Longest Substring with Same Letters after Replacement (hard)

We'll cover the following

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

Problem Statement

Given a string with lowercase letters only, if you are allowed to **replace no more than 'k' letters** with any letter, find the **length of the longest substring having the same letters** after replacement.

Example 1:

```
Input: String="aabccbb", k=2
Output: 5
Explanation: Replace the two 'c' with 'b' to have a longest repeating substring "b
bbbb".
```

Example 2:

```
Input: String="abbcb", k=1
Output: 4
Explanation: Replace the 'c' with 'b' to have a longest repeating substring "bbbb"
.
```

Example 3:

```
Input: String="abccde", k=1
Output: 3
Explanation: Replace the 'b' or 'd' with 'c' to have the longest repeating substri
ng "ccc".
```

Try it yourself

Try solving this question here:

educative

Java Python3 JS JS C++

```
def length_of_longest_substring(str, k):
  # 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 No-repeat Substring (https://www.educative.io/collection/page/5668639101419520/5671464854355968/5485010335301632/). We can use a HashMap to count the frequency of each letter.

We'll iterate through the string to add one letter at a time in the window. We'll also keep track of the count of the maximum repeating letter in any window (let's call it `maxRepeatLetterCount`). So at any time, we know that we can have a window which has one letter repeating `maxRepeatLetterCount` times, this means we should try to replace the remaining letters. If we have more than 'k' remaining letters, we should shrink the window as we are not allowed to replace more than 'k' letters.

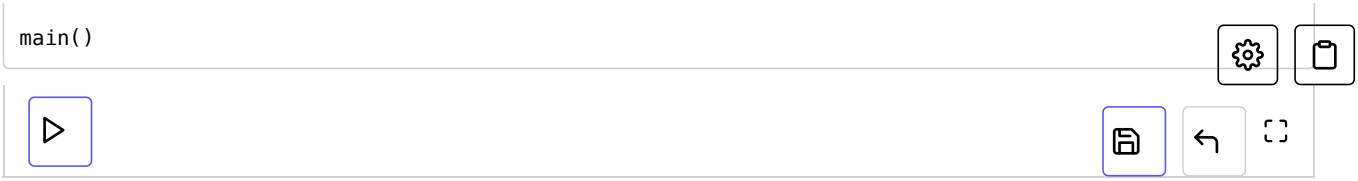
Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
def length_of_longest_substring(str1, k):
  window_start, max_length, max_repeat_letter_count = 0, 0, 0
  frequency_map = {}

  # Try to extend the range [window_start, window_end]
  for window_end in range(len(str1)):
    right_char = str1[window_end]
    if right_char not in frequency_map:
      frequency_map[right_char] = 0
    frequency_map[right_char] += 1
    max_repeat_letter_count = max(
      max_repeat_letter_count, frequency_map[right_char])

    # Current window size is from window_start to window_end, overall we have a letter which is
    # repeating 'max_repeat_letter_count' times, this means we can have a window which has one letter
    # repeating 'max_repeat_letter_count' times and the remaining letters we should replace.
    # if the remaining letters are more than 'k', it is the time to shrink the window as we
    # are not allowed to replace more than 'k' letters
    if (window_end - window_start + 1 - max_repeat_letter_count) > k:
      left_char = str1[window_start]
      frequency_map[left_char] -= 1
      window_start += 1

    max_length = max(max_length, window_end - window_start + 1)
  return max_length


def main():
  print(length_of_longest_substring("aabccbb", 2))
  print(length_of_longest_substring("abbcb", 1))
  print(length_of_longest_substring("abccde", 1))
```

```
main()
```



Time Complexity

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

Space Complexity

As we are expecting only the lower case letters in the input string, we can conclude that the space complexity will be $O(26)$, to store each letter's frequency in the **HashMap**, which is asymptotically equal to $O(1)$.

Back

No-repeat Substring (hard)

Next

Longest Subarray with Ones after Rep...

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/longest-substring-with-same-letters-after-replacement-hard__pattern-sliding-window__grokking-the-coding-interview-patterns-for-coding-questions)

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.