Reverse Linked List II Documentation

Problem Statement

Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

Example 1:

Input: head = [1, 2, 3, 4, 5], left = 2, right = 4

Output: [1, 4, 3, 2, 5]

Example 2:

Input: head = [5], left = 1, right = 1

Output: [5]

Constraints

- The number of nodes in the list is n.
- 1 <= n <= 500
- -500 <= Node.val <= 500
- 1 <= left <= right <= n

Follow Up

- Could you do it in one pass?

Detailed Steps

1. **Edge Case Handling:**

- If the head is None or if left is equal to right, there is no need to reverse anything. Simply return the head.

2. **Initialization:**

- Create a dummy node (dummy) to handle edge cases easily (e.g., reversing the first few nodes). Set dummy.next to head.
- Initialize prev to dummy.

3. **Move prev to the Node Before the Reversing Part:**

- Move prev to the node just before the part of the list that needs to be reversed. This is done by iterating left 1 times.

4. **Reversing the Sublist:**

- Set curr to prev.next, the first node in the part to be reversed.
- Use a loop to reverse the nodes from position left to right. During each iteration:
- Move the next_node after curr to the front of the sublist.
- Update the links accordingly to reverse the nodes.

5. **Return the New Head:**

- Return dummy.next as the new head of the reversed list.

Complexity Analysis

- **Time Complexity:** O(n), where n is the number of nodes in the linked list. This is because we traverse the list only once.

- **Space Complexity:** O(1), since we are using a constant amount of extra space.

Example Walkthrough

Consider the list [1, 2, 3, 4, 5] with left = 2 and right = 4.

1. **Initialization:**

- dummy -> [0, 1, 2, 3, 4, 5]
- prev = dummy

2. **Move prev:**

- *Move prev to the node just before position left:*
- prev -> [1, 2, 3, 4, 5]

3. **Reversing the Sublist:**

- curr -> [2, 3, 4, 5]
- *After first iteration:*
  - next_node -> 3
  - *Re-link nodes:*
    - ✓ prev -> [1, 3, 2, 4, 5]
- *After second iteration:*
  - next_node -> 4
  - *Re-link nodes:*
    - ✓ prev -> [1, 4, 3, 2, 5]

4. **Return the Result:**

- The new head is [1, 4, 3, 2, 5].