Advanced Programming

Sorted Lists

Sources: Data Structures and Abstractions with Java, by Carrano
chapter 16, 3rd edition or chapter 13, 2nd edition
Contents

• Specifications for the ADT Sorted List
  ▪ Using the ADT Sorted List

• A Linked Implementation
  ▪ The Method add
  ▪ The Efficiency of the Linked Implementation

• An Implementation That Uses the ADT List
  ▪ Efficiency Issues
Objectives

- Use sorted list in a program
- Describe differences between ADT list and ADT sorted list
- Implement ADT sorted list by using chain of linked nodes
- Implement ADT sorted list by using operations of ADT list
Sorted Lists

• We extend the capability of a list
  ▪ Previous example used list to organize names in alphabetical order

• Consider need to keep list sorted in numerical or alphabetic order after list established
  ▪ We add or remove an element
  ▪ The ADT handles keeping elements in order
Specifications for the ADT Sorted List

• Possible operations
  ▪ For simplicity, duplicate entries allowed
  ▪ Must determine where in list element is added
  ▪ Can ask if list contains specified entry
  ▪ Must be able to remove an entry
Abstract Data Type: Sorted List

• Data
  ▪ A collection of objects in sorted order and having same data type
  ▪ Number of objects in collection

• Operations
  ▪ add(newEntry)
  ▪ remove(anEntry)
  ▪ getPosition(anEntry)
Abstract Data Type: Sorted List

- Operations used from ADT list (Ch. 12)
  - `getEntry(givenPosition)`
  - `contains(anEntry)`
  - `remove(givenPosition)`
  - `clear()`
  - `getLength()`
  - `isEmpty()`
  - `toArray()`

- Interface, [Listing 16-1](#)
A Linked Implementation

- Linked implementation of the ADT sorted list, Listing 16-2
- Note different versions of method `add`
  - Iterative version
  - Recursive version
Figure 16-1 Places to insert names into a sorted chain of linked nodes
Figure 16-2 Recursively adding *Luke* to a sorted chain of names

Luke > Bob, so add Luke to the rest of the chain
Figure 16-2 Recursively adding *Luke* to a sorted chain of names

_Luke > Jill, so add Luke to the rest of the chain_
Figure 16-2 Recursively adding *Luke* to a sorted chain of names

\[ Luke < Mike, \text{ so add } Luke \text{ here, at the beginning of the rest of the chain} \]
(a) The list before any additions

(b) As `add("Ally", firstNode)` begins execution

Figure 16-3 Recursively adding a node at the beginning of a chain
(c) After a new node is created (the base case)

The private method returns the reference that is in `currentNode`

(d) After the public `add` assigns the returned reference to `firstNode`

Figure 16-3 Recursively adding a node at the beginning of a chain
Figure 16-4 Recursively adding a node between existing nodes in a chain
Figure 16-4 Recursively adding a node between existing existing nodes in a chain.
(e) After the returned reference is assigned to `nodeAfter`

```
firstNode → Bob → Jill → Mike → Sue
```

```
currentNode → Luke
```

```
nodeAfter
```

(f) After `currentNode.setNextNode(nodeAfter)` executes

```
firstNode → Bob → Jill → Mike → Sue
```

```
currentNode → Luke
```

Figure 16-4 Recursively adding a node between existing nodes in a chain
Efficiency of the Linked Implementation

<table>
<thead>
<tr>
<th>ADT Sorted List Operation</th>
<th>Array</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(newEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>getPosition(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>getEntry(givenPosition)</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>contains(anEntry)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove(givenPosition)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>toArray()</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>clear(), getLength(), isEmpty()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

FIGURE 16-5 The worst-case efficiencies of the operations on the ADT sorted list for two implementations
Implementation That Uses the ADT List

- Sorted list a natural application for ADT list
- Possible ways
  - Use list as data field within class that implements sorted list
  - Use inheritance to derive sorted list from list
- Our class SortedList will implement the interface SortedListInterface
- View source code, Listing 16-A
Figure 16-6 An instance of a sorted list that contains a list of its entries.
Figure 16-7 A sorted list in which Jamie belongs after Carlos but before Sarah
## Efficiency Issues

Figure 16-8 The worst-case efficiencies of selected ADT list operations for array-based and linked implementations

<table>
<thead>
<tr>
<th>ADT List Operation</th>
<th>Array</th>
<th>Linked</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getEntry(givenPosition)</code></td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><code>add(newPosition, newEntry)</code></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><code>remove(givenPosition)</code></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><code>contains(anEntry)</code></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><code>toArray()</code></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><code>clear()</code>, <code>getLength()</code>, <code>isEmpty()</code></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>ADT Sorted List Operation</td>
<td>List Implementation</td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Array</td>
<td>Linked</td>
</tr>
<tr>
<td>add(new Entry)</td>
<td>O(n)</td>
<td>O(n&lt;sup&gt;2&lt;/sup&gt;)</td>
</tr>
<tr>
<td>remove(anEntry)</td>
<td>O(n)</td>
<td>O(n&lt;sup&gt;2&lt;/sup&gt;)</td>
</tr>
<tr>
<td>getPosition(anEntry)</td>
<td>O(n)</td>
<td>O(n&lt;sup&gt;2&lt;/sup&gt;)</td>
</tr>
<tr>
<td>getEntry(givenPosition)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>contains(anEntry)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>remove(givenPosition)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>toArray()</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>clear(), getLength(), isEmpty()</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

Figure 16-9 The worst-case efficiencies of the ADT sorted list operations when implemented using an instance of the ADT list.
Data Structures and Abstractions with Java, by Carrano
chapter 16, 3rd edition or
chapter 13, 2nd edition