1. Implement Node Class
2. Implement DoublyLinkedList Class
3. Implement Basic Methods of DoublyLinkedList

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

1. قم بتنفيذ فئة Node:

java

Copy

public class Node<T> {

private T element;

private Node<T> prev;

private Node<T> next;

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

this.element = element;

this.prev = prev;

this.next = next;

}

public T getElement() {

return element;

}

public Node<T> getPrev() {

return prev;

}

public void setPrev(Node<T> prev) {

this.prev = prev;

}

public Node<T> getNext() {

return next;

}

public void setNext(Node<T> next) {

this.next = next;

}

}

فئة Node تمثل عقدة في القائمة المزدوجة المرتبطة، حيث تحتوي على عنصر ومرجع إلى العقدة السابقة والعقدة التالية.

1. قم بتنفيذ فئة DoublyLinkedList:

java

Copy

public class DoublyLinkedList<T> {

private Node<T> header;

private Node<T> trailer;

public DoublyLinkedList() {

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

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

header.setNext(trailer);

}

// ... باقي الطرق ...

}

فئة DoublyLinkedList تمثل القائمة المزدوجة المرتبطة، حيث تحتوي على عقدة الرأس (header) وعقدة الذيل (trailer) وتعيد القائمة فارغة في البداية.

1. قم بتنفيذ الطرق الأساسية لفئة DoublyLinkedList:

java

Copy

public class DoublyLinkedList<T> {

// ... الكود السابق ...

public boolean isEmpty() {

return header.getNext() == trailer;

}

public int size() {

int count = 0;

Node<T> current = header.getNext();

while (current != trailer) {

count++;

current = current.getNext();

}

return count;

}

public T first() {

if (isEmpty()) {

return null;

}

return header.getNext().getElement();

}

public T last() {

if (isEmpty()) {

return null;

}

return trailer.getPrev().getElement();

}

public void addFirst(T element) {

addBetween(element, header, header.getNext());

}

public void addLast(T element) {

addBetween(element, trailer.getPrev(), trailer);

}

public T removeFirst() {

if (isEmpty()) {

return null;

}

return remove(header.getNext());

}

public T removeLast() {

if (isEmpty()) {

return null;

}

return remove(trailer.getPrev());

}

// ... باقي الطرق ...

}

الطرق المنفذة تشمل:

* isEmpty(): تحقق مما إذا كانت القائمة المزدوجة المرتبطة فارغة أم لا.
* size(): تعيد حجم القائمة المزدوجة المرتبطة.
* first(): تعيد العنصر الأول في القائمة المزدوجة المرتبطة.
* last(): تعيد العنصر الأخير في القائمة المزدوجة المرتبطة.
* addFirst(): تضيف عنصرًا في بداية القائمة المزدوجة المرتبطة.
* addLast(): تضيف عنصرًا في نهاية القائمة المزدوجة المرتبطة.
* removeFirst(): تقوم بإزالة وإرجاع العنصر الأول في القائمة المزدوجة المرتبطة.
* removeLast(): تقوم بإزالة وإرجاع عنصر الأخير في القائمة المزدوجة المرتبطة.

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.”
2. Give an implementation of the size( ) method for the DoublyLinkedList class, assuming that we did not maintain size as an instance variable.
3. Implement the equals( ) method for the DoublyLinkedList class.
4. Give an algorithm for concatenating two doubly linked lists L and M, with header and trailer sentinel nodes, into a single list L′.
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.
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.
7. Implement the clone( ) method for the DoublyLinkedList class.
8. وصف طريقة للعثور على العقدة المتوسطة في القائمة المزدوجة المرتبطة مع وجود عقدة رأس وعقدة ذيل "بواسطة القفز بين الروابط" وبدون الاعتماد على معرفة صريحة بحجم القائمة. في حالة وجود عدد زوجي من العقدات، أعرض العقدة القليلة يسار المركز كـ "الوسط".

java

Copy

public Node<T> findMiddleNode() {

Node<T> slow = header.getNext();

Node<T> fast = header.getNext();

while (fast != trailer && fast.getNext() != trailer) {

slow = slow.getNext();

fast = fast.getNext().getNext();

}

return slow;

}

هذه الطريقة تستخدم مبدأ "القفز بين الروابط" للعثور على العقدة المتوسطة. يتم استخدام متغيرين، slow و fast، حيث يقوم slow بالتقدم بخطوة واحدة في كل مرة ويقوم fast بالتقدم بخطوتين في كل مرة. عندما يصل fast إلى نهاية القائمة أو العقدة الأخيرة، يكون slow قد وصل إلى العقدة المتوسطة.

1. تنفيذ طريقة size() لفئة DoublyLinkedList مع افتراض أننا لم نحتفظ بحجم القائمة كمتغير عضو:

java

Copy

public int size() {

int count = 0;

Node<T> current = header.getNext();

while (current != trailer) {

count++;

current = current.getNext();

}

return count;

}

تتمثل فكرة هذه الطريقة في الانتقال عبر القائمة من العقدة الرأس إلى العقدة الذيل وعد العقدات على الطريق.

1. تنفيذ طريقة equals() لفئة DoublyLinkedList:

java

Copy

@Override

public boolean equals(Object obj) {

if (this == obj) {

return true;

}

if (!(obj instanceof DoublyLinkedList)) {

return false;

}

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

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

return false;

}

Node<T> current = header.getNext();

Node<?> otherCurrent = otherList.header.getNext();

while (current != trailer) {

if (!current.getElement().equals(otherCurrent.getElement())) {

return false;

}

current = current.getNext();

otherCurrent = otherCurrent.getNext();

}

return true;

}

تقارن هذه الطريقة بين قائمتين مزدوجتين مرتبطتين. تتطابق القوائم إذا كان حجمهما متساوٍ وكل عنصر في القائمة الأولى يطابق عنصرًا مقابلًا في القائمة الثانية.