Length of Longest Substring without any Repeating Character

Problem Statement: Given a String, find the length of longest substring without any repeating character.

Examples:

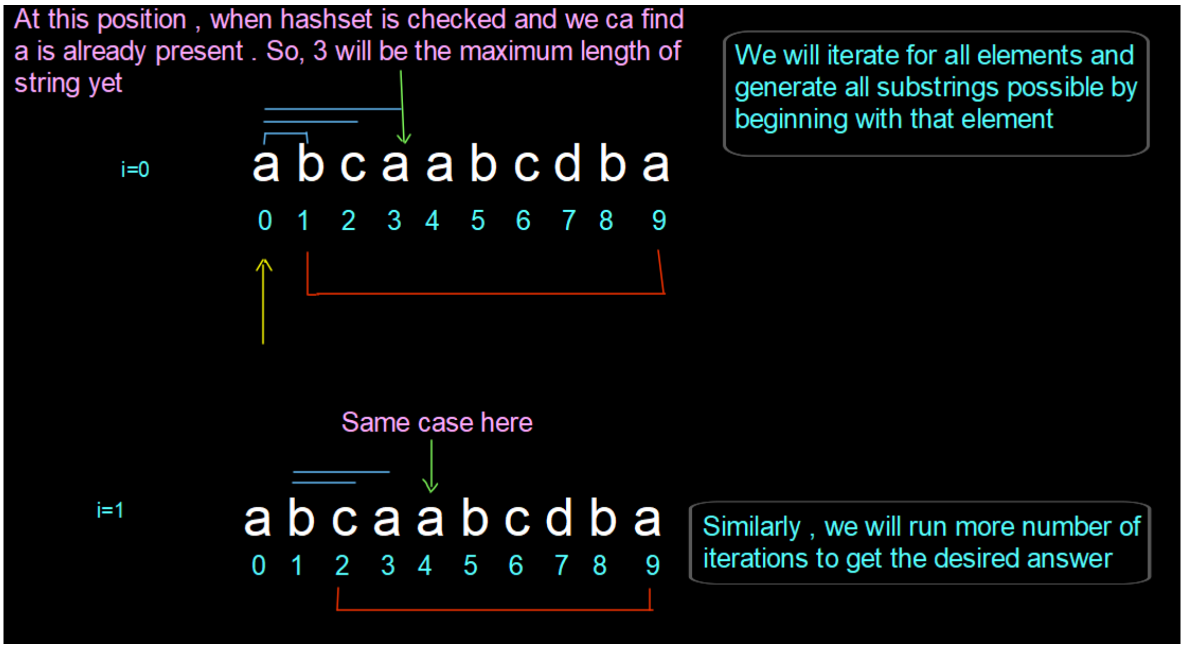
```
Example 1:
Input: s = "abcabcbb" Output: 3
Explanation: The answer is abc with length of 3.

Example 2:
Input: s = "bbbbb" Output: 1
Explanation: The answer is b with length of 1 units.
```

Solution

Solution 1: Brute force Approach

Approach: This approach consists of taking the two loops one for traversing the string and another loop - nested loop for finding different substrings and after that, we will check for all substrings one by one and check for each and every element if the element is not found then we will store that element in HashSet otherwise we will break from the loop and count it.


At this position , when hashset is checked and we ca find a is already present . So, 3 will be the maximum length of string yet
We will iterate for all elements and generate all substrings possible by beginning with that element
i=0
a b c a a b c d b a
0 1 2 3 4 5 6 7 8 9
Same case here
i=1
a b c a a b c d b a
0 1 2 3 4 5 6 7 8 9
Similarly , we will run more number of iterations to get the desired answer

Code:

```
import java.util.*;
public class Main {
    static int solve(String str) {

        if(str.length()==0)
             return 0;
        int maxans = Integer.MIN_VALUE;
        for (int i = 0; i < str.length(); i++)
            // outer loop for traversing the string
        {
            Set < Character > se = new HashSet < > ();
            for (int j = i; j < str.length(); j++)
            // nested loop for getting different
            string starting with str[i]
            {
                if (se.contains(str.charAt(j)))
                  // if element if found so mark it as ans
                and break from the loop
                {
                    maxans = Math.max(maxans, j - i);
                    break;
                }
                se.add(str.charAt(j));
            }
        }
        return maxans;
    }

    public static void main(String args[]) {
        String str = "takeUforward";
        System.out.println("The length of the longest substring without
repeating
        characters is " + solve(str));

    }
}
```

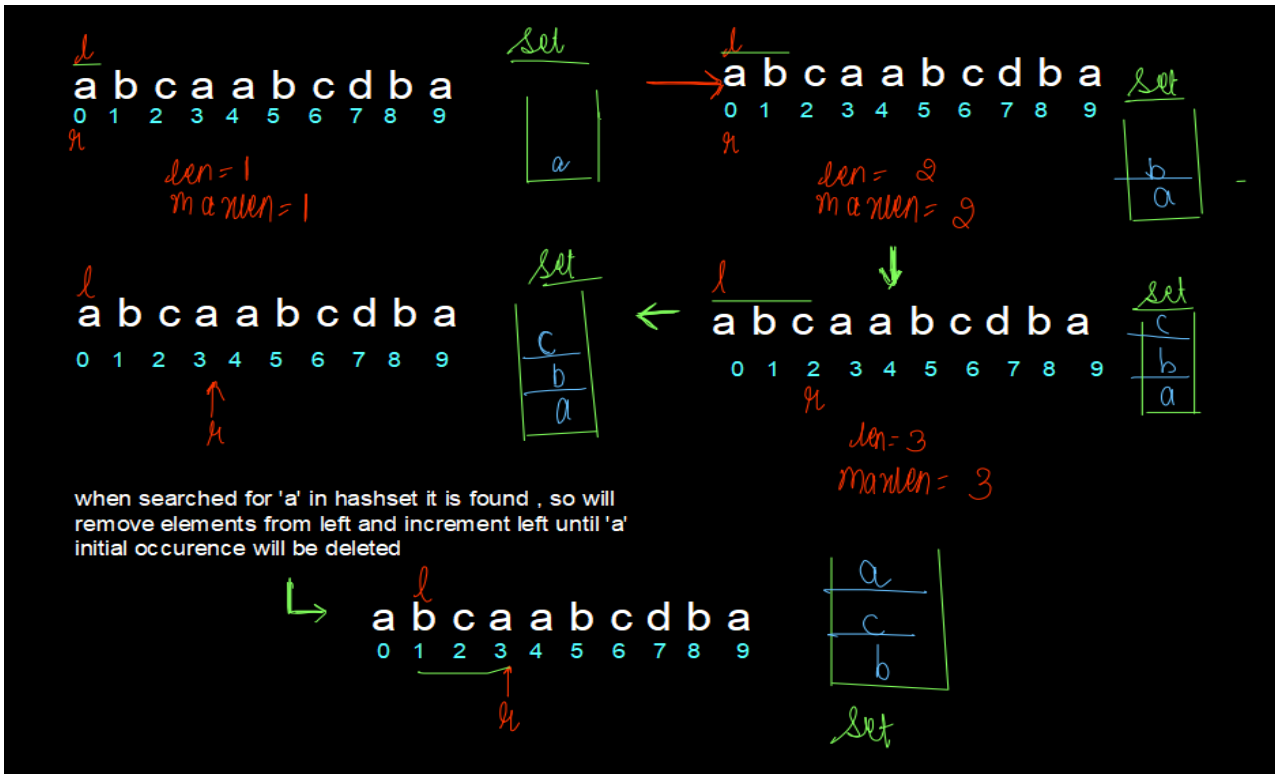
Output: The length of the longest substring without repeating characters is 9

Time Complexity: O(N2)

Space Complexity: O(N) where N is the size of HashSet taken for storing the elements

Solution 2: Optimised Approach 1

Approach: We will have two pointers left and right. Pointer ‘left’ is used for maintaining the starting point of the substring while ‘right’ will maintain the endpoint of the substring.’ right’ pointer will move forward and check for the duplicate occurrence of the current element if found then the ‘left’ pointer will be shifted ahead so as to delete the duplicate elements.


when searched for 'a' in hashset it is found , so will remove elements from left and increment left until 'a' initial occurence will be deleted

Code:

```
import java.util.*;
public class Main {
    static int solve(String str) {

        if(str.length()==0)
             return 0;
        int maxans = Integer.MIN_VALUE;
        Set < Character > set = new HashSet < > ();
        int l = 0;
        for (int r = 0; r < str.length(); r++)
            // outer loop for traversing the string
        {
            if (set.contains(str.charAt(r))) //if duplicate element is found
            {
                while (l < r && set.contains(str.charAt(r))) {
                    set.remove(str.charAt(l));
                    l++;
                }
            }
            set.add(str.charAt(r));
            maxans = Math.max(maxans, r - l + 1);
        }
        return maxans;
    }

    public static void main(String args[]) {
        String str = "takeUforward";
        System.out.println("The length of the longest substring without
repeating characters is " + solve(str));
      }
}
```

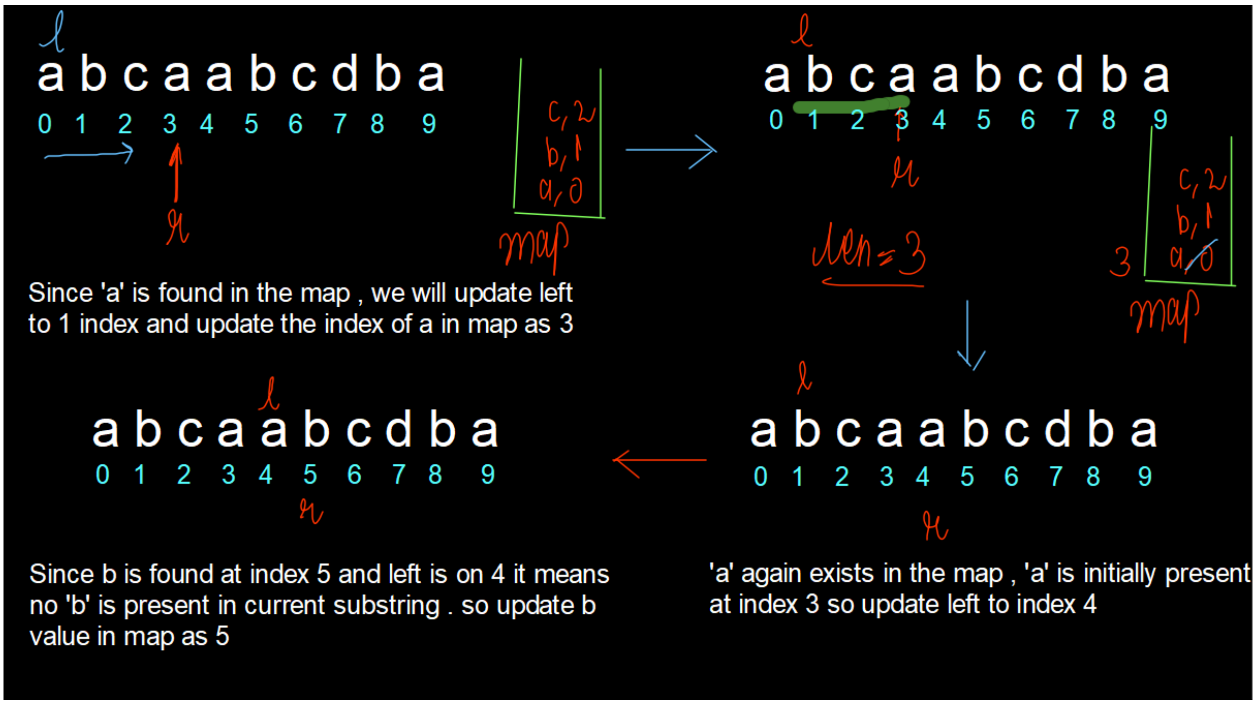
Output: The length of the longest substring without repeating characters is 9

Time Complexity: O(2*N) (sometimes left and right both have to travel complete array)

Space Complexity: O(N) where N is the size of HashSet taken for storing the elements

Solution 3: Optimised Approach 2

Approach: In this approach, we will make a map that will take care of counting the elements and maintaining the frequency of each and every element as a unity by taking the latest index of every element.


a b c a a b c d b a
0 1 2 3 4 5 6 7 8 9
l
r
c, 2
b, 1
a, 0
map
Since 'a' is found in the map , we will update left to 1 index and update the index of a in map as 3
len=3
3
Since b is found at index 5 and left is on 4 it means no 'b' is present in current substring . so update b value in map as 5
'a' again exists in the map , 'a' is initially present at index 3 so update left to index 4

Code:

```
import java.util.*;
public class Main {
    static int solve(String s) {
        HashMap < Character, Integer > mpp = new HashMap < Character, Integer >
();

        int left = 0, right = 0;
        int n = s.length();
        int len = 0;
        while (right < n) {
            if (mpp.containsKey(s.charAt(right))) left =
Math.max(mpp.get(s.charAt(right)) + 1, left);

            mpp.put(s.charAt(right), right);

            len = Math.max(len, right - left + 1);
            right++;
        }
        return len;
    }

    public static void main(String args[]) {
        String str = "takeUforward";
        System.out.println("The length of the longest substring without
repeating
        characters is " + solve(str));

    }
}
```

Output: The length of the longest substring without repeating characters is 9

Time Complexity: O(N)

Space Complexity: O(N) where N represents the size of HashSet where we are storing our elements