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 static Node findMiddleNode(Node head) {

    if (head.next == head) {

    }


    Node slow = head.next;
    Node fast = head.next.next;


    while (fast != head && fast.next != head) {
    slow = slow.next;
    fast = fast.next.next;
    }
    if (fast == head) {
    return slow;
    } else {
    return slow.next;
    }
    }
```

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<T> {

  private Node<T> head;
  private Node<T> tail;

  public int size() {
    int count = 0;
    Node<T> current = head;


    while (current != null && current != tail) {
      count++;
      current = current.next;
    }

    return count;
  }

}

private class Node<T> {
  T data;
  Node<T> next;
  Node<T> prev;


}
```

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

```
public class DoublyLinkedList<T> {

  private Node<T> head;
  private Node<T> tail;

  @Override
  public boolean equals(Object obj) {
    if (obj == this) {

    }
    if (!(obj instanceof DoublyLinkedList)) {
        DoublyLinkedList<T> other = (DoublyLinkedList<T>) obj;


        if (head == null && other.head == null) {
            return true; // Both empty
        }


        if (size() != other.size()) {
            re
        }


        Node<T> current1 = head;
        Node<T> current2 = other.head;
        while (current1 != null) {
            if (!current1.data.equals(current2.data)) {
                return false;
            }
            current1 = current1.next;
            current2 = current2.next;
        }

        return true;
    }


}

private class Node<T> {
    T data;
    Node<T> next;
    Node<T> prev;

    // ... constructor and other methods of the Node 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′.

```
public static void concatenate(DoublyLinkedList<T> L, DoublyLinkedList<T> M) {

        if (L.isEmpty() || M.isEmpty()) {
        return; // Nothing to concatenate
        }

        Node<T> lastL = L.tail.prev;


        lastL.next = M.head.next;
        M.head.next.prev = lastL;
        M.tail.prev = L.tail;
        L.tail.next = M.tail;
  L.tail = M.tail;
        }
```

Data Structure Lab6 : Doubly Linked List 2022-2023

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<T> {

    private Node<T> sentinel;

    public DoublyLinkedList() {
        sentinel = new Node<>(null);
        sentinel.next = sentinel;
        sentinel.prev = sentinel;
    }

    public boolean isEmpty() {
        return sentinel.next == sentinel;
    }

    public void addFirst(T data) {
        Node<T> newNode = new Node<>(data);
        newNode.next = sentinel.next;
        sentinel.next.prev = newNode;
        newNode.prev = sentinel;
        sentinel.next = newNode;
    }

    public void addLast(T data) {
        Node<T> newNode = new Node<>(data);
        newNode.prev = sentinel.prev;
        sentinel.prev.next = newNode;
        newNode.next = sentinel;
        sentinel.prev = newNode;
    }

    public T removeFirst() {
        if (isEmpty()) {
            throw new NoSuchElementException("List is empty");
        }
        Node<T> removed = sentinel.next;
        sentinel.next = removed.next;
        removed.next.prev = sentinel;
        return removed.data;
    }

    public T removeLast() {
        if (isEmpty()) {
            throw new NoSuchElementException("List is empty");
        }
        Node<T> removed = sentinel.prev;
        sentinel.prev = removed.prev;
        removed.prev.next = sentinel;
        return removed.data;
    }



    private class Node<T> {
        T data;
        Node<T> next;
        Node<T> prev;

        public Node(T data) {
            this.data = data;
        }
    }
}
```

5. 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<T> {

    private Node<T> head;

    public CircularDoublyLinkedList() {
        head = null;
    }

    // ... other methods like isEmpty(), size(), addFirst(), addLast(), removeFirst(), removeLast() as in the
original DoublyLinkedList

    public void rotate() {
        if (isEmpty()) {

        }
        Node<T> last = head.prev;
        last.next = last;
        head = head.next;
        head.prev = last;
    }

    public void rotateBackward() {
        if (isEmpty()) {
        }
        Node<T> last = head.prev;
        last.next = head;
        head.prev = last;
        head = last;
    }

    private class Node<T> {
        T data;
        Node<T> next;
        Node<T> prev;
        public Node(T data) {
            this.data = data;
        }
    }
}
```

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

```
public class DoublyLinkedList<T> implements Cloneable {

    private Node<T> head;
    private Node<T> tail;

    @Override
    public DoublyLinkedList<T> clone() throws CloneNotSupportedException {
        if (!super.cloneSupported()) {
            throw new CloneNotSupportedException("DoublyLinkedList cannot be cloned");
        }

        // Create a new DoublyLinkedList object
        DoublyLinkedList<T> clone = new DoublyLinkedList<>();

        if (head != null) {
            Node<T> current = head;
            Node<T> prevCloneNode = null;
            while (current != null) {
                Node<T> clonedNode = new Node<>(current.data);
                if (prevCloneNode != null) {
                    prevCloneNode.next = clonedNode;
                    clonedNode.prev = prevCloneNode;
                } else {
                    clone.head = clonedNode;
                }
                prevCloneNode = clonedNode;
                current = current.next;
            }
            clone.tail = prevCloneNode;
        }

        return clone;
    }



    private class Node<T> {
        T data;
        Node<T> next;
        Node<T> prev;

        public Node(T data) {
            this.data = data;
        }
    }
}
```