💡 \*\*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

**class** Node:

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

self**.**data **=** data

self**.**next **=** **None**

**def** delete\_middle\_node(head):

**if** head **is** **None** **or** head**.**next **is** **None**:

**return** head

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

**return** head

*# Create a linked list: 1->2->3->4->5*

head **=** Node(1)

head**.**next **=** Node(2)

head**.**next**.**next **=** Node(3)

head**.**next**.**next**.**next **=** Node(4)

head**.**next**.**next**.**next**.**next **=** Node(5)

print("Original Linked List:")

current **=** head

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

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

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

print()

head **=** delete\_middle\_node(head)

print("Modified Linked List:")

current **=** head

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

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

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

print()

#output-

Original Linked List:

1 2 3 4 5

Modified Linked List:

1 2 4 5

💡 \*\*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** Node:

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

self**.**data **=** data

self**.**next **=** **None**

**def** detect\_loop(head):

slow\_ptr **=** head

fast\_ptr **=** head

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

slow\_ptr **=** slow\_ptr**.**next

fast\_ptr **=** fast\_ptr**.**next**.**next

**if** slow\_ptr **==** fast\_ptr:

**return** **True**

**return** **False**

*# Create a linked list: 1->2->3->4->5->2 (loop)*

head **=** Node(1)

head**.**next **=** Node(2)

head**.**next**.**next **=** Node(3)

head**.**next**.**next**.**next **=** Node(4)

head**.**next**.**next**.**next**.**next **=** Node(5)

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

loop\_exists **=** detect\_loop(head)

print("Loop exists:", loop\_exists)

#output- Loop exists: True

**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** Node:

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

self**.**data **=** data

self**.**next **=** **None**

**def** find\_nth\_from\_end(head, n):

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

**return** **-**1

slow\_ptr **=** head

fast\_ptr **=** head

*# Move the fast pointer to the Nth node*

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

**if** fast\_ptr **is** **None**:

**return** **-**1

fast\_ptr **=** fast\_ptr**.**next

*# Move both pointers until the fast pointer reaches the end*

**while** fast\_ptr **is** **not** **None**:

slow\_ptr **=** slow\_ptr**.**next

fast\_ptr **=** fast\_ptr**.**next

**if** slow\_ptr **is** **None**:

**return** **-**1

**return** slow\_ptr**.**data

*# Create a linked list: 1->2->3->4->5->6->7->8->9*

head **=** Node(1)

head**.**next **=** Node(2)

head**.**next**.**next **=** Node(3)

head**.**next**.**next**.**next **=** Node(4)

head**.**next**.**next**.**next**.**next **=** Node(5)

head**.**next**.**next**.**next**.**next**.**next **=** Node(6)

head**.**next**.**next**.**next**.**next**.**next**.**next **=** Node(7)

head**.**next**.**next**.**next**.**next**.**next**.**next**.**next **=** Node(8)

head**.**next**.**next**.**next**.**next**.**next**.**next**.**next**.**next **=** Node(9)

n **=** 2

nth\_node **=** find\_nth\_from\_end(head, n)

print("Nth Node from the End:", nth\_node)

#output- Nth Node from the End: 8

💡 **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

**class** Node:

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

self**.**data **=** data

self**.**next **=** **None**

**def** is\_palindrome(head):

**if** head **is** **None** **or** head**.**next **is** **None**:

**return** **True**

slow\_ptr **=** head

fast\_ptr **=** head

*# Find the middle node of the linked list*

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

slow\_ptr **=** slow\_ptr**.**next

fast\_ptr **=** fast\_ptr**.**next**.**next

*# Reverse the second half of the linked list*

prev\_ptr **=** **None**

current **=** slow\_ptr

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

next\_ptr **=** current**.**next

current**.**next **=** prev\_ptr

prev\_ptr **=** current

current **=** next\_ptr

*# Compare the first half and reversed second half of the linked list*

ptr1 **=** head

ptr2 **=** prev\_ptr

**while** ptr2 **is** **not** **None**:

**if** ptr1**.**data **!=** ptr2**.**data:

**return** **False**

ptr1 **=** ptr1**.**next

ptr2 **=** ptr2**.**next

**return** **True**

*# Create a palindrome linked list: R->A->D->A->R*

head **=** Node('R')

head**.**next **=** Node('A')

head**.**next**.**next **=** Node('D')

head**.**next**.**next**.**next **=** Node('A')

head**.**next**.**next**.**next**.**next **=** Node('R')

is\_pal **=** is\_palindrome(head)

print("Is Palindrome:", is\_pal)

#output- Is Palindrome: True

💡 \*\*Question 5\*\*

Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must be **unique** and you may return the result in **any order**.

**Example 1:**

Input: nums1 = [1,2,2,1], nums2 = [2,2]

Output: [2]

**def** intersection(nums1, nums2):

set1 **=** set(nums1)

set2 **=** set(nums2)

**return** list(set1**.**intersection(set2))

*# Example 1*

nums1 **=** [1, 2, 2, 1]

nums2 **=** [2, 2]

print(intersection(nums1, nums2)) *# Output: [2]*

*# Example 2*

nums1 **=** [4, 9, 5]

nums2 **=** [9, 4, 9, 8, 4]

print(intersection(nums1, nums2)) *# Output: [9, 4]*

**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.

**class** Node:

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

self**.**data **=** data

self**.**next **=** **None**

**def** detect\_and\_remove\_loop(head):

**if** head **is** **None** **or** head**.**next **is** **None**:

**return**

slow\_ptr **=** head

fast\_ptr **=** head

*# Detect the loop in the linked list*

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

slow\_ptr **=** slow\_ptr**.**next

fast\_ptr **=** fast\_ptr**.**next**.**next

**if** slow\_ptr **==** fast\_ptr:

**break**

**if** slow\_ptr **==** fast\_ptr:

*# Move the slow pointer to the head and*

*# move both pointers one step at a time*

slow\_ptr **=** head

**while** slow\_ptr**.**next **!=** fast\_ptr**.**next:

slow\_ptr **=** slow\_ptr**.**next

fast\_ptr **=** fast\_ptr**.**next

*# Remove the loop by setting the next pointer of the last node to None*

fast\_ptr**.**next **=** **None**

**def** print\_linked\_list(head):

current **=** head

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

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

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

print()

*# Create a linked list with a loop: 1->3->4->2->5->2 (loop)*

head **=** Node(1)

head**.**next **=** Node(3)

head**.**next**.**next **=** Node(4)

head**.**next**.**next**.**next **=** Node(2)

head**.**next**.**next**.**next**.**next **=** Node(5)

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

print("Original Linked List:")

print\_linked\_list(head)

detect\_and\_remove\_loop(head)

print("Linked List after removing the loop:")

print\_linked\_list(head)