Assignment 12 Solutions

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

In [33]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def delete_middle_node(head):
    if head is None or head.next is None:
        return None

    slow = fast = head
    prev = None

    while fast is not None and fast.next is not None:
        fast = fast.next.next
        prev = slow
        slow = slow.next

    prev.next = slow.next
    slow.next = None

    return head
```

In [34]:
```
# Example 1
# Input: 1->2->3->4->5
# Output: 1->2->4->5

# Create the linked list
head1 = ListNode(1)
head1.next = ListNode(2)
head1.next.next = ListNode(3)
head1.next.next.next = ListNode(4)
head1.next.next.next.next = ListNode(5)

# Delete the middle node(s)
head1 = delete_middle_node(head1)

# Print the modified linked list
current = head1
while current is not None:
    print(current.val, end=" ")
    current = current.next
# Output: 1 2 4 5
```

```
1 2 4 5
```

In [35]:
```
# Example 2
# Input: 2->4->6->7->5->1
# Output: 2->4->6->5->1

# Create the linked list
```

```
head2 = ListNode(2)
head2.next = ListNode(4)
head2.next.next = ListNode(6)
head2.next.next.next = ListNode(7)
head2.next.next.next.next = ListNode(5)
head2.next.next.next.next.next = ListNode(1)

# Delete the middle node(s)
head2 = delete_middle_node(head2)

# Print the modified linked list
current = head2
while current is not None:
    print(current.val, end=" ")
    current = current.next
# Output: 2 4 6 5 1
```

```
2 4 6 5 1
```

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.

In [36]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def has_loop(head):
    if head is None:
        return False

    slow = fast = head

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

```
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            return True

    return False
```

In [37]:
```
# Example 1
# Input: N = 3, value[] = {1, 3, 4}, x = 2
# Output: True

# Create the linked list
head1 = ListNode(1)
head1.next = ListNode(3)
head1.next.next = ListNode(4)
head1.next.next.next = head1.next  # Create a loop by connecting the last node to the xth node

# Check if the linked list has a loop
result1 = has_loop(head1)
print(result1)
# Output: True
```

```
True
```

In [38]:
```
# Example 2
# Input: N = 4, value[] = {1, 8, 3, 4}, x = 0
# Output: False

# Create the linked list
head2 = ListNode(1)
head2.next = ListNode(8)
head2.next.next = ListNode(3)
head2.next.next.next = ListNode(4)

# Check if the linked list has a loop
result2 = has_loop(head2)
print(result2)
# Output: False
```

```
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.

In [39]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def find_nth_from_end(head, n):
    if head is None or n <= 0:
        return -1

    fast = slow = head

    # Move the fast pointer N nodes ahead
    for _ in range(n):
        if fast is None:
            return -1
        fast = fast.next

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

    return slow.val
```

In [40]:
```
# Example 1
# Input: N = 2, LinkedList: 1->2->3->4->5->6->7->8->9
# Output: 8

# Create the linked list
head1 = ListNode(1)
head1.next = ListNode(2)
head1.next.next = ListNode(3)
head1.next.next.next = ListNode(4)
head1.next.next.next.next = ListNode(5)
head1.next.next.next.next.next = ListNode(6)
head1.next.next.next.next.next.next = ListNode(7)
head1.next.next.next.next.next.next.next = ListNode(8)
head1.next.next.next.next.next.next.next.next = ListNode(9)

# Find the Nth node from the end of the linked list
result1 = find_nth_from_end(head1, 2)
print(result1)
# Output: 8
```

```
8
```

In [41]:
```
# Example 2
# Input: N = 5, LinkedList: 10->5->100->5
# Output: -1

# Create the linked list
head2 = ListNode(10)
head2.next = ListNode(5)
head2.next.next = ListNode(100)
head2.next.next.next = ListNode(5)

# Find the Nth node from the end of the linked list
result2 = find_nth_from_end(head2, 5)
print(result2)
# Output: -1
```

```
-1
```

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

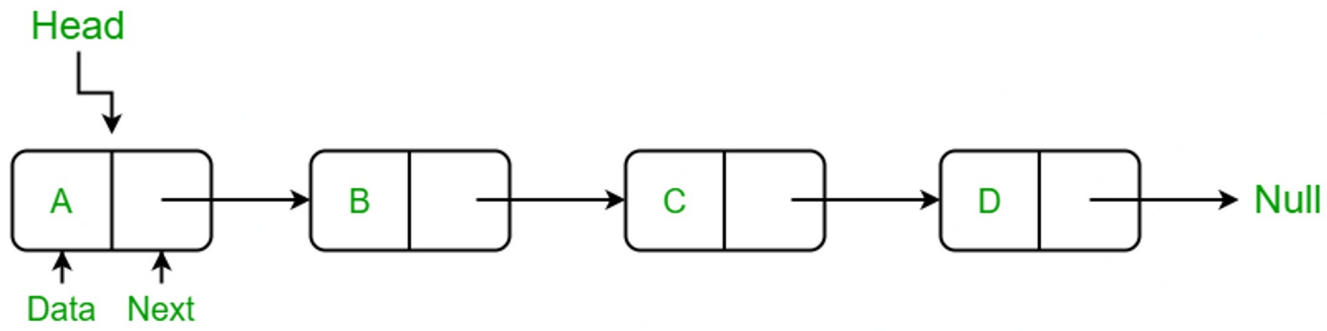
Examples:

> Input: R->A->D->A->R->NULL
>
> **Output:** Yes
>
> **Input:** C->O->D->E->NULL
>
> **Output:** No

In [42]:
```
from IPython import display
```

```
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\LLdrawio.png")
```

Out[42]:



In [43]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def is_palindrome(head):
    if head is None or head.next is None:
        return True

    slow = fast = head
    stack = []

    while fast is not None and fast.next is not None:
        stack.append(slow.val)
        slow = slow.next
        fast = fast.next.next

    # Skip the middle node if the number of nodes is odd
    if fast is not None:
        slow = slow.next

    while slow is not None:
        if slow.val != stack.pop():
            return False
        slow = slow.next

    return True
```

In [44]:
```
# Example 1
# Input: R->A->D->A->R->NULL
# Output: Yes

# Create the linked list
head1 = ListNode('R')
head1.next = ListNode('A')
head1.next.next = ListNode('D')
head1.next.next.next = ListNode('A')
head1.next.next.next.next = ListNode('R')

# Check if the linked list is a palindrome
result1 = is_palindrome(head1)
print(result1)
# Output: True
```

```
True
```

In [45]:
```
# Example 2
# Input: C->O->D->E->NULL
# Output: No

# Create the linked list
head2 = ListNode('C')
head2.next = ListNode('O')
head2.next.next = ListNode('D')
head2.next.next.next = ListNode('E')

# Check if the linked list is a palindrome
result2 = is_palindrome(head2)
print(result2)
# Output: False
```

```
False
```

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.

In [46]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def detect_and_remove_loop(head):
    if head is None or head.next is None:
        return head

    slow = fast = head

    # Detect the loop
    while fast is not None and fast.next is not None:
        slow = slow.next
        fast = fast.next.next
```

```
        if slow == fast:
            break

    # If there is no loop, return the head of the linked list
    if fast is None or fast.next is None:
        return head

    # Move the slow pointer to the head and find the starting point of the loop
    slow = head
    while slow.next != fast.next:
        slow = slow.next
        fast = fast.next

    # Break the loop by setting the next pointer of the node to None
    fast.next = None

    return head
```

In [47]:
```
# Example 1
# Input: N = 3, value[] = {1, 3, 4}, X = 2
# Output: 1

# Create the linked list
head1 = ListNode(1)
head1.next = ListNode(3)
head1.next.next = ListNode(4)
head1.next.next.next = head1.next  # Create a loop by connecting the last node to the Xth node

# Remove the loop from the linked list
result1 = detect_and_remove_loop(head1)
print(result1.val)
# Output: 1
```

```
1
```

In [48]:
```
# Example 2
# Input: N = 4, value[] = {1, 8, 3, 4}, X = 0
# Output: 1

# Create the linked list
head2 = ListNode(1)
head2.next = ListNode(8)
head2.next.next = ListNode(3)
head2.next.next.next = ListNode(4)

# Remove the loop from the linked list
result2 = detect_and_remove_loop(head2)
print(result2.val)
# Output: 1
```

```
1
```

In [49]:
```
# Example 3
# Input: N = 4, value[] = {1, 2, 3, 4}, X = 1
# Output: 1

# Create the linked list
head3 = ListNode(1)
head3.next = ListNode(2)
head3.next.next = ListNode(3)
head3.next.next.next = ListNode(4)
head3.next.next.next.next = head3  # Create a loop by connecting the last node to the first node

# Remove the loop from the linked list
result3 = detect_and_remove_loop(head3)
print(result3.val)
# Output: 1
```

```
1
```

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

In [50]:
```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def retain_delete(head, M, N):
    if head is None or M <= 0:
        return head

    curr = head

    while curr is not None:
        # Skip M nodes
        for _ in range(M - 1):
            if curr.next is not None:
                curr = curr.next
            else:
                return head

        # Delete N nodes
        next_node = curr.next
        for _ in range(N):
            if next_node is not None:
                next_node = next_node.next
            else:
                curr.next = None
                return head

        curr.next = next_node
        curr = next_node

    return head
```

In [51]:
```python
# Example 1
# Input: M = 2, N = 2, Linked List: 1->2->3->4->5->6->7->8
# Output: Linked List: 1->2->5->6

# Create the linked list
head1 = ListNode(1)
head1.next = ListNode(2)
head1.next.next = ListNode(3)
head1.next.next.next = ListNode(4)
head1.next.next.next.next = ListNode(5)
head1.next.next.next.next.next = ListNode(6)
head1.next.next.next.next.next.next = ListNode(7)
head1.next.next.next.next.next.next.next = ListNode(8)

# Modify the linked list
result1 = retain_delete(head1, 2, 2)

# Print the modified linked list
curr = result1
while curr is not None:
    print(curr.val, end=" ")
```

```
    curr = curr.next
# Output: 1->2->5->6
```

```
1 2 5 6
```

In [52]:
```
# Example 2
# 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

# Create the linked list
head2 = ListNode(1)
head2.next = ListNode(2)
head2.next.next = ListNode(3)
head2.next.next.next = ListNode(4)
head2.next.next.next.next = ListNode(5)
head2.next.next.next.next.next = ListNode(6)
head2.next.next.next.next.next.next = ListNode(7)
head2.next.next.next.next.next.next.next = ListNode(8)
head2.next.next.next.next.next.next.next.next = ListNode(9)
head2.next.next.next.next.next.next.next.next.next = ListNode(10)

# Modify the linked list
result2 = retain_delete(head2, 3, 2)

# Print the modified linked list
curr = result2
while curr is not None:
    print(curr.val, end=" ")
    curr = curr.next
# Output: 1->2->3->6->7->8
```

```
1 2 3 6 7 8
```

In [53]:
```
# Example 3
# 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

# Create the linked list
head3 = ListNode(1)
head3.next = ListNode(2)
head3.next.next = ListNode(3)
head3.next.next.next = ListNode(4)
head3.next.next.next.next = ListNode(5)
head3.next.next.next.next.next = ListNode(6)
head3.next.next.next.next.next.next = ListNode(7)
head3.next.next.next.next.next.next.next = ListNode(8)
head3.next.next.next.next.next.next.next.next = ListNode(9)
head3.next.next.next.next.next.next.next.next.next = ListNode(10)

# Modify the linked list
result3 = retain_delete(head3, 1, 1)

# Print the modified linked list
curr = result3
while curr is not None:
    print(curr.val, end=" ")
    curr = curr.next
# Output: 1->3->5->7->9
```

```
1 3 5 7 9
```

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.

In [54]:
```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next


def insert_at_alternate_positions(first, second):
    if first is None:
        return second
```

```
    if second is None:
        return first

    curr1 = first
    curr2 = second

    while curr1 is not None and curr2 is not None:
        next1 = curr1.next
        next2 = curr2.next

        curr1.next = curr2
        curr2.next = next1

        curr1 = next1
        curr2 = next2

    if curr2 is not None:
        curr1.next = curr2

    return first
```

In [55]:
```
# Example
# First list: 5->7->17->13->11
# Second list: 12->10->2->4->6
# Expected output:
# First list: 5->12->7->10->17->2->13->4->11->6
# Second list: None

# Create the first list
head1 = ListNode(5)
head1.next = ListNode(7)
head1.next.next = ListNode(17)
head1.next.next.next = ListNode(13)
head1.next.next.next.next = ListNode(11)

# Create the second list
head2 = ListNode(12)
head2.next = ListNode(10)
head2.next.next = ListNode(2)
head2.next.next.next = ListNode(4)
head2.next.next.next.next = ListNode(6)

# Insert nodes of the second list into the first list at alternate positions
result = insert_at_alternate_positions(head1, head2)

# Print the modified first list
curr = result
while curr is not None:
    print(curr.val, end="->")
    curr = curr.next
print("None")

# Print the modified second list
curr = head2
while curr is not None:
    print(curr.val, end="->")
    curr = curr.next
print("None")
```

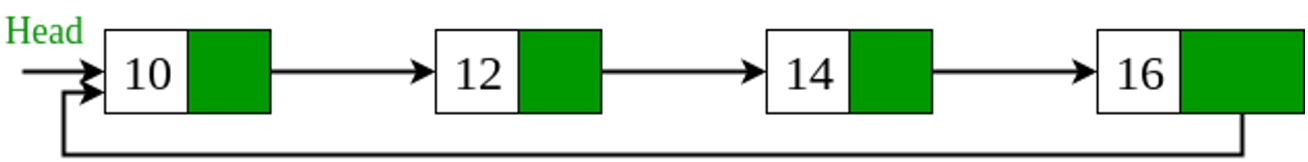
```
5->12->7->10->17->2->13->4->11->6->None
12->7->10->17->2->13->4->11->6->None
```

Question 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.

In [56]:
```
from IPython import display
display.Image(r"C:\Users\hrush\OneDrive\Pictures\Saved Pictures\asd.png")
```

Out[56]:



In [57]:
```
class ListNode:
    def __init__(self, val=0, next=None):
```

```
        self.val = val
        self.next = next


def is_circular(head):
    if head is None:
        return False

    slow = head
    fast = head

    while fast is not None and fast.next is not None:
        slow = slow.next
        fast = fast.next.next

        if slow == fast:
            return True

    return False
```

In [58]:
```
# Example
# Linked list: 1->2->3->4->5->2 (circular)
# Expected output: True

# Create the circular linked list
head = ListNode(1)
head.next = ListNode(2)
head.next.next = ListNode(3)
head.next.next.next = ListNode(4)
head.next.next.next.next = ListNode(5)
head.next.next.next.next.next = head.next

# Check if the linked list is circular
result = is_circular(head)

print(result)  # Output: True
```

```
True
```

In [ ]:

In [ ]: