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.”

```
public class DoublyLinkedList {
    private Node header;
    private Node trailer;

    public Node findMiddle() {
        Node slow = header.next;
        Node fast = header.next;

        while (fast != trailer && fast.next != trailer) {
            slow = slow.next;
            fast = fast.next.next;
        }

        return slow;
    }


    private class Node {
        private int data;
        private Node prev;
        private Node next;

        public Node(int data) {
            this.data = data;
            this.prev = null;
            this.next = null;
        }
    }

}
```

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

```
public class DoublyLinkedList {
    private Node header;
    private Node trailer;




    public int size() {
        int counter = 0;
        Node cur = header.next;

        while (cur != trailer) {
            counter++;
            cur = cur.next;
        }
  return counter;
    }
}
```

3. Implement the equals( ) method for the DoublyLinkedList class.

```
public class DoublyLinkedList {
    private Node header;
    private Node trailer;



    public boolean equals(DoublyLinkedList otherList) {
        if (this.size() != otherList.size()) {
            return false;
        }

        Node curThis = header.next;
        Node curOther = otherList.header.next;

        while (curThis != trailer) {
            if (curThis.data != curOther.data) {
                return false;
            }

            curThis = curThis.next;
            curOther = curOther.next;
        }

        return true;
    }

}
```

4. Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.

```
public class DoublyLinkedList {
    private Node header;
    private Node trailer;



    public DoublyLinkedList concatenate(DoublyLinkedList listx) {
        DoublyLinkedList concatenatedList = new DoublyLinkedList();


        concatenatedList.header = this.header;
        concatenatedList.trailer = this.trailer;


        this.trailer.prev.next = listM.header.next;
        listM.header.next.prev = this.trailer;


        listx.trailer.prev.next = concatenatedList.trailer;
        concatenatedList.trailer.prev = listM.trailer.prev;


        Node cur = listx.header.next;
        while (cur != listx.trailer) {
            current.prev = concatenatedList.trailer.prev;
            current.next = concatenatedList.trailer;
            concatenatedList.trailer.prev.next = current;
            concatenatedList.trailer.prev = current;
            cur = curr.next;
        }

        return concatenatedList;
    }
}
```

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.

```
public class DoublyLinkedList {
    private Node sentinel;

    private class Node {
        private int data;
        private Node prev;
        private Node next;

        public Node(int data) {
            this.data = data;
            this.prev = null;
            this.next = null;
        }
    }

    public DoublyLinkedList() {
        sentinel = new Node(-1);
        sentinel.prev = sentinel;
        sentinel.next = sentinel;
    }

    public void add(int data) {
        Node newNode = new Node(data);
        newNode.next = sentinel.next;
        newNode.prev = sentinel;
        sentinel.next.prev = newNode;
        sentinel.next = newNode;
    }
    public void remove(int data) {
        Node cur = sentinel.next;
        while (cur != sentinel) {
            if (cur.data == data) {
                cur.prev.next = cur.next;
                cur.next.prev = cur.prev;
                return;
            }
            cur= curr.next;
        }
    }

    public void display() {
        Node cur = sentinel.next;
        while (cur != sentinel) {
            System.out.print(cur.data + " ");
            cur = cur.next;
        }
        System.out.println();
    }
}
```

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.

```
public class CircularDoublyLinkedList {
    private Node head;
    private int size;

    private class Node {
        private int data;
        private Node p;
        private Node n;
        public Node(int data) {
            this.data = data;
            this.p = null;
            this.n = null;
        }
    }

    public CircularDoublyLinkedList() {
        head = null;
        size = 0;
    }

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void add(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            newNode.p = newNode;
            newNode.n = newNode;
            head = newNode;
        } else {
            newNode.p = head.p;
            newNode.n = head;
            head.p.n = newNode;
            head.p = newNode;
        }
        size++;
    }
```

```
public void remove(int data) {
    if (isEmpty()) {
        return;
    }

    Node cur = head;
    while (cur != null) {
        if (cur.data == data) {
            if (size == 1) {
                head = null;
            } else {
                cur.p.n = cur.n;
                cur.n.p = cur.p;
                if (cur == head) {
                    head = cur.n;
                }
            }
            size--;
            return;
        }
        cur = cur.n;
        if (cur == head) {
            break;
        }
    }
}

public void display() {
    if (isEmpty()) {
        return;
    }

    Node cur = head;
```

```
        while (cur != null) {
            System.out.print(cur.data + " ");
            cur = cur.n;
            if (cur == head) {
                break;
            }
        }
        System.out.println();
    }

    public void rotate() {
        if (isEmpty()) {
            return;
        }
        head = head.n;
    }

    public void rotateBackward() {
        if (isEmpty()) {
            return;
        }
        head = head.p;
    }
}
```

7. Implement the clone( ) method for the DoublyLinkedList class.

```
public class DoublyLinkedList {
    private Node header;
    private Node trailer;

    private class Node {
        private int data;
        private Node prev;
        private Node next;
        public Node(int data) {
            this.data = data;
            this.prev = null;
            this.next = null;
        }
    }


    public DoublyLinkedList clone() {
        DoublyLinkedList clonedList = new DoublyLinkedList();

        Node current = header.next;
        Node clonedCurrent = null;
        while (current != trailer) {
            Node newNode = new Node(current.data);
            if (clonedCurrent == null) {
                clonedList.header.next = newNode;
                newNode.prev = clonedList.header;
            } else {
                clonedCurrent.next = newNode;
                newNode.prev = clonedCurrent;
            }
            newNode.next = clonedList.trailer;
            clonedList.trailer.prev = newNode;

            clonedCurrent = newNode;
            current = current.next;
        }

        return clonedList;
    }
}
```