Data Structure Lab5 : Circularly Linked List 2022- 2023

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.

Answer:

public void addFirst(E e) {

if (size == 0) {

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

tail.setNext(tail); // link to itself circularly

} else {

tail.setNext(new Node<>(e, tail.getNext()));

}

size++;

}

2. Give an implementation of the size( ) method for the

CircularlyLinkedList class, assuming that we did not maintain size as an instance variable.

Answer:

public int size() {

if (tail == null) { // If the list is empty

return 0;

}

int count = 1; // Start with 1 for the node after the tail

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

while (current != tail) { // Traverse until we loop back to the tail

count++;

current = current.getNext();

}

return count;

}

1. 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.

Answer:

@Override

public boolean equals(Object obj) {

if (this == obj) { // Check if they are the same object

return true;

}

if (obj == null || !(obj instanceof CircularlyLinkedList<?>)) { // Check if obj is a CircularlyLinkedList

return false;

}

CircularlyLinkedList<?> other = (CircularlyLinkedList<?>) obj;

// Check if the sizes are different

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

return false;

}

// Compare elements one by one

if (this.size() == 0) { // Both lists are empty

return true;

}

Node<E> currentThis = this.tail.getNext(); // Start from head of this list

Node<?> currentOther = other.tail.getNext(); // Start from head of the other list

do {

if (!currentThis.getElement().equals(currentOther.getElement())) { // Compare elements

return false;

}

currentThis = currentThis.getNext();

currentOther = currentOther.getNext();

} while (currentThis != this.tail.getNext()); // Loop until we return to the start

return true; // All elements matched

}

1. 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 static <E> boolean areCircularListsEqual(CircularlyLinkedList<E> L, CircularlyLinkedList<E> M) {

// Step 1: Check for null or empty lists

if (L == null && M == null) return true; // Both lists are null

if (L == null || M == null) return false; // One list is null

if (L.size() == 0 && M.size() == 0) return true; // Both lists are empty

if (L.size() != M.size()) return false; // Sizes are different, cannot be equal

// Step 2: Find a matching starting point in M

Node<E> headL = L.tail.getNext(); // Head of L

Node<E> currentM = M.tail.getNext(); // Start at the head of M

// Traverse M to find a potential starting point

for (int i = 0; i < M.size(); i++) {

if (currentM.getElement().equals(headL.getElement())) {

// Step 3: Compare the sequences

Node<E> tempL = headL;

Node<E> tempM = currentM;

boolean match = true;

for (int j = 0; j < L.size(); j++) {

if (!tempL.getElement().equals(tempM.getElement())) {

match = false;

break;

}

tempL = tempL.getNext();

tempM = tempM.getNext();

}

if (match) return true; // The sequences match

}

currentM = currentM.getNext(); // Move to the next node in M

}

return false; // No matching sequence found

}

Data Structure Lab5 : Circularly Linked List 2022- 2023

1. 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.
2. **Find the midpoint**:

* Use two pointers (slow and fast).
* slow moves one step at a time, and fast moves two steps. Stop when fast completes one full cycle.

1. **Create two lists**:

* L1L1L1: Start from the head of LLL (tail.getNext()) and end at slow.
* L2L2L2: Start from slow.getNext() and end at the original tail.

1. **Adjust links**:

* Set the next of slow to point back to the head of L1L1L1.
* Set the next of the original tail to point back to the head of L2L2L2.

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

@Override

@SuppressWarnings("unchecked")

public CircularlyLinkedList<E> clone() throws CloneNotSupportedException {

// Create a new CircularlyLinkedList

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

// If the original list is empty, return the empty cloned list

if (size == 0) {

return clonedList;

}

// Traverse the original list and copy nodes

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

do {

clonedList.addLast(current.getElement()); // Add each element to the cloned list

current = current.getNext();

} while (current != tail.getNext()); // Stop when we loop back to the start

return clonedList;

}