**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 Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

def delete\_middle\_node(head):

if head is None or head.next is None:

return None

slow\_ptr = head

fast\_ptr = head

prev\_ptr = None

while fast\_ptr is not None and fast\_ptr.next is not None:

fast\_ptr = fast\_ptr.next.next

prev\_ptr = slow\_ptr

slow\_ptr = slow\_ptr.next

prev\_ptr.next = slow\_ptr.next

slow\_ptr.next = None

del slow\_ptr

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.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

# Definition for singly-linked list.

# class ListNode:

#     def \_\_init\_\_(self, x):

#         self.val = x

#         self.next = None

class Solution:

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

        s, f = head,head

        while f and f.next:

            s = s.next

            f = f.next.next

            if f == s: 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.**

# A Linked List Node

class Node:

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

        self.data = data

        self.next = next

# Recursive function to return the k'th node from the end in a linked list

def findKthNode(node, k):

    # base case

    if node is None:

        return 0

    count = findKthNode(node.next, k) + 1

    if count == k:

        print('k\'th node from the end is', node.data)

    return count

if \_\_name\_\_ == '\_\_main\_\_':

    head = None

    for i in reversed(range(5)):

        head = Node(i + 1, head)

    k = 3

    findKthNode(head, k)

**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.** Solution from my leetcode :- <https://leetcode.com/gopsa2001/>

# Definition for singly-linked list.

# class ListNode:

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

#         self.val = val

#         self.next = next

class Solution:

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

        fast, slow  = head,head

        while fast and fast.next:

            slow = slow.next

            fast = fast.next.next

        prev = None

        while slow:

            temp = slow.next

            slow.next = prev

            prev = slow

            slow = temp

        while head and prev:

            if head.val != prev.val:

                return False

            else:

                head = head.next

                prev = prev.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.

**Ans.**

def removeCycle(slow, head):

curr = head

while curr:

ptr = slow

while ptr.next is not slow and ptr.next is not curr:

ptr = ptr.next

if ptr.next == curr:

ptr.next = None

return

curr = curr.next

def identifyCycle(head):

slow = fast = head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

return slow

return None

**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.**

**def** printList(self):

        temp **=** self.head

**while**(temp):

**print** (temp.data,end**=**" ")

            temp **=** temp.next

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

        curr **=** self.head

        # The main loop that traverses through the

        # whole list

**while**(curr):

            # Skip M nodes

**for** count **in** range(1, M):

**if** curr **is** None:

**return**

                curr **=** curr.next

**if** curr **is** None :

**return**

            # Start from next node and delete N nodes

            t **=** curr.next

**for** count **in** range(1, N**+**1):

**if** t **is** None:

**break**

                t **=** t.next

            # Link the previous list with remaining nodes

            curr.next **=** t

            # Set Current pointer for next iteration

            curr **=** t

**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.**

**def** printList(self):

        temp **=** self.head

**while** temp !**=** None:

**print**(temp.data)

            temp **=** temp.next

    # Main function that inserts nodes of linked list q into p at alternate positions.

    # Since head of first list never changes

    # but head of second list/ may change,

    # we need single pointer for first list and double pointer for second list.

**def** merge(self, p, q):

        p\_curr **=** p.head

        q\_curr **=** q.head

        # swap their positions until one finishes off

**while** p\_curr !**=** None **and** q\_curr !**=** None:

            # Save next pointers

            p\_next **=** p\_curr.next

            q\_next **=** q\_curr.next

            # make q\_curr as next of p\_curr

            q\_curr.next **=** p\_next  # change next pointer of q\_curr

            p\_curr.next **=** q\_curr  # change next pointer of p\_curr

            # update current pointers for next iteration

            p\_curr **=** p\_next

            q\_curr **=** q\_next

            q.head **=** q\_curr

**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.**

**def** Circular(head):

**if** head **==** None:

**return** True

    # Next of head

    node **=** head.next

    i **=** 0

    # This loop would stop in both cases (1) If

    # Circular (2) Not circular

**while**((node **is** **not** None) **and** (node **is** **not** head)):

        i **=** i **+** 1

        node **=** node.next

**return**(node **==** head)