Longest Substring Without Repeating Characters - GeeksforGeeks

Source: https://www.geeksforgeeks.org/length-of-the-longest-substring-without-repeating-characters/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Longest Substring Without Repeating Characters Last Updated : 20 Jan, 2026 Given a string s having lowercase characters, find the length of the longest substring without repeating characters. Examples: Input : s = "geeksforgeeks" Output : 7 Explanation : The longest substrings without repeating characters are "eksforg" and "ksforge", with lengths of 7. Input : s = "aaa" Output : 1 Explanation : The longest substring without repeating characters is "a" Input: s = "abcdefabcbb" Output: 6 Explanation: The longest substring without repeating characters is "abcdef". Try it on GfG Practice Table of Content [Naive Approach] Substrings Starting From Every Index - O(n^2) Time and O(1) Space [Expected Approach 1] Using Sliding Window - O(n) Time and O(1) Space [Expected Approach 2] Using Last Index of Each Character - O(n) Time and O(1) Space [Naive Approach] Substrings Starting From Every Index - O(n^2) Time and O(1) Space The idea is to find length of longest substring with distinct characters starting from every index and maximum of all such lengths will be our answer. To find the length of the longest substring with distinct characters starting from an index, we create a new visited array of size = 26 to keep track of included characters in the substring. vis[0] checks for 'a', vis[1] checks for 'b', vis[2] checks for 'c' and so on. Note: The array size is fixed at 26, representing the lowercase English alphabet as a constant C++ #include <iostream> #include <vector> using namespace std ; int longestUniqueSubstr (string & s){ int n = s . size (); int res = 0 ; for (int i = 0 ; i < n ; i ++){ // Initializing all characters as not visited vector < bool > vis (26 , false); for (int j = i ; j < n ; j ++){ // If current character is visited // Break the loop if (vis [s [j] - 'a'] == true) break ; // Else update the result if this window is larger, // and mark current character as visited. else { res = max (res , j - i + 1); vis [s [j] - 'a'] = true ; } } } return res ; } int main (){ string s = "geeksforgeeks" ; cout << longestUniqueSubstr (s); return 0 ; } Java class GfG { static int longestUniqueSubstr (String s){ int n = s . length (); int res = 0 ; for (int i = 0 ; i < n ; i ++) { // Initializing all characters as not visited boolean [] vis = new boolean [26] ; for (int j = i ; j < n ; j ++) { // If current character is visited // Break the loop if (vis [s . charAt (j) - 'a'] == true) break ; // Else update the result if this window is // larger, and mark current character as // visited. else { res = Math . max (res , j - i + 1); vis [s . charAt (j) - 'a'] = true ; } } } return res ; } public static void main (String [] args){ String s = "geeksforgeeks" ; System . out . println (longestUniqueSubstr (s)); } } Python def longestUniqueSubstr (s): n = len (s) res = 0 for i in range (n): # Initializing all characters as not visited vis = [False] * 26 for j in range (i , n): # If current character is visited # Break the loop if vis [ord (s [j]) - ord ('a')] == True : break # Else update the result if this window is larger, # and mark current character as visited. else : res = max (res , j - i + 1) vis [ord (s [j]) - ord ('a')] = True return res if __name__ == "__main__" : s = "geeksforgeeks" print (longestUniqueSubstr (s)) C# using System ; class GfG { static int longestUniqueSubstr (string s) { int n = s . Length ; int res = 0 ; for (int i = 0 ; i < n ; i ++) { // Initializing all characters as not visited bool [] vis = new bool [26]; for (int j = i ; j < n ; j ++) { // If current character is visited // Break the loop if (vis [s [j] - 'a'] == true) break ; // Else update the result if this window is larger, // and mark current character as visited. else { res = Math . Max (res , j - i + 1); vis [s [j] - 'a'] = true ; } } } return res ; } static void Main () { string s = "geeksforgeeks" ; Console . WriteLine (longestUniqueSubstr (s)); } } JavaScript function longestUniqueSubstr (s) { let n = s . length ; let res = 0 ; for (let i = 0 ; i < n ; i ++) { // Initializing all characters as not visited let vis = new Array (26). fill (false); for (let j = i ; j < n ; j ++) {

// If current character is visited // Break the loop if (vis [s . charCodeAt (j) - 'a' . charCodeAt (0)] === true) break ; // Else update the result if this window is larger, // and mark current character as visited. else { res = Math . max (res , j - i + 1); vis [s . charCodeAt (j) - 'a' . charCodeAt (0)] = true ; } } } return res ; } // Driver Code let s = "geeksforgeeks" ; console . log (longestUniqueSubstr (s)); Output 7 Time Complexity: O(n 2), The outer loop runs O(n) times and the inner loop may also run up to O(n) in the worst case. The visited array initialization costs O(26), which is constant and does not affect overall complexity. Auxiliary Space: O(1), vis array has size 26 which is constant. [Expected Approach 1] Using Sliding Window - O(n) Time and O(1) Space The idea is to maintain a window of distinct characters. The window is initialized as single character. We keep extending the window on the right side till we see distinct characters. When we see a repeating character, we remove characters from the left side of the window. We keep track of the maximum length window. Below are the detailed steps: Initialize two pointers left and right with 0 , which define the current window being considered. The right pointer moves from left to right, extending the current window. If the character at right pointer is not visited, it's marked as visited. If the character at right pointer is visited, it means there is a repeating character. The left pointer moves to the right while marking visited characters as false until the repeating character is no longer part of the current window. The length of the current window (right - left + 1) is calculated and answer is updated accordingly. Working: C++ #include <iostream> #include <vector> using namespace std ; int longestUniqueSubstr (string & s) { if (s . length () == 0 || s . length () == 1) return s . length (); int res = 0 ; vector < bool > vis (26 , false); // left and right pointer of sliding window int left = 0 , right = 0 ; while (right < s . length ()) { // If character is repeated, move left pointer marking // visited characters as false until the repeating // character is no longer part of the current window while (vis [s [right] - 'a'] == true) { vis [s [left] - 'a'] = false ; left ++ ; } vis [s [right] - 'a'] = true ; // The length of the current window (right - left + 1) // is calculated and answer is updated accordingly. res = max (res , (right - left + 1)); right ++ ; } return res ; } int main () { string s = "geeksforgeeks" ; cout << longestUniqueSubstr (s); return 0 ; } Java class GfG { static int longestUniqueSubstr (String s) { if (s . length () == 0 || s . length () == 1) return s . length (); int res = 0 ; boolean [] vis = new boolean [26] ; // left and right pointer of sliding window int left = 0 , right = 0 ; while (right < s . length ()) { // If character is repeated, move left pointer marking // visited characters as false until the repeating // character is no longer part of the current window while (vis [s . charAt (right) - 'a'] == true) { vis [s . charAt (left) - 'a'] = false ; left ++ ; } vis [s . charAt (right) - 'a'] = true ; // The length of the current window (right - left + 1) // is calculated and answer is updated accordingly. res = Math . max (res , (right - left + 1)); right ++ ; } return res ; } public static void main (String [] args) { String s = "geeksforgeeks" ; System . out . println (longestUniqueSubstr (s)); } } Python MAX_CHAR = 26 def longestUniqueSubstr (s): if len (s) == 0 or len (s) == 1 : return len (s) res = 0 vis = [False] * 26 # left and right pointer of sliding window left = 0 right = 0 while right < len (s): # If character is repeated, move left pointer marking # visited characters as false until the repeating # character is no longer part of the current window while vis [ord (s [right]) - ord ('a')] == True : vis [ord (s [left]) - ord ('a')] = False left += 1 vis [ord (s [right]) - ord ('a')] = True # The length of the current window (right - left + 1) # is calculated and answer is updated accordingly. res = max (res , (right - left + 1)) right += 1 return res if __name__ == "__main__" : s = "geeksforgeeks" print (longestUniqueSubstr (s)) C# using System ; class GfG { static int longestUniqueSubstr (string s) { if (s . Length == 0 || s . Length == 1) return s . Length ; int res = 0 ; bool [] vis = new bool [26]; // left and right pointer of sliding window int left = 0 , right = 0 ; while (right < s . Length) { while (vis [s [right] - 'a'] == true) { vis [s [left] - 'a'] = false ; left ++ ; } vis [s [right] - 'a'] = true ; // The length of the current window (right - left + 1) // is calculated and answer is updated accordingly. res = Math . Max (res , (right - left + 1)); right ++ ; } return res ; } static void Main () { string s = "geeksforgeeks" ; Console . WriteLine (longestUniqueSubstr (s)); } } JavaScript function longestUniqueSubstr (s) { if (s . length === 0 || s . length === 1) return s . length ; let res = 0 ; let vis = new Array (26). fill (false); // left and right pointer of sliding window let left = 0 , right = 0 ; while (right < s . length) { while (vis [s [right]. charCodeAt (0) - 'a' . charCodeAt (0)] === true) { vis [s [left]. charCodeAt (0) - 'a' . charCodeAt (0)] = false ; left ++ ; } vis [s [right]. charCodeAt (0) - 'a' . charCodeAt (0)] = true ; res = Math . max (res , (right - left + 1)); right ++ ; } return res ; } // Driver Code const s = "geeksforgeeks" ; console . log (longestUniqueSubstr (s)); Output 7 [Expected Approach 2] Using Last Index of Each Character - O(n) Time and O(1) Space The approach stores the last indexes of already visited characters. The idea is to maintain a window of distinct characters. Start from the first character, and keep extending the window on the right side till we see distinct characters. When we see a repeating character, we check for the last index of the repeated character: If last index of repeated character >= starting inde x of the current window, then we update the starting index of the

current window to last index of repeated character + 1 to remove the repeated character. If last index of repeated character < starting inde x of the current window, then it means that the repeated character is already outside the current window so the window size remains unchanged. After iterating over all the characters, the largest window size will be our answer. Working: C++ #include <iostream> #include <vector> using namespace std ; int longestUniqueSubstr (string & s) { int n = s . size (); int res = 0 ; vector < int > lastIndex (26 , -1); // Initialize start of current window int start = 0 ; // Move end of current window for (int end = 0 ; end < n ; end ++) { start = max (start , lastIndex [s [end] - 'a'] + 1); // Update result if we get a larger window res = max (res , end - start + 1); // Update last index of s[end] lastIndex [s [end] - 'a'] = end ; } return res ; } int main () { string s = "geeksforgeeks" ; cout << longestUniqueSubstr (s); return 0 ; } Java class GfG { static int longestUniqueSubstr (String s) { int n = s . length (); int res = 0 ; // last index of all characters is initialized as -1 int [] lastIndex = new int [26] ; for (int i = 0 ; i < 26 ; i ++) { lastIndex [i] = - 1 ; } // Initialize start of current window int start = 0 ; // Move end of current window for (int end = 0 ; end < n ; end ++) { // Find the last index of s[end] // Update starting index of current window as // maximum of current value of end and last index + 1 start = Math . max (start , lastIndex [s . charAt (end) - 'a'] + 1); // Update result if we get a larger window res = Math . max (res , end - start + 1); // Update last index of s[end] lastIndex [s . charAt (end) - 'a'] = end ; } return res ; } public static void main (String [] args) { String s = "geeksforgeeks" ; System . out . println (longestUniqueSubstr (s)); } } Python def longestUniqueSubstr (s): n = len (s) res = 0 lastIndex = [- 1] * 26 # Initialize start of current window start = 0 # Move end of current window for end in range (n): start = max (start , lastIndex [ord (s [end]) - ord ('a')] + 1) # Update result if we get a larger window res = max (res , end - start + 1) # Update last index of s[end] lastIndex [ord (s [end]) - ord ('a')] = end return res if __name__ == "__main__" : s = "geeksforgeeks" print (longestUniqueSubstr (s)) C# using System ; class GfG { const int MAX_CHAR = 26 ; public static int longestUniqueSubstr (string s) { int n = s . Length ; int res = 0 ; // last index of all characters is initialized as -1 int [] lastIndex = new int [MAX_CHAR]; for (int i = 0 ; i < MAX_CHAR ; i ++) { lastIndex [i] = - 1 ; } // Initialize start of current window int start = 0 ; // Move end of current window for (int end = 0 ; end < n ; end ++) { start = Math . Max (start , lastIndex [s [end] - 'a'] + 1); // Update result if we get a larger window res = Math . Max (res , end - start + 1); // Update last index of s[end] lastIndex [s [end] - 'a'] = end ; } return res ; } static void Main () { string s = "geeksforgeeks" ; Console . WriteLine (longestUniqueSubstr (s)); } } JavaScript function longestUniqueSubstr (s) { const n = s . length ; let res = 0 ; // last index of all characters is initialized as -1 const lastIndex = new Array (26). fill (- 1); // Initialize start of current window let start = 0 ; // Move end of current window for (let end = 0 ; end < n ; end ++) { start = Math . max (start , lastIndex [s . charCodeAt (end) - 'a' . charCodeAt (0)] + 1); // Update result if we get a larger window res = Math . max (res , end - start + 1); // Update last index of s[end] lastIndex [s . charCodeAt (end) - 'a' . charCodeAt (0)] = end ; } return res ; } // Driver Code const s = "geeksforgeeks" ; console . log (longestUniqueSubstr (s)); Output 7 Comment Article Tags: Article Tags: Strings DSA Microsoft Amazon Morgan Stanley Housing.com + 2 More