CS W3134: Data Structures in Java

Lecture #12: Linked lists cont'd., recursion
10/14/04
Janak J Parekh

Administrivia

- HW#2 due today
- I'll put up both HW#1 and HW#2 solutions before the midterm so you have them as a resource
  - I'd return HW#1 today, but I'm waiting on Matthew. :-/
- HW#3 out this afternoon
- Today's lecture technically last topic for midterm
  - Although we will reinforce today's concepts next time
  - Use the syllabus to help you study

Agenda

- Finish linked list basics
- Start recursion

Doubling up

- Double-ended lists
  - Contains pointer to last element
  - Makes insertLast() much faster (how much?)
- Doubly-linked lists
  - Keep a back (prev) pointer at every node
  - Advantage: faster to go backwards
  - Disadvantage: more memory and bookkeeping
- Be careful of syntax!
  - What does last.prev.next = null mean?

Linked list complexity?

- Similar to arrays
- O(1) insert/delete at beginning (also end of list for double-ended)
- Other operations take O(N), but faster than array if "sliding" is needed in array
- Memory?
  - Linked list more efficient, although it has to keep lots of references

Revisit abstraction

- Book finally covers abstraction here
- We can redo all of our previous data structures, previously *array-backed*, as *linked list-backed*
- *Interface* – high-level contract, while the dirty details are hidden
- How to do a stack?
- How to do a queue?
- You should read through this section

Other linked-list considerations

- Sorted List: how to do?
  - Cases when inserting at beginning, middle, or end
- Sorting an unsorted List
  - Insertion sort is faster than the other two sorts, since "sliding" is very easy to do

Iterators

- With lists, frequently need to walk through a list
  - Increase minimum wages of all employees, etc.
- But there's no array index! How to step through?
- One way is to keep references to current cell, but requires "outsider" to know the internals of how the list works

Iterators (II)

- Structure: list, current, and previous references
- Methods – book suggests:
  - reset() – go back to beginning
  - nextLink()
  - getCurrent()
  - atEnd() – *last* element, not after it
  - insertAfter()
  - insertBefore()
  - deleteCurrent()

Iterators (III)

- Java has its own, simpler, Iterator, with next() and hasNext(), and that's it
  - Supports more than linked lists

Iteration vs. Recursion

- So, what is iteration, anyway?
  - Dictionary.com: "The process of repeating a set of instructions a specified number of times or until a specific result is achieved."
- Any other way of repeating over and over?
- Well, let's think about it…

How to calculate…

- What's the sequence 1, 3, 6, 10, 15, 21, 28, 36…
  - *Triangle* numbers
  - How to do as loop?
  - How to do *as addition on previous result*?
    - Recursion!

A better example

- Simpler, you say?
- What's the sequence 1, 1, 2, 3, 5, 8, …
  - Easy to define in terms of recursion, right?
  - How to iterate over this?
  - In other words, there are problems that are more intuitive recursively

Formalizing Recursion

- Recursive algorithms have the following properties
  - They call themselves
  - They call themselves to solve a smaller problem, and then work with the result
  - There's a *stopping* condition, e.g., a call which is simple enough to solve explicitly
  - Generally avoid explicit loops

Recursion vs. Iteration

- Recursion is, *generally*:
  - A bit less intuitive at first…
  - Simpler to implement / elegant
  - Less efficient
- Conceptually simpler

Some more examples

- FindMax
- Recursive binary search (p. 268)
- Divide-and-conquer approach
  - Take a big problem, split into smaller problems, solve separately
  - Very powerful methodology, works well with recursion
  - Usually two recursive calls

Next time…

- Finish up recursion
  - Mergesort