**ASSIGNMENT-XII**

**Question 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

**Example 1:**

Input:

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

Output:1 2 4 5

**Example 2:**

Input:

LinkedList: 2->4->6->7->5->1

Output:2 4 6 5 1

**Ans**: class ListNode:

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

self.val = val

self.next = next

def deleteMiddleNode(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

**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.

**Ans:** class ListNode:

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

self.val = val

self.next = next

def hasLoop(head):

slow = head

fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

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.

**Ans:** class ListNode:

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

self.val = val

self.next = next

def findNthFromEnd(head, N):

fast = head

slow = head

for \_ in range(N):

if fast is None:

return -1

fast = fast.next

while fast:

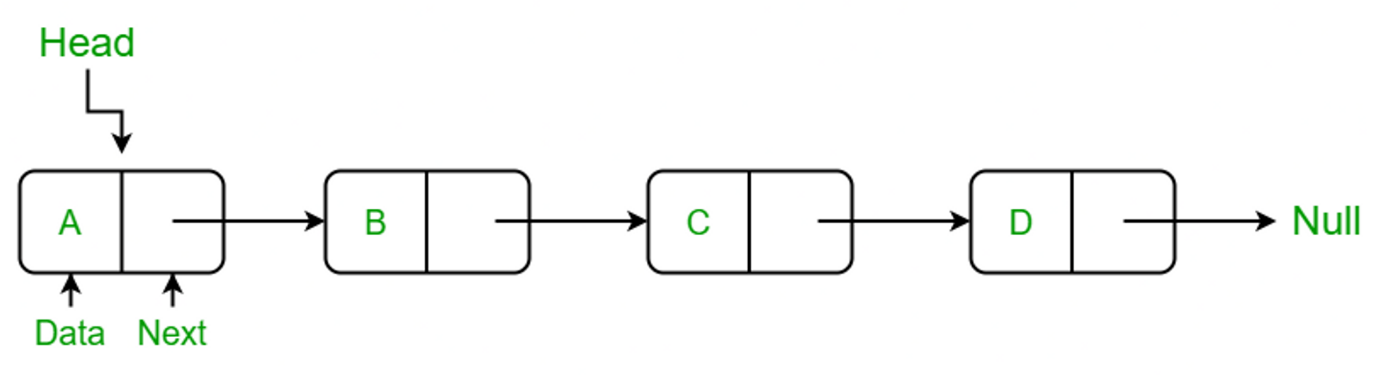
slow = slow.next

fast = fast.next

return slow.val

**Question 4**

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



**Examples:**

Input: R->A->D->A->R->NULL

**Output:** Yes

**Input:** C->O->D->E->NULL

**Output:** No

**Ans**: class ListNode:

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

self.val = val

self.next = next

def isPalindrome(head):

if not head or not head.next:

return True

slow = head

fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

prev = None

curr = slow

while curr:

next\_node = curr.next

curr.next = prev

prev = curr

curr = next\_node

first\_half = head

second\_half = prev

while second\_half:

if first\_half.val != second\_half.val:

return False

first\_half = first\_half.next

second\_half = second\_half.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.

**Ans:**

**class ListNode:**

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

**self.val = val**

**self.next = next**

**def detectAndRemoveLoop(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**

**Question 6**

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.

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

**Ans:**

**class ListNode:**

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

**self.val = val**

**self.next = next**

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

**if M == 0 or N == 0:**

**return None**

**curr = head**

**prev = None**

**while curr:**

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

**if curr:**

**prev = curr**

**curr = curr.next**

**else:**

**break**

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

**if curr:**

**curr = curr.next**

**else:**

**break**

**prev.next = curr**

**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 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.

**Ans:**

**class ListNode:**

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

**self.val = val**

**self.next = next**

**def insertAtAlternatePositions(firstHead, secondHead):**

**if not secondHead:**

**return firstHead**

**firstCurr = firstHead**

**secondCurr = secondHead**

**while firstCurr and secondCurr:**

**firstNext = firstCurr.next**

**secondNext = secondCurr.next**

**firstCurr.next = secondCurr**

**secondCurr.next = firstNext**

**firstCurr = firstNext**

**secondCurr = secondNext**

**if secondCurr:**

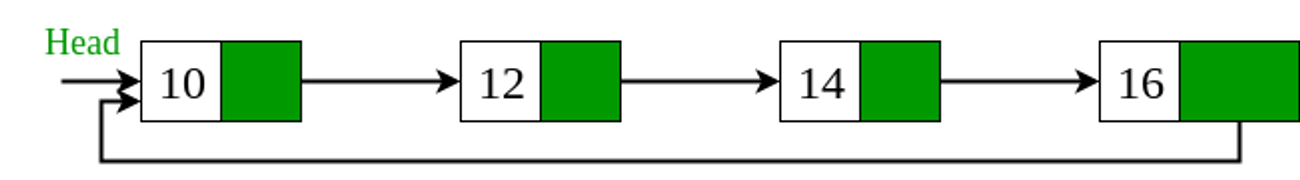
**firstCurr.next = secondCurr**

**return firstHead**

**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.



**Ans:** class ListNode:

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

self.val = val

self.next = next

def isCircular(head):

if not head:

return False

slow = head

fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

return True

return False