## 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 Node<E> findMiddle() {

if (isEmpty()) {

return null; // القائمة فارغة

}

Node<E> start = header.getNext(); // العقدة الأولى

Node<E> end = trailer.getPrev(); // العقدة الأخيرة

while (start != end && start.getNext() != end) {

start = start.getNext(); // تقدّم خطوة للأمام

end = end.getPrev(); // خطوة للخلف

}

return start; // العقدة الوسطى

}

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.getNext(); // العقدة الأولى

while (current != trailer) { // توقف عند العقدة الأخيرة

count++;

current = current.getNext(); // انتقل إلى العقدة التالية

}

return count;

}

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

@Override

public boolean equals(Object obj) {

if (obj == null || !(obj instanceof DoublyLinkedList)) {

return false;

}

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

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

return false;

}

Node<E> currentA = this.header.getNext();

Node<E> currentB = other.header.getNext();

while (currentA != this.trailer) {

if (!currentA.getElement().equals(currentB.getElement())) {

return false; // العناصر غير متساوية

}

currentA = currentA.getNext();

currentB = currentB.getNext();

}

return true; // القائمتان متساويتان

}

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

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

return;

}

Node<E> lastOfL = L.trailer.getPrev();

Node<E> firstOfM = M.header.getNext();

lastOfL.setNext(firstOfM);

firstOfM.setPrev(lastOfL);

L.trailer = M.trailer; // تحديث المؤشر الأخير

L.size += M.size; // تحديث الحجم

}

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 DoublyLinkedListSingleSentinel<E> {

private static class Node<E> {

E element;

Node<E> next;

Node<E> prev;

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

this.element = element;

this.prev = prev;

this.next = next;

}

}

private Node<E> sentinel;

private int size = 0;

public DoublyLinkedListSingleSentinel() {

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

sentinel.next = sentinel;

sentinel.prev = sentinel;

}

public void addFirst(E e) {

addBetween(e, sentinel, sentinel.next);

}

public void addLast(E e) {

addBetween(e, sentinel.prev, sentinel);

}

private void addBetween(E e, Node<E> prev, Node<E> next) {

Node<E> newNode = new Node<>(e, prev, next);

prev.next = newNode;

next.prev = newNode;

size++;

}

public E removeFirst() {

if (isEmpty()) return null;

return remove(sentinel.next);

}

public E removeLast() {

if (isEmpty()) return null;

return remove(sentinel.prev);

}

private E remove(Node<E> node) {

Node<E> prev = node.prev;

Node<E> next = node.next;

prev.next = next;

next.prev = prev;

size--;

return node.element;

}

public boolean isEmpty() {

return size == 0;

}

public int size() {

return size;

}

}

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> tail = null;

private int size = 0;

public void rotate() {

if (tail != null) {

tail = tail.next;

}

}

public void rotateBackward() {

if (tail != null) {

tail = tail.prev;

}

}

public void addFirst(E e) {

if (size == 0) {

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

tail.next = tail;

tail.prev = tail;

} else {

Node<E> newNode = new Node<>(e, tail, tail.next);

tail.next.prev = newNode;

tail.next = newNode;

}

size++;

}

public void addLast(E e) {

addFirst(e);

tail = tail.next;

}

}

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

@Override

public DoublyLinkedList<E> clone() {

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

Node<E> current = this.header.getNext();

while (current != this.trailer) {

copy.addLast(current.getElement());

current = current.getNext();

}

return copy;

}