**ASSIGNMENT 12**

**Question 1**

Given a singly linked list, delete the **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then the 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 the 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

**Example 1:**

Input:

LinkedList: 1->2->3->4->5

Output:1 2 4 5

class Solution:

    def deleteMiddle(self, head: Optional[ListNode]) -> Optional[ListNode]:

        if head == None:

            return head

        slow = head

        fast = head

        prev = None

        while (fast and fast.next):

            prev = slow

            slow = slow.next

            fast = fast.next.next

        prev.next = prev.next.next

        slow = slow.next

        return head

**Question 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.

**Example 1:**

Input:

N = 3

value[] = {1,3,4}

x(position at which tail is connected) = 2

Output:True

Explanation:In above test case N = 3.

The linked list with nodes N = 3 is

given. Then value of x=2 is given which

means last node is connected with xth

node of linked list. Therefore, there

exists a loop.

**Example 2:**

Input:

N = 4

value[] = {1,8,3,4}

x = 0

Output:False

Explanation:For N = 4 ,x = 0 means

then lastNode->next = NULL, then

the Linked list does not contains

any loop.

class Solution:

    def hasCycle(self, head: Optional[ListNode]) -> bool:

        if head == None:

            return False

        slow = head

        while(head and head.next):

            slow = slow.next

            head = head.next.next

            if head == slow:

                return True

        return False

**Question 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.

**Example 1:**

Input:

N = 2

LinkedList: 1->2->3->4->5->6->7->8->9

Output:8

Explanation:In the first example, there

are 9 nodes in linked list and we need

to find 2nd node from end. 2nd node

from end is 8.

**Example 2:**

Input:

N = 5

LinkedList: 10->5->100->5

Output:-1

Explanation:In the second example, there

are 4 nodes in the linked list and we

need to find 5th from the end. Since 'n'

is more than the number of nodes in the

linked list, the output is -1.

class Solution:

    def removeNthFromEnd(self, head: Optional[ListNode], n: int) -> Optional[ListNode]:

        dummy = ListNode(0, head)

        left = dummy

        right = head

# sbse pehle right ko nth position tk lekr jaate hai is loop se

        while n>0 and right:

            right = right.next

            n -= 1

# Ab is loop se dono left or right pointer ko ek-ek position move krwaate hai jbtk right Null na ho jaaye

        while right:

            left = left.next

            right = right.next

# Now we have reached a place jaha se hum nth element ko delete kr skte

        left.next = left.next.next

        return dummy.next

#Dummy.next becoz hume original list return krni haii, 0 add kri hui dummy node nahi.

**Question 4**

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

<https://media.geeksforgeeks.org/wp-content/uploads/20220816144425/LLdrawio.png>

**Examples:**

Input: R->A->D->A->R->NULL

**Output:** Yes

**Input:** C->O->D->E->NULL

**Output:** No

# Step1 - Find the middle of the list

# step2 - Reverse the 2nd half

# Step3- traverse and check

class Solution:

    def isPalindrome(self, head: Optional[ListNode]) -> bool:

        slow = head

        fast = head

        if head == None:

            return True

        # Find Middle

        while (fast and fast.next):

            slow = slow.next

            fast = fast.next.next

# Reverse the Second half

        prev = None

        while slow:

            temp = slow.next

            slow.next = prev

            prev = slow

            slow = temp

# Check if its palindrome

# traverse and check

# slow is at last

# slow = prev

        fast = head

        slow = prev

        while slow :

            if (fast.val != slow.val):

                return False

            slow = slow.next

            head = head.next

        return True

**Question 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.

**Example 1:**

Input:

N = 3

value[] = {1,3,4}

X = 2

Output:1

Explanation:The link list looks like

1 -> 3 -> 4

^ |

|\_\_\_\_|

A loop is present. If you remove it

successfully, the answer will be 1.

**Example 2:**

Input:

N = 4

value[] = {1,8,3,4}

X = 0

Output:1

Explanation:The Linked list does not

contains any loop.

**Example 3:**

Input:

N = 4

value[] = {1,2,3,4}

X = 1

Output:1

Explanation:The link list looks like

1 -> 2 -> 3 -> 4

^ |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_|

A loop is present.

If you remove it successfully,

the answer will be 1.

Approach:-

1. Traverse the complete linked list with two pointers (slow, fast) and if the two pointers meet at the same node then a loop exists.
2. Now, if a loop exists then take one pointer to the head and keep the other pointer at the same place where both pointers met.
3. Now check slow. next and fast. next, either they are the same if yes then make the fast. next = null. And if not then move both the pointers one step.

class Solution:

#Function to remove a loop in the linked list.

def removeLoop(self, head):

# code here

# remove the loop without losing any nodes

slow = head

fast = head

while (fast and fast.next):

slow = slow.next

fast = fast.next.next

if fast == slow:

break

slow = head

if slow.next == fast.next:

fast.next = None

else:

slow = slow.next

fast = fast.next

return head

**Question 6**

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

Difficulty Level: Rookie

**Examples**:

Input:

M = 2, N = 2

Linked List: 1->2->3->4->5->6->7->8

Output:

Linked List: 1->2->5->6

Input:

M = 3, N = 2

Linked List: 1->2->3->4->5->6->7->8->9->10

Output:

Linked List: 1->2->3->6->7->8

Input:

M = 1, N = 1

Linked List: 1->2->3->4->5->6->7->8->9->10

Output:

Linked List: 1->3->5->7->9

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** skipMdeleteN(head, M, N):

**if** head **is** **None** **or** M **<=** 0 **or** N **<=** 0:

**return** head

current **=** head

previous **=** **None**

**while** current **is** **not** **None**:

*# Skip M nodes*

**for** \_ **in** range(M):

**if** current **is** **None**:

**break**

previous **=** current

current **=** current**.**next

*# Delete N nodes*

**for** \_ **in** range(N):

**if** current **is** **None**:

**break**

current **=** current**.**next

*# Link previous node to the next node after N nodes*

**if** previous **is** **not** **None**:

previous**.**next **=** current

return head

**Question 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 the second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and the second list should become empty. The nodes of the second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and the second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and the 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.

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** insertAtAlternatePositions(first, second):

**if** first **is** **None**:

**return** second

**if** second **is** **None**:

**return** first

first\_curr **=** first

second\_curr **=** second

**while** first\_curr **is** **not** **None** **and** second\_curr **is** **not** **None**:

first\_next **=** first\_curr**.**next

second\_next **=** second\_curr**.**next

second\_curr**.**next **=** first\_next

first\_curr**.**next **=** second\_curr

first\_curr **=** first\_next

second\_curr **=** second\_next

*# Append remaining nodes of second list to the end of first list*

**if** second\_curr **is** **not** **None**:

first\_curr**.**next **=** second\_curr

**return** first

**Question 8**

Given a singly linked list, find if the linked list is [circular](https://www.geeksforgeeks.org/circular-linked-list/amp/) 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.

**CODE:**

**class** ListNode:

**def** \_\_init\_\_(self, val**=**0, next**=None**):

self**.**val **=** val

self**.**next **=** next

**def** isCircular(head):

**if** head **is** **None**:

**return** **False**

slow **=** head

fast **=** head**.**next

**while** fast **is** **not** **None** **and** fast**.**next **is** **not** **None**:

**if** slow **==** fast:

**return** **True**

slow **=** slow**.**next

fast **=** fast**.**next**.**next

**return** **False**