Data Structure in Java - Midterm Review

Paul Blaer

March 3, 2017

Weiss Textbook Chapters

- Chapter 1 (entirely)
- Chapter 2 (entirely)
- Chapter 3 (entirely)
- Chapter 4.1, 4.2, 4.3, 4.4, and 4.6 (note, 4.6 covers tree traversals the way we covered them in class and makes for good supplemental reading)

Material from Outside the Textbook

- Tower of Hanoi
- Josephus Problem

General Concepts

- Abstract Data Types vs. Data Structures.
- Recursion.
- Basic proofs by induction.

Java Concepts

- Basic Java OOP: Classes / Methods / Fields. Visibility modifiers.
- Generics.
- Inner classes (static vs. non-static).
- Interfaces.

- `Iterator`/`Iterable`.
- `Comparable`.

Analysis of Algorithms

- Big-O notation for asymptotic running time: $O(f(n))$, $\Theta(f(n))$, $\Omega(f(n))$.
- Typical growth functions for algorithms.
- Worst case, best case, average case.
- *Skills:* Compare growth of functions using big-O notation. Given an algorithm (written in Java), estimate the asymptotic run time (including nested loops and simple recursive calls).
- Basic understanding of recursion (Towers of Hanoi, Binary Search) and runtime behavior of recursive programs. Logarithms in the runtime. Tail recursion.

Lists

- List ADT, including typical List operations.
- ArrayList:
  - running time for insert, remove, get, contains at different positions in the list.
  - increasing the array capacity when the array is full.
- LinkedList:
  - single vs. doubly linked list.
  - running time for insert, remove, get, contains at different positions in the list.
  - sentinel (head/tail) nodes.
- *Skills:* Implement iterators. Implement additional algorithms on lists (removing duplicates, intersection, etc.).
- Lists in the Java Collections API.

Stacks and Queues

- Stack ADT and operations (push, pop, peek). LIFO.
- Queue ADT and operations (enqueue, dequeue). FIFO.
- All operations should run in $O(1)$.

- Stack implementation using List data structures, and directly on an array.
- Stack applications:
  - symbol balancing, detecting palindromes, ...
  - reordering sequences (in-order to post-order, train cars,...).
  - storing intermediate computations on a stack (evaluating post-order expressions).
  - building expression trees.
- Tail recursion.
- Queue implementation using Linked List.
- Stacks and Queues in the Java Collections API (java.util.LinkedList supports all stack operations).
- *Skills:* Implement stacks and queues. Use stacks and queues in applications.

Trees

- Tree terminology (parent, children, root, leafs, path, depth, height)
- Different tree implementations (one field per child, general trees: siblings as linked list).
- Binary trees:
  - full / complete / perfect binary trees.
  - tree traversals: in-order, pre-order, post-order.
  - expression trees - pre-fix, post-fix (reverse Polish notation), and in-fix notation.
  - Constructing an expression tree using a stack.
  - Relation between number of nodes and height of a binary tree.
  - Inductive proofs over binary trees.
- *Skills:* Perform tree traversals on paper. Implement different tree traversals using recursion (different versions). Use these traversals to implement operations on trees. Convert between in-fix, post-fix, pre-fix notation using a tree. Simple inductive proofs for tree properties.

Binary Search Trees

- BST property.
- BST operations: contains, findMin, findMax, insert, remove
- Runtime performance of these operations, depending on the height of the tree.
- Lazy Deletion
- *Skills:* Perform BST operations on paper.

AVL Trees

- Balanced BSTs. AVL balancing property.
- Maintaining AVL balance property on insert:
  - Outside imbalance, single rotation.
  - Inside imbalance, double rotation.
  - Verifying that a tree is balanced. Finding the location of an imbalance (bottom-up).
- *Skills:* Perform AVL rotations on paper, detect imbalances.