Documentation

To solve the problem of checking if a singly linked list is a palindrome, we need an efficient approach that works in (O(n)) time and (O(1)) space. The problem asks us to determine if the list reads the same forwards and backward. This requirement suggests a two-part approach: identifying the midpoint of the list and then checking symmetry by comparing the first half with the reversed second half. In the initial step, we aim to find the middle of the list using the "fast and slow pointer" technique. Here, we move one pointer (the fast pointer) two nodes at a time while moving the other (the slow pointer) one node at a time. By the time the fast pointer reaches the end of the list, the slow pointer will be positioned at the midpoint. This allows us to split the list in half effectively.

Once we have identified the middle, the second step is to reverse the second half of the linked list. By reversing this part, we can now compare the first half of the list with the reversed second half in a straightforward manner. Reversing a linked list is straightforward; we iteratively adjust the `next` pointers of nodes in the second half so that they point backward. This way, the list structure of the second half mirrors the order of the first half, and we can directly compare nodes from the start and the mirrored second half. This reversal is crucial because it enables us to compare in a single pass from both ends toward the center, ensuring that the space complexity remains constant.

In the next phase, we compare the values in the nodes of the first half with those in the reversed second half. We use two pointers for this, one at the start of the list and one at the head of the reversed section. We move both pointers one step at a time and check if the values at each position are identical. If all values match as we traverse, we confirm the list is a palindrome and return `true`. However, the list is not symmetrical if any values differ, and we return `false`. This comparison ensures that the algorithm correctly identifies palindrome lists in a single pass after reversing.

An optional but practical step is to restore the original list structure. Although this is not strictly required to solve the problem, reversing the second half back to its original state can be beneficial if the linked list will be used later. This step involves reversing the second half once more, effectively "undoing" the previous reversal. Restoring the list can be useful in contexts where the original list needs to be preserved for further operations.

In summary, this approach efficiently checks for palindromic linked lists by breaking the solution into four key stages: finding the midpoint, reversing the second half, comparing both halves and optionally restoring the list. This method is optimal in terms of both time and space, with a time complexity of (O(n)) for traversing the list and (O(1)) space for storing only a few pointers. By working within these constraints, the solution meets the challenge of determining palindrome lists effectively, even for very large inputs.