COMP20010

Data Structures and Algorithms I

03 - Tutorial: Linked Lists

Dr. Aonghus Lawlor
aonghus.lawlor@ucd.ie

Linked Lists

- In this tutorial we will work through the implementation of a doubly linked list in Java.

- Consider the interface class for List

- Iteration built in to the List class

- reversing a linked list

Doubly Linked List

- Get some starter code at the Github link

GitHub

DoublyLinkedList

```java
private static class Node<E> {

  /** The element stored at this node */
  private E element;                // reference to the element
stored at this node

  /** A reference to the preceding node in the list */
  private Node<E> prev;             // reference to the previous
node in the list

  /** A reference to the subsequent node in the list */
  private Node<E> next;             // reference to the subsequent
node in the list

  /**
   * Creates a node with the given element and next node.
   *
   * @param e  the element to be stored
   * @param p  reference to a node that should precede the new
node
   * @param n  reference to a node that should follow the new node
   */
  public Node(E e, Node<E> p, Node<E> n) {
    element = e;
    prev = p;
    next = n;
  }
```

```java
  // public accessor methods
  /**
   * Returns the element stored at the node.
   * @return the element stored at the node
   */
  public E getElement() { return element; }

  /**
   * Returns the node that precedes this one (or null if no such
node).
   * @return the preceding node
   */
  public Node<E> getPrev() { return prev; }

  /**
   * Returns the node that follows this one (or null if no such
node).
   * @return the following node
   */
  public Node<E> getNext() { return next; }

  // Update methods
  /**
   * Sets the node's previous reference to point to Node n.
   * @param p    the node that should precede this one
   */
  public void setPrev(Node<E> p) { prev = p; }
```

DoublyLinkedList

```
// instance variables of the DoublyLinkedList
/** Sentinel node at the beginning of the list */
private Node<E> header;                        // header sentinel

/** Sentinel node at the end of the list */
private Node<E> trailer;                       // trailer sentinel

/** Number of elements in the list (not including sentinels) */
private int size = 0;                          // number of elements in the list

/** Constructs a new empty list. */
public DoublyLinkedList() {
  header = new Node<>(null, null, null);         // create header
  trailer = new Node<>(null, header, null);      // trailer is preceded by header
  header.setNext(trailer);                       // header is followed by trailer
}
```

DoublyLinkedList

```java
// public update methods
/**
 * Adds an element to the front of the list.
 * @param e   the new element to add
 */
public void addFirst(E e) {
  addBetween(e, header, header.getNext());    // place just after the header
}

/**
 * Adds an element to the end of the list.
 * @param e   the new element to add
 */
public void addLast(E e) {
  addBetween(e, trailer.getPrev(), trailer);  // place just before the trailer
}

/**
 * Removes and returns the first element of the list.
 * @return the removed element (or null if empty)
 */
public E removeFirst() {
  if (isEmpty()) return null;                  // nothing to remove
  return remove(header.getNext());             // first element is beyond header
}

/**
 * Removes and returns the last element of the list.
 * @return the removed element (or null if empty)
 */
public E removeLast() {
  if (isEmpty()) return null;                  // nothing to remove
  return remove(trailer.getPrev());            // last element is before trailer
}
```

DoublyLinkedList

```java
// private update methods
/**
 * Adds an element to the linked list in between the given nodes.
 * The given predecessor and successor should be neighboring each
 * other prior to the call.
 *
 * @param predecessor   node just before the location where the new element is inserted
 * @param successor     node just after the location where the new element is inserted
 */
private void addBetween(E e, Node<E> predecessor, Node<E> successor) {
  // create and link a new node
  Node<E> newest = new Node<>(e, predecessor, successor);
  predecessor.setNext(newest);
  successor.setPrev(newest);
  size++;
}

/**
 * Removes the given node from the list and returns its element.
 * @param node    the node to be removed (must not be a sentinel)
 */
private E remove(Node<E> node) {
  Node<E> predecessor = node.getPrev();
  Node<E> successor = node.getNext();
  predecessor.setNext(successor);
  successor.setPrev(predecessor);
  size--;
  return node.getElement();
}
```