Reverse Singly Linked List in K-groups LeetCode

Approach 1: Reverse k nodes in a group using Iterative and recursive approach.

This approach aims to reverse nodes in groups of size k within a linked list. It combines both iterative and recursive techniques.

1. It starts by checking if the current group has at least k nodes remaining. If not, it means there are not enough nodes to reverse, and the function returns the original head of the list.
2. When there are enough nodes, the approach enters an iterative loop that reverses the current group of k nodes. It maintains three pointers: **prevNode**, **currNode**, and **nextNode**.
3. During each iteration, **currNode** points to the current node being processed. The **nextNode** pointer is used to store the next node in the original list to maintain progress during the reversal. The **prevNode** pointer is updated to point to **currNode**, effectively reversing the link direction.
4. After reversing k nodes, the head of the reversed group (**prevNode**) is connected to the recursively reversed next group (if any). Then, the head of the current group is updated to **prevNode**.
5. The process continues recursively for the remaining nodes in the list.
6. **Time Complexity: O(N), where N is the number of nodes in the linked list. Since each node is visited only once in a single iteration, and there are a total of N/k iterations (where k is the group size), the time complexity is linear.**
7. **Space Complexity: O(k), where k is the group size. The space complexity comes from the recursive call stack, which can hold up to k frames at any given time.**

Approach 2: Reverse k nodes in a group using a stack and recursive approach.

This approach also aims to reverse nodes in groups of size k within a linked list. It employs a stack to temporarily store nodes for reversal.

1. Similar to the first approach, it checks if there are at least k nodes remaining in the current group. If not, it returns the original head.
2. When there are enough nodes, the approach uses a stack to store the first k nodes of the group in reverse order.
3. After pushing k nodes onto the stack, it pops them off in reverse order, effectively reversing the order of the nodes.

4. The approach then connects the end of the reversed group to the recursively reversed next group (if any) and updates the head of the reversed group to the top node of the stack.

5. The process continues recursively for the remaining nodes in the list.

6. **Time Complexity: O(N), where N is the number of nodes in the linked list. Similar to the first approach, the time complexity is linear because each node is visited exactly once, and there are N/k iterations.**
7. **Space Complexity: O(k), where k is the group size. The stack is used to store up to k nodes at a time during recursion.**