EX - 10 LINKED IMPLEMENTATION OF LIST, STACK AND QUEUE

T SADAKOPA RAMAKRISHNAN | IT - B | 3122225002109

fQ1) Design and implement Linked List with the following operations.

- Isempty
- Display
- Find
- append
- Insert by pos
- Delete by pos
- Insert by specifying the previous value.
- Delete by specifying the previous value.

```
class Node:
    __slots__ = ['item', 'next']
    def __init__(self, item = None, next = None):
        self.item = item
        self.next = next


from node import Node

class SLL:
    def __init__(self):
        self.head = self.tail = Node()
        self.size = 0

    def isempty(self):
        return (self.head == self.tail)

    def append(self, val):
        temp = Node(val)
        self.tail.next = temp
        self.tail = temp
        self.size += 1

    def display(self):
```

```
    pos = self.head.next
    while (pos != None):
        print(str(pos.item) + "->", end = " ")
        pos = pos.next
    print("END")

def insertAtFirst(self, val):
    temp = Node(val)
    temp.next = self.head.next
    self.head.next = temp
    self.size += 1

def insert(self, index, val):
    pos = self.head.next
    for _ in range(index - 1):
        pos = pos.next
    temp = Node(val)
    temp.next = pos.next
    pos.next = temp
    self.size += 1

def find_prev(self,sr):
    if (self.isempty()):
        print("Empty")
    pos = self.head.next
    while (pos != None):
        if (pos.next.item == sr):
            return pos
        else:
            pos = pos.next
    return None

def remove(self, ele):
    if self.isempty():
        print("Empty")
    else:
        if ele == self.head.next.item:
            delnode = self.head.next
```

```
            self.head.next = delnode.next
            self.size -= 1
        else:
          prev = self.find_prev(ele)
        if prev is None:
            print("Element not present")
            delnode = None
        else:
            delnode = prev.next
            prev.next = delnode.next
            self.size -= 1
    return delnode

def __str__(self):
    out = ''
    pos = self.head.next
    while (pos != None):
        out += str(pos.item) + " -> "
        pos = pos.next
    out += "END"
    return out

def __len__(self):
    return self.size

def removeatfirst(self):
    remnode = self.head.next
    self.head.next = remnode.next
    if self.head == None:
        self.tail = None
    self.size -= 1

def removeatlast(self):
    if self.size <= 1:
        self.removeatfirst()
    pos = self.head
    for i in range(self.size-1):
        pos = pos.next
```

```
        pos.next = None
        self.tail = pos

    def find(self, ele):
        pos = self.head.next
        while (pos != None):
            if pos.item == ele:
                return True
            pos = pos.next
        return False

    def removebypos(self,pos):
        if pos < 0 or pos >= self.size:
            print("Invalid Position")
            return

        curr_pos = 0
        curr_node = self.head.next
        prev = self.head

        while curr_node != None and curr_pos < pos:
            prev = curr_node
            curr_node = curr_node.next
            curr_pos += 1

        if curr_node == None:
            print("Invalid Position")
            return

        prev.next = curr_node.next
        self.size -= 1

if __name__ == "__main__":
    s = SLL()
    s.append(3)
    s.append(4)
    s.append(2)
    s.append(1)
```

```
s.append(10)
s.insertAtFirst(100)
s.insert(3,50)

print("Original:")
s.display()
print()

print("Removing at first:")
s.removeatfirst()
s.display()
print()

print("Removing at last:")
s.removeatlast()
s.display()
print()

print("Removing 50")
s.remove(50)
s.display()
print()

print("Printing using print statement:",s)
print()
print("Length of linked list is:", len(s))
print()

print(s.find(4))

s.removebypos(1)
s.display()
```

```
from node import Node

class DNode(Node):
    __slots__ = ['prev']
    def __init__(self,item, next, prev = None):
        super().__init__(item, next)
        self.prev = prev

from dnode import DNode

class DoublyLinkedList:
    def __init__(self):
        self.head = self.tail = DNode(None, None, None)
        self.size = 0

    def isempty(self):
        return (self.head == self.tail)

    def append(self, ele):
        temp = DNode(ele, None)
        self.tail.next = temp
        temp.prev = self.tail
        self.tail = temp
        self.size += 1

    def display(self):
        pos = self.head.next
        while (pos != None):
            if pos.item != None:
                print(pos.item, end = " ")
                pos = pos.next

    def reverse_display(self):
        pos = self.tail
        while (pos != None):
            if pos.item == None:
                break
            print(pos.item, end = " ")
```

```
        pos = pos.prev

def find(self, ele):
    pos = self.head.next
    while (pos != None):
        if pos.item == ele:
            return pos
    return None

def insert(self, position, ele):
    pos = self.head
    for i in range(position - 1):
        pos = pos.next
    temp = DNode(ele, None, None)

    if pos.next == None:
        self.append(ele)
        return self.display()
    else:
        temp.prev = pos
        temp.next = pos.next
        pos.next.prev = temp
        pos.next = temp
    self.size += 1
    return self.display()

def find_prev(self, data):
    pos = self.head.next
    while (pos.next != None):
        if (pos.next.item == data):
            return pos
        else:
            pos = pos.next
    return None
```

```
D = DoublyLinkedList()
#appending
D.append(10)
D.append(20)
D.append(30)
D.append(40)
D.append(50)

#displaying initial linked list
print("Initial linked list:")
D.display()
print()

#Reversed linked list
print("Reverse Linked List:")
D.reverse_display()
print()

#inserting 40
print("Insert 40:")
print(D.insert(4,100))
print()

#Removing 40
print("Removing 40")
print(D.remove(40))
print()

#checking if empty
print("Check if empty")
print(D.isempty())
print()
```

Q2) Design and implement Linked stack and Queue ADT with various operations

```
class Node:
    __slots__ = ['item', 'next']
    def __init__(self, item = None, next = None):
        self.item = item
        self.next = next

class LinkedStack:
    def __init__(self):
        self.top = Node()
        self.size = 0

    def __len__(self):
        return self.size

    def isempty(self):
        return self.top.next == None

    def top(self):
        if self.isempty():
            raise Empty("Stack Empty")
        return self.top.next.item

    def push(self, ele):
        self.top = Node(ele, self.top)
        self.size += 1

    def __str__(self):
        out = ''
        pos = self.top
        while (pos.next != None):
            out += str(pos.item) + " -> "
            pos = pos.next
        out += "END"
        return out

    def pop(self):
        delnode = self.top
```

```
        self.top = delnode.next
        self.size -= 1
        return delnode


class Empty(Exception):
    pass

s = LinkedStack()
s.push(1)
s.push(2)
s.push(3)
s.push(4)
s.push(5)
print(s)
s.pop()
print(s)

class Node:
    __slots__ = ['item', 'next']
    def __init__(self, item = None, next = None):
        self.item = item
        self.next = next

class LinkedQueue:
    __slots__ = ['front', 'rear','size']
    def __init__(self):
        self.front = self.rear = Node()
        self.size = 0

    def __len__(self):
        return self.size

    def isempty(self):
        return self.front.next == None

    def enqueue(self, ele):
        self.rear.next = Node(ele)
```

```
        self.rear = self.rear.next
        self.size += 1

    def dequeue(self):
        if self.isempty():
            raise Empty("Queue is empty")
        ele = self.front.next.item
        self.front = self.front.next
        self.size -= 1
        return ele

    def __str__(self):
        out = ''
        pos = self.front.next
        while (pos != None):
            out += str(pos.item) + " -> "
            pos = pos.next
        out += "END"
        return out

class Empty(Exception):
    pass

q = LinkedQueue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
q.enqueue(4)
q.enqueue(5)
print(q)
print(q.dequeue())
print(q)
```