

Length of the longest substring without repeating characters

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

G	E	E	K	S	F	O	R	G	E	E	K	S
---	---	---	---	---	---	---	---	---	---	---	---	---

G	E	E	K	S	F	O	R	G	E	E	K	S
---	---	---	---	---	---	---	---	---	---	---	---	---

G	E	E	K	S	F	O	R	G	E	E	K	S
---	---	---	---	---	---	---	---	---	---	---	---	---

The desired time complexity is $O(n)$ where n is the length of the string.

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. Time complexity of this solution would be $O(n^3)$.

C++

```
#include <bits/stdc++.h>

using namespace std;

bool areDistinct(string str, int i, int j)
```

```
{

    vector<bool> visited(26);

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

        if (visited[str[k] - 'a'] == true)

            return false;

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

    }

    return true;

}

int longestUniqueSubsttr(string str)
{

    int n = str.size();

    int res = 0;

    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;

}

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;
}
```

Java

```
import java.io.*;

import java.util.*;

class GFG{

    public static Boolean areDistinct(String str,

                                     int i, int j)

    {

        boolean[] visited = new boolean[26];

        for(int k = i; k <= j; k++)

        {

            if (visited[str.charAt(k) - 'a'] == true)

                return false;

            visited[str.charAt(k) - 'a'] = true;

        }

        return true;

    }

}
```

```
public static int longestUniqueSubsttr(String str)
{
    int n = str.length();

    int res = 0;

    for(int i = 0; i < n; i++)
        for(int j = i; j < n; j++)
            if (areDistinct(str, i, j))
                res = Math.max(res, j - i + 1);

    return res;
}

public static void main(String[] args)
{
    String str = "geeksforgeeks";

    System.out.println("The input string is " + str);

    int len = longestUniqueSubsttr(str);

    System.out.println("The length of the longest " +
        "non-repeating character " +
        "substring is " + len);
}
```

Python3

```
def areDistinct(strr, i, j):  
  
    visited = [0] * (26)  
  
    for k in range(i, j + 1):  
        if (visited[ord(strr[k]) -  
            ord('a')] == True):  
            return False  
  
        visited[ord(strr[k]) -  
            ord('a')] = True  
  
    return True  
  
def longestUniqueSubsttr(strr):  
  
    n = len(strr)  
  
    res = 0  
  
    for i in range(n):  
        for j in range(i, n):  
            if (areDistinct(strr, i, j)):  
                res = max(res, j - i + 1)
```

```
        return res

if __name__ == '__main__':

    strr = "geeksforgeeks"

    print("The input is ", strr)


    len = longestUniqueSubsttr(strr)

    print("The length of the longest "

          "non-repeating character substring is ", len)
```

C#

```
using System;

class GFG{

public static bool areDistinct(string str,

                               int i, int j)

{

    bool[] visited = new bool[26];

    for(int k = i; k <= j; k++)

    {

        if (visited[str[k] - 'a'] == true)
```

```
        return false;

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

    return true;
}

public static int longestUniqueSubsttr(string str)
{
    int n = str.Length;

    int res = 0;

    for(int i = 0; i < n; i++)
        for(int j = i; j < n; j++)
            if (areDistinct(str, i, j))
                res = Math.Max(res, j - i + 1);

    return res;
}

public static void Main()
{
    string str = "geeksforgeeks";
```

```
Console.WriteLine("The input string is " + str);

int len = longestUniqueSubsttr(str);

Console.WriteLine("The length of the longest " +
                  "non-repeating character " +
                  "substring is " + len);
}
}
```

Output

The input string is geeksforgeeks

The length of the longest non-repeating character substring is 11

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.

C++

```
#include <bits/stdc++.h>

using namespace std;

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

    int res = 0;

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



```
vector<bool> visited(256);

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

    if (visited[str[j]] == true)

        break;

    else {

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

        visited[str[j]] = true;

    }

}

visited[str[i]] = false;

}

return res;

}

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;

}
```

Java

```
import java.io.*;

import java.util.*;

class GFG{

public static int longestUniqueSubsttr(String str)
{

    int n = str.length();

    int res = 0;

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

        boolean[] visited = new boolean[256];

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

```
{

    if (visited[str.charAt(j)] == true)

        break;

    else
    {
        res = Math.max(res, j - i + 1);
        visited[str.charAt(j)] = true;
    }
}

visited[str.charAt(i)] = false;
}

return res;
}

public static void main(String[] args)
{
    String str = "geeksforgeeks";

    System.out.println("The input string is " + str);

    int len = longestUniqueSubstr(str);

    System.out.println("The length of the longest " +
```

```
        "non-repeating character " +  
        "substring is " + len);  
    }  
}
```

Python3

```
def longestUniqueSubsttr(str):  
  
    n = len(str)  
  
    res = 0  
  
    for i in range(n):  
  
        visited = [0] * 256  
  
        for j in range(i, n):  
  
            if (visited[ord(str[j])] == True):  
  
                break
```

```
        else:

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

            visited[ord(str[j])] = True

        visited[ord(str[i])] = False

    return res

str = "geeksforgeeks"

print("The input is ", str)

len = longestUniqueSubsttr(str)

print("The length of the longest "

      "non-repeating character substring is ", len)
```

C#

```
using System;

class GFG{

    static int longestUniqueSubsttr(string str)

    {
```

```
int n = str.Length;

int res = 0;

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

    bool[] visited = new bool[256];

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

        if (visited[str[j]] == true)
            break;

        else
        {

            res = Math.Max(res, j - i + 1);
```

```
        visited[str[j]] = true;

    }

}

    visited[str[i]] = false;

}

return res;

}

static void Main()

{

    string str = "geeksforgeeks";

    Console.WriteLine("The input string is " + str);

    int len = longestUniqueSubsttr(str);

    Console.WriteLine("The length of the longest " +

        "non-repeating character " +

        "substring is " + len );

}

}
```

Output

The input string is geeksforgeeks

The length of the longest non-repeating character substring is

Method 3 (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.

- 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 present in the previous window. To check if the current character was present in the previous window or not, we store last index of every character in an array **lastIndex[]**. If `lastIndex[str[j]] + 1` is more than previous start, then we updated the start index **i**. Else we keep same **i**.

Below is the implementation of the above approach :

C++

```
#include <bits/stdc++.h>

using namespace std;

#define NO_OF_CHARS 256

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

    int res = 0;

    vector<int> lastIndex(NO_OF_CHARS, -1);

    int i = 0;

    for (int j = 0; j < n; j++) {
```



```
        i = max(i, lastIndex[str[j]] + 1);

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

        lastIndex[str[j]] = j;
    }

    return res;
}

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;
}
```

Java

```
import java.util.*;

public class GFG {
```

```
static final int NO_OF_CHARS = 256;

static int longestUniqueSubsttr(String str)
{
    int n = str.length();

    int res = 0;

    int [] lastIndex = new int[NO_OF_CHARS];
    Arrays.fill(lastIndex, -1);

    int i = 0;

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

        i = Math.max(i, lastIndex[str.charAt(j)] + 1);

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

        lastIndex[str.charAt(j)] = j;
    }

    return res;
}
```

```
public static void main(String[] args)
{
    String str = "geeksforgeeks";

    System.out.println("The input string is " + str);

    int len = longestUniqueSubsttr(str);

    System.out.println("The length of "

                        + "the longest non repeating character is " +
len);
}
}
```

Python3

```
def longestUniqueSubsttr(string):

    last_idx = {}

    max_len = 0

    start_idx = 0

    for i in range(0, len(string)):

        if string[i] in last_idx:
```

```
        start_idx = max(start_idx, last_idx[string[i]] + 1)

    max_len = max(max_len, i-start_idx + 1)

    last_idx[string[i]] = i

    return max_len

string = "geeksforgeeks"

print("The input string is " + string)

length = longestUniqueSubsttr(string)

print("The length of the longest non-repeating character" +
      " substring is " + str(length))
```

C#

```
using System;

public class GFG
{
    static int NO_OF_CHARS = 256;

    static int longestUniqueSubsttr(string str)
    {
        int n = str.Length;

        int res = 0;

        int [] lastIndex = new int[NO_OF_CHARS];

        Array.Fill(lastIndex, -1);
```

```
int i = 0;

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

    i = Math.Max(i, lastIndex[str[j]] + 1);

    res = Math.Max(res, j - i + 1);

    lastIndex[str[j]] = j;
}

return res;
}

static public void Main ()
{

    string str = "geeksforgeeks";

    Console.WriteLine("The input string is " + str);

    int len = longestUniqueSubsttr(str);

    Console.WriteLine("The length of "+

        "the longest non repeating character is " +

        len);
}
```

```
}  
  
}
```

Output

The input string is geeksforgeeks

The length of the longest non-repeating character substring is

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++

```
#include <bits/stdc++.h>  
  
using namespace std;  
  
int longestUniqueSubsttr(string s)  
{  
  
    map<char, int> seen ;  
  
    int maximum_length = 0;  
  
    int start = 0;  
  
    for(int end = 0; end < s.length(); end++)  
    {
```

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

            start = max(start, seen[s[end]] + 1);

        }

        seen[s[end]] = end;

        maximum_length = max(maximum_length,
                               end - start + 1);

    }

    return maximum_length;
}

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;

}
```

Java

```
import java.util.*;

class GFG {

    static int longestUniqueSubsttr(String s)
    {

        HashMap<Character, Integer> seen = new HashMap<>();

        int maximum_length = 0;

        int start = 0;

        for(int end = 0; end < s.length(); end++)
        {

            if(seen.containsKey(s.charAt(end)))
            {

                start = Math.max(start, seen.get(s.charAt(end)) + 1);

            }

            seen.put(s.charAt(end), end);

            maximum_length = Math.max(maximum_length, end-start + 1);

        }

        return maximum_length;

    }

}
```



```
public static void main(String []args)

{

    String s = "geeksforgeeks";

    System.out.println("The input String is " + s);

    int length = longestUniqueSubsttr(s);

    System.out.println("The length of the longest non-repeating
character substring is " + length);

}

}
```

Python

```
def longestUniqueSubsttr(string):

    seen = {}

    maximum_length = 0

    start = 0

    for end in range(len(string)):

        if string[end] in seen:
```

```
        start = max(start, seen[string[end]] + 1)

        seen[string[end]] = end

        maximum_length = max(maximum_length, end-start + 1)

    return maximum_length
```

```
string = "geeksforgeeks"

print("The input string is", string)

length = longestUniqueSubsttr(string)

print("The length of the longest non-repeating character substring is",
length)
```

C#

```
using System;

using System.Collections.Generic;

class GFG {

    static int longestUniqueSubsttr(string s)

    {

        Dictionary<char, int> seen = new Dictionary<char, int>();

        int maximum_length = 0;

        int start = 0;

        for(int end = 0; end < s.Length; end++)
```

```
{

    if(seen.ContainsKey(s[end]))
    {

        start = Math.Max(start, seen[s[end]] + 1);

    }

    seen[s[end]] = end;

    maximum_length = Math.Max(maximum_length, end-start + 1);

}

return maximum_length;

}

static void Main() {

    string s = "geeksforgeeks";

    Console.WriteLine("The input string is " + s);

    int length = longestUniqueSubsttr(s);

    Console.WriteLine("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

As an exercise, try the modified version of the above problem where you need to print the maximum length NRCS also (the above program only prints the length of it).

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the [DSA Self Paced Course](#) at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more, please refer [Complete Interview Preparation Course](#).

In case you wish to attend live classes with industry experts, please refer [Geeks Classes Live](#) and [Geeks Classes Live USA](#)