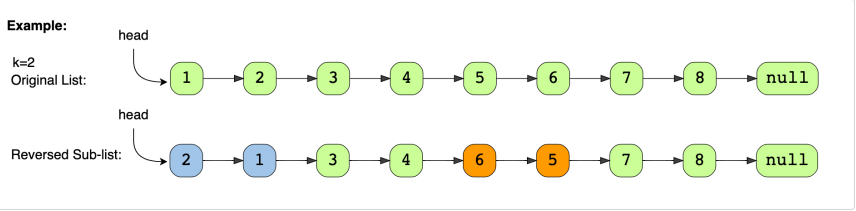
Solution Review: Problem Challenge 1

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

```
    while (temp !== null) {
      process.stdout.write(`${temp.value} `);
      temp = temp.next;
    }
    console.log();
  }
}

function reverse_alternate_k_elements(head, k) {
  if (k <= 1 || head === null) {
    return head;
  }

  let current = head,
    previous = null;
  while (true) {
    const last_node_of_previous_part = previous;
    // after reversing the LinkedList 'current' will become the last node of the sub-list
    const last_node_of_sub_list = current;
    let next = null; // will be used to temporarily store the next node

    // reverse 'k' nodes
    let i = 0;
    while (current !== null && i < k) {
      next = current.next;
      current.next = previous;
      previous = current;
```

RUN | SAVE | RESET

Close

Output 2.339s

```
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
```

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 $O(1)$.

+ Create

Back
Problem Challenge 1

Next
Problem Challenge 2

Mark as Completed