Lecture 13: Review for Midterm

The midterm exam will be Thurs, Mar 15. The exam is closed-book and closed-notes, but you will be allowed one sheet of notes (front and back).

Read: You are responsible only for the material covered in class, but it is a good idea to consult Weiss's book and Samet's notes for additional insights on the concepts discussed in class. We have covered Section 1.2, Chapters 2, 3, 4, and Section 10.4.2 in Weiss and portions of Chapter 1 and Chapter 5 (Sections 5.1 and 5.2) in Samet's notes.

Overview: So far this semester we have covered some of the basic elements of data structures and search trees. Here is an overview of the major topics that we have seen. We have also discussed Java, but you are not responsible for this on the exam.

Basics of Analysis: Asymptotic notation, induction proofs, summations, and recurrences.

Basic Data Structures: Stacks, queues, deques, graphs (undirected and directed) and their representations. In particular we discussed both sequential (array) and linked allocations for these objects. For graphs and digraphs we discussed the adjacency matrix and the adjacency list representations and Prim's algorithm for computing minimum spanning trees.

Trees: We discussed multiway and binary trees and their representations. For complete trees we discussed sequential allocation (as used in the binary heap) and discussed traversals and threaded binary trees.

Search Trees: Next we discussed the dictionary abstract data type and search trees (as well as skip lists). We presented a number of data structures, each of which provided some special features.

Unbalanced binary search trees: Expected height $O(\log n)$ assuming random insertions. Expected height over a series of random insertions and deletions is $O(\sqrt{n})$, but this is due to the skew in the choice of replacement node (always selecting the minimum from the right subtree). If we randomly select min from right, and max from left, the trees will be $O(\log n)$ height on average (assuming random insertions and deletions).

AVL trees: Height balanced binary search trees. These guarantee $O(\log n)$ time for insertions, deletions, and finds. Use single and double rotations to restore balance. Because deletion is somewhat more complex, we discussed lazy deletion as an alternative approach (in which deleted nodes are just marked rather than removed).

Splay trees: Self organizing binary search trees, which are based on the splay operation, which brings a given key to the root. Splay trees guarantee $O(m \log n)$ time for a series of m insertions, deletions, and finds. The tree stores no balance information or makes any structural assumptions. Any one operation could be slow, but any long series of operations will be fast. It is a good for data sets where the frequency of access of elements is nonuniform (a small fraction of elements are accessed very often), since frequently accessed elements tend to rise to the top of the tree.

Skip lists: A simple randomized data structure with $O(\log n)$ expected time for insert, delete, and find (with high probability). Unlike balanced binary trees, skip lists perform this well on average, no matter what the input looks like (in other words, there are no bad inputs, just unlucky coin tosses). They also tend to be very efficient in practice because the underlying manipulations are essentially the same as simple linked list operations.

B-trees: Widely considered the best data structure for disk access, since node size can be adjusted to the size of a disk page. Guarantees $O(\log n)$ time for dictionary operations (but is rather messy to code). The basic restructuring operations are node splits, node merges, and key-rotations.