ASSIGNMENT 12

1. Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def delete_middle_node(head):
    if not head or not head.next:
        return None

    slow = head
    fast = head
    prev = None

    while fast and fast.next:
        fast = fast.next.next
        prev = slow
        slow = slow.next

    prev.next = slow.next

    return head
```

2. Given a linked list of **N** nodes. The task is to check if the linked list has a loop. Linked list can contain self loop.

```
    class ListNode:
        def __init__(self, val=0, next=None):
            self.val = val
            self.next = next


    def has_loop(head):
        if not head or not head.next:
            return False

        slow = head
        fast = head.next

        while fast and fast.next:
            if slow == fast:
                return True

            slow = slow.next
            fast = fast.next.next
    return False
```

3. Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the **N**th node from the end of the linked list.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def find_nth_from_end(head, n):
    if not head:
        return None

    first = head
    second = head

    # Move the first pointer N nodes ahead
    for _ in range(n):
        if first is None:
            return None
        first = first.next

    # Move both pointers until the first pointer reaches the end
    while first is not None:
        first = first.next
        second = second.next

    return second
```

4. Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def is_palindrome(head):
    if not head or not head.next:
        return True

    # Find the middle node
    slow = head
    fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # Reverse the second half of the linked list
    second_half = reverse_linked_list(slow)

    # Compare the values of the first half and reversed second half
    first_half = head
    while second_half:
```

```
            if first_half.val != second_half.val:
                return False
            first_half = first_half.next
            second_half = second_half.next

        return True


    def reverse_linked_list(head):
        prev = None
        current = head

        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node

        return prev
```

5. Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def detect_and_remove_loop(head):
    if not head or not head.next:
        return head
    slow = head
    fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            break
    if slow != fast:
        return head
    slow = head
    while slow.next != fast.next:
        slow = slow.next
        fast = fast.next
    fast.next = None
 return head
```

5. Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def delete_nodes(head, M, N):
    if not head:
        return head

    current = head
    prev = None

    while current:
        # Traverse M nodes
        for _ in range(M):
            if not current:
                return head
            prev = current
            current = current.next

        # Delete N nodes
        for _ in range(N):
            if not current:
                break
            current = current.next

        # Connect the previous node to the next node after deleting N nodes
        prev.next = current

    return head
```

7. Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def merge_lists(first, second):
    if not first:
        return second
```

```
    current_first = first
    current_second = second

    while current_first and current_second:
        next_first = current_first.next
        next_second = current_second.next

        current_first.next = current_second
        current_second.next = next_first

        current_first = next_first
        current_second = next_second

    return first


# Create the first list: 5 -> 7 -> 17 -> 13 -> 11
first = ListNode(5)
first.next = ListNode(7)
first.next.next = ListNode(17)
first.next.next.next = ListNode(13)
first.next.next.next.next = ListNode(11)

# Create the second list: 12 -> 10 -> 2 -> 4 -> 6
second = ListNode(12)
second.next = ListNode(10)
second.next.next = ListNode(2)
second.next.next.next = ListNode(4)
second.next.next.next.next = ListNode(6)

# Merge the second list into the first list
merge_lists(first, second)

current = first
while current:
    print(current.val, end=" -> ")
    current = current.next

# Output: 5 -> 12 -> 7 -> 10 -> 17 -> 2 -> 13 -> 4 -> 11 -> 6 ->

# Print the modified second list
current = second
while current:
    print(current.val, end=" -> ")
    current = current.next
```

8. Given a singly linked list, find if the linked list is circular or not.

A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def is_circular(head):
    if not head or not head.next:
        return False

    slow = head
    fast = head.next

    while fast and fast.next:
        if slow == fast:
            return True
        slow = slow.next
        fast = fast.next.next

    return False
```