Reverse Doubly Linked List in K groups CodeStudio

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

1. This approach aims to reverse k nodes in a group within a doubly linked list. It uses a combination of iterative and recursive techniques.
2. It first checks if there are at least k nodes remaining in the current group. If not, it returns the original head of the list.
3. It then enters an iterative loop that reverses the current group of k nodes. During each iteration:
    1. It maintains three pointers: **prevNode**, **currNode**, and **nextNode**.
    2. **currNode** points to the current node being processed.
    3. **nextNode** stores the next node in the original list.
    4. **prevNode** is updated to reverse the next and prev pointers.
4. After reversing k nodes, it connects the head of the reversed group to the next reversed group (recursively).
5. The head of the current group is updated to the head of the reversed group, and the process continues recursively for the remaining nodes.
6. **Time Complexity: O(N), where N is the number of nodes in the doubly linked list. Similar to the singly linked list case, the time complexity is linear since each node is visited exactly once, and there are N/k iterations (where k is the group size).**
7. **Space Complexity: O(k), where k is the group size. This 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.

1. Similar to the first approach, this approach aims to reverse k nodes in a group within a doubly linked list. It uses a stack to temporarily store nodes for reversal.
2. It checks if there are at least k nodes remaining in the current group. If not, it returns the original head.
3. It then initializes a pointer **currNode** to traverse through the list. It also uses a stack to store k nodes.
4. It pushes k nodes onto the stack and reverses the order of nodes within the stack. While popping nodes off the stack, it updates the next and prev pointers to reverse the links.

5. The head of the reversed group is set to the top node of the stack.
6. After reversing the nodes in the current group, it recursively connects the end of the reversed group to the next reversed group.
7. **Time Complexity: O(N), where N is the number of nodes in the doubly linked list. As with the first approach, the time complexity is linear due to the same reasons.**
8. **Space Complexity: O(k), where k is the group size. In this approach, the additional space is used for the stack to store k nodes during each iteration.**