

Longest Substring with Distinct Characters

Given a string **str**, find the length of the longest substring without repeating characters.

Example:

For "ABDEFGABEF", the longest substring are "BDEFGA" and "DEFGAB", with length 6.

For "BBBB" the longest substring is "B", with length 1.

For "GEEKSFORGEEKS", there are two longest substrings shown in the below diagrams, with length 7



Method 1 (Simple : $O(n^3)$): We can consider all substrings one by one and check for each substring whether it contains all unique characters or not. There will be $n * (n+1)/2$ substrings. Whether a substring contains all unique characters or not can be checked in linear time by scanning it from left to right and keeping a map of visited characters.

C++

Java

```
// C++ program to find the length of the longest substring  
// without repeating characters
```

Track Progress

58 of 67 Complete. (87%)

Dash



Articles

// Note : Default values in visited are false

```
vector<bool> visited(26);
```



Videos

```
for (int k = i; k <= j; k++) {  
    if (visited[str[k] - 'a'] == true)  
        return false;
```



Problems

```
    visited[str[k] - 'a'] = true;
```

```
}
```

```
return true;
```



Quiz

```
}
```

```
// Returns length of the longest substring
```

```
// with all distinct characters.
```

```
int longestUniqueSubsttr(string str)
```

```
{
```

```
    int n = str.size();
```

```
    int res = 0; // result
```

```
    for (int i = 0; i < n; i++)
```

```
        for (int j = i; j < n; j++)
```

```
            if (areDistinct(str, i, j))
```

```
                res = max(res, j - i + 1);
```

```
    return res;
```

```
}
```

```
// Driver code
```

```
int main()
```

```
{
```

```
    string str = "geeksforgeeks";
```

```
    cout << "The input string is " << str << endl;
```

```
    int len = longestUniqueSubsttr(str);
```

```
    cout << "The length of the longest non-repeating "  
            "character substring is "
```

```
        << len;
```

```
    return 0;
```

```
}
```



Menu

Track Progress

58 of 67 Complete. (87%)

Output

Dash



All

Time Complexity: $O(n^3)$ since we are processing n^2 substrings with maximum length n .



Articles

Auxiliary Space: $O(1)$



Videos

Method 2 (Better : $O(n^2)$) The idea is to use window sliding. Whenever we see repetition, we remove the previous occurrence and slide the window.



Problems

C++

Java



Quiz

```
// C++ program to find the length of the longest substring
// without repeating characters
#include <bits/stdc++.h>
using namespace std;

int longestUniqueSubsttr(string str)
{
    int n = str.size();
    int res = 0; // result

    for (int i = 0; i < n; i++) {

        // Note : Default values in visited are false
        vector<bool> visited(256);

        for (int j = i; j < n; j++) {

            // If current character is visited
            // Break the loop
            if (visited[str[j]] == true)
                break;

            // Else update the result if
            // this window is larger, and mark
            // current character as visited.
```

Menu



Track Progress

58 of 67 Complete. (87%)

}

Dash



All

```
// window
visited[str[i]] = false;
}
return res;
}

// Driver code
int main()
{
    string str = "geeksforgeeks";
    cout << "The input string is " << str << endl;
    int len = longestUniqueSubsttr(str);
    cout << "The length of the longest non-repeating "
           "character substring is "
           << len;
    return 0;
}
```



Articles



Videos



Problems



Output

The input string is geeksforgeeks

The length of the longest non-repeating character substring is 7

Time Complexity: $O(n^2)$ since we are traversing each window to remove all repetitions.

Auxiliary Space: $O(1)$

Method 3 (Linear Time): Using this solution the problem can be solved in linear time using the window sliding technique. Whenever we see repetition, we remove the window till the repeated string.

C++

Java

Menu

#include <iostream>



Track Progress

58 of 67 Complete. (87%)

Dash



```
if (str.length() == 0)
    return 0;
```



```
// if string length 1
if (str.length() == 1)
    return 1;
```



```
// if string length is more than 2
int maxLength = 0;
bool visited[256] = { false };
```



```
// left and right pointer of sliding window
int left = 0, right = 0;
for (; right < str.length(); right++) {
```



```
// if character is not visited then mark visited
if (visited[str[right]] == false)
    visited[str[right]] = true;
```

```
// if character is visited
else {
    /* capture the unique string from [left ,
    right). Not including right since repeating
    character is at index right.
    */
    maxLength = max(maxLength, (right - left));
```

```
/* Mark all characters until repeating
    character as unvisited but not the repeating
    character as it is in the new unique string.
    However move window past the repeating
    character.
    */
```

```
while (left < right) {
    if (str[left] != str[right])
        visited[str[left]] = false;
    else {
```

Menu

Track Progress

58 of 67 Complete. (87%)

}

Dash



All

}



Articles

// compare current left,right with previous result

`maxLength = max(maxLength, (right - left));`

Videos

`return maxLength;`

}



Problems

`int main()`

{

`string s = "GeeksForGeeks!";`

Quiz

`cout << longestUniqueSubsttr(s) << endl;`

//expected result : 7

//unique substring = eksForG

`return 0;`

}



Output

7

Time Complexity: $O(n)$ since we slide the window whenever we see any repetitions.

Auxiliary Space: $O(1)$

Method 4 (Linear Time): Let us talk about the linear time solution now. This solution uses extra space to store the last indexes of already visited characters. The idea is to scan the string from left to right, keep track of the maximum length Non-Repeating Character Substring seen so far in **res**. When we traverse the string, to know the length of current window we need two indexes.

Menu

1) Ending index (**j**) : We consider current index as ending index.

2) Starting index (**i**) : It is same as previous window if current character was not



Track Progress

58 of 67 Complete. (87%)

lastIndex[]. If `lastIndex[str[j]] + 1` is more than previous start, then we updated the



C++

Java



Articles



Videos



Problems



Quiz

```
// C++ program to find the length of the longest substring
// without repeating characters
#include <bits/stdc++.h>
using namespace std;
#define NO_OF_CHARS 256

int longestUniqueSubsttr(string str)
{
    int n = str.size();

    int res = 0; // result

    // last index of all characters is initialized
    // as -1
    vector<int> lastIndex(NO_OF_CHARS, -1);

    // Initialize start of current window
    int i = 0;

    // Move end of current window
    for (int j = 0; j < n; j++) {

        // Find the last index of str[j]
        // Update i (starting index of current window)
        // as maximum of current value of i and last
        // index plus 1
        i = max(i, lastIndex[str[j]] + 1);

        // Update result if we get a larger window
        res = max(res, j - i + 1);

        // Update last index of j.
        lastIndex[str[j]] = j;
    }
```

Menu

Track Progress

58 of 67 Complete. (87%)

Dash



All

```
string str = "geeksforgeeks";
cout << "The input string is " << str << endl;
int len = longestUniqueSubsttr(str);
cout << "The length of the longest non-repeating "
      "character substring is "
      << len;
return 0;
}
```


Problems

Quiz

Output

The input string is geeksforgeeks
The length of the longest non-repeating character substring is 7

Time Complexity: $O(n + d)$ where n is length of the input string and d is number of characters in input string alphabet. For example, if string consists of lowercase English characters then value of d is 26.

Auxiliary Space: $O(d)$

Alternate Implementation :

C++

Java

```
#include <bits/stdc++.h>
using namespace std;

int longestUniqueSubsttr(string s)
{
    // Creating a map to store the last positions
    // of occurrence
    map<char, int> seen ;
    int maximum_length = 0;

    // Starting the initial point of window to index 0
```

Menu

Track Progress

58 of 67 Complete. (87%)

{

Dash



```
if (seen.find(s[end]) != seen.end())
{
```

Articles

```
// If we have seen the number, move the start
// pointer to position after the last occurrence
start = max(start, seen[s[end]] + 1);
```

}



```
// Updating the last seen value of the character
seen[s[end]] = end;
maximum_length = max(maximum_length,
end - start + 1);
```

Quiz

```
}
return maximum_length;
```

}

// Driver code

int main()

{

```
string s = "geeksforgeeks";
cout << "The input String is " << s << endl;
int length = longestUniqueSubsttr(s);
```

```
cout<<"The length of the longest non-repeating character "
<<"substring is "
<< length;
```

}

Output

The input String is geeksforgeeks

The length of the longest non-repeating character substring is 7



Time Complexity: $O(n + d)$ where n is length of the input string and d is number of characters in input string alphabet. For example, if string consists of lowercase

Track Progress

58 of 67 Complete. (87%)

As an exercise, try the modified version of the above problem where you need to



Articles



Videos

Method 5 (Linear time): In this method we will apply KMP Algorithm technique, to solve the problem. We maintain an Unordered Set to keep track of the maximum non repeating char sub string (Instead of standard LPS array of KMP). When ever we find a repeating char, then we clear the Set and reset len to zero. Rest everything is almost similar to KMP.



C++

Java



Problems

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

int longestSubstrDistinctChars(string s)
{
    if (s.length() == 0)
        return 0;
    int n = s.length();
    set<char> st;
    int len = 1;
    st.insert(s[0]);
    int i = 1;
    int maxlen = 0;
    while (i < n)
    {
        if (s[i] != s[i - 1] && st.find(s[i]) == st.end())
            len++;
        i++;
        if (len > maxlen)
        {
            maxlen = len;
        }
    }
    else
```

90% Money-Back!

Courses

Tutorials

Jobs

Practice

Contests

Menu



P



Track Progress

58 of 67 Complete. (87%)

`i++;`

Dash



All

```
        st.clear();
        i = i - len + 1;
        len = 0;
    }
}
return max(maxLen, len);
}

// Driver program to test above function
int main()
{
    string str = "abcabcbb";
    cout << "The input string is " << str << endl;
    int len = longestSubstrDistinctChars(str);
    cout << "The length of the longest non-repeating character substring
    return 0;
}
```



Articles



Videos



Problems



Quiz



Output

The input string is abcabcbb

The length of the longest non-repeating character substring 3

Time Complexity : $O(n)$ where n is the input string length

Auxiliary Space: $O(m)$ where m is the length of the resultant sub string

[Mark as Read](#)[Report An Issue](#)[Menu](#)

If you are facing any issue on this page. Please let us know.

[Track Progress](#)

58 of 67 Complete. (87%)