Longest Substring Without Repeating Characters

Difficulty	Medium
Category	Sliding Window
Question	https://leetcode.com/problems/longest-substring-without-repeating-characters/
Solution	https://youtu.be/wiGpQwVHdE0
Status	Done

Question

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

Example

Example 1:

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

Example 2:

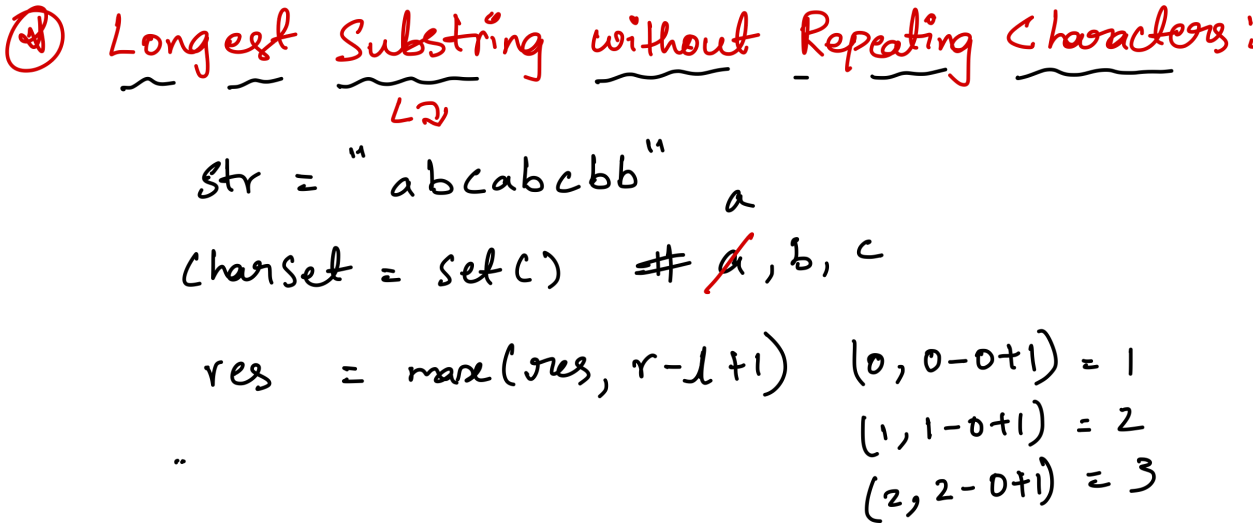
```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

Idea

💡 Sliding window: implement right pointer when no repeat. Use set to keep track of characters. When repeat happens, check from the left pointer. Implement left pointer while removing the character from set to check which one is the repeated one. When the repeat character found from left pointer, implement right pointer again. Use right-left to find the length;



Solution

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        charSet = set()  # Initialize an empty set to keep track of unique characters
        l = 0  # Initialize the left pointer to 0
        res = 0  # Initialize the result (length of the longest substring) to 0

        for r in range(len(s)):  # Iterate through the characters in the input string 's'
            while s[r] in charSet:
                # If the current character 's[r]' is already in the set, remove the character at 'l' and move the left pointer 'l' to the r
                charSet.remove(s[l])
                l += 1
            charSet.add(s[r])  # Add the current character 's[r]' to the set
            res = max(res, r - l + 1)  # Update the result by taking the maximum of the current result and the length of the current substr
        return res  # Return the result
```

Explanation

Time and Space Complexity

💡 The time complexity of this code is O(n), where 'n' is the length of the input string 's', because we iterate through the string once.

💡 The space complexity is O(min(n, m)), where 'm' is the size of the character set (26 for lowercase English letters), because we store unique characters in the `charSet` set.