No-repeat Substring (hard)

We'll cover the following

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 character
s is "abc".
```

Example 2:

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

Example 3:

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

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```
import java.util.*;

class NoRepeatSubstring {
        public static int findLength(String str) {
                char pre = 0;
                int end = 0;
                int max = 0;
                List<Character> arr = new ArrayList<>();
                for (end = 0; end < str.length(); end++) {
                        arr.add(str.charAt(end));
                        if (arr.size() - 1 >= 0 && arr.get(arr.size() -1).equals(pre
                                max = Math.max(max, arr.size() - 1);
                                arr.clear();
                                arr.add(str.charAt(end));
                        }
                        pre = str.charAt(end);
                }
                return max;
        }
}
```

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/5668639101419520/567146485435596 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

```
import java.util.*;

class NoRepeatSubstring {
  public static int findLength(String str) {
    int windowStart = 0, maxLength = 0;
    Map<Character, Integer> charIndexMap = new HashMap<>();
    // try to extend the range [windowStart, windowEnd]
    for (int windowEnd = 0; windowEnd < str.length(); windowEnd++) {
      char rightChar = str.charAt(windowEnd);
      // if the map already contains the 'rightChar', shrink the window from the beg
      // we have only one occurrence of 'rightChar'
      if (charIndexMap.containsKey(rightChar)) {
        // this is tricky; in the current window, we will not have any 'rightChar' af
        // and if 'windowStart' is already ahead of the last index of 'rightChar', we
        windowStart = Math.max(windowStart, charIndexMap.get(rightChar) + 1);
      }
      charIndexMap.put(str.charAt(windowEnd), windowEnd); // insert the 'rightChar' i
      maxLength = Math.max(maxLength, windowEnd - windowStart + 1); // remember the m
    }

    return maxLength;
  }

  public static void main(String[] args) {
    System.out.println("Length of the longest substring: " + NoRepeatSubstring.findLe
    System.out.println("Length of the longest substring: " + NoRepeatSubstring.findLe
    System.out.println("Length of the longest substring: " + NoRepeatSubstring.findLe
  }
}
```

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 $K <= N$, 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 (/courses/grokking-the-coding-interview/Bn2KLlOR0lQ)**

Fruits into Baskets (medium)

☑ **Completed**

Next (/courses/grokking-the-coding-interview/R8DVgjq78y

Longest Substring with Same Letters

Stuck? Get help on **DISCUSS** (https://discuss.educative.io/c/grokking-the-coding-interview-patterns-for-coding-questions-design-gurus/pattern-sliding-window-no-repeat-substring-hard)

Send feedback

♡ 9 Recommendations