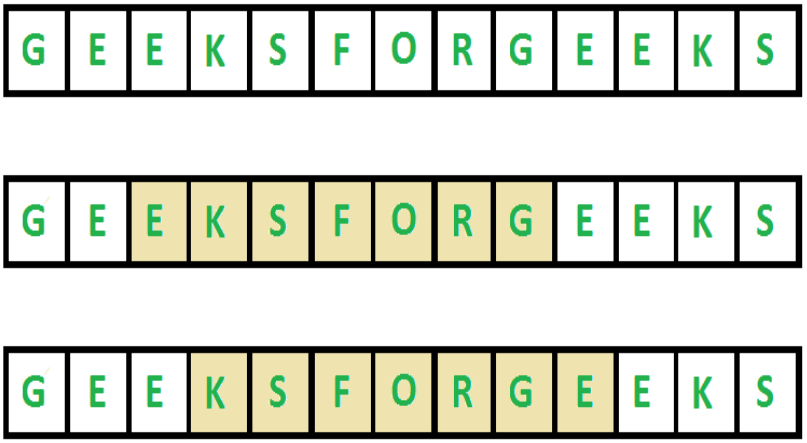
Length of the longest substring without repeating characters

Given a string, find the length of the longest substring without repeating characters. For example,the longest substrings without repeating characters for “ABDEFGABEF” 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.



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

Method 1 (Simple)
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 substirng 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).

Method 2 (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 (NRCS) seen so far. Let the maximum length be max_len. When we traverse the string, we also keep track of length of the current NRCS using cur_len variable. For every new character, we look for it in already processed part of the string (A temp array called visited[] is used for this purpose). If it is not present, then we increase the cur_len by 1. If present, then there are two cases:

a) The previous instance of character is not part of current NRCS (The NRCS which is under process). In this case, we need to simply increase cur_len by 1.
b) If the previous instance is part of the current NRCS, then our current NRCS changes. It becomes the substring staring from the next character of previous instance to currently scanned character. We also need to compare cur_len and max_len, before changing current NRCS (or changing cur_len).

Implementation

```
#include<stdlib.h>
#include<stdio.h>
#define NO_OF_CHARS 256

int min(int a, int b);

int longestUniqueSubsttr(char *str)
{
```

```
    int n = strlen(str);
    int cur_len = 1;  // To store the lenght of current substring
    int max_len = 1;  // To store the result
    int prev_index;  // To store the previous index
    int i;
    int *visited = (int *)malloc(sizeof(int)*NO_OF_CHARS);

    /* Initialize the visited array as -1, -1 is used to indicate that
       character has not been visited yet. */
    for (i = 0; i < NO_OF_CHARS;  i++)
        visited[i] = -1;

    /* Mark first character as visited by storing the index of first
       character in visited array. */
    visited[str[0]] = 0;

    /* Start from the second character. First character is already processed
       (cur_len and max_len are initialized as 1, and visited[str[0]] is set */
    for (i = 1; i < n; i++)
    {
        prev_index =  visited[str[i]];

        /* If the currentt character is not present in the already processed
           substring or it is not part of the current NRCS, then do cur_len++ */
        if (prev_index == -1 || i - cur_len > prev_index)
            cur_len++;

        /* If the current character is present in currently considered NRCS,
           then update NRCS to start from the next character of previous instance
        else
        {
            /* Also, when we are changing the NRCS, we should also check whether
              length of the previous NRCS was greater than max_len or not.*/
            if (cur_len > max_len)
                max_len = cur_len;

            cur_len = i - prev_index;
        }

        visited[str[i]] = i; // update the index of current character
    }

    // Compare the length of last NRCS with max_len and update max_len if needed
    if (cur_len > max_len)
        max_len = cur_len;


    free(visited); // free memory allocated for visited

    return max_len;
}

/* A utility function to get the minimum of two integers */
int min(int a, int b)
{
    return (a>b)?b:a;
}

/* Driver program to test above function */
int main()
{
    char str[] = "ABDEFGABEF";
    printf("The input string is %s \n", str);
    int len =  longestUniqueSubsttr(str);
    printf("The length of the longest non-repeating character substring is %d",

    getchar();
    return 0;
}
```

Output

```
The input string is ABDEFGABEF
The length of the longest non-repeating character substring is 6
```

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)
Algorithmic Paradigm: Dynamic Programming

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 length of it).