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 DoublyLinkedListNode<E> findMiddle() {

if (header.getNext() == trailer) {

return null;

}

DoublyLinkedListNode<E> slow = header.getNext();

DoublyLinkedListNode<E> fast = header.getNext();

while (fast != trailer && fast.getNext() != trailer) {

slow = slow.getNext();

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

}

return slow;

}

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

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

public int size() {

int count = 0;

DoublyLinkedListNode<E> current = header.getNext();

while (current != trailer) {

count++;

current = current.getNext();

}

return count;

}

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

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || !(obj instanceof DoublyLinkedList<?>)) {

return false;

}

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

DoublyLinkedListNode<E> currentThis = header.getNext();

DoublyLinkedListNode<?> currentOther = other.header.getNext();

while (currentThis != trailer && currentOther != other.trailer) {

if (!currentThis.getElement().equals(currentOther.getElement())) {

return false;

}

currentThis = currentThis.getNext();

currentOther = currentOther.getNext();

}

return currentThis == trailer && currentOther == other.trailer;

}

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 void concatenate(DoublyLinkedList<E> L, DoublyLinkedList<E> M) {

if (L.header.getNext() == L.trailer) { // If L is empty

L = M;

return;

}

if (M.header.getNext() == M.trailer) { // If M is empty

return;

}

DoublyLinkedListNode<E> lastL = L.trailer.getPrev();

DoublyLinkedListNode<?> firstM = M.header.getNext();

lastL.setNext(firstM);

firstM.setPrev(lastL);

L.trailer.setPrev(M.trailer.getPrev());

M.trailer.getPrev().setNext(L.trailer);

M.header.setNext(M.trailer); // Empty M list after concatenation

M.trailer.setPrev(M.header);

}

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.

DoublyLinkedList<Integer> list = new DoublyLinkedList<>();

list.addFirst(1);

list.addLast(2);

list.addLast(3);

System.out.println("Size: " + list.size()); // Output: 3

System.out.println("First: " + list.getFirst()); // Output: 1

System.out.println("Last: " + list.getLast()); // Output: 3

list.removeFirst();

System.out.println("First after removal: " + list.getFirst()); // Output: 2

Data Structure Lab6 : Doubly Linked List 2022-2023

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<E> {

private DoublyLinkedListNode<E> head;

private int size;

public CircularDoublyLinkedList() {

head = null;

size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public void addFirst(E element) {

DoublyLinkedListNode<E> newNode = new DoublyLinkedListNode<>(element);

if (isEmpty()) {

head = newNode;

head.setNext(head);

head.setPrev(head);

} else {

newNode.setNext(head);

newNode.setPrev(head.getPrev());

head.getPrev().setNext(newNode);

head.setPrev(newNode);

head = newNode;

}

size++;

}

public void rotate() {

if (size > 1) head = head.getNext();

}

public void rotateBackward() {

if (size > 1) head = head.getPrev();

}

class DoublyLinkedListNode<E> {

E element;

DoublyLinkedListNode<E> next, prev;

DoublyLinkedListNode(E element) { this.element = element; }

void setNext(DoublyLinkedListNode<E> next) { this.next = next; }

void setPrev(DoublyLinkedListNode<E> prev) { this.prev = prev; }

}

}

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

public DoublyLinkedList<E> clone() {

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

if (isEmpty()) return clonedList;

DoublyLinkedListNode<E> current = head;

do {

clonedList.addLast(current.getElement());

current = current.getNext();

} while (current != head);

return clonedList;

}