Assignment 13 Solutions

Q1. Given two linked list of the same size, the task is to create a new linked list using those linked lists. The condition is that the greater node among both linked list will be added to the new linked list.

Examples:

Input: list1 = 5->2->3->8

list2 = 1->7->4->5

Output: New list = 5->7->4->8

Input:list1 = 2->8->9->3

list2 = 5->3->6->4

Output: New list = 5->8->9->4

In [24]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def create_new_linked_list(list1, list2):
    head1 = list1
    head2 = list2
    newList = None
    tail = None

    while head1 is not None or head2 is not None:
        if head1 is None:
            tail.next = head2
            break
        if head2 is None:
            tail.next = head1
            break

        if head1.data >= head2.data:
            new_node = Node(head1.data)
            head1 = head1.next
        else:
            new_node = Node(head2.data)
            head2 = head2.next

        if newList is None:
            newList = new_node
            tail = new_node
        else:
            tail.next = new_node
            tail = new_node

    return newList
```

In [25]:
```
# Example 1
list1 = Node(5)
list1.next = Node(2)
list1.next.next = Node(3)
list1.next.next.next = Node(8)

list2 = Node(1)
list2.next = Node(7)
list2.next.next = Node(4)
list2.next.next.next = Node(5)

new_list = create_new_linked_list(list1, list2)

current = new_list
while current is not None:
    print(current.data, end=" ")
    current = current.next
```

```
5 2 3 8 1 7 4 5
```

In [26]:
```
# Example 2
list1 = Node(2)
```

```
list1.next = Node(8)
list1.next.next = Node(9)
list1.next.next.next = Node(3)

list2 = Node(5)
list2.next = Node(3)
list2.next.next = Node(6)
list2.next.next.next = Node(4)

new_list = create_new_linked_list(list1, list2)

current = new_list
while current is not None:
    print(current.data, end=" ")
    current = current.next
```

```
5 3 6 4 2 8 9 3
```

Q2 Write a function that takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

Example 1:

Input:

LinkedList:

11->11->11->21->43->43->60

Output:

11->21->43->60

Example 2:

Input:

LinkedList:

10->12->12->25->25->25->34

Output:

10->12->25->34

In [27]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

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

    current = head
    nextDistinct = head.next

    while current is not None:
        if current.data == nextDistinct.data:
            current.next = nextDistinct.next
        else:
            current = nextDistinct
            nextDistinct = nextDistinct.next

    return head
```

In [ ]:
```
# Example 1
list1 = Node(11)
list1.next = Node(11)
list1.next.next = Node(11)
list1.next.next.next = Node(21)
list1.next.next.next.next = Node(43)
list1.next.next.next.next.next = Node(43)
list1.next.next.next.next.next.next = Node(60)

new_list = remove_duplicates(list1)

current = new_list
```

```
while current is not None:
    print(current.data, end=" ")
    current = current.next

# Example 2
list2 = Node(10)
list2.next = Node(12)
list2.next.next = Node(12)
list2.next.next.next = Node(25)
list2.next.next.next.next = Node(25)
list2.next.next.next.next.next = Node(25)
list2.next.next.next.next.next.next = Node(34)

new_list = remove_duplicates(list2)

current = new_list
while current is not None:
    print(current.data, end=" ")
    current = current.next
```

Q3 Given a linked list of size **N**. The task is to reverse every **k** nodes (where k is an input to the function) in the linked list. If the number of nodes is not a multiple of *k* then left-out nodes, in the end, should be considered as a group and must be reversed (See Example 2 for clarification).

Example 1:

Input:

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

K = 4

Output:4 2 2 1 8 7 6 5

Explanation:

The first 4 elements 1,2,2,4 are reversed first and then the next 4 elements 5,6,7,8. Hence, the resultant linked list is 4->2->2->1->8->7->6->5.

Example 2:

Input:

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

K = 3

Output:3 2 1 5 4

Explanation: The first 3 elements are 1,2,3 are reversed first and then elements 4,5 are reversed.Hence, the resultant linked list is 3->2->1->5->4.

In [28]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def reverse_k_nodes(head, k):
    if head is None or head.next is None or k == 1:
        return head

    dummy = Node(0)
    dummy.next = head
    prev = dummy
    current = head
    count = 0

    while current is not None:
        count += 1
        if count % k == 0:
            prev = reverse_group(prev, current.next)
            current = prev.next
        else:
            current = current.next

    return dummy.next
```

```
def reverse_group(prev, next):
    last = prev.next
    current = last.next

    while current != next:
        last.next = current.next
        current.next = prev.next
        prev.next = current
        current = last.next

    return last
```

In [29]:
```
# Example 1
list1 = Node(1)
list1.next = Node(2)
list1.next.next = Node(2)
list1.next.next.next = Node(4)
list1.next.next.next.next = Node(5)
list1.next.next.next.next.next = Node(6)
list1.next.next.next.next.next.next = Node(7)
list1.next.next.next.next.next.next.next = Node(8)

k = 4

new_list = reverse_k_nodes(list1, k)

current = new_list
while current is not None:
    print(current.data, end=" ")
    current = current.next
```

```
4 2 2 1 8 7 6 5
```

In [30]:
```
# Example 2
list2 = Node(1)
list2.next = Node(2)
list2.next.next = Node(3)
list2.next.next.next = Node(4)
list2.next.next.next.next = Node(5)

k = 3

new_list = reverse_k_nodes(list2, k)

current = new_list
while current is not None:
    print(current.data, end=" ")
    current = current.next
```

```
3 2 1 4 5
```

Q4 Given a linked list, write a function to reverse every alternate k nodes (where k is an input to the function) in an efficient way. Give the complexity of your algorithm.

Example:

Inputs: 1->2->3->4->5->6->7->8->9->NULL and k = 3

Output: 3->2->1->4->5->6->9->8->7->NULL.

In [31]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


def reverse_alternate_k_nodes(head, k):
    current = head
    next = None
    prev = None
    count = 0

    while current is not None and count < k:
        next = current.next
        current.next = prev
        prev = current
        current = next
        count += 1

    if head is not None:
```

```
        head.next = current

    count = 0
    while count < k - 1 and current is not None:
        current = current.next
        count += 1

    if current is not None:
        current.next = reverse_alternate_k_nodes(current.next, k)

    return prev

def print_linked_list(head):
    while head is not None:
        print(head.data, end=" ")
        head = head.next
    print()
```

In [32]:
```
# Create a sample linked list
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)

print("Original Linked List:")
print_linked_list(head)

k = 3
head = reverse_alternate_k_nodes(head, k)

print("Reversed Linked List:")
print_linked_list(head)
```

```
Original Linked List:
1 2 3 4 5 6 7 8 9
Reversed Linked List:
3 2 1 4 5 6 9 8 7
```

Q5 Given a linked list and a key to be deleted. Delete last occurrence of key from linked. The list may have duplicates.

Examples:

Input: 1->2->3->5->2->10, key = 2

Output: 1->2->3->5->10

In [33]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


def delete_last_occurrence(head, key):
    prev = None
    lastOccur = None
    toDelete = None
    current = head

    while current is not None:
        if current.data == key:
            lastOccur = prev
            toDelete = current

        prev = current
        current = current.next

    if lastOccur is None:
        return head
    elif toDelete == head:
        head = head.next
    else:
        lastOccur.next = toDelete.next

    return head
```

In [34]:
```
def print_linked_list(head):
```

In [34]:
```
def print_linked_list(head):
    while head is not None:
        print(head.data, end=" ")
        head = head.next
    print()
```

In [35]:
```
head = Node(1)
head.next = Node(2)
head.next.next = Node(3)
head.next.next.next = Node(5)
head.next.next.next.next = Node(2)
head.next.next.next.next.next = Node(10)

print("Original Linked List:")
print_linked_list(head)

key = 2
head = delete_last_occurrence(head, key)

print("Modified Linked List:")
print_linked_list(head)
```

```
Original Linked List:
1 2 3 5 2 10
Modified Linked List:
1 2 3 5 10
```

Q6 Given two sorted linked lists consisting of **N** and **M** nodes respectively. The task is to merge both of the lists (in place) and return the head of the merged list.

Examples:

Input: a: 5->10->15, b: 2->3->20

Output: 2->3->5->10->15->20

Input: a: 1->1, b: 2->4

Output: 1->1->2->4

In [36]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


def merge_sorted_lists(head1, head2):
    dummy = Node(0)
    mergePtr = dummy

    while head1 is not None and head2 is not None:
        if head1.data <= head2.data:
            mergePtr.next = head1
            head1 = head1.next
        else:
            mergePtr.next = head2
            head2 = head2.next
        mergePtr = mergePtr.next

    if head1 is not None:
        mergePtr.next = head1
    if head2 is not None:
        mergePtr.next = head2

    return dummy.next
```

In [37]:
```
def print_linked_list(head):
    while head is not None:
        print(head.data, end=" ")
        head = head.next
    print()
```

In [38]:
```
a = Node(5)
a.next = Node(10)
a.next.next = Node(15)

b = Node(2)
b.next = Node(3)
b.next.next = Node(20)
```

```
print("First Linked List:")
print_linked_list(a)

print("Second Linked List:")
print_linked_list(b)

merged = merge_sorted_lists(a, b)

print("Merged Linked List:")
print_linked_list(merged)
```

```
First Linked List:
5 10 15
Second Linked List:
2 3 20
Merged Linked List:
2 3 5 10 15 20
```

Q7 Given a **Doubly Linked List**, the task is to reverse the given Doubly Linked List.

Example:

Original Linked list 10 8 4 2

Reversed Linked list 2 4 8 10

In [39]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None


def reverse_doubly_linked_list(head):
    current = head
    temp = None

    while current is not None:
        temp = current.prev
        current.prev = current.next
        current.next = temp
        current = current.prev

    if temp is not None:
        head = temp.prev

    return head
```

In [40]:
```
def print_doubly_linked_list(head):
    while head is not None:
        print(head.data, end=" ")
        head = head.next
    print()
```

In [41]:
```
head = Node(10)
head.next = Node(8)
head.next.prev = head
head.next.next = Node(4)
head.next.next.prev = head.next
head.next.next.next = Node(2)
head.next.next.next.prev = head.next.next

print("Original Doubly Linked List:")
print_doubly_linked_list(head)

head = reverse_doubly_linked_list(head)

print("Reversed Doubly Linked List:")
print_doubly_linked_list(head)
```

```
Original Doubly Linked List:
10 8 4 2
Reversed Doubly Linked List:
2 4 8 10
```

Q8 Given a doubly linked list and a position. The task is to delete a node from given position in a doubly linked list.

Example 1:

Input:

LinkedList = 1 <--> 3 <--> 4

x = 3

Output:1 3

Explanation:After deleting the node at position 3 (position starts from 1), the linked list will be now as 1->3.

Example 2:

Input:

LinkedList = 1 <--> 5 <--> 2 <--> 9

x = 1

Output:5 2 9

In [42]:
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None


def delete_node_at_position(head, position):
    if position < 1:
        return head

    current = head
    count = 1

    while current is not None and count < position:
        current = current.next
        count += 1

    if current is None:
        return head

    if current.prev is not None:
        current.prev.next = current.next
    else:
        head = current.next

    if current.next is not None:
        current.next.prev = current.prev

    return head
```

In [43]:
```
def print_doubly_linked_list(head):
    while head is not None:
        print(head.data, end=" ")
        head = head.next
    print()
```

In [44]:
```
head = Node(1)
head.next = Node(5)
head.next.prev = head
head.next.next = Node(2)
head.next.next.prev = head.next
head.next.next.next = Node(9)
head.next.next.next.prev = head.next.next

print("Original Doubly Linked List:")
print_doubly_linked_list(head)

position = 1
head = delete_node_at_position(head, position)

print("Modified Doubly Linked List:")
print_doubly_linked_list(head)
```

```
Original Doubly Linked List:
1 5 2 9
Modified Doubly Linked List:
5 2 9
```

In [ ]: