Description

234. Palindrome Linked List

```
Given a singly linked list, determine if it is a palindrome.
Follow up:
Could you do it in O(n) time and O(1) space?
```

Idea

The primitive idea is to check the first and last element, then check the second and second to the last element. To get to the end of the linked list, it takes O(n) time, so this would take O(n^2) time to check if this linked list is palindrome. How do we speed up?

Java

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null) {
            return true;
        }

        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode mid = getMid(dummy);

        ListNode cur = mid.next;
        mid.next = null;
        ListNode prev = null;
        while (cur != null) {
            ListNode next = cur.next;
            cur.next = prev;
            prev = cur;
            cur = next;
        }

        ListNode left = head;
        ListNode right = prev;
        while (left != null && right != null) {
            if (left.val != right.val) {
                return false;
            }
            left = left.next;
            right = right.next;
        }

        return true;
    }

    private ListNode getMid(ListNode head) {
        ListNode fast = head;
        while (fast != null && fast.next != null) {
            head = head.next;
            fast = fast.next.next;
        }
        return head;
    }
}
```

We can modulize most of the functions

Java

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        if (head == null) {
            return true;
        }

        // get mid
        ListNode mid = getMid(head);

        // reverse 2nd half
        ListNode right = reverse(mid.next);

        // check palindrom
        return isPalindrom(head, right);
    }

    private ListNode getMid(ListNode head) {
        ListNode fast = head;
        while (fast.next != null && fast.next.next != null) {
            head = head.next;
            fast = fast.next.next;
        }
        return head;
    }

    // return the head of the reversed linked list
    private ListNode reverse(ListNode cur) {
        ListNode prev = null;
        while (cur != null) {
            ListNode next = cur.next;
            cur.next = prev;
            prev = cur;
            cur = next;
        }
        return prev;
    }

    private boolean isPalindrom(ListNode left, ListNode right) {
        while (left != null && right != null) {
            if (left.val != right.val) {
                return false;
            }
            left = left.next;
            right = right.next;
        }
        return true;
    }
}
```

C++

```
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        if (head == NULL) {
            return true;
        }

        ListNode* mid = getMid(head);
        ListNode* right = reverse(mid->next);
        return isPalindrome(head, right);
    }

    ListNode* getMid(ListNode* head) {
        ListNode* fast = head;
        while (fast->next != NULL && fast->next->next != NULL) {
            head = head->next;
            fast = fast->next->next;
        }
        return head;
    }

    ListNode* reverse(ListNode* cur) {
        ListNode* prev = NULL;
        while (cur != NULL) {
            ListNode* next = cur->next;
            cur->next = prev;
            prev = cur;
            cur = next;
        }
        return prev;
    }

    bool isPalindrome(ListNode* left, ListNode* right) {
        while (left != NULL && right != NULL) {
            if (left->val != right->val) {
                return false;
            }
            left = left->next;
            right = right-> next;
        }
        return true;
    }
};
```

Summary

- Consider modify linked list structure if require O(1) extra space.
- Try modulize the functions.