

Grokking the Coding Interview: Patterns for Coding Questions

53% completed



Challenge 1

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)
- 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
- Solution Review: Problem Challenge 2

Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem

Solution Review: Problem Challenge 1

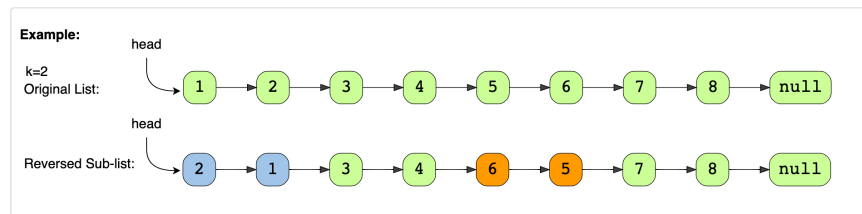
We'll cover the following

- Reverse alternating K-element Sub-list (medium)
- Solution
 - Code
 - Time complexity
 - Space complexity

Reverse alternating K-element Sub-list (medium)

Given the head of a LinkedList and a number 'k', **reverse every alternating 'k' sized sub-list** starting from the head.

If, in the end, you are left with a sub-list with less than 'k' elements, reverse it too.



Solution

The problem follows the **In-place Reversal of a LinkedList** pattern and is quite similar to [Reverse every K-element Sub-list](#). The only difference is that we have to skip 'k' alternating elements. We can follow a similar approach, and in each iteration after reversing 'k' elements, we will skip the next 'k' elements.

Code

Most of the code is the same as [Reverse every K-element Sub-list](#); only the highlighted lines have a majority of the changes:

JavaPython3C++JS

```
8 let temp = this;
9 while (temp !== null) {
10   process.stdout.write(`${temp.value} `);
11   temp = temp.next;
12 }
13 console.log();
14 }
15 }
16
17 function reverse_alternate_k_elements(head, k) {
18   if (k <= 1 || head === null) {
19     return head;
20   }
21
22   let current = head,
23       previous = null;
24   while (true) {
25     const last_node_of_previous_part = previous;
26     // after reversing the LinkedList 'current' will become the last node of the sub-list
27     const last_node_of_sub_list = current;
28     let next = null; // will be used to temporarily store the next node
29
30     // reverse 'k' nodes
31     let i = 0;
32     while (current !== null && i < k) {
33       next = current.next;
34       current.next = previous;
35       previous = current;
```

RUNSAVERESET

Output

Nodes of original LinkedList are: 1 2 3 4 5 6 7 8
Nodes of reversed LinkedList are: 2 1 3 4 6 5 7 8

2.339sClose

Time complexity

The time complexity of our algorithm will be $O(N)$ where 'N' is the total number of nodes in the LinkedList.

Space complexity

We only used constant space, therefore, the space complexity of our algorithm is $O(1)$.

friend

Challenge 1

Pattern: Subsets

○ Introduction

● Subsets (easy)

● Subsets With Duplicates (easy)

● Permutations (medium)

○ String Permutations by changing case (medium)

○ Balanced Parentheses (hard)

○ Unique Generalized Abbreviations (hard)

○ Problem Challenge 1

○ Solution Review: Problem Challenge 1

○ Problem Challenge 2

○ Solution Review: Problem

+

Create

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ

← Back

Next →

Problem Challenge 1

Problem Challenge 2

✓ Mark as Completed

🔔 Report an Issue 🤔 Ask a Question