## Data Structure Lab6 : Doubly Linked List 2022-2023 Topics

## 1. Implement Node Class 2. Implement DoublyLinkedList Class 3. Implement Basic Methods of DoublyLinkedList ● isEmpty() ● size() ● first() ● last() ● addFirst() ● addLast() ● removeFirst() ● removeLast()

## Homework

## 1. Describe a method for finding the middle node of a doubly linked list with header and trailer sentinels by “link hopping,” and without relying on explicit knowledge of the size of the list. In the case of an even number of nodes, report the node slightly left of center as the “middle.”

## 2. Give an implementation of the size( ) method for the DoublyLinkedList class, assuming that we did not maintain size as an instance variable.

## 3. Implement the equals( ) method for the DoublyLinkedList class. 4. Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.

## 5. Our implementation of a doubly linked list relies on two sentinel nodes, header and trailer, but a single sentinel node that guards both ends of the list should suffice. Reimplement the DoublyLinkedList class using only one sentinel node.Data Structure Lab6 : Doubly Linked List 2022-2023

6. Implement a circular version of a doubly linked list, without any sentinels, that supports all the public behaviors of the original as well as two new update methods, rotate( ) and rotateBackward. 7. Implement the clone( ) method for the DoublyLinkedList class.

**Solution**

Class Node {

Int data;

Node prev;

Node next;

Node(int data) {

This.data = data;

This.prev = null;

This.next = null;

}

}

Class DoublyLinkedList {

Private Node head;

Private Node tail;

DoublyLinkedList() {

Head = null;

Tail = null;

}

Boolean isEmpty() {

Return head == null;

}

Int size() {

Int count = 0;

Node current = head;

While (current != null) {

Count++;

Current = current.next;

}

Return count;

}

Void addFirst(int data) {

Node newNode = new Node(data);

If (isEmpty()) {

Head = newNode;

Tail = newNode;

} else {

newNode.next = head;

head.prev = newNode;

head = newNode;

}

}

Void addLast(int data) {

Node newNode = new Node(data);

If (isEmpty()) {

Head = newNode;

Tail = newNode;

} else {

Tail.next = newNode;

newNode.prev = tail;

tail = newNode;

}

}

Int removeFirst() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Int removedData = head.data;

If (head == tail) {

Head = null;

Tail = null;

} else {

Node newHead = head.next;

newHead.prev = null;

head.next = null;

head = newHead;

}

Return removedData;

}

Int removeLast() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Int removedData = tail.data;

If (head == tail) {

Head = null;

Tail = null;

} else {

Node newTail = tail.prev;

newTail.next = null;

tail.prev = null;

tail = newTail;

}

Return removedData;

}

Int getFirst() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Return head.data;

}

Int getLast() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Return tail.data;

}

}