Lecture 10: Doubly Linked Lists

- Doubly Linked Lists
- Java Collections
Recursive Definition of Doubly Linked Lists

- A doubly linked list is either empty (null) or a node having a reference to a doubly linked list.
- **Node**: is a data type that holds any kind of data and two references to the previous and next node.
Node

private class Node {
    Item item;
    Node next;
    Node prev;
}

DOUBLY LINKED LISTS

Standard Operations

- `DoublyLinkedList()`: Constructs an empty doubly linked list.
- `isEmpty()`: Returns true if the doubly linked list does not contain any item.
- `size()`: Returns the number of items in the doubly linked list.
- `get(int index)`: Returns the item at the specified index.
- `addFirst(Item item)`: Inserts the specified item at the head of the doubly linked list.
- `addLast(Item item)`: Inserts the specified item at the tail of the doubly linked list.
- `add(int index, Item item)`: Inserts the specified item at the specified index.
- `removeFirst()`: Retrieves and removes the head of the doubly linked list.
- `removeLast()`: Retrieves and removes the tail of the doubly linked list.
- `remove(int index)`: Retrieves and removes the item at the specified index.
DoublyLinkedList(): Constructs an empty DLL

DoublyLinkedList<String> dll = new DoublyLinkedList<String>();

head = null

tail = null

n = 0

dll.addFirst("CS062");
addFirst(Item item): Inserts the specified item at the head of the doubly linked list

dll.addFirst("CS062");
n=1

dll.addFirst("ROCKS");

What should happen?
addFirst(Item item): Inserts the specified item at the head of the doubly linked list

dl.addFirst("ROCKS")

What should happen?

dl.addLast("!");

n=2
addLast(Item item): Inserts the specified item at the tail of the doubly linked list

```
dll.addLast("!");
n=3
```

What should happen?

```
dll.add(1,"?");
```

Head/Beginning/Front/First

Tail/End/Back/Last
add(int index, Item item): Adds item at the specified index

dll.add(1, "?")
n=4

What should happen?
dll.removeFirst();
removeFirst(): Retrieves and removes the head of the doubly linked list

```python
dll.removeFirst()
n=3
```

What should happen?
```javascript
dll.removeLast();
```
removeLast(): Retrieves and removes the tail of the doubly linked list

dl.removeLast()

n=2

What should happen?
dl.remove(1);
remove(int index): Retrieves and removes the item at the specified index.

```python
dll.remove(1)
```

```
n=1
```
Our own implementation of Doubly Linked Lists

- We will follow the textbook style.
  - It does not offer a class for this so we will build our own.
- We will work with generics because we don’t want to offer multiple implementations.
- We will use an inner class Node and we will keep track of how many elements we have in our doubly linked list.
Instance variables and inner class

```java
public class DoublyLinkedList<Item> implements Iterable<Item> {
    private Node first; // head of the doubly linked list
    private Node last;  // tail of the doubly linked list
    private int n;      // number of nodes in the doubly linked list

    /**
     * This nested class defines the nodes in the doubly linked list with a value
     * and pointers to the previous and next node they are connected.
     */
    private class Node {
        Item item;
        Node next;
        Node prev;
    }
```
Check if is empty and how many items

```java
/**
 * Returns true if the doubly linked list does not contain any item.
 * @return true if the doubly linked list does not contain any item
 */
public boolean isEmpty() {
    return size() == 0;
}

/**
 * Returns the number of items in the doubly linked list.
 * @return the number of items in the doubly linked list
 */
public int size() {
    return n;
}
```
Retrieves item from specified index

```java
/**
 * Returns item at the specified index.
 * 
 * @param index the index of the item to be returned
 * @return the item at specified index
 */
public Item get(int index) {
    rangeCheck(index);

    if (index == 0)
        return first.item;

    else if (index == size() - 1)
        return last.item;

    Node finger = first;
    // search for index-th element or end of list
    while (index > 0) {
        finger = finger.next;
        index--;
    }
    return finger.item;
}
```
Insert item at head of doubly linked list

```java
/**
 * Inserts the specified item at the head of the doubly linked list.
 * @param item the item to be inserted
 */
public void addFirst(Item item) {
    // Save the old node
    Node oldfirst = first;

    // Make a new node and assign it to head. Fix pointers.
    first = new Node();
    first.item = item;
    first.next = oldfirst;
    first.prev = null;

    // if first node to be added, adjust tail to it.
    if (last == null)
        last = first;
    else
        oldfirst.prev = first;

    n++; // increase number of nodes in doubly linked list.
}
```
Insert item at tail of doubly linked list

```java
/**
 * Inserts the specified item at the tail of the doubly linked list.
 *
 * @param item the item to be inserted
 */
public void addLast(Item item) {
    // Save the old node
    Node oldlast = last;

    // Make a new node and assign it to tail. Fix pointers.
    last = new Node();
    last.item = item;
    last.next = null;
    last.prev = oldlast;

    // if first node to be added, adjust head to it.
    if (first == null)
        first = last;
    else
        oldlast.next = last;

    n++;
}
```
Check if index is >=0 and <n

/**
 * A helper method to check if an index is in range 0<=index<n
 *
 * @param index the index to check
 */
private void rangeCheck(int index) {
    if (index >= n || index < 0)
        throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
}
/**
 * Inserts the specified item at the specified index.
 *
 * @param index the index to insert the item
 * @param item  the item to insert
 */
public void add(int index, Item item) {
    rangeCheck(index);
    if (index == 0) {
        addFirst(item);
    } else if (index == size()) {
        addLast(item);
    } else {
        Node previous = null;
        Node finger = first;
        // search for index-th position
        while (index > 0) {
            previous = finger;
            finger = finger.next;
            index--;
        }
        // create new value to insert in correct position
        Node current = new Node();
        current.item = item;
        current.next = finger;
        current.prev = previous;
        previous.next = current;
        finger.prev = current;
        n++;
    }
}
Retrieve and remove head

/**
 * Retrieves and removes the head of the doubly linked list.
 *
 * @return the head of the doubly linked list.
 */

public Item removeFirst() {
    Node oldFirst = first;
    // Fix pointers.
    first = first.next;
    // at least 1 nodes left.
    if (first != null) {
        first.prev = null;
    } else {
        last = null; // remove final node.
    }
    oldFirst.next = null;

    n--;

    return oldFirst.item;
}
/**
 * Retrieves and removes the tail of the doubly linked list.
 * @return the tail of the doubly linked list.
 */
public Item removeLast() {
    Node temp = last;
    last = last.prev;

    // if there was only one node in the doubly linked list.
    if (last == null) {
        first = null;
    } else {
        last.next = null;
    }
    n--;
    return temp.item;
}
Retrieve and remove element from a specific index

```java
/**
 * Retrieves and removes the item at the specified index.
 *
 * @param index the index of the item to be removed
 * @return the item previously at the specified index
 */
public Item remove(int index) {
    rangeCheck(index);

    if (index == 0) {
        return removeFirst();
    } else if (index == size() - 1) {
        return removeLast();
    } else {
        Node previous = null;
        Node finger = first;
        // search for value indexed, keep track of previous
        while (index > 0) {
            previous = finger;
            finger = finger.next;
            index--;
        }
        previous.next = finger.next;
        finger.next.prev = previous;

        n--;
        // finger’s value is old value, return it
        return finger.item;
    }
}
```
Lecture 10: Doubly Linked Lists

- Doubly Linked Lists
- Java Collections
The Java Collections Framework

Collections

LinkedLIst in Java Collections

- Doubly linked list implementation of the List and Deque (stay tuned) interfaces.

```java
java.util.LinkedList;

public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>
```
Lecture 10: Doubly Linked Lists

- Doubly Linked Lists
- Java Collections
ASSIGNED READINGS AND PRACTICE PROBLEMS

Readings:

- Oracle’s guides:
  - Collections: https://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - Linked Lists: https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

- Textbook:
  - Chapter 1.3 (Page 142-146)

- Textbook Website:
  - Linked Lists: https://algs4.cs.princeton.edu/13stacks/

Practice Problems:

- 1.3.18-1.3.27 (approach them as doubly linked lists).