MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

HOMEWORK –4 (Reverse Nodes in k-Group)

Aswin Sampath
saswin@uci.edu
(53844684)

Problem List

Description | Editorial | Solutions (6.8K) | Submissions

25. Reverse Nodes in k-Group

Hard 12.8K 628

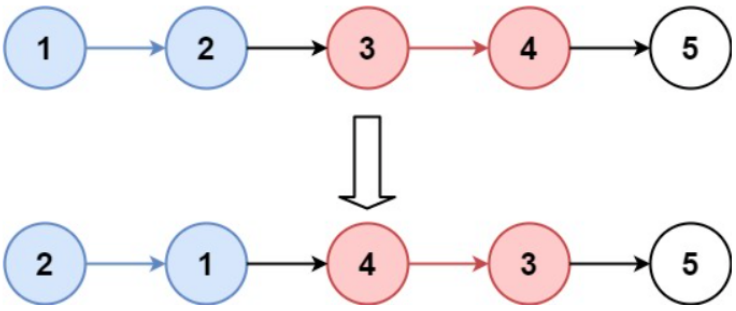
Companies

Given the `head` of a linked list, reverse the nodes of the list `k` at a time, and return *the modified list*.

`k` is a positive integer and is less than or equal to the length of the linked list. If the number of nodes is not a multiple of `k` then left-out nodes, in the end, should remain as it is.

You may not alter the values in the list's nodes, only nodes themselves may be changed.

Example 1:




```
Input: head = [1,2,3,4,5], k = 2
Output: [2,1,4,3,5]
```

Understanding the Problem:

The problem requires reversing the nodes of a linked list in groups of size k. The goal is to return the modified list after reversing. If the number of nodes is not a multiple of k, the remaining nodes at the end should remain as they are.

Identifying Edge Cases:

- When the linked list is empty (i.e., head is nullptr), the function should return nullptr since there are no nodes to reverse.
- When k is less than or equal to 1, the function should return the original list since reversing in groups of size 1 or less would have no effect.

Effective Test Cases:

- **Test Case 1: Linked List of Size Greater than k**
  - Input: Linked list with nodes 1, 2, 3, 4, 5
  - k is set to 2
  - Expected Output: The linked list is reversed in groups of 2, resulting in a modified list: 2 -> 1 -> 4 -> 3 -> 5.
- **Test Case 2: Linked List of Size Less than k**
  - Input: Linked list with nodes 1, 2
  - k is set to 3
  - Expected Output: The linked list has fewer nodes than k, so it remains unchanged: 1 -> 2.
- **Test Case 3: Linked List of Size That Is a Multiple of k**
  - Input: Linked list with nodes 1, 2, 3, 4, 5, 6
  - k is set to 2
  - Expected Output: The linked list is reversed in groups of 2, resulting in a modified list: 2 -> 1 -> 4 -> 3 -> 6 -> 5.
- **Test Case 4: Empty Linked List (Edge Case)**
  - Input: Empty linked list
  - k is set to 4
  - Expected Output: Since there are no nodes to reverse, the result is an empty list.
- **Test Case 5: k Equal to 1**
  - Input: Linked list with nodes 1, 2, 3, 4
  - k is set to 1
  - Expected Output: When k is 1, no nodes are reversed, so the list remains the same: 1 -> 2 -> 3 -> 4.

Algorithmic Solution:

1. Initialize a current pointer to head and a count variable to 0.
2. Count the number of nodes in the current group by moving current k times or until current becomes nullptr.
3. If there are at least k nodes in the group, reverse them by:
4. Recursively calling reverseKGroup on the next group starting from current.
5. Reversing the current group in-place using a while loop.
6. Return the new head of the list after all reversals.

Time and Space Complexity Analysis:

Time Complexity: The time complexity is O(N), where N is the total number of nodes in the linked list. This is because each node is visited once during the reversal.

Space Complexity: The space complexity is O (1), as the algorithm uses a constant amount of additional space for temporary pointers and variables during the reversal process. The recursion stack space also contributes to space usage, but it is proportional to the depth of recursion, which is at most N/k levels deep.

Code:

```cpp
class Solution {
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        ListNode* current = head;
        int count = 0;

        // Count the number of nodes in the current group
        while (current != nullptr && count < k) {
            current = current->next;
            count++;
        }

        // If there are at least k nodes in the group, reverse them
        if (count == k) {
            current = reverseKGroup(current, k); // Recursively reverse the next group
            while (count > 0) {
                ListNode* next = head->next;
                head->next = current;
                current = head;
                head = next;
                count--;
            }
            head = current;
        }

        return head;
    }
};
```

Output: