

Grokking the Coding Interview: Patterns for Coding Questions

29% completed

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)

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

Reverse every K-element Sub-list (medium)

We'll cover the following

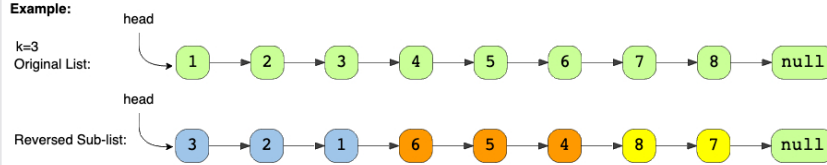
- Problem Statement
- Try it yourself
- Solution
 - Code
 - Time complexity
 - Space complexity

Problem Statement

Given the head of a LinkedList and a number 'k', **reverse every '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.

Example:



Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 class Node {
2   constructor(value, next=null){
3     this.value = value;
4     this.next = next;
5   }
6
7   get_list() {
8     result = "";
9     temp = this;
10    while (temp !== null) {
11      result += temp.value + " ";
12      temp = temp.next;
13    }
14    return result;
15  }
16 };
17
18
19
20 const reverse_every_k_elements = function(head, k) {
21   // TODO: Write your code here
22   return head;
23 }
24
25
26 head = new Node(1)
27 head.next = new Node(2)
28 head.next.next = new Node(3)
```

RUNSAVERESET

Solution

The problem follows the **In-place Reversal of a LinkedList** pattern and is quite similar to [Reverse a Sub-list](#). The only difference is that we have to reverse all the sub-lists. We can use the same approach, starting with the first sub-list (i.e. $p=1$, $q=k$) and keep reversing all the sublists of size 'k'.

Code

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

JavaPython3C++JS

```
1 class Node {
2   constructor(value, next = null) {
3     this.value = value;
4     this.next = next;
5   }
6
7   print_list() {
8     let temp = this;
9     while (temp !== null) {
```

Introduction

Find the Median of a Number Stream (medium)

Sliding Window Median (hard)

Maximize Capital (hard)

Problem Challenge 1

Solution Review: Problem 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 Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

Introduction

Order-agnostic Binary Search (easy)

Ceiling of a Number (medium)

Next Letter (medium)

Number Range (medium)

Search in a Sorted Infinite Array (medium)

Minimum Difference Element (medium)

Bitonic Array Maximum (easy)

Problem Challenge 1

Solution Review: Problem Challenge 1

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

10

process.stdout.write(`\${temp.value}`);

11

temp = temp.next;

12

}

13

console.log();

14

}

15

}

16

}

17

}

18

function reverse_every_k_elements(head, k) {

19

if (k <= 1 || head === null) {

20

return head;

21

}

22

}

23

let current = head,

24

previous = null;

25

while (true) {

26

const last_node_of_previous_part = previous;

27

// after reversing the LinkedList 'current' will become the last node of the sub-list

28

const last_node_of_sub_list = current;

RUN

SAVE

RESET

Close

Output

2.443s

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

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

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 $O(1)$.

Interviewing soon? We've partnered with **Hired** so that companies apply to you, instead of the other way around. [See how](#)

← Back

Reverse a Sub-list (medium)

MARK AS COMPLETED

Next →

Problem Challenge 1

Report an Issue