## 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.”

you can use two pointers, one moving at twice the speed of the other. Start both pointers at the header sentinel node. Move the faster pointer two nodes at a time and the slower pointer one node at a time. When the faster pointer reaches the trailer sentinel node, the slower pointer will be at the middle (or slightly left of center in the case of an even number of nodes)

1. 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;

Node<E> current = header.next;

while (current != trailer) {

count++;

current = current.next;

}

return count;

}

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

public boolean equals(Object obj) {

if (obj == this) {

return true;

}

if (!(obj instanceof DoublyLinkedList)) {

return false;

}

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

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

return false;

}

Iterator<E> it1 = this.iterator();

Iterator<?> it2 = other.iterator();

while (it1.hasNext()) {

if (!Objects.equals(it1.next(), it2.next())) {

return false;

}

}

return true;

}

1. 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 <E> DoublyLinkedList<E> concatenate(DoublyLinkedList<E> L, DoublyLinkedList<E> M) {

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

for (E element : L) {

concatenatedList.addLast(element);

}

for (E element : M) {

concatenatedList.addLast(element);

}

return concatenatedList;

}

1. 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<E> {

private Node<E> sentinel;

public DoublyLinkedList() {

sentinel = new Node<>(null, null, null);

sentinel.next = sentinel;

sentinel.prev = sentinel;

}

}

1. 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 Node<E> head;

// Constructor and other methods

public void rotate() {

if (head != null) {

head = head.next;

}

}

public void rotateBackward() {

if (head != null) {

head = head.prev; }}}

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

public DoublyLinkedList<E> clone() throws CloneNotSupportedException {

DoublyLinkedList<E> clonedList = (DoublyLinkedList<E>) super.clone();

clonedList.header = new Node<>(null, null, null);

clonedList.trailer = new Node<>(null, clonedList.header, null);

Node<E> current = header.next;

Node<E> prevNode = clonedList.header;

while (current != trailer) {

Node<E> newNode = new Node<>(current.element, prevNode, null);

prevNode.next = newNode;

prevNode = newNode;

current = current.next;

}

prevNode.next = clonedList.trailer;

clonedList.trailer.prev = prevNode;

return clonedList;

}