Data Structure Lab5 : Circularly Linked List 20222023

Topics

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

Homework

1. Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.

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

3. Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.

4. Suppose you are given two circularly linked lists, L and M. Describe an algorithm for telling if L and M store the same sequence of elements (but perhaps with different starting points).

Data Structure Lab5 : Circularly Linked List 20222023

5. Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size. 6. Implement the clone( ) method for the CircularlyLinkedList class.

**Solution**

Class Node<T> {

T data;

Node<T> next;

Node(T data) {

This.data = data;

This.next = null;

}

}

Class CircularlyLinkedList<T> {

Private Node<T> tail;

Private int size;

CircularlyLinkedList() {

Tail = null;

Size = 0;

}

Boolean isEmpty() {

Return size == 0;

}

Int size() {

Return size;

}

T first() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Return tail.next.data;

}

T last() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Return tail.data;

}

Void addFirst(T data) {

Node<T> newNode = new Node<>(data);

If (isEmpty()) {

newNode.next = newNode;

tail = newNode;

} else {

newNode.next = tail.next;

tail.next = newNode;

}

Size++;

}

Void addLast(T data) {

addFirst(data);

tail = tail.next;

}

T removeFirst() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Node<T> head = tail.next;

If (head == tail) {

Tail = null;

} else {

Tail.next = head.next;

}

Size--;

Return head.data;

}

Void rotate() {

If (!isEmpty()) {

Tail = tail.next;

}

}

Boolean equals(CircularlyLinkedList<T> otherList) {

If (size() != otherList.size()) {

Return false;

}

Node<T> currentSelf = tail.next;

Node<T> currentOther = otherList.tail.next;

While (currentSelf != tail) {

If (!currentSelf.data.equals(currentOther.data)) {

Return false;

}

currentSelf = currentSelf.next;

currentOther = currentOther.next;

}

Return true;

}

Int calculateSize() {

Int count = 0;

If (!isEmpty()) {

Node<T> current = tail.next;

Do {

Count++;

Current = current.next;

} while (current != tail.next);

}

Return count;

}

Void splitList() {

If (isEmpty() || size() % 2 != 0) {

Throw new UnsupportedOperationException("List is empty or has an odd number of nodes");

}

Int halfSize = size() / 2;

Node<T> current = tail.next;

For (int i = 0; i < halfSize – 1; i++) {

Current = current.next;

}

Node<T> secondHalfTail = current;

Node<T> secondHalfHead = current.next;

secondHalfTail.next = tail.next;

tail.next = secondHalfHead;

tail = secondHalfTail;

}

CircularlyLinkedList<T> clone() {

CircularlyLinkedList<T> clonedList = new CircularlyLinkedList<>();

If (!isEmpty()) {

Node<T> current = tail.next;

Do {

clonedList.addLast(current.data);

current = current.next;

} while (current != tail.next);

}

Return clonedList;

}

}