

## Grokking the Coding Interview: Patterns for Coding Questions

3% completed

### Introduction

- Who should take this course?
- Course Overview

### Pattern: Sliding Window

- Introduction
- Maximum Sum Subarray of Size K (easy)
- Smallest Subarray with a given sum (easy)
- Longest Substring with K Distinct Characters (medium)
- Fruits into Baskets (medium)
- No-repeat Substring (hard)
- Longest Substring with Same Letters after Replacement (hard)**
- Longest Subarray with Ones after Replacement (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3
- Problem Challenge 4
- Solution Review: Problem Challenge 4

### Pattern: Two Pointers

### Pattern: Fast & Slow pointers

### Pattern: Merge Intervals

### Pattern: Cyclic Sort

### Pattern: In-place Reversal of a LinkedList

### Pattern: Tree Breadth First Search

### Pattern: Tree Depth First Search

### Pattern: Two Heaps

### Pattern: Subsets

### Pattern: Modified

## 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 "bbbbb".
```

**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 substring "ccc".
```

### Try it yourself

Try solving this question here:

Java

Python3

JS

C++

```
1 const length_of_longest_substring = function(str, k) {
2   // TODO: Write your code here
3   return -1;
4 };
5
```

TEST

SAVE

RESET

### Solution

This problem follows the **Sliding Window** pattern and we can use a similar dynamic sliding window strategy as discussed in [No-repeat Substring](#). 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

```
1 function length_of_longest_substring(str, k) {
2   let windowStart = 0,
3       maxLength = 0,
4       maxRepeatLetterCount = 0,
5       frequencyMap = {};
6
7   // Try to extend the range [windowStart, windowEnd]
8   for (let windowEnd = 0; windowEnd < str.length; windowEnd++) {
9     const rightChar = str[windowEnd];
10    if (!(rightChar in frequencyMap)) {
11      frequencyMap[rightChar] = 0;
12    }
13    frequencyMap[rightChar] += 1;
14    maxRepeatLetterCount = Math.max(maxRepeatLetterCount, frequencyMap[rightChar]);
15
16    // Current window size is from windowStart to windowEnd, overall we have a letter which is
17    // repeated maxRepeatLetterCount times, so we need to delete the rest of the letters.
```

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

Binary Search

Pattern: Bitwise XOR

Pattern: Top 'K' Elements

Pattern: K-way merge

Pattern : 0/1 Knapsack (Dynamic Programming)

Pattern: Topological Sort (Graph)

Miscellaneous

Conclusions

Where to Go from Here

Mark Course as Completed

```
17 // repeating 'maxRepeatLetterCount' times, this means we can have a window which has one letter
18 // repeating 'maxRepeatLetterCount' times and the remaining letters we should replace.
19 // if the remaining letters are more than 'k', it is the time to shrink the window as we
20 // are not allowed to replace more than 'k' letters
21 if ((windowEnd - windowStart + 1 - maxRepeatLetterCount) > k) {
22     leftChar = str[windowStart];
23     frequencyMap[leftChar] -= 1;
24     windowStart += 1;
25 }
26
27 maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
28 }
```

RUN

SAVE

RESET

**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

MARK AS COMPLETED

Next →

No-repeat Substring (hard)

Longest Subarray with Ones after Rep...

Report an Issue