 0) :
                cnt -= 1

            # Shrink the window by
            # incrementing start by 1
            start += 1

        # If there are curr_unique
        # characters and each character
        # is atleast k times
        if ((cnt == curr_unique) and (count_k == curr_unique)) :

            # Update the overall
            # maximum length
            ans = max(ans, end - start)

    # Print the answer
    print(ans)
```

```
# This code is contributed by divyesh072019.
```

C#

```
// C# program to implement
// the above approach
using System;

class GFG
{

// Function to find the length of
// the longest substring
static void longestSubstring(string s, int k)
{

    // Store the required answer
    int ans = 0;

    // Create a frequency map of the
    // characters of the string
    int[] freq = new int[26];

    // Store the length of the string
    int n = s.Length;

    // Traverse the string, s
    for (int i = 0; i < n; i++)

        // Increment the frequency of
        // the current character by 1
        freq[s[i] - 'a']++;

    // Stores count of unique characters
    int unique = 0;

    // Find the number of unique
    // characters in string
    for (int i = 0; i < 26; i++)
```

```
for (int curr_unique = 1;
     curr_unique <= unique;
     curr_unique++)
{

    // Initialize frequency of all
    // characters as 0
    for (int i = 0; i < freq.Length; i++)
    {
        freq[i] = 0;
    }

    // Stores the start and the
    // end of the window
    int start = 0, end = 0;

    // Stores the current number of
    // unique characters and characters
    // occurring atleast K times
    int cnt = 0, count_k = 0;
    while (end < n)
    {
        if (cnt <= curr_unique)
        {
            int ind = s[end] - 'a';

            // New unique character
            if (freq[ind] == 0)
                cnt++;
            freq[ind]++;

            // New character which
            // occurs atleast k times
            if (freq[ind] == k)
                count_k++;

            // Expand window by
            // incrementing end by 1
            end++;
        }
        else
```

```
                    // is present atleast k times
                    if (freq[ind] == k)
                        count_k--;
                    freq[ind]--;

                    // Check if this character
                    // is unique
                    if (freq[ind] == 0)
                        cnt--;

                    // Shrink the window by
                    // incrementing start by 1
                    start++;
                }

                // If there are curr_unique
                // characters and each character
                // is atleast k times
                if (cnt == curr_unique
                    && count_k == curr_unique)

                    // Update the overall
                    // maximum length
                    ans = Math.Max(ans, end - start);
            }
        }

        // Print the answer
        Console.Write(ans);
    }

    // Driver Code
    public static void Main()
    {
       string S = "aabbba";
        int K = 3;
        longestSubstring(S, K);
    }
    }

    // This code is contributed by splevel62.
```

```
// Javascript program to implement
// the above approach

// Function to find the length of
// the longest substring
function longestSubstring(s, k)
{

    // Store the required answer
    let ans = 0;

    // Create a frequency map of the
    // characters of the string
    let freq = new Array(26);
    freq.fill(0);

    // Store the length of the string
    let n = s.length;

    // Traverse the string, s
    for (let i = 0; i < n; i++)

        // Increment the frequency of
        // the current character by 1
        freq[s[i].charCodeAt() -
           'a'.charCodeAt()]++;

    // Stores count of unique characters
    let unique = 0;

    // Find the number of unique
    // characters in string
    for (let i = 0; i < 26; i++)
        if (freq[i] != 0)
            unique++;

    // Iterate in range [1, unique]
    for (let curr_unique = 1;
         curr_unique <= unique;
         curr_unique++)
```

```
for (let i = 0; i < freq.length; i++)
{
    freq[i] = 0;
}

// Stores the start and the
// end of the window
let start = 0, end = 0;

// Stores the current number of
// unique characters and characters
// occurring atleast K times
let cnt = 0, count_k = 0;
while (end < n)
{
    if (cnt <= curr_unique)
    {
        let ind = s[end].charCodeAt() -
                  'a'.charCodeAt();

        // New unique character
        if (freq[ind] == 0)
            cnt++;
        freq[ind]++;

        // New character which
        // occurs atleast k times
        if (freq[ind] == k)
            count_k++;

        // Expand window by
        // incrementing end by 1
        end++;
    }
    else
    {
        let ind = s[start].charCodeAt() -
        'a'.charCodeAt();

        // Check if this character
        // is present atleast k times
```

```
                    // Check if this character
                    // is unique
                    if (freq[ind] == 0)
                        cnt--;

                    // Shrink the window by
                    // incrementing start by 1
                    start++;
                }

                // If there are curr_unique
                // characters and each character
                // is atleast k times
                if (cnt == curr_unique
                    && count_k == curr_unique)

                    // Update the overall
                    // maximum length
                    ans = Math.max(ans, end - start);
            }
        }

        // Print the answer
        document.write(ans);
    }

    let S = "aabbba";
    let K = 3;
    longestSubstring(S, K);

</script>
```

Output

```
6
```

Time Complexity: O(N)

Auxiliary Space: O(1)

Previous

L&T Technology Services Interview Experience for Associate Engineer Profile | On-Campus 2021

Next

Maximize count of 1s in an array by repeated division of array elements by 2 at most K times

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Largest substring where all characters appear at least K times | Set 2**
23, Jun 20

02 **Largest sub-string where all the characters appear at least K times**
07, Jun 19

03 **Check if substring S1 appear after any occurrence of substring S2 in given sentence**
05, Jan 22

04 **Count pairs (p, q) such that p occurs in array at least q times and q occurs at least p times**
25, Feb 19

05 **Remove characters that appear more than k times**
07, Nov 18

06 **Longest substring with atmost K characters from the given set of characters**
30, Apr 20

07 **Given an array of size n and a number k, find all elements that appear more than n/k times**
31, May 13

08 **Remove elements that appear strictly less than k times**
10, Nov 18

Article Contributed By :

single__loop
@single__loop

Vote for difficulty

Current difficulty : Hard

Easy | Normal | Medium | Hard | Expert

Improved By : splevel62, 29AjayKumar, divyesh072019, suresh07, sagar0719kumar, adityadas285, nikhatkhan11

Article Tags : frequency-counting, sliding-window, substring, Hash, Searching, Strings

Practice Tags : sliding-window, Searching, Hash, Strings

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

Improve Article | Report Issue

HTML

JavaScript

Bootstrap

ReactJS

NodeJS

Pick Topics to Write

Write Interview Experience

Internships

Video Internship