Back To Course Home

Grokking the Coding Interview: Patterns for Coding Questions

37% completed

Search Course

(medium)
Problem Challenge 1
Solution Review: Problem Challenge 1
Problem Challenge 2
Solution Review: Problem Challenge 2

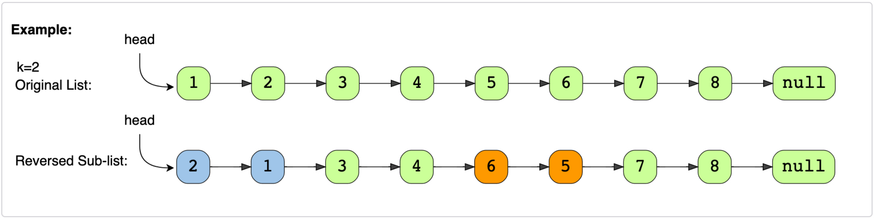
Solution Review: Problem Challenge 1

We'll cover the following

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:

Java | Python3 | C++ | JS

```java
import java.util.*;

class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class ReverseAlternatingKElements {

  public static ListNode reverse(ListNode head, int k) {
    if (k <= 1 || head == null)
      return head;

    ListNode current = head, previous = null;
    while (current != null) { // break if we've reached the end of the list
      ListNode lastNodeOfPreviousPart = previous;
      // after reversing the LinkedList 'current' will become the last node of the sub-list
      ListNode lastNodeOfSubList = current;
      ListNode next = null; // will be used to temporarily store the next node
      // reverse 'k' nodes
      for (int i = 0; current != null && i < k; i++) {
        next = current.next;
        current.next = previous;
        previous = current;
```

Run | Save | Reset

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 $O(1)$.