COMS W3137 - Honors Data Structures

Midterm Review - Spring 22

Daniel Bauer

March 1, 2022

Topics marked with will not be tested on the midterm.

Weiss Textbook Chapters

- Chapter 1 (entirely)
- Chapter 2 (entirely)
- Chapter 3 (entirely)
- Chapter 4.1, 4.2, 4.3, 4.4, and 4.6 (and 4.7*)

Material from Outside the Textbook

- Towers of Hanoi.
- Amortized Analysis (the Weiss textbook discusses Amortized Analysis in chatper 11, but using data structures we did not yet discuss in class. Instead, please refer to the chapter from the Cormen, Leiserson, Rivest, and Stein Algorithms textbook, which you can find as a PDF on Courseworks).
- Immutable data structures.
- Scala (see below, please refer to the recitation nodes).

General Concepts

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

Java Concepts

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

Scala Concepts

There will not be a Scala programming problem on the midterm.

- REPL vs. compiled scala code.
- `var` vs. `val`.
- Everything is an expression.
- Basic functional programming: First class and higher-order functions, function literals (vs. methods defined with `def`).
- Basic OOP in Scala (classes, methods, fields/instance variables).
- Case classes and pattern matching.
- Multiple return values with tuples.
- Immutable lists in scala (and using them as stacks).
- map, and folds.
- Banker's queue.
- Binary tree implementation and tree traversals.

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.

- Recursion (Towers of Hanoi, recursive Fibonacci implementation, Binary Search) and runtime behavior of recursive programs. Logarithms in the runtime. Tail recursion.
- *Skills:* Compare growth of functions using big-O notation. Given an algorithm (written in Java or Scala), estimate the asymptotic run time (including nested loops and simple recursive calls).
- Basic understanding of amortized analysis (Aggregate method, Banker's method, Physicists/Potential method).*.

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 (beginMarker/head and endMarker/tail) nodes.
- *Skills:* Implement iterators. Implement additional list operations (such as reversal, but think of others, such as removing duplicates 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, directly on an array, or using immutable lists.
- Stack applications:
  - Symbol balancing, detecting palindromes.

  - Reordering sequences (in-order to post-order etc.).
  - Storing intermediate computations on a stack (evaluating post-order expressions).
  - Building expression trees.
- Method call stack, stack frames *, relation between stacks and recursion.
- Tail recursion.
- Queue implementation using Linked List.
- Queue implementation using a Circular Array.
- Banker's queue.
- 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

- Recursive definition of a tree.
- Tree terminology (parent, children, root, leafs, path, depth, height)
- Different tree implementations (one instance variable per child, list of children, 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 (a.k.a. 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.
  - Structural induction over binary trees (two versions: induction over height, induction over number of nodes).
- *Skills:* Perform tree traversals on paper. Implement different tree traversals using recursion. Use these traversals to implement operations on trees (for example, computing tree properties, such as height). Convert between in-fix, post-fix, pre-fix notation using a tree. Structural induction proofs for binary tree properties.

Binary Search Trees

- Map ADT.
- BST property.
- BST operations: contains, findMin, findMax, insert, remove (three cases for remove).
- Runtime performance of these operations, depending on the height of the tree.
- Lazy deletion *.
- *Skills:* Perform BST operations (contains, insert, delete) 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.

B-Trees *