

No-repeat Substring (hard)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time Complexity
 - Space Complexity

Problem Statement

Given a string, find the **length of the longest substring** which has **no repeating characters**.

Example 1:

Input: String="aabccbb"
Output: 3
Explanation: The longest substring without any repeating characters is "abc".

Example 2:

Input: String="abbbb"
Output: 2
Explanation: The longest substring without any repeating characters is "ab".

Example 3:

Input: String="abccde"
Output: 3
Explanation: Longest substrings without any repeating characters are "abc" & "cde".

Try it yourself

Try solving this question here:

 Java

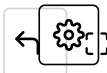
 Python3

 JS

 C++

```
1 def non_repeat_substring(str):  
2     # TODO: Write your code here  
3     return -1  
4
```





Solution

This problem follows the **Sliding Window** pattern and we can use a similar dynamic sliding window strategy as discussed in Longest Substring with K Distinct Characters

(<https://www.educative.io/collection/page/5668639101419520/5671464854355968/5698217712812032/>). We can use a **HashMap** to remember the last index of each character we have processed. Whenever we get a repeating character we will shrink our sliding window to ensure that we always have distinct characters in the sliding window.

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

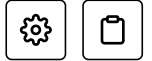
```
1 def non_repeat_substring(str1):
2     window_start = 0
3     max_length = 0
4     char_index_map = {}
5
6     # try to extend the range [windowStart, windowEnd]
7     for window_end in range(len(str1)):
8         right_char = str1[window_end]
9         # if the map already contains the 'right_char', shrink the window from the beginning
10        # we have only one occurrence of 'right_char'
11        if right_char in char_index_map:
12            # this is tricky; in the current window, we will not have any 'right_char' after its
13            # and if 'window_start' is already ahead of the last index of 'right_char', we'll keep
14            window_start = max(window_start, char_index_map[right_char] + 1)
15        # insert the 'right_char' into the map
16        char_index_map[right_char] = window_end
17        # remember the maximum length so far
18        max_length = max(max_length, window_end - window_start + 1)
19    return max_length
20
21
22 def main():
23     print("Length of the longest substring: " + str(non_repeat_substring("aabccbb")))
24     print("Length of the longest substring: " + str(non_repeat_substring("abbbb")))
25     print("Length of the longest substring: " + str(non_repeat_substring("abccde")))
26
27
28 main()
```



Time Complexity

The time complexity of the above algorithm will be $O(N)$ where 'N' is the number of characters in the input string.

Space Complexity



The space complexity of the algorithm will be $O(K)$ where K is the number of distinct characters in the input string. This also means $K \leq N$, because in the worst case, the whole string might not have any repeating character so the entire string will be added to the **HashMap**. Having said that, since we can expect a fixed set of characters in the input string (e.g., 26 for English letters), we can say that the algorithm runs in fixed space $O(1)$; in this case, we can use a fixed-size array instead of the **HashMap**.

[< Back](#)[Next >](#)

Fruits into Baskets (medium)

Longest Substring with Same Letters ...

 **Mark as Completed**



Report an
Issue



Ask a Question

(https://discuss.educative.io/tag/no-repeat-substring-hard__pattern-sliding-window__grokking-the-coding-interview-patterns-for-coding-questions)