linked-list-assignment-2-solution

August 13, 2024

[1]:

```
"Mar_5^24_Assignment_17_Linked_List_Assignment_2_Solution"
```

[1]:

```
'Mar_5^24_Assignment_17_Linked_List_Assignment_2_Solution'
```

[18]:

```
#1.Define a doubly linked list[ Will be done in the class ]
class Node:
    def __init__(self,data=None,next=None,prev=None):
        self.data=data
        self.next=next
        self.prev=prev
class DoublyLinkedList:
    def __init__(self):
        self.head=None
        self.tail=None
    def addNode(self,data):
        newNode=Node(data)
        if self.head is None:
            self.head=newNode
            self.tail=newNode
        else:
            newNode.prev=self.tail
            self.tail.next=newNode
            self.tail=newNode
    def traverse(self):
        temp=self.head
        while(temp):
            print(temp.data,end="->")
            temp=temp.next
dll=DoublyLinkedList()
dll.addNode(1)
dll.addNode(2)
dll.addNode(3)
dll.addNode(4)
dll.addNode(5)
dll.traverse()
```

```
1->2->3->4->5->
```

[26]:

```
#2.Write a function to reverse a linked list in-place.
class Node:
    def __init__(self,data=None,next=None):
        self.data = data
        self.next = next
def reverseLL(head):
    curr=head
    prev=None
    while curr is not None:
        next_node=curr.next
        curr.next=prev
        prev=curr
        curr=next_node
    head=prev
    return head
def traverse(head):
    temp=head
    while temp:
        print(temp.data,end="->")
        temp=temp.next
#Create a linked list>>Collection of link Nodes
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
node5=Node(5)
#Create the linkage
head.next=node2
node2.next=node3
node3.next=node4
node4.next=node5
traverse(head)
print()
rev_head=reverseLL(head)
traverse(rev_head)
```

```
1->2->3->4->5->
5->4->3->2->1->
```

[31]:

```
#3.Detect cicle in a linked list.
class Node:
    def __init__(self,data=None,next=None):
        self.data = data
        self.next = next
def isCyclePresent(head):
    slow=head
    fast=head
    while(fast and fast.next):
        slow=slow.next
        fast=fast.next.next
        if(fast and slow.data==fast.data):
            return True
    return False
#Create a linked list>>Collection of link Nodes
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
node5=Node(5)
#Create the linkage
head.next=node2
node2.next=node3
node3.next=node4
node4.next=node5
node5.next=node2
print("Is cycle present:",isCyclePresent(head))
```

```
Is cycle present: True
```

[17]:

```
'''4. Merge two sorted linked list into one
1->3->5->7->null and 2->4->6->8->null should be merged to make
1->2->3->4->5->6->7->8'''
class Node:
    def __init__(self,data=None,next=None):
        self.data = data
        self.next = next
def merge_sortedLL(head1,head2):
    dummy=Node()
    tail=dummy
    while head1 and head2:
        if head1.data<=head2.data:
            tail.next=head1
            head1=head1.next
        else:
            tail.next=head2
            head2=head2.next
        tail=tail.next
    if head1:
        tail.next=head1
    else:
        tail.next=head2
    return dummy.next
def traverse(head):
    temp=head
    while(temp):
        print(temp.data,end="->")
        temp=temp.next
#LL1
head1=Node(1)
head1.next=Node(3)
head1.next.next=Node(5)
head1.next.next.next=Node(7)
#LL2
head2=Node(2)
head2.next=Node(4)
head2.next.next=Node(6)
head2.next.next.next=Node(8)
print("Linked list 1:")
traverse(head1)
print()
print("Linked list 2:")
traverse(head2)
print()
merged_head=merge_sortedLL(head1,head2)
print("Merged Sorted Linked List:")
traverse(merged_head)
```

```
Linked list 1:
1->3->5->7->
Linked list 2:
2->4->6->8->
Merged Sorted Linked List:
1->2->3->4->5->6->7->8->
```

[15]:

```
'''5.Write a function to remove nth node from the end in a linked list
1->2->3->4->5->6, removing 2nd node from end will return 1->2->3->4->6'''
class Node:
    def __init__(self,data=None,next=None):
        self.data=data
        self.next=next
#Find the length of the linked list
def length(head):
    lenLL=0
    while head:
        lenLL+=1
        head=head.next
    return lenLL

def remove_nth_from_end(head,n):
    prev=None
    curr=head
    #Calculate the position of the node to be removed from the beginning␣
 ↪(length-n)
    for i in range(length(head)-n):
        #Traverse the list to the node just before the node to be removed
        prev=curr
        curr=curr.next
    #Update pointers to skip the node to be removed
    if prev:
        prev.next=curr.next
    else:
        head=curr.next
    return head #Return the updated head of the linked list
def traverse(head):
    temp=head
    while(temp):
        print(temp.data,end="->")
        temp=temp.next
#Create a linked list>>Collection of link Nodes
head=Node(1)
node2=Node(2)
node3=Node(3)
node4=Node(4)
node5=Node(5)
node6=Node(6)
#Create the linkage
head.next=node2
node2.next=node3
node3.next=node4
node4.next=node5
node5.next=node6
print("Input LL:")
traverse(head)
print()
print("Resulting LL:")
r_head=remove_nth_from_end(head,2)
traverse(r_head)
```

```
Input LL:
1->2->3->4->5->6->
Resulting LL:
1->2->3->4->6->
```

[7]:

```
'''6.Remove duplicates from a sorted linked list
1->2->3->3->4->4->4->5  should be changed to 1->2->3->4->5'''
class ListNode:
    def __init__(self,val=None,next=None):
        self.val=val
```

```python
        self.next=next
def remove_duplicates(head):
    if not head:
        return head
    #Initialize the current
    curr=head
    #Check if any other element in the list
    while(curr.next):
        if curr.val==curr.next.val:
            curr.next=curr.next.next
        else:
            curr=curr.next
    return head
def traverse(head):
    temp=head
    while(temp):
        print(temp.val,end="->")
        temp=temp.next
#Input LL
head=ListNode(1)
head.next=ListNode(2)
head.next.next=ListNode(3)
head.next.next.next=ListNode(3)
head.next.next.next.next=ListNode(4)
head.next.next.next.next.next=ListNode(4)
head.next.next.next.next.next.next=ListNode(4)
head.next.next.next.next.next.next.next=ListNode(5)
print("Input LL:")
traverse(head)
print()
r_head=remove_duplicates(head)
print("Output LL:")
traverse(r_head)
```

```
Input LL:
1->2->3->3->4->4->4->5->
Output LL:
1->2->3->4->5->
```

[33]:

```python
'''7.Find the intersection of the two linked lists
1->2->3->4->8->6->9  5->1->6->7  , intersection 1->6'''
class ListNode:
    def __init__(self,val=None,next=None):
        self.val=val
        self.next=next
def Find_intersection(head1,head2):
    if not head1 or not head2:
        return None

    curr1,curr2=head1,head2
    len1,len2=0,0
    while(curr1):
        len1+=1
        curr1=curr1.next
    while(curr2):
        len2+=1
        curr2=curr2.next
    while(len1>len2):
        head1=head1.next
        len1-=1
    while(len1<len2):
        head2=head2.next
        len2-=1
    while(head1!=head2):
        head1=head1.next
        head2=head2.next
    return head1
def traverse(head):
    temp=head
    while(temp):
        print(temp.val,end="->")
        temp=temp.next
#(Create LL1 the collection of ListNode 1)
head1=ListNode(1)
head1.next=ListNode(2)
head1.next.next=ListNode(3)
head1.next.next.next=ListNode(4)
head1.next.next.next.next=ListNode(8)
head1.next.next.next.next.next=ListNode(6)
head1.next.next.next.next.next.next=ListNode(9)
print("LinkedList1:")
traverse(head1)
print()
#Create LL2 the collection of ListNode 1
head2=ListNode(5)
head2.next=head1
head2.next.next=head1.next.next.next.next.next
head2.next.next.next=ListNode(7)
print("LinkeList2:")
traverse(head2)
print()
head2.next.next.next=None
print("Intersecting Nodes:")
traverse(Find_intersection(head1,head2))
```

```
#Sir I have Tried Multiple ways, but Required Output has not come...But this is␣
 ↪what i did, sorry if not right way!!!
```

```
LinkedList1:
1->2->3->4->8->6->9->
LinkeList2:
5->1->6->7->
Intersecting Nodes:
1->6->
```

[48]:

```
'''8.Rotate a linked list by k positions to the right
1->2->3->4->8->6->9 , after rotating for 2 times becomes ,␣
 ↪3->4->8->6->9->1->2'''
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

def rotateLeft(head, k):
    if not head or not head.next or k == 0:
        return head

    # Find the length of the linked list
    length = 1
    tail = head
    while tail.next:
        tail = tail.next
        length += 1

    # Calculate the actual rotation index
    rotation_index = k % length
    if rotation_index == 0:
        return head

    # Find the node before the new head
    new_head_index = rotation_index - 1
    new_head = head
    for i in range(new_head_index):
        new_head = new_head.next

    # Perform the rotation
    new_tail = new_head
    while new_tail.next:
        new_tail = new_tail.next
    new_tail.next = head
    head = new_head.next
    new_head.next = None
    return head

def traverse(head):
    temp = head
    while temp:
        print(temp.data, end="->")
        temp = temp.next

#Create the linked list
#Create the 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(8)
head.next.next.next.next.next = Node(6)
head.next.next.next.next.next.next = Node(9)

#Print the initial linked list
print("Initial Linked List:")
traverse(head)
print()
#Rotate the linked list to the left
k = 2
rotated_head = rotateLeft(head, k)
#Print the rotated linked list
print("Rotated Linked List:")
traverse(rotated_head)
```

```
Initial Linked List:
1->2->3->4->8->6->9->
Rotated Linked List:
3->4->8->6->9->1->2->
```

[56]:

```
'''9.Add Two Numbers Represented by LinkedLists:
Given two non-empty linked lists representing two non-negative integers, where␣
 ↪the digits are stored in
reverse order, add the two numbers and return it as a linked list.'''
class Node:
    def __init__(self, data=None, next=None):
        self.data = data
        self.next = next

def addTwoNumbers(head1, head2):
    dummy = Node()
    current = dummy
    carry = 0

    while head1 or head2 or carry:
        sum = carry

        if head1:
            sum += head1.data
            head1 = head1.next

        if head2:
            sum += head2.data
            head2 = head2.next

        carry = sum // 10
        current.next = Node(sum % 10)
        current = current.next

    return dummy.next
def traverse(head):
    temp = head
    while temp:
        print(temp.data, end="->")
        temp = temp.next

#Creating the first linked list
head1 = Node(2)
node2 = Node(4)
node3 = Node(3)
head1.next = node2
node2.next = node3
print("Linked list1:")
traverse(head1)
print()
#Creating the second linked list
head2 = Node(5)
node4 = Node(6)
node5 = Node(4)
head2.next = node4
node4.next = node5
print("Linked list2:")
traverse(head2)
print()
#Calling the addTwoNumbers function
result = addTwoNumbers(head1, head2)
print("Sum of Linked list1 and list2:")
traverse(result)
print()
```

```
Linked list1:
2->4->3->
Linked list2:
5->6->4->
Sum of Linked list1 and list2:
7->0->8->
```

[63]:

```
'''10.Clone a Linked List with next and Random Pointer
Given a linked list of size N where each node has two links: one pointer points␣
↪to the next node and the
second pointer points to any node in the list.The task is to create a clone of␣
↪this linked list in O(N) time.
Note: The pointer pointing to the next node is ‘next‘ pointer and the one␣
↪pointing to an arbitrary node is
called ‘arbit’ pointer as it can point to any arbitrary node in the linked list.
↪'''

class Node:
    def __init__(self, data=None,next=None):
        self.data = data
        self.next = next
        self.random = None

#Function to clone a linked list with next and random pointers
def cloneLinkedList(head):
    if not head:
        return None

    #Create a hashmap to store the mapping between original and cloned nodes
    node_map = {}

    #Create a new head node for the cloned list
    cloned_head = Node(head.data)
    node_map[head] = cloned_head

    #Traverse the original list
    curr = head
    cloned_curr = cloned_head

    while curr:
        #Clone the next pointer
        if curr.next:
            if curr.next not in node_map:
                node_map[curr.next] = Node(curr.next.data)
            cloned_curr.next = node_map[curr.next]

        #Clone the random pointer
        if curr.random:
            if curr.random not in node_map:
                node_map[curr.random] = Node(curr.random.data)
            cloned_curr.random = node_map[curr.random]

        #Move to the next node
        curr = curr.next
        cloned_curr = cloned_curr.next

    return cloned_head
#Create the original linked list
node1 = Node(1)
node2 = Node(2)
node3 = Node(3)
node4 = Node(4)

node1.next = node2
node2.next = node3
node3.next = node4

node1.random = node3
node2.random = node1
node3.random = node4
node4.random = node2

#Clone the linked list
cloned_head = cloneLinkedList(node1)

#Print the original and cloned linked lists
print("Original Linked List:")
curr = node1
while curr:
    print("Data:", curr.data, "Next:", curr.next.data if curr.next else None,␣
  ↪"Random:", curr.random.data if curr.random else None)
    curr = curr.next

print("\nCloned Linked List:")
cloned_curr = cloned_head
while cloned_curr:
    print("Data:", cloned_curr.data, "Next:", cloned_curr.next.data if␣
  ↪cloned_curr.next else None, "Random:", cloned_curr.random.data if␣
  ↪cloned_curr.random else None)
    cloned_curr = cloned_curr.next
```

```
Original Linked List:
Data: 1 Next: 2 Random: 3
Data: 2 Next: 3 Random: 1
```

```
Data: 3 Next: 4 Random: 4
Data: 4 Next: None Random: 2

Cloned Linked List:
Data: 1 Next: 2 Random: 3
Data: 2 Next: 3 Random: 1
Data: 3 Next: 4 Random: 4
Data: 4 Next: None Random: 2
```

[ ]: