

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 Binary Search

Longest Subarray with Ones after Replacement (hard)

We'll cover the following

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

Problem Statement

Given an array containing 0s and 1s, if you are allowed to **replace no more than 'k' 0s with 1s**, find the length of the **longest contiguous subarray having all 1s**.

Example 1:

```
Input: Array=[0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1], k=2
Output: 6
Explanation: Replace the '0' at index 5 and 8 to have the longest contiguous subarray of 1s having length 6.
```

Example 2:

```
Input: Array=[0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1], k=3
Output: 9
Explanation: Replace the '0' at index 6, 9, and 10 to have the longest contiguous subarray of 1s having length 9.
```

Try it yourself

Try solving this question here:

Java

Python3

JS

C++

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

TEST

SAVE

RESET

Solution

This problem follows the **Sliding Window** pattern and is quite similar to **Longest Substring with same Letters after Replacement**. The only difference is that, in the problem, we only have two characters (1s and 0s) in the input arrays.

Following a similar approach, we'll iterate through the array to add one number at a time in the window. We'll also keep track of the maximum number of repeating 1s in the current window (let's call it **maxOnesCount**). So at any time, we know that we can have a window which has 1s repeating **maxOnesCount** time, so we should try to replace the remaining 0s. If we have more than 'k' remaining 0s, we should shrink the window as we are not allowed to replace more than 'k' 0s.

Code

Here is how our algorithm will look like:

Java

Python3

C++

JS

```
1 function length_of_longest_substring(arr, k) {
2   let windowStart = 0,
3       maxLength = 0,
4       maxOnesCount = 0;
5
6   // Try to extend the range [windowStart, windowEnd]
7   for (windowEnd = 0; windowEnd < arr.length; windowEnd++) {
8     if (arr[windowEnd] === 1) {
9       maxOnesCount += 1;
10    }
11
12    // Current window size is from windowStart to windowEnd, overall we have a maximum of 1s
13    // repeating 'maxOnesCount' times, this means we can have a window with 'maxOnesCount' 1s
14    // and the remaining are 0s which should replace with 1s.
15    // now, if the remaining 1s are more than 'k', it is the time to shrink the window as we
16    // are not allowed to replace more than 'k' 0s
17    if ((windowEnd - windowStart + 1 - maxOnesCount) > k) {
18      if (arr[windowStart] === 1) {
19        maxOnesCount -= 1;
20      }
21      windowStart++;
22    }
23  }
24  return maxLength;
25}
```

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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

```
21     windowStart += 1;
22 }
23
24     maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
25 }
26     return maxLength;
27 }
28
```

RUN

SAVE

RESET

Time Complexity

The time complexity of the above algorithm will be $O(N)$ where 'N' is the count of numbers in the input array.

Space Complexity

The algorithm runs in constant space $O(1)$.

← Back

MARK AS COMPLETED

Next →

Longest Substring with Same Letters ...

Problem Challenge 1

Report an Issue