CSE 12 – Basic Data Structures

Prof. Christine Alvarado
[Slides borrowed/adapted from slides by Cynthia Lee]
Discussion sections

1. Both will have 30 min review of basic topics
2. Last 20 mins will be:
   1. Monday: student directed questions on current HW
   2. Wednesday: preview of next HW
Announcements

1. HW2: How’s it going?
   A. I haven’t even looked at it.
   B. I’ve read it
   C. I’ve started programming (or tried to)
   D. I’m well into the assignment
   E. I’m done!
Announcements

1. HW2: How confused are you about HW2?
   A. N/A: I haven’t looked at it
   B. Completely lost, I don’t have a clue
   C. Pretty confused, but I get a few things
   D. A little confused, but I think I’ve got it
   E. Not confused. I know what I need to do, and more or less how I’ll do it
Today’s Topics

1. Implementing the List interface: Linked Lists
2. Implementing a LinkedList iterator
Single Linked List: Picture

MySingleLinkedList object

head

size 2

Node object

next

data null

Node object

next

data

Node object

next null

data

Objects of type E:
What does the list of nodes look like @ the end of main? (choices next slide)

class Node<E> {
    E data;
    Node next;

    public Node() {
        data = null;
        next = null;
    }

    public Node(E theData, Node newNodePred) {
        data = theData;
        next = newNodePred.next;
        newNodePred.next = this;
    }
}

public static void main() {
    Node<Integer> n0 = new Node<Integer>();
    Node<Integer> n1 = new Node( new Integer(1), n0);
    Node<Integer> n2 = new Node( new Integer(2), n0);
}
A
n0
null → n1 → n2
1 → 2 → null

B
n0
null → n2 → n1
2 → 1 → null

C
n0
null → n2
2 → null

D Other
Single Linked List: Code

class MySingleLinkedList<E> extends AbstractList
{
    Node<E> head;
    int size;

    public MySingleLinkedList<E>() {
        head = new Node<E>();
        size = 0;
    }
    //... more here
}

After calling the constructor:

<table>
<thead>
<tr>
<th>Node object</th>
</tr>
</thead>
<tbody>
<tr>
<td>head</td>
</tr>
<tr>
<td>size 0</td>
</tr>
</tbody>
</table>

Dummy (sentinel) head node

<table>
<thead>
<tr>
<th>Node object</th>
</tr>
</thead>
<tbody>
<tr>
<td>next null</td>
</tr>
<tr>
<td>data null</td>
</tr>
</tbody>
</table>
Single Linked List: Code

class MySingleLinkedList<E> extends AbstractList
{
    Node<E> head;
    int size;

    public MySingleLinkedList<E>() {
        head = new Node<E>();
        size = 0;
    }
    //... more here
}

After calling the constructor:

Dummy (sentinel) head node

head

After calling the constructor:

NULL NULL
addFirst (E item)

addLast (E item)

remove (E item)
addFirst (E item)

addLast (E item)

remove (E item)
addFirst (E item)

head

WARNING
These methods are similar to what you’ll implement in HW2, but not the same.
Read the documentation for HW2 carefully to understand what your HW2 methods have to do and how they are different from what you see here.

remove (E item)
// In MySingleLinkedList<E> class (NOT Node class)
public void addFirst (E newItem) {
    Node<E> newNode = new Node<E>(newItem, head);
________________________
    size++;
}

What line of code will complete this method correctly (in the blank)?
A) No line is needed. The code is correct as written.
B) head = head.next;
C) head = newNode;
D) newNode.next = head;
public void addLast (E newItem) {
    Node<E> current = head;
    while (current != null) {
        current = current.next;
    }
    new Node<E>(newItem, current);
    size++;
}

A) current == head
B) current != null
C) current.next != null
D) head != null
Another implementation of `addLast`

```java
public void addLast (E newItem) {
    Node<E> current = head;
    int currIndex = 0;
    while (currIndex < size){
        current = current.next;
        currIndex++;
    }
    new Node<E>(newItem, current);
    size++;
}
```

Change to insert at an arbitrary location?
Removal from Linked List

// In MySingleLinkedList<E> class
public E remove (int position){
    // Removes the element at index position from the
    // list and returns the element.
    ...
}

head

NULL  NULL
Suppose we have a reference (current) to the node containing the item to be removed.

What additional information do we need to successfully remove the node?

A) Nothing additional.
B) A reference to the node immediately prior to the deleted node.
C) A reference to the node immediately after the node to be deleted.
D) Both B and C.
Suppose we have a reference (current) to the node containing the item to be removed.

What additional information do we need to successfully remove the node?

A) Nothing additional.
B) A reference to the node immediately prior to the deleted node.
C) A reference to the node immediately after the node to be deleted.
D) Both B and C.

This is different for your Double linked list implementation (where you already have a reference to the node before, in the prev pointer.)
Removal from Linked List

// In MySingleLinkedList<E> class
public E remove (int position){
    // Removes the element at index position from the list and returns the element.
HW2: Doubly linked lists

Objects of type E:

Linked list object stores pointer to the tail

MyLinkedList object

head

size 1
tail

Node object

next

data null

prev null

Node object

next

data

prev

Node object

next

data null

prev

Dummy head and tail nodes
Iterator objects: Picture

head

ListIterator

left

right

idx

0

canRemove false

forward true

NULL
T next()

Return the next element in the list when going forward.
Throw NoSuchElementException if there is no such element

What would be returned by a call to it.next()?
A. The Node referenced by right
B. The Node referenced by left
C. The item stored in right.data
D. The item stored in left.data
E. The method would throw a NoSuchElementException
T next()
Return the next element in the list when going forward.
Throw NoSuchElementException if there is no such element

What instance variables would change value in it during a call to it.next()?
A. left, right
B. left, right, idx
C. right, idx
D. left, right, idx, canRemove, forward
E. Other
HW2 tip:
DRAW THE PICTURES BEFORE YOU CODE!!
Next time

- Running time analysis