

Solution Review: Problem Challenge 1

We'll cover the following

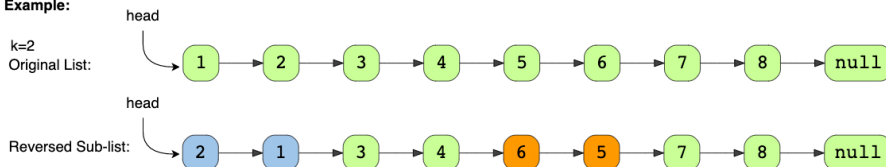
- Reverse alternating K-element Sub-list (medium)
- Solution
 - Code
 - Time complexity
 - Space complexity

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.

Example:



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:

```
1 import java.util.*;
2
3 class ListNode {
4     int value = 0;
5     ListNode next;
6
7     ListNode(int value) {
8         this.value = value;
9     }
10 }
11
12 class ReverseAlternatingKElements {
13
14     public static ListNode reverse(ListNode head, int k) {
15         if (k <= 1 || head == null)
16             return head;
17
18         ListNode current = head, previous = null;
19         while (current != null) { // break if we've reached the end of the list
20             ListNode lastNodeOfPreviousPart = previous;
21             // after reversing the LinkedList 'current' will become the last node of the sub-list
22             ListNode lastNodeOfSubList = current;
23             ListNode next = null; // will be used to temporarily store the next node
24             // reverse 'k' nodes
25             for (int i = 0; current != null && i < k; i++) {
26                 next = current.next;
27                 current.next = previous;
28                 previous = current;
```

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 $O(1)$.



[← Back](#)

Problem Challenge 1

[Next →](#)

Problem Challenge 2

 Mark as Completed

 Report an Issue  Ask a Question