|  |  |
| --- | --- |
| Homework\_6 |  |
| 1. 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(DoublyLinkedList<E> list) {  // Initialize the forward and backward pointers  Node<E> forward = list.header.next; // First node  Node<E> backward = list.trailer.prev; // Last node  // Link hopping until forward meets or crosses backward  while (forward != backward && forward != backward.next) {  forward = forward.next; // Move forward pointer  backward = backward.prev; // Move backward pointer  }  // Return the node slightly left of center  return backward;  } |
| 1. 2Give 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; // Counter to keep track of the number of nodes  Node<E> current = header.next; // Start at the first valid node (after the header)  // Traverse the list until reaching the trailer sentinel  while (current != trailer) {  count++; // Increment the counter for each node  current = current.next; // Move to the next node  }  return count; // Return the total count  } |
| 1. Implement the equals( ) method for the DoublyLinkedList class. | public boolean equals(Object other) {  // Check if the objects are the same  if (this == other) return true;  // Check if 'other' is null or not of the same type  if (other == null || !(other instanceof DoublyLinkedList)) return false;  // Cast the other object to DoublyLinkedList  DoublyLinkedList<?> otherList = (DoublyLinkedList<?>) other;  // Compare sizes  if (this.size() != otherList.size()) return false;  // Compare elements node by node  Node<E> currentA = this.header.next; // Start from the first valid node of this list  Node<?> currentB = otherList.header.next; // Start from the first valid node of other list  while (currentA != this.trailer) { // Traverse until the end of this list  if (!currentA.element.equals(currentB.element)) return false; // Check equality of elements  currentA = currentA.next;  currentB = currentB.next;  }  return true; // All elements matched  } |
| 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 DoublyLinkedList<E> concatenate(DoublyLinkedList<E> L, DoublyLinkedList<E> M) {  // Check if either list is empty  if (L.isEmpty()) return M; // If L is empty, return M  if (M.isEmpty()) return L; // If M is empty, return L  // Get the last valid node of L and the first valid node of M  Node<E> lastL = L.trailer.prev; // Last valid node in L  Node<E> firstM = M.header.next; // First valid node in M  // Link L and M  lastL.next = firstM; // Connect the last node of L to the first node of M  firstM.prev = lastL; // Connect the first node of M to the last node of L  // Update the trailer of L to the trailer of M  L.trailer = M.trailer; // L's trailer becomes M's trailer  L.trailer.prev.next = L.trailer; // Update the trailer's prev reference  // Disconnect M's header and trailer  M.header.next = M.trailer;  M.trailer.prev = M.header;  return L; // Return the concatenated list  } |
| 1. 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.    الإضافة والإزالة:   * التعقيد الزمني: O(1)O(1)O(1) (كل العمليات تتم بتحديث المراجع فقط).    الحجم:   * يتم تتبع الحجم باستخدام متغير size، مما يجعل عملية الحصول على الحجم أيضًا O(1)O(1)O(1 | public class DoublyLinkedList<E> {  private Node<E> sentinel; // عقدة حارس واحدة  private int size = 0; // حجم القائمة  // صف العقدة  private static class Node<E> {  E element;  Node<E> next;  Node<E> prev;  Node(E element, Node<E> next, Node<E> prev) {  this.element = element;  this.next = next;  this.prev = prev;  }  }  // المُنشئ  public DoublyLinkedList() {  sentinel = new Node<>(null, null, null);  sentinel.next = sentinel; // الحارس يشير لنفسه  sentinel.prev = sentinel;  }  // الحصول على الحجم  public int size() {  return size;  }  // التحقق من إذا كانت القائمة فارغة  public boolean isEmpty() {  return size == 0;  }  // إضافة عنصر في البداية  public void addFirst(E e) {  addBetween(e, sentinel, sentinel.next);  }  // إضافة عنصر في النهاية  public void addLast(E e) {  addBetween(e, sentinel.prev, sentinel);  }  // إزالة العنصر الأول  public E removeFirst() {  if (isEmpty()) return null;  return remove(sentinel.next);  }  // إزالة العنصر الأخير  public E removeLast() {  if (isEmpty()) return null;  return remove(sentinel.prev);  }  // وظيفة مساعدة للإضافة بين عقدتين  private void addBetween(E e, Node<E> predecessor, Node<E> successor) {  Node<E> newNode = new Node<>(e, successor, predecessor);  predecessor.next = newNode;  successor.prev = newNode;  size++;  }  // وظيفة مساعدة للإزالة  private E remove(Node<E> node) {  Node<E> predecessor = node.prev;  Node<E> successor = node.next;  predecessor.next = successor;  successor.prev = predecessor;  size--;  return node.element;  }  } [\\مشابه](file:///\\مشابه) لجافا |
| 1. 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.    لإضافة والإزالة:   1. الزمن: O(1)O(1)O(1) لكل عملية.    الدوران:   1. الزمن: O(1)O(1)O(1) لكل عملية. | public class CircularDoublyLinkedList<E> {  private Node<E> tail; // تشير إلى العنصر الأخير في القائمة  private int size = 0; // عدد العناصر  // صف العقدة  private static class Node<E> {  E element;  Node<E> next;  Node<E> prev;  Node(E element, Node<E> next, Node<E> prev) {  this.element = element;  this.next = next;  this.prev = prev;  }  }  // التحقق من إذا كانت القائمة فارغة  public boolean isEmpty() {  return size == 0;  }  // الحصول على الحجم  public int size() {  return size;  }  // إضافة عنصر في البداية  public void addFirst(E e) {  if (isEmpty()) {  Node<E> newNode = new Node<>(e, null, null);  newNode.next = newNode; // يشير لنفسه  newNode.prev = newNode;  tail = newNode; // العنصر الوحيد يصبح هو الذيل  } else {  Node<E> head = tail.next;  Node<E> newNode = new Node<>(e, head, tail);  tail.next = newNode;  head.prev = newNode;  }  size++;  }  // إضافة عنصر في النهاية  public void addLast(E e) {  addFirst(e); // أضف في البداية  tail = tail.next; // اجعل العنصر الجديد هو الذيل  }  // إزالة العنصر الأول  public E removeFirst() {  if (isEmpty()) return null;  Node<E> head = tail.next;  if (size == 1) {  tail = null; // إذا كانت القائمة تحتوي على عنصر واحد  } else {  Node<E> newHead = head.next;  tail.next = newHead;  newHead.prev = tail;  }  size--;  return head.element;  }  // إزالة العنصر الأخير  public E removeLast() {  if (isEmpty()) return null;  tail = tail.prev; // اجعل الذيل هو العنصر السابق  return removeFirst(); // احذف العنصر الأول  }  // الدوران للأمام  public void rotate() {  if (!isEmpty()) {  tail = tail.next; // العنصر الأول يصبح الأخير  }  }  // الدوران للخلف  public void rotateBackward() {  if (!isEmpty()) {  tail = tail.prev; // العنصر الأخير يصبح الأول  }  }  } |
| 1. 7\_Implement the clone( ) method for the DoublyLinkedList class. 2. التعقيد الزمني:    * O(n)O(n)O(n): يتم المرور على جميع العقد في القائمة الأصلية. 3. التعقيد المكاني:    * O(n)O(n)O(n): يتم إنشاء عقد جديدة لكل عنصر في القائمة. | public DoublyLinkedList<E> clone() {  // إنشاء قائمة جديدة  DoublyLinkedList<E> clonedList = new DoublyLinkedList<>();    // إذا كانت القائمة الأصلية فارغة  if (this.isEmpty()) {  return clonedList; // إرجاع نسخة فارغة  }  // البدء من أول عقدة  Node<E> current = this.header.next; // العقدة الأولى في القائمة الأصلية    // المرور على العقد ونسخها  while (current != this.trailer) {  clonedList.addLast(current.element); // نسخ كل عنصر وإضافته إلى القائمة الجديدة  current = current.next; // الانتقال إلى العقدة التالية  }    return clonedList; // إرجاع النسخة  } |

تكليف الطالبة/سمية عبدة مصلال