Data Structures and Algorithms

Lists

Chris Brooks

Department of Computer Science
University of San Francisco
5-0: Abstract Data Types

- An Abstract Data Type is a definition of a type based on the operations that can be performed on it.
  - It is a List because we can insert, remove, clear, etc.
  - Not because of how elements are stored.
- An ADT is an interface
- Data in an ADT cannot be manipulated directly – only through operations defined in the interface
- A data structure is an implementation of an ADT
5-1: List ADT

- A List is an ordered collection of elements
- Each element in the list has a position
  - Element 0, Element 1, Element 2, ...
- We can access elements in the list through an *iterator*
5-2: List ADT Operations

- Make list empty
- Add an element to the list
  - At the end
  - At the current position
- Remove an element
- Get the length of the list
- Check if the list is empty
- Print list
- Get an iterator to traverse the list
Think of an iterator as a generic way of accessing the elements in an ADT.

Provides a *logical* index into the ADT.

Separates out the functionality of accessing elements from the ADT itself.

Some operation on iterators:
- Move iterator forward in the data structure
- Move iterator backward in the data structure
- Insert a new element at the current location
- Return the element at the current location
5-4: List Iterator

- Set the iterator to the front of the list
- Move the iterator forward one element
- Move the iterator back one element
- Set the iterator to index $i$
- Get the object at the current position
- Remove the object at the current position
- Insert an element right before the current position
A Java interface is a set of methods.

Any class that implements an interface must implement all of these methods.
5-6: Java List Interface

Without iterators:

```java
public interface List {
    public void clear();
    public void append(Object item);
    public Object remove();
    public int length();
    public void print();
    public boolean isEmpty();
    // All the following methods require a user to know about the 'curr' pointer.
    public void insert(Object item);
    public void next();
    public void prev();
    public void setPosition(int pos);
    public void setValue(Object val);
    public Object currValue();
}
```
With iterators:

```java
public interface List {
  public void clear();
  public void append(Object elem);
  public int length();
  public void print();
  public boolean isEmpty();
  public ListIterator iterator();
}
```
The iterator is an example of *delegation*

```java
public interface ListIterator

    public void first();
    public void next();
    public void previous();
    public boolean inList();
    public Object current();
    public void setPosition(int position);
    public void insert(Object elem);
    public Object remove();
    public void setCurrent(Object value);
```
5-9: Array Implementation

- Data is stored in an array
- Iterator stores index of current location
- To add an element to the current position:
  - Shift other elements to the right
- To remove an element from the middle of the array:
  - Shift other elements to the left
- List has a maximum size (unless we use growable arrays)
5-10: Array Implementation

$O()$ Running Time for each operation:
- insert
- append
- remove
- setFirst
- next
- previous
- length
- setPosition
- currentValue
5-11: Array Implementation

$O()$ Running Time for each operation:

- insert $O(n)$
- append $O(1)$
- remove $O(n)$
- setFirst $O(1)$
- next $O(1)$
- previous $O(1)$
- length $O(1)$
- setPosition $O(1)$
- currentValue $O(1)$
5-12: Linked-List Implementation

- Data is stored in a linked list
  - We’ll use an auxiliary class called Link to hold elements
- Maintain a pointer to first and last element in list
- Iterator maintains a pointer to the current element
- To find the ith element:
  - Start at the front of the list
  - Skip past i elements

How do we insert an element before the current element? How do we remove the current element?
Data is stored in a linked list

Maintain a pointer to first and last element in list

Iterator maintains a pointer to the element before the current element

To find the ith element:
  ▶ Start at the front of the list
  ▶ Skip past (i) elements

What should “current” pointer be when the first element in the list is the current element?
**5-14: Linked-List Implementation**

- Data is stored in a linked list – with a dummy first element
- Maintain a pointer to first (dummy) and last element in list
- Iterator maintains a pointer to the element before the current element
- To find the ith element:
  - Start at the front of the list
  - Skip past (i+1) elements
5-15: **Linked List Implementation**

$O()$ Running Time for each operation:
- insert
- append
- remove
- setFirst
- next
- previous
- length
- setPosition
- currentValue
$O()$ Running Time for each operation:

insert $O(1)$ *
append $O(1)$
remove $O(1)$ *
setFirst $O(1)$
next $O(1)$
previous $O(n)$ *
length $O(1)$
setPosition $O(n)$ *
currentValue $O(1)$

* Different from array implementation
Each element in the list has two pointers – next and previous

- Can locate the previous element of any element in the list in time $O(1)$, instead of time $O(n)$
- More space is required (two pointers for each element, instead of one)
- Does “current” still need to point to the element before the current element?
- Do we still need a “dummy” element?