Topic 11
Linked Lists

"All the kids who did great in high school writing pong games in BASIC for their Apple II would get to
college, take CompSci 101, a data structures
course, and when they hit the pointers business their
brains would just totally explode, and the next thing
you knew, they were majoring in Political Science
because law school seemed like a better idea."

-Joel Spolsky

Thanks to Don Slater of CMU for use of his slides.

Dynamic Data Structures

- *Dynamic* data structures
  - They grow and shrink one element at a time, normally without some of the inefficiencies of
  arrays
  - as opposed to a static container such as an array

- Big O of Array Manipulations
  - Access the kth element
  - Add or delete an element in the middle of the
  array while maintaining relative order
  - adding element at the end of array? space
  avail? no space avail?
  - add element at beginning of an array

Clicker Question 1

- What is output by the following code?
  ```java
  ArrayList<Integer> a1 = new ArrayList<Integer>();
  ArrayList<Integer> a2 = new ArrayList<Integer>();
  a1.add(12);
  a2.add(12);
  System.out.println( a1 == a2 );
  ```

A. No output due to syntax error
B. No output due to runtime error
C. false
D. true

Object References

- Recall that an *object reference* is a variable
  that stores the address of an object

- A reference can also be called a *pointer*

- They are often depicted graphically:

```
  student
  ---
  John Smith
  40725
  3.57
```
References as Links

- Object references can be used to create \textit{links} between objects

- Suppose a \texttt{Student} class contained a reference to another \texttt{Student} object

\begin{itemize}
  \item John Smith
    \begin{itemize}
      \item 40725
      \item 3.57
    \end{itemize}

  \item Jane Jones
    \begin{itemize}
      \item 58821
      \item 3.72
    \end{itemize}
\end{itemize}

\begin{tikzpicture}
  \node (john) at (0,0) {John Smith
    \begin{itemize}
      \item 40725
      \item 3.57
    \end{itemize}};
  \node (jane) at (1.5,0) {Jane Jones
    \begin{itemize}
      \item 58821
      \item 3.72
    \end{itemize}};
  \draw[->] (john) -- (jane);
\end{tikzpicture}

References as Links

- References can be used to create a variety of linked structures, such as a \textit{linked list}:

\begin{itemize}
  \item \texttt{studentList}
  \item \texttt{null}
  \item \texttt{next}
  \item \texttt{null}
  \item \texttt{next}
  \item \texttt{null}
\end{itemize}

Linked Lists

- A \textbf{linear} collection of self-referential objects, called \texttt{nodes}, connected by other \texttt{links}

  - \texttt{linear}: for every node in the list, there is one and only one node that precedes it (except for possibly the first node, which may have no predecessor,) and there is one and only one node that succeeds it, (except for possibly the last node, which may have no successor)

  - \texttt{self-referential}: a \texttt{node} that has the ability to refer to another \texttt{node} of the same \texttt{type}, or even to refer to itself

  - \texttt{node}: \texttt{contains data} of any \texttt{type}, including a reference to another \texttt{node} of the same \texttt{data type}, or to \texttt{nodes} of different \texttt{data types}

  - Usually a list will have a beginning and an \texttt{end}; the first \texttt{element} in the list is accessed by a reference to that \texttt{class}, and the last \texttt{node} in the list will have a reference that is set to \texttt{null}

Advantages of linked lists

- Linked lists are dynamic, they can grow or shrink as necessary

- Linked lists are \textit{non-contiguous}; the logical sequence of \texttt{items} in the structure is decoupled from any physical ordering in memory
Nodes and Lists

- A different way of implementing a list
- Each element of a Linked List is a separate Node object.
- Each Node tracks a single piece of data plus a reference (pointer) to the next
- Create a new Node very time we add something to the List
- Remove nodes when item removed from list and allow garbage collector to reclaim that memory

A Node Class

```java
public class Node<E> {
    private E myData;
    private Node myNext;

    public Node() {
        myData = null; myNext = null;    }
    public Node(E data, Node<E> next) {
        myData = data; myNext = next;    }

    public E getData() {
        return myData;    }
    public Node<E> getNext() {
        return myNext;    }
    public void setData(E data) {
        myData = data;    }
    public void setNext(Node<E> next) {
        myNext = next;    }
}
```

A Linked List Implementation

```java
public class LinkedList<E> implements IList<E> {
    private Node<E> head;
    private Node<E> tail;
    private int size;

    public LinkedList(){
        head = null;
        tail = null;
        size = 0;
    }
    LinkedList<String> list = new LinkedList<String>();
```

<table>
<thead>
<tr>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>myHead</td>
</tr>
<tr>
<td>iMySize</td>
</tr>
<tr>
<td>myTail</td>
</tr>
</tbody>
</table>
Writing Methods

- When trying to code methods for Linked Lists draw pictures!
  - If you don't draw pictures of what you are trying to do it is very easy to make mistakes!

Add Element - List Empty (Before)

<table>
<thead>
<tr>
<th>head</th>
<th>tail</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>0</td>
</tr>
</tbody>
</table>

Add Element - List Empty (After)

<table>
<thead>
<tr>
<th>head</th>
<th>tail</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Object

Item

String

Node

myData

myNext

null
Add Element - List Not Empty (Before)

Add Element - List Not Empty (After)

Code for default add

- public void add(E obj)

Clicker Question 2

- What is the worst case Big O for adding to the end of an array based list and a linked list? The lists already contain N items.

Array based | Linked
---|---
A. $O(1)$ | $O(1)$
B. $O(N)$ | $O(N)$
C. $O(\log N)$ | $O(1)$
D. $O(1)$ | $O(N)$
E. $O(N)$ | $O(1)$
Code for addFront

- add to front of list
- public void addFront(E obj)
- How does this compare to adding at the front of an array based list?

Clicker Question 3

- What is the Big O for adding to the front of an array based list and a linked list? The lists already contain N items.
  
<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>B. O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>C. O(logN)</td>
<td>O(1)</td>
</tr>
<tr>
<td>D. O(1)</td>
<td>O(N)</td>
</tr>
<tr>
<td>E. O(N)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>

Code for Insert

- public void insert(int pos, E obj)
- Must be careful not to break the chain!
- Where do we need to go?
- Special cases?

Clicker Question 4

- What is the Big O for inserting an element into the middle of an array based list and into the middle of a linked list? Each list already contains N items.
  
<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. O(N)</td>
<td>O(N)</td>
</tr>
<tr>
<td>B. O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>C. O(logN)</td>
<td>O(1)</td>
</tr>
<tr>
<td>D. O(logN)</td>
<td>O(logN)</td>
</tr>
<tr>
<td>E. O(1)</td>
<td>O(N)</td>
</tr>
</tbody>
</table>
**Clicker Question 5**

- What is the Big O for getting an element based on position from an array based list and from a linked list? Each list contains N items. In other words \(E \text{ get}(\text{int pos})\)

<table>
<thead>
<tr>
<th>Array based</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. (O(1))</td>
<td>(O(N))</td>
</tr>
<tr>
<td>B. (O(N))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>C. (O(\log N))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>D. (O(\log N))</td>
<td>(O(N))</td>
</tr>
<tr>
<td>E. (O(N))</td>
<td>(O(N))</td>
</tr>
</tbody>
</table>

**Code for get**

- public \(E \text{ get}(\text{int pos})\)
- The downside of Linked Lists

**Code for remove**

- public \(E \text{ remove}(\text{int pos})\)

**Why Use Linked List**

- What operations with a Linked List faster than the version from ArrayList?
Iterators for Linked Lists

What is the Big O of the following code?

```java
LinkedList<Integer> list;
list = new LinkedList<Integer>();
// code to fill list with N elements

// Big O of following code?
for (int i = 0; i < list.size(); i++)
    System.out.println(list.get(i));
```

A. O(N)  B. O(2^N)  C. O(NlogN)
D. O(N^2)  E. O(N^3)

---

Other Possible Features of Linked Lists

- Doubly Linked
- Circular
- Dummy Nodes for first and last node in list

```java
public class DLNode<E> {
    private E myData;
    private DLNode<E> myNext;
    private DLNode<E> myPrevious;
}
```

---

Dummy Nodes

- Use of Dummy Nodes for a Doubly Linked List removes most special cases
- Also could make the Double Linked List circular

---

Doubly Linked List add

- public void add(E obj)
Insert for Doubly Linked List

- public void insert(int pos, E obj)