Click on the logo to find Problem Statement

Intuition

1. **Three Pointers:**
    - When reversing a linked list iteratively, you maintain three pointers: `before`, `temp`, and `after`.
    - `before` points to the previously reversed part of the list.
    - `temp` points to the current node that needs to be reversed.
    - `after` points to the next node that will be reversed.
2. **Reversing the Pointers:**
    - During each iteration of the loop, you update the `temp.next` pointer to point back to the `before` node. This effectively reverses the link of the current node.
    - Then, you move the `before`, `temp`, and `after` pointers one step forward in the list.
3. **Iterative Process:**
    - The process continues iteratively, reversing the pointers one node at a time.
    - Eventually, `temp` becomes `None`, indicating that the entire list has been reversed. At this point, `before` points to the new head of the reversed list.
4. **Returning the New Head:**
    - Finally, you return `before` as the new head of the reversed list.

Approach

1. **Initialize Pointers:**

- Initialize three pointers: `before` (initialized to `None`), `temp` (initialized to the head of the linked list), and `after` (initialized to `temp.next`).

2. **Iterative Reversal:**
   - Use a `while` loop to iterate through the list until `temp` becomes `None`.
   - Within the loop:
     - Update `after` to store the next node (`temp.next`) to prevent losing the reference.
     - Update `temp.next` to point back to the `before` node, reversing the link.
     - Move `before` and `temp` pointers one step forward. `before` becomes `temp`, and `temp` becomes `after`.

3. **Return the New Head:**
   - Once the loop exits (`temp` becomes `None`), `before` will be pointing to the new head of the reversed list.
   - Return `before` as the new head of the reversed linked list.

Complexity

- Time complexity: O(n)
- Space complexity: O(1)

Code

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        temp = head
        after = temp
        before = None
        while after is not None:
            after = temp.next
            temp.next = before
            before = temp
            temp =  after
        return before
```


GitHub