**Leetcode Problem 3:**

**Longest Substring Without Repeating Characters**

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

**Example 1:**

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

**Example 2:**

Input: s = "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

Solution code along with the comments:

class Solution {

public int lengthOfLongestSubstring(String s) {

// This Problem can be solved using sliding windows technique

// use set to store all the characters till the final index

// use telusko channel to learn about set

// set is basically interface

Set <Character> uniqueChar = new HashSet<>();

// variable to store the actual length of longest substring without repeating characters

int max= 0;

// variable to hold the current max

int max\_so\_far = -1;

// let's convert the string into character array so that the we can work at index level

// we can use java inbuilt function str.toCharArray() to convert string in to character array

char[] c = s.toCharArray();

//Create variable initial and final to track the ends of sliding windows

int intial=0;

int fin=0;

//now time to traverse the character array c so that you can track the maximum lengh substring

//use while loop because we are not increasing the fin at each step

while(fin<s.length())

{

//search if the set contains the character at the final index of c. If if contains shrink the

// window by incrementing intial index and then go for finding max\_so\_far

//very clever way to add method of the set in the if condition because this method will only

//only run when the char at index become unique

if(uniqueChar.add(c[fin]))

{

//update the max\_so\_far because some addition and substraction has taken place

max\_so\_far=uniqueChar.size();

if(max\_so\_far>=max)

{

max=max\_so\_far;

}

fin++;

}

else

{

// we are in the else condition meaning the add operation has been rejected

//carry on the remove operation till then

if(intial<s.length())

uniqueChar.remove(c[intial]);

//incremented the intial in this case

intial++;

max\_so\_far = uniqueChar.size();

if(max\_so\_far>=max)

{

max=max\_so\_far;

}

}

}

return max;

}

}

