COMS W3137 - Honors Data Structures

Final Review - Spring 22

Daniel Bauer

April 27, 2022

Topics marked with * will not be tested on the final.

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*)
- Chapter 5.1 through 5.5.
- Chapter 6.1 through 6.5, and 6.6.
- Chapter 7.1 through 7.3, 7.5 through 7.7, and 7.11
- Chapter 8.1 through 8.5 (this is about the disjoint set. Not all details are needed, just the basic data structure with path compression).
- Chapter 9.1 through 9.3.3, 9.5, 9.6.2*, 9.6.3., 9.7
- Chapter 10.1.2, 10.2.1, 10.3.1

Material from Outside the Textbook

- Towers of Hanoi.
- Amortized Analysis (the Weiss textbook discusses Amortized Analysis in chapter 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).
- Longest increasing subsequence problem.

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.
- Leftist heaps.

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.
- Master theorem for divide and conquer algorithms *.
- Basic understanding of amortized analysis (Aggregate method, Banker's method, Physicists/Potential method).*.
- *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). Solving recurrences using the recursion tree method and 'unrolling'.

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 *

Hash Tables

- Hash functions.
- Collision Resolution Strategies
  - Separate Chaining
  - Linear Probing
  - Quadratic Probing
  - Double Hashing
- Rehashing
- Lazy deletion
- HashSets and HashMaps in the Java Collections API *.
- Quadratic Probing Theorem *
- *Skills:* Perform hash inserts, lookups, and rehashes on paper. Understand the consequences of the various collision resolution strategies. Be able to recognize a bad hash function. Implement hashCode and equals in Java classes.

Priority Queues

- Binary Heaps.
- Tree and Array representations of Binary Heap.
- Complete tree and heap order property.
- Min-heaps vs. Max-heaps.
- Min-max heaps *.
- The insert and deleteMin/deleteMax operations.
- Percolate up and percolate down.
- Calculating the parent and child nodes in the array representation.
- The buildheap/heapify operation and its cost.
- The decreaseKey operation and why it is useful.
- Leftist heaps (leftist property, merge operation and how to use it to implement insert and deleteMin).
- *Skills:* Perform heap operations (insert/deleteMin/buildheap) on paper. Merge two leftists heaps. Understand BinaryHeap implementation in Java.

Sorting

- Insertion sort
- Selection sort (not in the Weiss book)
- Heap Sort (in-place, that is, with replacement)
- Merge Sort, recursive implementation
- Iterative Merge Sort *.
- Quick Sort, partitioning, and pivot selection (median-of-three).
- $\Omega(nlogn)$ lower-bound for comparison-based sorting. (decision tree proof *)
- Bucket sort, counting sort, radix sort.
- *Skills:* Know the big-O costs of all of the algorithms listed (best and worst case). Be able to perform the sorting algorithms on paper. Understand when it might be appropriate to use one algorithm over the other. Implement basic sorting algorithms in Java, or at least understand the various implementations discussed in class.

Graphs

- Basic definitions - vertices and edges, directed vs. undirected, weighted vs. unweighted, Directed Acyclic Graphs (DAGs), connectivity (strong vs. weak in the case of directed graphs), complete graphs. Dense vs. sparse.
- Graph representations: adjacency matrix vs. adjacency lists.
- Topological Sorting on DAGs.
- Earliest completion time on event node graphs *.
- Breadth First Search (BFS) vs. Depth First Search (DFS).
- Single-source unweighted shortest paths using BFS.
- Dijkstra's algorithm for single-source weighted shortest paths.
- Minimum Spanning Trees:
  - Prim's algorithm
  - Kruskal's algorithm (using union-find a.k.a. disjoint set data structure).
- DFS spanning trees.
- Biconnected components and articulation points *.

- Euler paths/circuits (preconditions and linear time algorithm for finding them).
- Hamiltonian cycles.
- The Traveling Salesperson Problem (TSP).
- deterministic vs. non-deterministic machines *.
- P vs. NP, NP-hard and NP-complete.
- Reductions. (* you will not be asked to do a reduction on the final, but you should understand their significance).
- Nearest Neighbor Approximation vs. Brute Force Solution to TSP.
- MST approximation for TSP *. 2-opt *.
- Undecidability and the Halting Problem *.
- *Skills:* Perform topological sort, BFS for unweighted shortest paths, Dijkstra's, Prim's on paper (table format discussed in class and on homework 4). Compute a DFS spanning tree. Perform the Euler circuit algorithm on paper. Understand various graph implementations in Java.

Algorithm Design Strategies

- Greedy algorithms (examples include nearest neighbor approximation for TSP, Dijkstra's/Prim's, huffman coding).
- Divide-and-conquer algorithms and the master theorem (*).
- Dynamic programming
  - Fibonacci sequence.
  - Longest increasing subsequence.
  - Edit distance *.
- *Skills:* Perform LCS DP algorithm on paper (fill out table).