CS W3134: Data Structures in Java

Lecture #11: Linked lists
10/12/04
Janak J Parekh

Administrivia

- HW#2 questions?
  - enqueue / push / insert
  - dequeue / pop / remove
  - Yes, you can *use* what you dequeue!
- I'll put up HW#1 solutions shortly – I have one situation to resolve
- Midterm next Thursday

Agenda

- Linked lists
- Recursion, if time allows

Linked lists

- Arrays are rather limited, cumbersome data structures – cells are "fixed" together, limited length
- What if we could break apart the cells?
- We *can*!
- In fact, linked list-style structures are used more frequently unless you need very fast random index-based access
- Trees, graphs, etc. are generalizations of linked lists

Linked List structure

- Two basic objects:
  - The list "parent" itself
  - An "element" (book calls "link"), with data
  - Technically, we don't need both
- Parent contains reference to the first element
- *Each element contains a reference to the next element*
- Last element's "next" is set to null
- Meaning of the "." operator, reviewed

Basic Linked List operations

- How to tell if empty?
- Insertions
  - insertFirst()
  - deleteFirst()
  - displayList()
  - insertLast()
- More complex operations
  - How to find an arbitrary element?
  - How to delete arbitrary element?

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!

Next time…

- Continue recursion