**Algorithm for Concatenating Two Doubly Linked Lists:**

1. **Initial Checks:**
   * If either list LLL or list MMM is empty, the concatenation is straightforward:
     + If LLL is empty, L′L'L′ will be identical to MMM.
     + If MMM is empty, L′L'L′ will be identical to LLL.
2. **Modify Pointers for Concatenation:**
   * Link the last node of LLL (the node just before the trailer sentinel) to the first node of MMM (the node just after the header sentinel of MMM).
   * Link the last node of MMM (the node just before the trailer sentinel of MMM) to the trailer of LLL (the trailer sentinel of LLL).
   * Adjust the next pointer of the last node in LLL to point to the first node in MMM, and adjust the prev pointer of the first node in MMM to point to the last node of LLL.
3. **Update the Trailer Pointer:**
   * Set the trailer pointer of LLL to the trailer of MMM.

**Steps in Code:**

Here is how the algorithm can be implemented in Java for concatenating two doubly linked lists LLL and MMM.

**Java Code Implementation:**

public class DoublyLinkedList {

// تعريف الكلاس Node لتمثيل العقدة

private static class Node {

Object element;

Node prev;

Node next;

public Node(Object element) {

this.element = element;

this.prev = null;

this.next = null;

}

}

private Node header;

private Node trailer;

public DoublyLinkedList() {

header = new Node(null); // عقدة رأسية

trailer = new Node(null); // عقدة نهائية

header.next = trailer;

trailer.prev = header;

}

// دالة لإضافة عنصر إلى نهاية القائمة

public void addLast(Object element) {

Node newNode = new Node(element);

newNode.prev = trailer.prev;

newNode.next = trailer;

trailer.prev.next = newNode;

trailer.prev = newNode;

}

// دالة لدمج قائمتين (L و M) إلى قائمة جديدة L'

public static DoublyLinkedList concatenate(DoublyLinkedList L, DoublyLinkedList M) {

// إنشاء قائمة جديدة L' تكون بداية بقائمة L

DoublyLinkedList result = new DoublyLinkedList();

// إذا كانت قائمة L فارغة، استخدم قائمة M مباشرة

if (L.size() == 0) {

result.header.next = M.header.next;

M.header.next.prev = result.header;

result.trailer.prev = M.trailer.prev;

M.trailer.prev.next = result.trailer;

} else {

// إذا كانت قائمة M فارغة، استخدم قائمة L مباشرة

if (M.size() == 0) {

result.header.next = L.header.next;

L.header.next.prev = result.header;

result.trailer.prev = L.trailer.prev;

L.trailer.prev.next = result.trailer;

} else {

// ربط آخر عنصر في L إلى أول عنصر في M

L.trailer.prev.next = M.header.next;

M.header.next.prev = L.trailer.prev;

// ربط آخر عنصر في M إلى عقدة trailer في L

L.trailer.prev = M.trailer.prev;

M.trailer.prev.next = L.trailer;

}

}

return result;

}

// دالة لإرجاع حجم القائمة

public int size() {

int count = 0;

Node current = header.next;

while (current != trailer) {

count++;

current = current.next;

}

return count;

}

// دالة لطباعة القائمة

public void printList() {

Node current = header.next;

while (current != trailer) {

System.out.print(current.element + " ");

current = current.next;

}

System.out.println();

}

// اختبار دالة concatenate

public static void main(String[] args) {

DoublyLinkedList L = new DoublyLinkedList();

L.addLast(1);

L.addLast(2);

L.addLast(3);

DoublyLinkedList M = new DoublyLinkedList();

M.addLast(4);

M.addLast(5);

M.addLast(6);

System.out.println("List L:");

L.printList();

System.out.println("List M:");

M.printList();

DoublyLinkedList L\_prime = concatenate(L, M);

System.out.println("Concatenated List L':");

L\_prime.printList();

}

}