Data Structure in Java - Final Review

Daniel Bauer

December 6, 2015

This review sheet only lists topics covered after the midterm. Make sure to revisit the midterm review sheet as well.

Priority Queues / Heaps

- Priotity Queue ADT including typical operations (`insert`, `deleteMin`).
- Implementation as a Heap:
  - Storing a complete binary tree in an array. Calculating parent/child addresses.
  - MinHeap vs. MaxHeap.
  - Implementation of insert and delete using percolate up/down.
  - Building a heap bottom-up in O(N).
- Algorithms that use Priority Queues
  - Selecting the k-th largest element.
  - Retaining the k-largest elements.
  - HeapSort.
  - Greedy algorithms (e.g. Dijkstra's, Prim's, Kruskal's, Huffman Code...).

Sorting

- Comparison-based sorting (e.g. insertion sort) vs. count-based sorting (e.g. bucket sort).
- Sorting algorithms. Need to know run time (worst/best/average), space requirements, stability.
  - Insertion Sort.
  - Heap Sort.

- Merge Sort: Top-down divide-and-conquer approach. Iterative bottom-up version.
- Quick Sort. Median-of-Three pivot selection strategy.

Comparison-Based Sorting Algorithms

	T_{Worst}	T_{Best}	T_{Avg}	Space	Stable?
Insertion Sort	$\Theta(N^2)$	$\Theta(N)$	$\Theta(N^2)$	$O(1)$	✓
Shell Sort	$\Theta(N^{3/2})^*$	$\Theta(N)$	$\Theta(N^{3/2})^*$	$O(1)$	✗
Heap Sort	$\Theta(NlogN)$	$\Theta(NlogN)$	$\Theta(NlogN)$	$O(1)$	✗
Merge Sort	$\Theta(NlogN)$	$\Theta(NlogN)$	$\Theta(NlogN)$	$O(N)$	✓
Quick Sort	$\Theta(N^2)$	$\Theta(NlogN)$	$\Theta(NlogN)$	$O(N)$	✗

*depends on increment sequence 23 gray entries: not shown in class

- Bucket Sort & Radix Sort.

Graphs

- Basic concepts:
  - Vertices, Edges. Adjacency relation.
  - Directed vs. undirected graphs.
  - Weighted vs. unweighted graphs.
  - Paths. Simple paths.
  - Cycles. Directed Acyclic Graphs (DAGs).
  - Connectivity: Weak and Strong connectivity in directed graphs.
  - Complete Graphs.
- Graph data structures
  - Adjacency matrices vs. adjacency lists.
  - Storing information in vertex objects vs. storing them in separate tables.
- Graph Traversals
  - Depth First Search (DFS) using a Stack or recursion.
  - Breadth First Search (BFS) using a Queue.

- Single source shortest paths
  - BFS for unweighted graphs.
  - Dijkstra's for weighted graphs, using a Heap.
  - Using backpointers to retrieve the shortest path.
  - Effect of negative weight edges.
- Algorithms on DAGs
  - Computing topological order.
  - Critical path analysis on event-node graphs. Computing earliest completion time.
- Spanning Trees
  - Minimum spanning trees (MSTs).
  - Prim's algorithm.
  - Kruskal's algorithm.
  - Hierarchical clustering using MSTs.
- Applications of DFS
  - Definition of Euler Circuit/Path and Hamiltonian Circuit/Path.
  - Conditions for Euler Circuits and Euler Paths.
  - Repeated DFS to find Euler Circuits.
  - Connectivity: Use DFS to determine if a graph is connected.
  - Biconnectivity:
    * Articulation points, biconnected components.
    * DFS spanning trees with back-edges.
    * Determining biconnected components using the DFS spanning tree.
  - Strongly connected Components:
    * Determine if a graph is strongly connected.
    * Finding strongly connected components using a stack.

Types of algorithms

- Greedy algorithms:
  * Usually using a heap.
  * List example algorithms.
  * Huffman code. Huffmans' algorithm.

- Divide and Conquer:
  - Divide problem into easier subproblems, solve subproblems (usually recursively), then combine solutions.
  - List example algorithms.
  - Understand recurrence relation. Know what the Master Theorem is.
- Dynamic Programming:
  - Basic concept: cache and re-use solutions to sub problems.
  - List example algorithms.
  - Solving the Coin-Change problem.
  - Computing Edit Distance.