Topics

* Implement Node Class
* Implement CircularlyLinkedList Class
* Implement Basic Methods of CircularlyLinkedList
* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* rotate()

public class Node<T> {

private T element;

private Node<T> next;

public Node(T element, Node<T> next) {

this.element = element;

this.next = next;

}

public T getElement() {

return element;

}

public void setElement(T element) {

this.element = element;

}

public Node<T> getNext() {

return next;

}

public void setNext(Node<T> next) {

this.next = next;

}

}

public class CircularlyLinkedList<T> {

private Node<T> tail;

private int size;

public CircularlyLinkedList() {

tail = null;

size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public T first() {

if (isEmpty()) {

return null;

}

return tail.getNext().getElement();

}

public T last() {

if (isEmpty()) {

return null;

}

return tail.getElement();

}

public void addFirst(T element) {

if (isEmpty()) {

tail = new Node<>(element, null);

tail.setNext(tail);

} else {

Node<T> newNode = new Node<>(element, tail.getNext());

tail.setNext(newNode);

}

size++;

}

public void addLast(T element) {

addFirst(element);

tail = tail.getNext();

}

public T removeFirst() {

if (isEmpty()) {

return null;

}

Node<T> head = tail.getNext();

if (head == tail) {

tail = null;

} else {

tail.setNext(head.getNext());

}

size--;

return head.getElement();

}

public void rotate() {

if (tail != null) {

tail = tail.getNext();

}

}

}

Homework

* 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.

public void addFirst(T element) {

Node<T> newest = new Node<>(element, null);

if (isEmpty()) {

newest.setNext(newest);

tail = newest;

} else {

newest.setNext(tail.getNext());

tail.setNext(newest);

}

size++;

}

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

public int size() {

if (isEmpty()) {

return 0;

}

int count = 1; // Start with 1 because there is at least one node (the tail)

Node<T> current = tail.getNext(); // Start from the head node

while (current != tail) {

count++;

current = current.getNext();

}

return count;

}

* 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.

public boolean equals(Object o) {

if (o == this) {

return true;

}

if (!(o instanceof CircularlyLinkedList)) {

return false;

}

CircularlyLinkedList other = (CircularlyLinkedList) o;

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

return false;

}

Node current1 = head;

Node current2 = other.head;

while (current1 != head) {

if (current1.data != current2.data) {

return false;

}

current1 = current1.next;

current2 = current2.next;

}

return true;

}

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).

public boolean hasSameSequence(CircularlyLinkedList<T> L, CircularlyLinkedList<T> M) {

if (L.isEmpty() && M.isEmpty()) {

return true;

}

if (L.size() != M.size()) {

return false;

}

Node<T> startNode = L.first();

Node<T> currentL = startNode;

Node<T> currentM = M.first();

do {

if (!currentL.getElement().equals(currentM.getElement())) {

return false;

}

currentL = currentL.getNext();

currentM = currentM.getNext();

} while (currentL != startNode);

return true;

}

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.

public Pair<CircularlyLinkedList<T>, CircularlyLinkedList<T>> splitCircularList(CircularlyLinkedList<T> L) {

if (L.isEmpty()) {

return new Pair<>(null, null);

}

CircularlyLinkedList<T> L1 = new CircularlyLinkedList<>();

CircularlyLinkedList<T> L2 = new CircularlyLinkedList<>();

Node<T> slow = L.first();

Node<T> fast = L.first();

do {

fast = fast.getNext().getNext();

slow = slow.getNext();

} while (fast != L.last() && fast.getNext() != L.last());

L.setTail(slow);

L1.setHead(L.first());

L2.setHead(slow.getNext());

L1.setTail(slow);

L2.setTail(L.last());

L2.getTail().setNext(L2.getHead());

return new Pair<>(L1, L2);

}

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

public CircularlyLinkedList clone() {

if (isEmpty()) {

return new CircularlyLinkedList();

}

CircularlyLinkedList clone = new CircularlyLinkedList();

Node current = head;

do {

clone.addLast(current.data);

current = current.next;

} while (current != head);

return clone;

}