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 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++

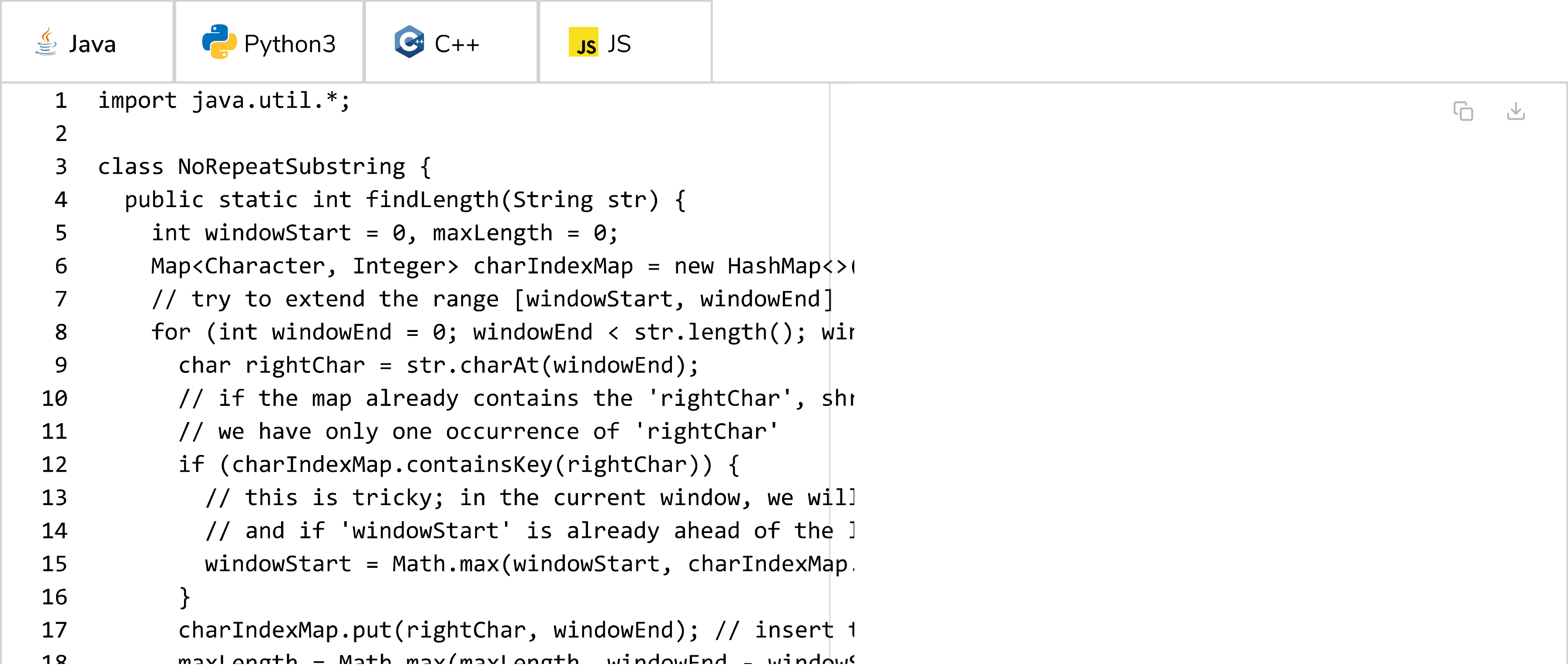
```
import java.util.*;

class NoRepeatSubstring {
  public static int findLength(String str) {
    // TODO: Write your code here
    return -1;
  }
}
```

Solution

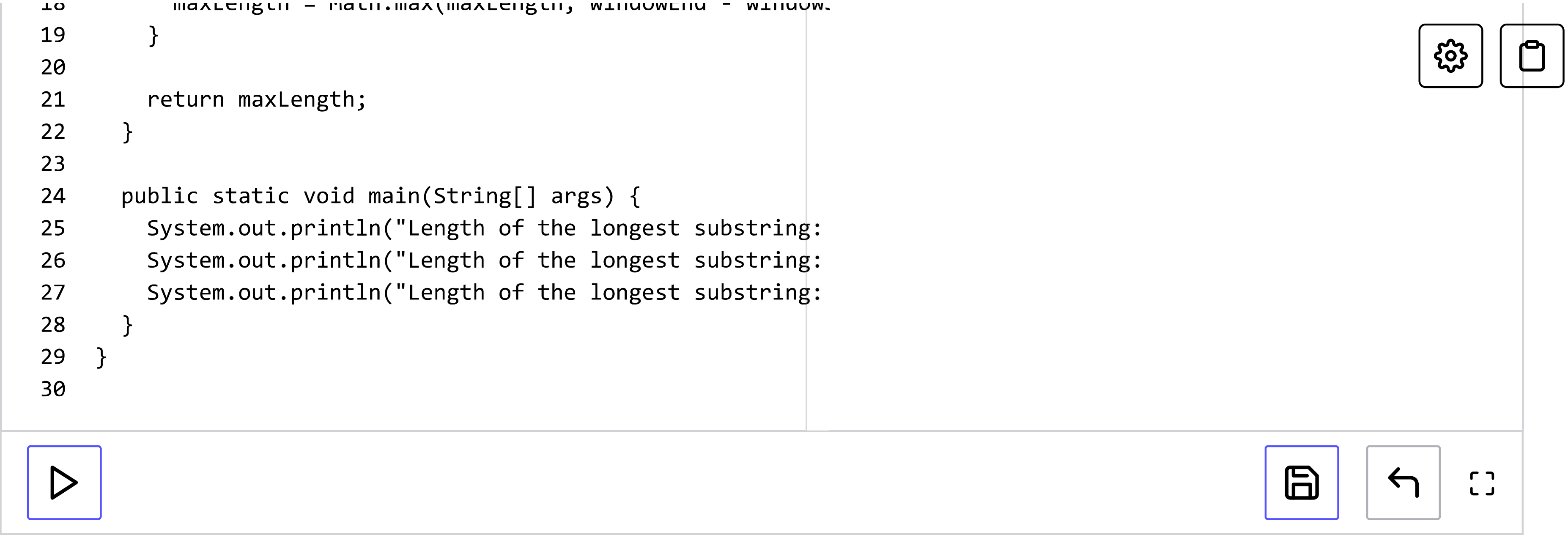
This problem follows the **Sliding Window** pattern, and we can use a similar dynamic sliding window strategy as discussed in Longest Substring with K Distinct Characters (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5698217712812032/). We can use a **HashMap** to remember 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:



```
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(); win
      char rightChar = str.charAt(windowEnd);
      // if the map already contains the 'rightChar', shr
      // we have only one occurrence of 'rightChar'
      if (charIndexMap.containsKey(rightChar)) {
        // this is tricky; in the current window, we will
        // and if 'windowStart' is already ahead of the l
        windowStart = Math.max(windowStart, charIndexMap.
      }
      charIndexMap.put(rightChar, windowEnd); // insert i
      maxLength = Math.max(maxLength, windowEnd - windowS
```



```
      maxLength = Math.max(maxLength, windowEnd - window
    }

    return maxLength;
  }

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

```

Time Complexity

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

Space Complexity

The algorithm's space complexity 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

Next →

Fruits into Baskets (medium)

Longest Substring with Same Letters ...

☑ Mark as Completed

3% completed, meet the criteria and claim your course certificate!

Claim Certificate

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)