

Grokking the Coding Interview: Patterns for Coding Questions

1% 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

- Introduction
- Pair with Target Sum (easy)
- Remove Duplicates (easy)
- Squaring a Sorted Array (easy)
- Triplet Sum to Zero (medium)
- Triplet Sum Close to Target (medium)
- Triplets with Smaller Sum (medium)
- Subarrays with Product Less than a Target (medium)
- Dutch National Flag Problem (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Fast & Slow pointers

- Introduction
- LinkedList Cycle (easy)
- Start of LinkedList Cycle (medium)

Longest Substring with K Distinct Characters (medium)

We'll cover the following

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

Problem Statement

Given a string, find the length of the **longest substring** in it with **no more than K distinct characters**.

Example 1:

```
Input: String="araaci", K=2
Output: 4
Explanation: The longest substring with no more than '2' distinct characters is "araa".
```

Example 2:

```
Input: String="araaci", K=1
Output: 2
Explanation: The longest substring with no more than '1' distinct characters is "aa".
```

Example 3:

```
Input: String="cbbebi", K=3
Output: 5
Explanation: The longest substrings with no more than '3' distinct characters are "cbbeb" & "bbebi".
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

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

TESTSAVERESET↺

Solution

This problem follows the **Sliding Window** pattern and we can use a similar dynamic sliding window strategy as discussed in [Smallest Subarray with a given sum](#). We can use a **HashMap** to remember the frequency of each character we have processed. Here is how we will solve this problem:

- First, we will insert characters from the beginning of the string until we have 'K' distinct characters in the **HashMap**.
- These characters will constitute our sliding window. We are asked to find the longest such window having no more than 'K' distinct characters. We will remember the length of this window as the longest window so far.
- After this, we will keep adding one character in the sliding window (i.e. slide the window ahead), in a stepwise fashion.
- In each step, we will try to shrink the window from the beginning if the count of distinct characters in the **HashMap** is larger than 'K'. We will shrink the window until we have no more than 'K' distinct characters in the **HashMap**. This is needed as we intend to find the longest window.
- While shrinking, we'll decrement the frequency of the character going out of the window and remove it from the **HashMap** if its frequency becomes zero.
- At the end of each step, we'll check if the current window length is the longest so far, and if so, remember its length.

Here is the visual representation of this algorithm for the Example-1:



- Happy Number (medium)
- Middle of the LinkedList (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Merge Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Cyclic Sort

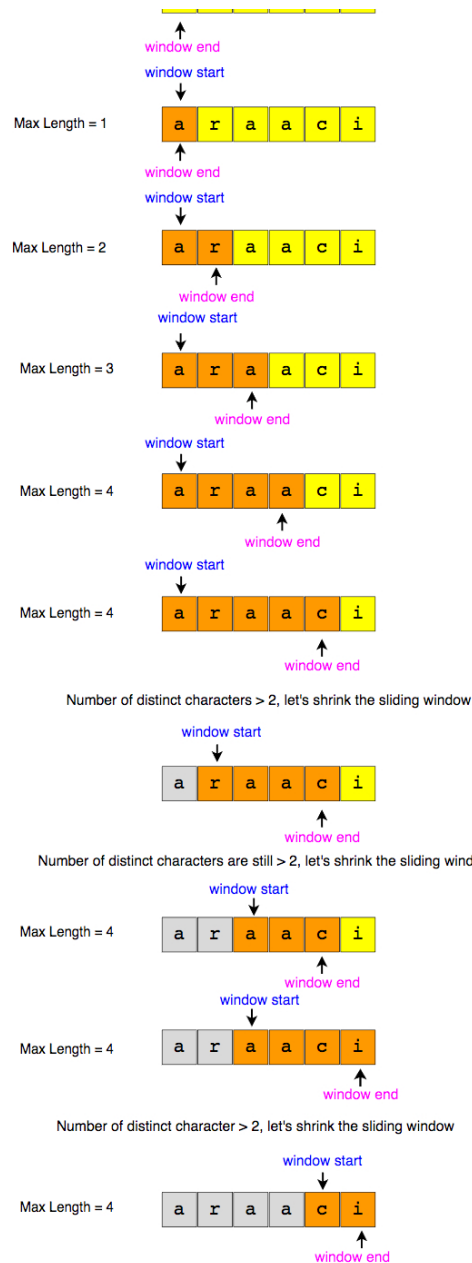
- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)



Code

Here is how our algorithm will look:

```

1 function longest_substring_with_k_distinct(str, k) {
2   let windowStart = 0,
3   maxLength = 0,
4   charFrequency = {};
5
6   // in the following loop we'll try to extend the range [window_start, window_end]
7   for (let windowEnd = 0; windowEnd < str.length; windowEnd++) {
8     const rightChar = str[windowEnd];
9     if (!(rightChar in charFrequency)) {
10      charFrequency[rightChar] = 0;
11    }
12    charFrequency[rightChar] += 1;
13    // shrink the sliding window, until we are left with 'k' distinct characters in the char_frequency
14    while (Object.keys(charFrequency).length > k) {
15      const leftChar = str[windowStart];
16      charFrequency[leftChar] -= 1;
17      if (charFrequency[leftChar] === 0) {
18        delete charFrequency[leftChar];
19      }
20      windowStart += 1; // shrink the window
21    }
22    // remember the maximum length so far
23    maxLength = Math.max(maxLength, windowEnd - windowStart + 1);
24  }
25  return maxLength;
26 }
27
28

```

Tracks

My Courses

Edpresso

Refer a Friend

Create

Level Order Successor (easy)

Connect Level Order Siblings (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

Introduction

Binary Tree Path Sum (easy)

All Paths for a Sum (medium)

Sum of Path Numbers (medium)

Path With Given Sequence (medium)

Count Paths for a Sum (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Time Complexity

The time complexity of the above algorithm will be $O(N)$ where 'N' is the number of characters in the input string. The outer **for** loop runs for all characters and the inner **while** loop processes each character only once, therefore the time complexity of the algorithm will be $O(N + N)$ which is asymptotically equivalent to $O(N)$.

Space Complexity

The space complexity of the algorithm is $O(K)$, as we will be storing a maximum of 'K+1' characters in the HashMap.

← Back

Smallest Subarray with a given sum (e...

✓ MARK AS COMPLETED

Next →

Fruits into Baskets (medium)

Report an Issue