## Topics

1. Implement Node Class

public class Node<E> {

private E element;

private Node<E> next;

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

this.element = element;

this.next = next;

}

public E getElement() {

return element;

}

public void setElement(E

element) {

this.element = element;

}

public Node<E> getNext() {

return next;

}

public void setNext(Node<E> next) {

this.next = next;

}

}

1. Implement DoublyLinkedList Class

public class CircularlyLinkedList<E> {

private Node<E> tail;

private int size;

public CircularlyLinkedList() {

tail = null;

size = 0;

}

public int size() {

return size;

}

public boolean isEmpty() {

return size == 0;

}

public E first() {

if (isEmpty()) {

return null;

}

return tail.getNext().getElement();

}

public E last() {

if (isEmpty()) {

return null;

}

return tail.getElement();

}

public void addFirst(E element) {

if (isEmpty()) {

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

tail.setNext(tail);

} else {

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

tail.setNext(newNode);

}

size++;

}

public void addLast(E element) {

addFirst(element);

tail = tail.getNext();

}

public E removeFirst() {

if (isEmpty()) {

return null;

}

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

if (head == tail) {

tail = null;

} else {

tail.setNext(head.getNext());

}

size--;

return head.getElement();

}

1. Implement Basic Methods of DoublyLinkedList

* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()
* removeLast()

public class CircularlyLinkedList<E> {

private Node<E> tail;

private int size;

public CircularlyLinkedList() {

tail = null;

size = 0;

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

public E first() {

if (isEmpty()) {

return null;

}

return tail.getNext().getElement();

}

public E last() {

if (isEmpty()) {

return null;

}

return tail.getElement();

}

public void addFirst(E element) {

if (isEmpty()) {

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

tail.setNext(tail);

} else {

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

tail.setNext(newNode);

}

size++;

}

public void addLast(E element) {

addFirst(element);

tail = tail.getNext();

}

public E removeFirst() {

if (isEmpty()) {

return null;

}

Node<E> 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

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 void addFirst(E e) {

Node<E> newest = new Node<>(e);

if (isEmpty()) {

newest.setNext(newest);

tail = newest;

} else {

newest.setNext(tail.getNext());

tail.setNext(newest);

}

size++;

}

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() {

if (isEmpty()) {

return 0;

}

int count = 1;

Node<E> current = tail.getNext();

while (current != tail) {

count++;

current = current.getNext();

}

return count;

}

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

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (obj == null || getClass() != obj.getClass()) {

return false;

}

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

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

return false;

}

if (isEmpty()) {

return true; // Both lists are empty

}

Node<E> currentThis = tail.getNext();

Node<?> currentOther = otherList.tail.getNext();

while (currentThis != tail) {

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

return false;

}

currentThis = currentThis.getNext();

currentOther = currentOther.getNext();

}

// Compare the last elements

return currentThis.getElement().equal

s(currentOther.getElement());

}

1. Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.

function compareCircularLists(L, M):

if L.isEmpty() and M.isEmpty():

return true

startingPointL = findStartingPoint(L)

startingPointM = findStartingPoint(M)

if startingPointL is null or startingPointM is null:

return false

currentNodeL = startingPointL

currentNodeM = startingPointM

completedIterationL = false

completedIterationM = false

while not (completedIterationL and completedIterationM):

if currentNodeL.getElement() != currentNodeM.getElement():

return false

currentNodeL = currentNodeL.getNext()

currentNodeM = currentNodeM.getNext()

if currentNodeL == startingPointL:

completedIterationL = true

if currentNodeM == startingPointM:

completedIterationM = true

if completedIterationL != completedIterationM:

return false

return true

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.

function splitCircularList(L):

if L is empty or L contains an odd number of nodes:

return null

// Step 1: Find the midpoint

Node<E> tortoise = L.getHead();

Node<E> hare = L.getHead();

do {

hare = hare.getNext();

hare = hare.getNext();

tortoise = tortoise.getNext();

} while (hare != L.getHead() && hare.getNext() != L.getHead());

// Step 2: Break circular link in the first half

Node<E> lastNodeFirstHalf = tortoise;

lastNodeFirstHalf.setNext(null);

// Step 3: Create circular link between the two halves

Node<E> lastNodeSecondHalf =

L.getHead();

while (lastNodeSecondHalf.getNext() != L.getHead()) {

lastNodeSecondHalf = lastNodeSecondHalf.getNext();

}

lastNodeSecondHalf.setNext(L.getHead());

// Step 4: Return the two halves

CircularlyLinkedList<E> firstHalf = new

CircularlyLinkedList<>();

firstHalf.setHead(L.getHead());

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

secondHalf.setHead(tortoise);

return new Pair<>(firstHalf, secondHalf);

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 CircularlyLinkedList<E> clone() {

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

if (isEmpty()) {

return newList; // Return an empty list

}

Node<E> current = tail.getNext();

Node<E> newTail = new Node<>(current.getElement());

newList.tail = newTail;

while (current.getNext() != tail.getNext()) {

current = current.getNext();

Node<E> newNode = new Node<>(current.getElement());

newTail.setNext(newNode);

newTail = newNode;

}

newTail.setNext(newList.tail); // Set the last node to point back to the new tail

newList.size = size;

return newList;

}

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

public DoublyLinkedList<E> clone() {

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

if (isEmpty()) {

return newList; // Return an empty list

}

Node<E> current = head;

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

newList.head = newNode;

Node<E> prevNode = newNode;

current = current.getNext();

while (current != null) {

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

prevNode.setNext(newNode);

prevNode = newNode;

current = current.getNext();

}

newList.tail = prevNode;

newList.size = size;

return newList;

}