Longest Common Substring - GeeksforGeeks

Source: https://www.geeksforgeeks.org/longest-common-substring/

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 Common Substring Last Updated : 23 Jan, 2026 Given two strings s1 and s2 , find the length of the longest common substring. A substring is a sequence of characters that appears contiguously in a string. Example: Input: s1 = "GeeksforGeeks", s2 = "GeeksQuiz" Output : 5 Explanation: The longest common substring is "Geeks" and is of length 5. Input: s1 = "abcdxyz", s2 = "xyzabcd" Output : 4 Explanation: The longest common substring is "abcd" and is of length 4. Input: s1 = "abc", s2 = "" Output : 0 Try it on GfG Practice Table of Content [Naive Approach] Using Recursion - O(m×n×k) Time and O(k) Space [Better Approach] Using Dynamic Programming - O(m×n) Time and O(m×n) Space [Efficient Approach] Using Space Optimized DP - O(m×n) Time and O(n) Space [Naive Approach] Using Recursion - O(m×n×k) Time and O(k) Space The idea is to use recursion to compute the longest common suffix ending at every pair of positions in both strings. For each (i, j) , we check if the characters match; if they do, we move one step back in both strings and add 1 to the suffix length. By exploring all such pairs for all positions, we find the maximum suffix length, which becomes the longest common substring. C++ #include <iostream> #include <string> using namespace std ; // Returns length of the longest common suffix // ending with the last characters int commonSuffix (string & s1 , string & s2 , int m , int n) { if (m == 0 || n == 0 || s1 [m - 1] != s2 [n - 1]) return 0 ; return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1); } int longCommSubstr (string & s1 , string & s2) { int res = 0 ; // Find the longest common substring // and take the maximum of all. for (int i = 1 ; i <= s1 . size (); i ++) { for (int j = 1 ; j <= s2 . size (); j ++) { res = max (res , commonSuffix (s1 , s2 , i , j)); } } return res ; } int main () { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; cout << longCommSubstr (s1 , s2) << endl ; return 0 ; } C #include <stdio.h> #include <string.h> #include <stdlib.h> // Returns length of the longest common suffix // ending with the last characters int commonSuffix (char * s1 , char * s2 , int m , int n) { if (m == 0 || n == 0 || s1 [m - 1] != s2 [n - 1]) return 0 ; return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1); } int longCommSubstr (char * s1 , char * s2) { int res = 0 ; int m = strlen (s1); int n = strlen (s2); // Find the longest common substring // and take the maximum of all. for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { res = res > commonSuffix (s1 , s2 , i , j) ? res : commonSuffix (s1 , s2 , i , j); } } return res ; } int main () { char s1 [] = "GeeksforGeeks" ; char s2 [] = "GeeksQuiz" ; printf ("%d \n " , longCommSubstr (s1 , s2)); return 0 ; } Java class GfG { // Returns length of the longest common suffix // ending with the last characters static int commonSuffix (String s1 , String s2 , int m , int n) { if (m == 0 || n == 0 || s1 . charAt (m - 1) != s2 . charAt (n - 1)) { return 0 ; } return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1); } static int longCommSubstr (String s1 , String s2) { int res = 0 ; int m = s1 . length (); int n = s2 . length (); // Find the longest common substring // and take the maximum of all. for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { res = Math . max (res , commonSuffix (s1 , s2 , i , j)); } } return res ; } public static void main (String [] args) { String s1 = "GeeksforGeeks" ; String s2 = "GeeksQuiz" ; System . out . println (longCommSubstr (s1 , s2)); } } Python # Returns length of the longest common suffix # ending with the last characters def commonSuffix (s1 , s2 , m , n): if m == 0 or n == 0 or s1 [m - 1] != s2 [n - 1]: return 0 return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1) def longCommSubstr (s1 , s2): res = 0 m = len (s1) n = len (s2) # Find the longest common substring # and take the maximum of all. for i in range (1 , m + 1): for j in range (1 , n + 1): res = max (res , commonSuffix (s1 , s2 , i , j)) return res if __name__ == "__main__" : s1 = "GeeksforGeeks" s2 = "GeeksQuiz" print (longCommSubstr (s1 , s2)) C# using System ; class GfG { // Returns length of the longest suffix // ending with the last characters static int commonSuffix (string s1 , string s2 , int m , int n) { if (m == 0 || n == 0 || s1 [m - 1] != s2 [n - 1]) { return 0 ; } return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1); } static int longCommSubstr (string s1 , string s2) { int res = 0 ; int m = s1 . Length ; int n = s2 . Length ; // Find the longest common substring // at every pair of characters for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { res = Math . Max (res , commonSuffix (s1 , s2 , i , j)); } } return res ; } static void Main () { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; Console . WriteLine (longCommSubstr (s1 , s2)); } } JavaScript // Returns length of the longest common substring // ending with the last characters function commonSuffix (s1 , s2 , m , n) { if (m === 0 || n === 0 || s1 [m - 1] !== s2 [n - 1]) { return 0 ; } return 1 + commonSuffix (s1 , s2 , m - 1 , n - 1); } function longCommSubstr (s1 , s2) { let res = 0 ; let m = s1 . length ; let n = s2 . length ; // Find the longest common substring // and take the maximum of all. for (let i = 1 ; i <= m ; i ++) { for (let j = 1 ; j <= n ; j ++) { res = Math . max (res , commonSuffix (s1 , s2 , i , j)); } } return res ; } //Driver Code let s1 = "GeeksforGeeks" ; let s2 = "GeeksQuiz" ; console . log (longCommSubstr (s1 , s2)); Output 5 Time Complexity: O(m×n×k), where k = min(n,m). Space Complexity: O(k). [Better Approach] Using Dynamic Programming - O(m×n) Time and O(m×n) Space From the recursive approach we can observe that the same (i, j) states are solved repeatedly while computing common suffixes, leading to overlapping subproblems. So, The idea is to convert the recursive suffix logic into a tabulated DP table, where each cell stores the length of the longest common suffix ending at those positions. For each pair of characters, we either extend the previous match (if characters match) or reset the suffix length to zero (if they don't). As we fill the table row by row, we continuously update the global maximum. This eliminates repeated recursion and efficiently computes the longest common substring. C++ #include <iostream> #include <string> #include <vector> using namespace std ; int longCommSubstr (string & s1 , string & s2) { int m = s1 . length (); int n = s2 . length (); // Create a table to store lengths of longest // common suffixes of substrings. vector < vector < int >> dp (m + 1 , vector < int > (n + 1 , 0)); // Build dp[m+1][n+1] in bottom-up fashion. int res = 0 ; for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { dp [i][j] = dp [i - 1][j - 1] + 1 ; res = max (res , dp [i][j]); } else { dp [i][j] = 0 ; } } } return res ; } int main () { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; cout << longCommSubstr (s1 , s2) << endl ; return 0 ; } C #include <stdio.h> #include <string.h> int longCommSubstr (char * s1 , char * s2) { int m = strlen (s1); int n = strlen (s2); // Create a table to store lengths of longest // common suffixes of substrings. int dp [m + 1][n + 1]; // Build dp[m+1][n+1] in bottom-up fashion. int res = 0 ; for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { dp [i][j] = dp [i - 1][j - 1] + 1 ; if (dp [i][j] > res) { res = dp [i][j]; } } else { dp [i][j] = 0 ; } } } return res ; } int main () { char s1 [] = "GeeksforGeeks" ; char s2 [] = "GeeksQuiz" ; printf ("%d \n " , longCommSubstr (s1 , s2)); return 0 ; } Java class GFG { public static int longCommSubstr (String s1 , String s2) { int m = s1 . length (); int n = s2 . length (); // Create a table to store lengths of longest // common suffixes of substrings int [][] dp = new int [m + 1][n + 1] ; int res = 0 ; // Build dp[m+1][n+1] in bottom up fashion. for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { if (s1 . charAt (i - 1) == s2 . charAt (j - 1)) { dp [i][j] = dp [i - 1][j - 1] + 1 ; res = Math . max (res , dp [i][j]); } else { dp [i][j] = 0 ; } } } return res ; } public static void main (String [] args) { String s1 = "GeeksforGeeks" ; String s2 = "GeeksQuiz" ; System . out . println (longCommSubstr (s1 , s2)); } } Python def longCommSubstr (s1 , s2): m = len (s1) n = len (s2) # Create a table to store lengths of longest # common suffixes of substrings. dp = [[0] * (n + 1) for _ in range (m + 1)] res = 0 # Build dp[m+1][n+1] in bottom-up fashion. for i in range (1 , m + 1): for j in range (1 , n + 1): if s1 [i - 1] == s2 [j - 1]: dp [i][j] = dp [i - 1][j - 1] + 1 res = max (res , dp [i][j]) else : dp [i][j] = 0 return res if __name__ == "__main__" : s1 = "GeeksforGeeks" s2 = "GeeksQuiz" print (longCommSubstr (s1 , s2)) C# using System ; public class GFG { public static int longCommSubstr (string s1 , string s2) { int m = s1 . Length ; int n = s2 . Length ; // Create a table to store lengths of longest // common suffixes of substrings. int [,] dp = new int [m + 1 , n + 1]; int res = 0 ; // Build dp[m+1][n+1] in bottom-up fashion. for (int i = 1 ; i <= m ; i ++) { for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { dp [i , j] = dp [i - 1 , j - 1] + 1 ; res = Math . Max (res , dp [i , j]); } else { dp [i , j] = 0 ; } } } return res ; } public static void Main (string [] args) { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; Console . WriteLine (longCommSubstr (s1 , s2)); } } JavaScript function longCommSubstr (s1 , s2) { let m = s1 . length ; let n = s2 . length ; // Create a table to store lengths of longest // common suffixes of substrings. let dp = Array . from (Array (m + 1), () => Array (n + 1). fill (0)); let res = 0 ; // Build dp[m+1][n+1] in bottom-up fashion. for (let i = 1 ; i <= m ; i ++) { for (let j = 1 ; j <= n ; j ++) { if (s1 [i - 1] === s2 [j - 1]) { dp [i][j] = dp [i - 1][j - 1] + 1 ; res = Math . max (res , dp [i][j]); } else { dp [i][j] = 0 ; } } } return res ; } //Driver Code let s1 = "GeeksforGeeks" ; let s2 = "GeeksQuiz" ; console . log (longCommSubstr (s1 , s2)); Output 5 [Efficient Approach] Using Space Optimized DP - O(m×n) Time and O(n) Space

From the DP state transition, we can observe that, dp[i][j] = 1 + dp[i–1][j–1], if s1[i–1] == s2[j–1] dp[i][j] = 0, otherwise Since each state dp[i][j] depends only on the diagonal value from the previous row, dp[i–1][j–1], we do not need to store the entire 2D DP table. Keeping only the previous row is sufficient, which allows us to optimize the space from O(m×n) to O(n). We keep a prev[] array for the last row and build a curr[] array for the current row. When the characters match, we extend the substring using prev[j–1]; when they don’t, we reset the length to 0. After completing a row, curr becomes the new prev. This keeps the logic exactly the same while using only a linear-size array instead of the full table. C++ #include <iostream> #include <vector> using namespace std ; int longCommSubstr (string & s1 , string & s2) { int m = s1 . length (); int n = s2 . length (); // Create a 1D array to store the previous row's results vector < int > prev (n + 1 , 0); int res = 0 ; for (int i = 1 ; i <= m ; i ++) { // Create a temporary array to store the current row vector < int > curr (n + 1 , 0); for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { curr [j] = prev [j - 1] + 1 ; res = max (res , curr [j]); } else { curr [j] = 0 ; } } // Move the current row's data to the previous row prev = curr ; } return res ; } int main () { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; cout << longCommSubstr (s1 , s2) << endl ; return 0 ; } C #include <stdio.h> #include <string.h> int longCommSubstr (char * s1 , char * s2) { int m = strlen (s1); int n = strlen (s2); // Create a 1D array to store the previous row's results int prev [n + 1]; memset (prev , 0 , sizeof (prev)); int res = 0 ; for (int i = 1 ; i <= m ; i ++) { // Create a temporary array to store the current row int curr [n + 1]; memset (curr , 0 , sizeof (curr)); for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { curr [j] = prev [j - 1] + 1 ; if (curr [j] > res) { res = curr [j]; } } else { curr [j] = 0 ; } } // Move the current row's data to the previous row memcpy (prev , curr , sizeof (curr)); } return res ; } int main () { char s1 [] = "GeeksforGeeks" ; char s2 [] = "GeeksQuiz" ; printf ("%d \n " , longCommSubstr (s1 , s2)); return 0 ; } Java class GfG { public static int longCommSubstr (String s1 , String s2) { int m = s1 . length (); int n = s2 . length (); // Create a 1D array to store the previous row's results int [] prev = new int [n + 1] ; int res = 0 ; for (int i = 1 ; i <= m ; i ++) { // Create a temporary array to store the current row int [] curr = new int [n + 1] ; for (int j = 1 ; j <= n ; j ++) { if (s1 . charAt (i - 1) == s2 . charAt (j - 1)) { curr [j] = prev [j - 1] + 1 ; res = Math . max (res , curr [j]); } else { curr [j] = 0 ; } } // Move the current row's data to the previous row prev = curr ; } return res ; } public static void main (String [] args) { String s1 = "GeeksforGeeks" ; String s2 = "GeeksQuiz" ; System . out . println (longCommSubstr (s1 , s2)); } } Python def longCommSubstr (s1 , s2): m = len (s1) n = len (s2) # Create a 1D array to store the previous row's results prev = [0] * (n + 1) res = 0 for i in range (1 , m + 1): # Create a temporary array to store the current row curr = [0] * (n + 1) for j in range (1 , n + 1): if s1 [i - 1] == s2 [j - 1]: curr [j] = prev [j - 1] + 1 res = max (res , curr [j]) else : curr [j] = 0 # Move the current row's data to the previous row prev = curr return res if __name__ == "__main__" : s1 = "GeeksforGeeks" s2 = "GeeksQuiz" print (longCommSubstr (s1 , s2)) C# using System ; public class GFG { public static int longCommSubstr (string s1 , string s2) { int m = s1 . Length ; int n = s2 . Length ; // Create a 1D array to store the previous row's results int [] prev = new int [n + 1]; int res = 0 ; for (int i = 1 ; i <= m ; i ++) { // Create a temporary array to store the current row int [] curr = new int [n + 1]; for (int j = 1 ; j <= n ; j ++) { if (s1 [i - 1] == s2 [j - 1]) { curr [j] = prev [j - 1] + 1 ; res = Math . Max (res , curr [j]); } else { curr [j] = 0 ; } } // Move the current row's data to the previous row prev = curr ; } return res ; } public static void Main (string [] args) { string s1 = "GeeksforGeeks" ; string s2 = "GeeksQuiz" ; Console . WriteLine (longCommSubstr (s1 , s2)); } } JavaScript function longCommSubstr (s1 , s2) { let m = s1 . length ; let n = s2 . length ; // Create a 1D array to store the previous row's results let prev = new Array (n + 1). fill (0); let res = 0 ; for (let i = 1 ; i <= m ; i ++) { // Create a temporary array to store the current row let curr = new Array (n + 1). fill (0); for (let j = 1 ; j <= n ; j ++) { if (s1 [i - 1] === s2 [j - 1]) { curr [j] = prev [j - 1] + 1 ; res = Math . max (res , curr [j]); } else { curr [j] = 0 ; } } // Move the current row's data to the previous row prev = curr ; } return res ; } // Driver Code let s1 = "GeeksforGeeks" ; let s2 = "GeeksQuiz" ; console . log (longCommSubstr (s1 , s2)); Output 5 Comment Article Tags: Article Tags: Strings Dynamic Programming DSA Microsoft Amazon Morgan Stanley + 2 More