Lists and Iterators

CSSE 221

Fundamentals of Software Development Honors

Rose-Hulman Institute of Technology

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Announcements

- Should be making progress on VectorGraphics.
  - Should have turned in CRC cards, UML, and User stories yesterday
  - These must be completed before you get a jump on the coding
- Questions on Exam 1?

Schedule

- OO software development in Java.
- 18 chapters in text!
  - Ch. 1-16.4, 18, 20
  - Only 6 more left…
- Lots of programming, including:
  - Each week' structured around a prog. assignment
  - 1 bigger team project
- Researching and presenting course material to classmates

	Topic	Project	Indep
1	Interfaces	BigRational	
2	Inher & Poly	BallWorlds	Research
3	GUI	Fifteen	Research
4	Exam	VectorGraphics	
5	Lists	VectorGraphics	Demo
6	Data Structs	Markov	Demo
7	TeamProject	TeamProject	
8	Sorting	TeamProject	Present
9	Searching	TeamProject	Present
10	Linked Lists	Linked Lists	

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

This week: VectorGraphics

- Today:
  - Lists and Iterators (capsule)
  - Review big-Oh
- Wednesday:
  - Threads (capsule)
  - Project workday
- Friday:
  - Stacks and Queues
  - Sets and Maps

Understanding the engineering trade-offs when storing data

Data Structures

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Data Structures and the Java Collections Framework

- Three aspects to understand:
  - Specification (methods in the interface)
  - Implementation (sometimes several alternate implementations)
  - Applications (when I should use it)

Data Structures

- Efficient ways to store data based on how we'll use it

- So far we've seen **ArrayLists**
  - Fast addition to end of list
  - Fast access to any existing position
  - Slow inserts to and deletes from middle of list
  - If sorted, can find in O(log n) time

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Another List Data Structure

- What if we have to add/remove data from a list frequently?
- A *LinkedList* supports this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements
  - Sketch one now

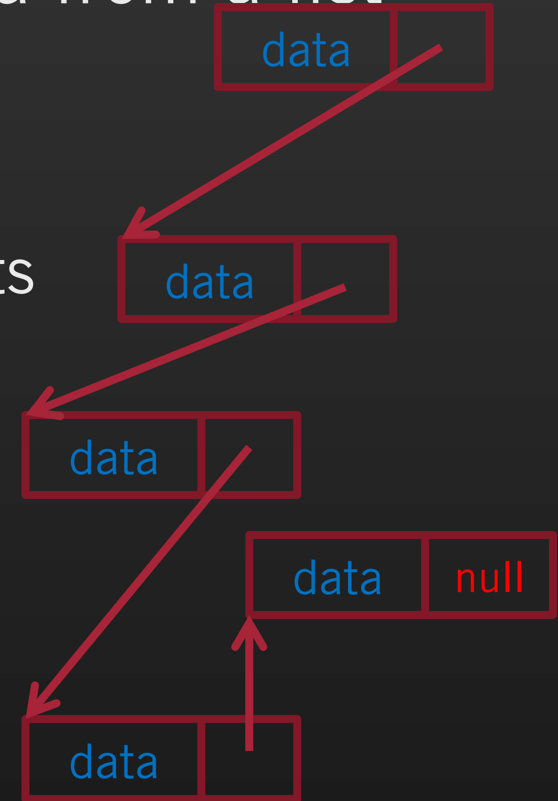
Insertion, per Wikipedia

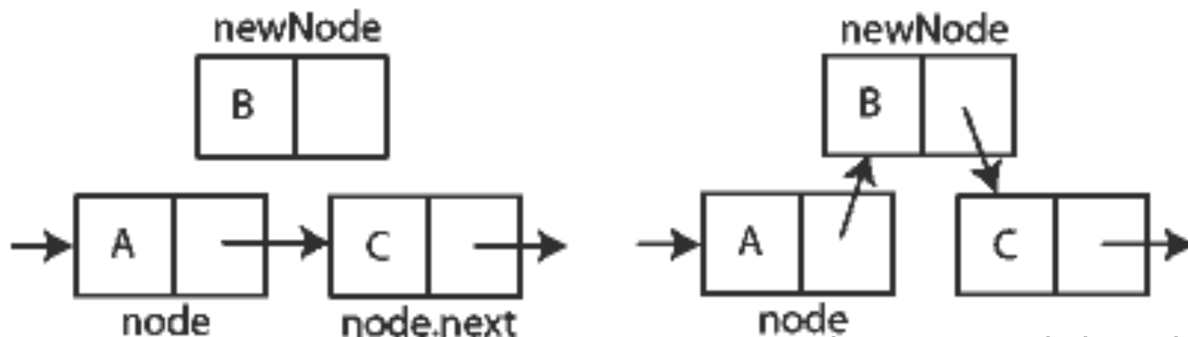
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Another List Data Structure

- What if we have to add/remove data from a list frequently?
- A ***LinkedList*** supports this:
  - Fast insertion and removal of elements
    - Once we know where they go
  - Slow access to arbitrary elements

"random access"



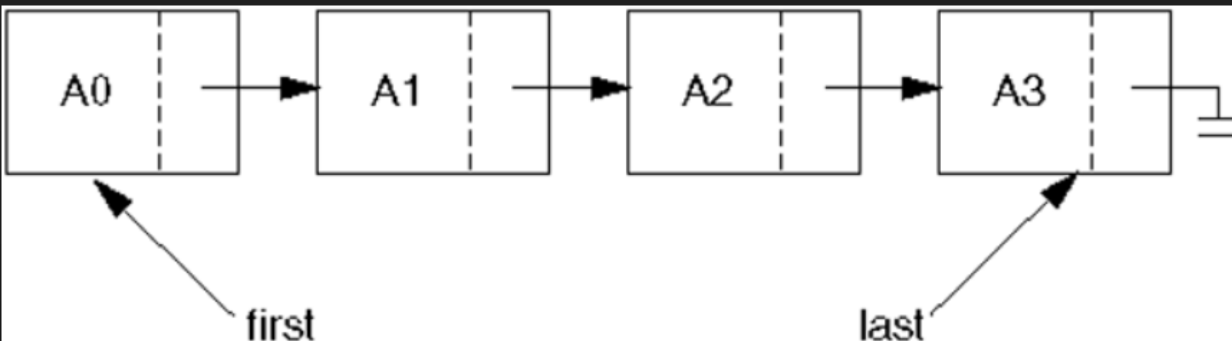


Insertion, per Wikipedia

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

LinkedList implementation of the List Interface

- Stores items (non-contiguously) in nodes; each contains a reference to the next node.
- Lookup by index is linear time (worst, average).
- Insertion or removal is constant time once we have found the location.
  - Insert A4 after A1.
- Even if Comparable list items are kept in sorted order, finding an item still takes linear time.
- Implementing these is fun, will defer until later



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

LinkedList<E> methods

- Manipulating the ends of a list is quick:
  - void addFirst(E e) [Stacks call it push()]
  - void addLast(E e) [Queues call it offer()]
  - E getFirst() [peek() or peekFirst]
  - E getLast() [peekLast()]
  - E removeFirst() [pop() in Stacks, poll() in Queues]
  - E removeLast() [pollLast()]

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

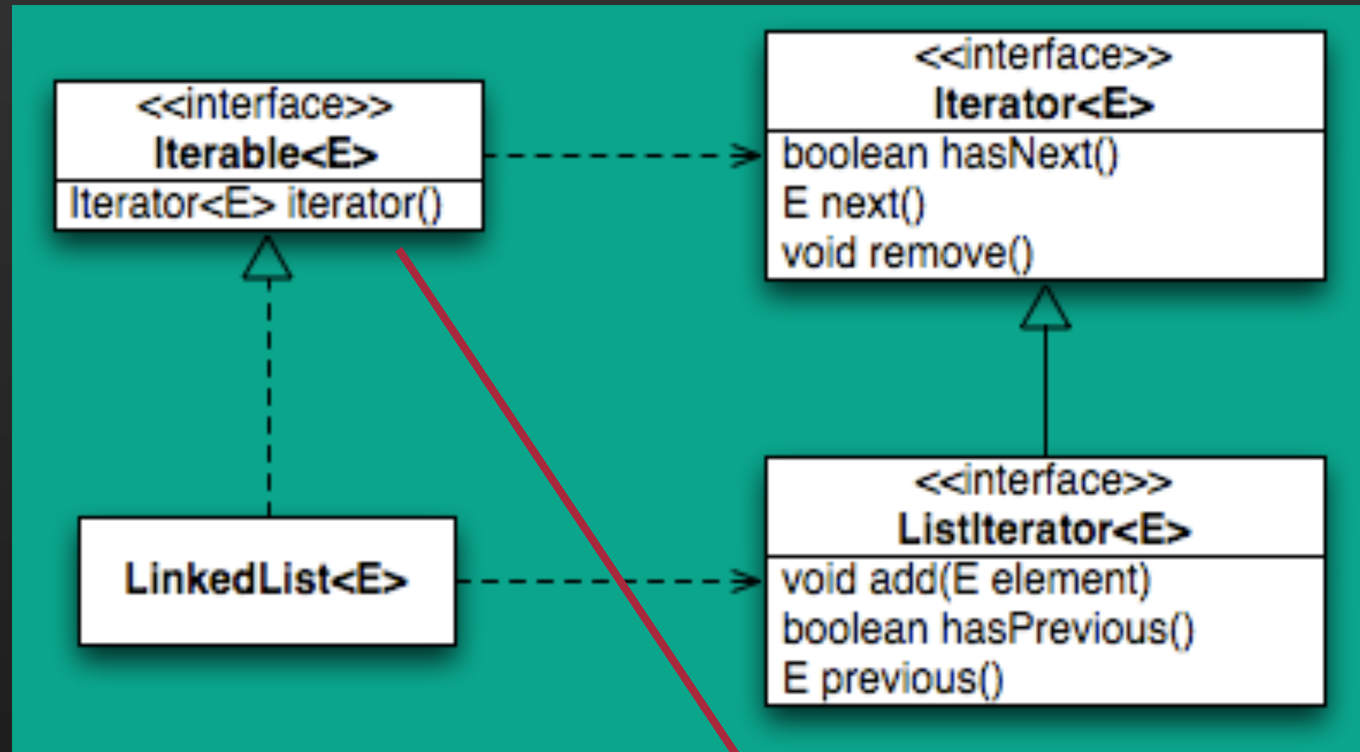
LinkedList<E> iterator

- What if you want to access the rest of the list?
- Iterator<E> iterator()
  - An iterator<E> has methods:
    - boolean hasNext()
    - E next()
    - E remove()

What should remove() remove?

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Accessing the Middle of a LinkedList



- iterator() is what is called a *factory method*: it returns a new concrete iterator, but using an interface type.

An Insider's View

Enhanced For Loop

- ```
  for (String s : list) {
    // do something
  }
  ```

What Compiler Generates

- ```
  Iterator<String> iter =
      list.iterator();

  while (iter.hasNext()) {
  ```
- ```
    String s = iter.next();
  ```
- ```
    // do something
  ```
- ```
  }
  ```
  ```

How to use linked lists and iterators

Demo

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

More with big-Oh

Abstract Data Types

Abstract Data Types (ADTs)

- Boil down data types (e.g., lists) to their essential operations

- Choosing a data structure for a project then becomes:
  - Identify the operations needed
  - Identify the abstract data type that most **efficiently** supports those operations

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Arrays

- Must declare its size when constructed.
- Access items by *index*

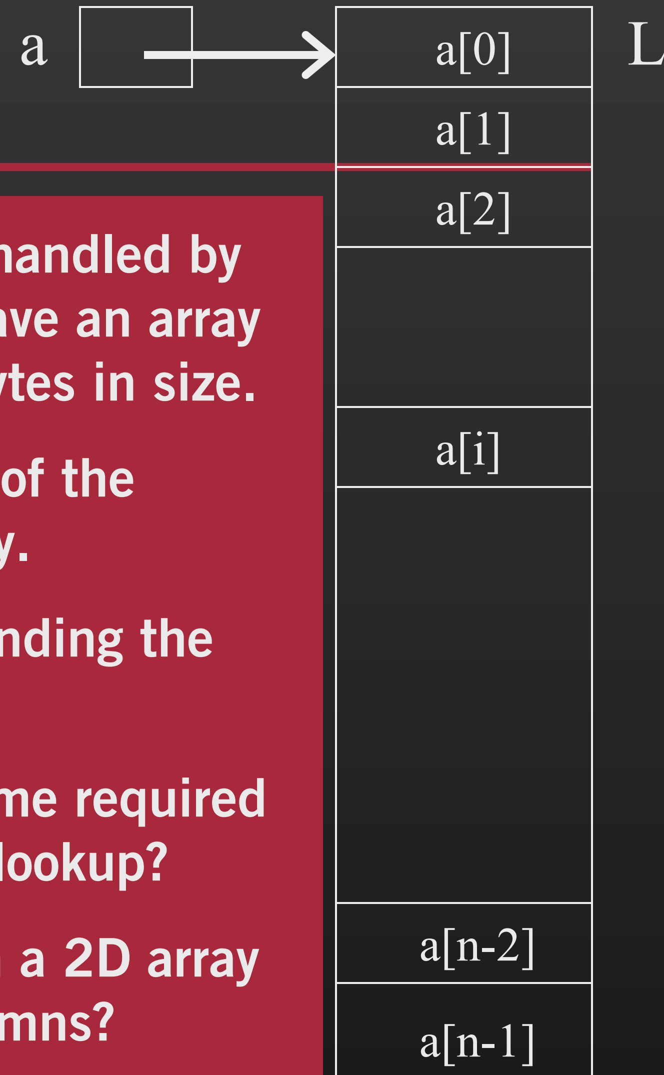
Implementation (handled by the compiler): We have an array of N items, each b bytes in size.

Let L be the address of the beginning of the array.

What is involved in finding the address of a[i]?

What is the Big-oh time required for an array-element lookup?

What about lookup in a 2D array of n rows and m columns?



E-HULMAN
TE OF TECHNOLOGY

ArrayLists use arrays internally

- So O(1) random access

- We said **Array Lists** have
  - Fast addition to end of list
  - Slow inserts to and deletes from middle of list
- Big-Oh runtimes of each?

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Runtimes of LinkedList methods

- void addFirst(E element)
- E getFirst()
- E removeFirst()

- E get(int k)

- To access the rest of the list: Iterator<E> iterator()
  - boolean hasNext()
  - E next()
  - E remove()

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Summary

Operations Provided	Array List Efficiency	Linked List Efficiency
Random access	O(1)	O(n)
Add/remove item	O(n)	O(1)

Which list to use?

Operation	ArrayList	LinkedList
Random access		
Accessing front or rear		
Insert in front		
Insert in middle		
Insert in back		

Which list to use?

Operation	ArrayList	LinkedList
Random access	O(1)	O(n)
Accessing front or rear	O(1)	O(1)
Insert in front	O(n)	O(1)
Insert in middle	O(n)	O(1) once found
Insert in back	O(1) average (O(n) resizing occurs rarely)	O(1)

Common ADTs

- Array List
- Linked List
- Stack
- Queue
- Set
- Map

- Look at the *Collection* interface now.

Implementations for all of these are provided by the Java Collections Framework in the **`java.util`** package.

A longer list

- Array (1D, 2D, …)
- List
  - ArrayList
  - LinkedList
- Stack
- Queue
- Set
- MultiSet
- Map (a.k.a. table, dictionary)
  - HashMap
  - TreeMap
- PriorityQueue
- Tree
- Graph
- Network

What is "special" about each data type?

What is each used for?

What can you say about time required for:
adding an element?
removing an element?
finding an element?

You will know these, inside and out, by the end of CSSE230.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY