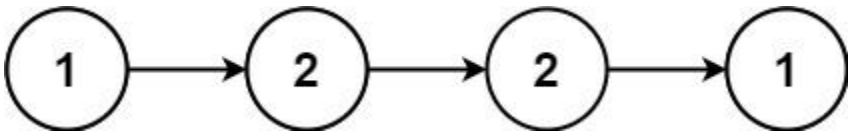
Problem Link:

https://leetcode.com/problems/palindrome-linked-list/

Problem Description:

Given the `head` of a singly linked list, return `true` *if it is a Palindrome or* `false` *otherwise*.



Input: head = [1,2,2,1]

Output: true

Problem Approach:

Split the list into two halves, reverse the first half, and then compare it with the second half.

Solution:

If the list has only one node or is empty, it is trivially a palindrome, so we return True. We use two pointers, slow and fast, to find the middle of the linked list. slow moves one step at a time, and fast moves two steps at a time. When fast reaches the end, slow will be in the middle. We reverse the first half of the linked list while reaching the middle. Once reversed, the leftPointer will point to the start of this reversed half. After finding the middle, we determine if the list is of even or odd length using fast. This ensures that the list comparison is handled correctly for both cases. Finally, we compare the values from the reversed first half (leftPointer) with the values from the second half (rightPointer). If all values match, the list is a palindrome.

The time complexity of this algorithm is O(n) and space complexity is O(1), making it an efficient solution.

Code (Python):

```
def isPalindrome(self, head: Optional[ListNode]) -> bool:
        if head is None or head.next is None:
            return True
        slow = head
        fast = head
        while fast.next is not None and fast.next.next is not None:
            slow = slow.next
            fast = fast.next.next
        rightPointer = slow.next
        is_even = fast.next is not None
        prev = None
        curr = head
        while curr != slow:
            next_node = curr.next
            curr.next = prev
            prev = curr
            curr = next_node
        if is_even:
            curr.next = prev
        else:
            curr = prev
        leftPointer = curr
        while leftPointer is not None:
            if leftPointer.val != rightPointer.val:
                return False
            leftPointer = leftPointer.next
            rightPointer = rightPointer.next
        return True
```