CSC207.01 2013F, Class 28: Linked Lists in Java

Overview

- Admin.
- Leftover topics.
- Java iterators.
- Linked lists.
- Implementation details.

Admin

- No readings for the rest of the week. Keep Heck Week sane!
- Assignment 7 available in draft form. Keep Heck Week insane!
- We will probably continue the work a little / talk a little approach, although I'd like to hear your opinions on how it went.
- Upcoming extra credit opportunities
  - Learning from Alumni: Eryn O'Neil '09
  - CS Extras: Max Mindock
  - CS Table: TBD
  - Others?
- Other things
  - Kington convo today

Leftover topics

- When we advanced beyond the end of the list and then inserted, we got some very strange output. Why?
  - We have three important fields:
    - list.values
    - list.size
    - iterator.pos
  - When we advance too far, we are incrementing pos
  - When we insert, we insert at the current pos (beyond the end) and then increment size
  - If we've advanced twice, there's a "hole" in the array
- Intuitively, deletion and insertion can screw up other iterators. How should we handle this?
  - E.g., it1 = stuff.front(); it2 = stuff.front(); s1 = stuff.get(it1); // Code that does not do anything to it1 s2 = stuff.get(it1); // Can we say anything about the relationship
    - What code in the middle might make s1 != s2
      - stuff.delete(it2);
      - stuff.insert("hello", it2);

        - stuff.prepend("ouch");
    - One solution: Every time you insert, delete, and otherwise modify, you can update all the iterators
    - A less painful solution: Postcondition: "All other iterators are now invalid"
    - An alternate strategy: Store the value in the iterator
    - Then you get strange things like s1 = stuff.get(it1); stuff.contains(s1) => FALSE
- Is this iterator valid?
    - Add a "number of mutations" counter to the list. If the number of mutations now is the same as when the iterator was created, the iterator is still valid.
    - If we don't want to invalidate the current iterator, we have to update its mutation count, too.
- Many of our procedures have the precondition that the iterator belongs to the list. How do we verify that precondition?

Java iterators

- Why see what the folks at Sunacle did?
    - Juxtaposing different designs can be useful - Help us think in new ways
    - We can learn from smart people
    - There may be aspects of Java lists that clients will expect of your lists (or other data structures)
    - And those can be incorporated in the languag ein diffeferent ways
- If you class implements Iterable, then you can write for (var : IterableObject) { doSomethingWith(var); }
    - And Java expands it to Iterator it = IterableObject.iterator(); while (it.hasNext()) { var = it.next(); doSomethingWith(Var); }

Linked lists

- Deficiency in array-based lists: Adding is often O(N)
- A different approach makes adding O(1)
- Idea: Linked nodes: Value plus link to next element
- Assume that a cursor is just a link to a node (but you can change that)
- Insertion:
    - Create a new node
    - Link that new node to our successor
    - Link from current node to new node
    - Constant time
- Delete *next* element
    - Make the next pointer the next of the next
- Delete *current* element?
    - Nodes have two pointers, rather than one - PITN
    - Start at the beginning and find the previous element - O(N)
    - Shove a "deleted" in the list, and the next time you iterate, delete the element
    - Some other PITNs

    - Copy data from next node and delete the next node. (Potential drawbacks)
    - Cursors store links to current and previous element
- Insert at front?
- Insert at end?

Implementation details

- Forthcoming