Lecture 11: More Linked Lists

CS 62
Spring 2015
Kim Bruce & America Chambers

Linked List Algos

- Constructor
- addFirst, removeFirst
- get(i)
- indexOf(e)
- add(i,o)
- remove(e), remove(i)
- iterator

What is worst-case complexity of each?

Variants of List

- If add a lot at end, add "tail" pointer
  - Makes adding at end faster
  - But harder to delete at end
  - More special cases -- e.g. add first when empty
  - See implementation when look at queues.
- Circular lists
  - Keep reference/pointer to end rather than beginning
    - What is the difference between adding to end & beginning?
    - getFirst vs getLast?
    - removeLast still hard!
    - How do you know when at end of list if searching?

Doubly-Linked List

- Doubly Linked Lists
  - Previous pointer as well as next
  - Useful if need to traverse in both directions
  - Provided by java.util.LinkedList (but we're using DoublyLinkedList from Bailey)
  - Must change twice as many links when adding or deleting!
  - Our class has head and tail pointers,
    - Doubly-linked lists often represented as circular!

Expectations

- You should be able to write any of these methods in any variant.
- Midterms always include such a question!

Stack

- Interface Stack<E> {
  - void push(E value)
  - E pop()
  - E peek()
- Example: Trays in cafeteria

- Last In - First Out (LIFO)

Stack Applications

- Run-time stack:
  - See sum and quicksort programs
- Backtracking
  - Solving Maze
- Evaluating expression in postfix form:
  - (52 - ((5 + 7) * 4) ⇒ 52 5 7 + 4 * - ⇒ 4
- Tools to parse programs

Stack Implementations

- ArrayList:
  - Which end should be head?
  - How complex for push, pop, peek?
- SinglyLinkedList: *Why not doubly-linked?*
  - Which end should be head?
  - How complex for push, pop, peek?
- Space differences?
  - What if there are several stacks?
- java.util.Stack based on Vector - don't use!