CS 240: Data Structures and Data Management
Module 4 Study Guide

Taylor J. Smith — Spring 2017

Key Concepts

- A **dictionary** is a collection of (typically unique) key-value pairs.
- Dictionary operations:
  - $\textsc{search}(k)$, searches for a key k—$\Theta(n)$ for unordered arrays and lists, $\Theta(\log(n))$ for ordered arrays
  - $\textsc{insert}(k, v)$, inserts key k with value v—$\Theta(1)$ for unordered arrays and lists, $\Theta(n)$ for ordered arrays
  - $\textsc{delete}(k)$, deletes key k—$\Theta(n)$ for unordered and ordered arrays and lists
- **Binary search trees** can be used to implement dictionaries, but their height may affect operation runtimes.
- **AVL trees** are BSTs with a balancing property to keep the height of the tree reasonable.
  - The balancing property ensures the height of an AVL tree with n nodes is $\Theta(\log(n))$
  - Operations on AVL trees are identical to BSTs, but we must also update the **balance factors** of nodes and possibly rebalance the tree
  - A node's balance factor $\in \{-1, 0, 1\}$ means the tree rooted at that node is balanced, but a balance factor $\in \{-2, 2\}$ means we must rebalance at that node
- Rebalancing is achieved using **rotations**.
  - Right rotations fix left-left imbalances
  - Left rotations fix right-right imbalances
  - Double-right rotations fix left-right imbalances (left, then right rotation; *not* two right rotations)
  - Double-left rotations fix right-left imbalances (right, then left rotation; *not* two left rotations)

Suggested Readings

- **Sedgewick:** 12.1 (Symbol-Table Abstract Data Type), 12.5 (Binary Search Trees), 12.6 (Performance Characteristics of BSTs)
- **CLRS:** Chapter 12 (Binary Search Trees)
- **Goodrich/Tamassia:** 3.1 (Ordered Dictionaries and Binary Search Trees), 3.2 (AVL Trees)

Practice Questions

Sedgewick

12.46. Draw the BST that results when you insert items with the keys E A S Y Q U T I O N, in that order, into an initially empty tree.

12.47. Draw the BST that results when you insert items with the keys E A S Y Q U E S T I O N, in that order, into an initially empty tree.

12.49. Inserting the keys in the order A S E R H I N G C into an initially empty tree gives the tree in Figure 12.6. Give 10 other orderings of these keys that produce the same result.

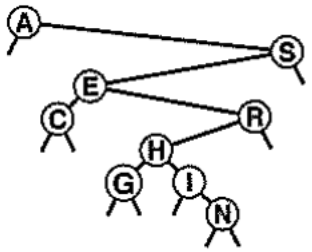

Figure 12.6: BST construction

12.54. Modify our BST implementation to keep items with duplicate keys in linked lists hanging from tree nodes. Change the interface to have SEARCH operate like SORT (for all the items with the search key).

CLRS

12.1-1. For the set of $\{1, 4, 5, 10, 16, 17, 21\}$ of keys, draw binary search trees of heights 2, 3, 4, 5, and 6.

12.1-2. What is the difference between the binary-search-tree property and the min-heap property? Can the min-heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Show how, or explain why not.

12.1-4. Give recursive algorithms that perform preorder and postorder tree walks in $\Theta(n)$ time on a tree of n nodes.

12.2-1. Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could *not* be the sequence of nodes examined?

(a) 2, 252, 401, 398, 330, 344, 397, 363.

(b) 924, 220, 911, 244, 898, 258, 362, 363.

(c) 925, 202, 911, 240, 912, 245, 363.

(d) 2, 399, 387, 219, 266, 382, 381, 278, 363.

(e) 935, 278, 347, 621, 299, 392, 358, 363.

Goodrich/Tamassia

R-3.1. Insert items with the following keys (in the given order) into an initially empty binary search tree: 30, 40, 24, 58, 48, 26, 11, 13. Draw the tree after each insertion.

R-3.4. Is the rotation done in Figure 3.12 a single or a double rotation? What about the rotation done in Figure 3.15?

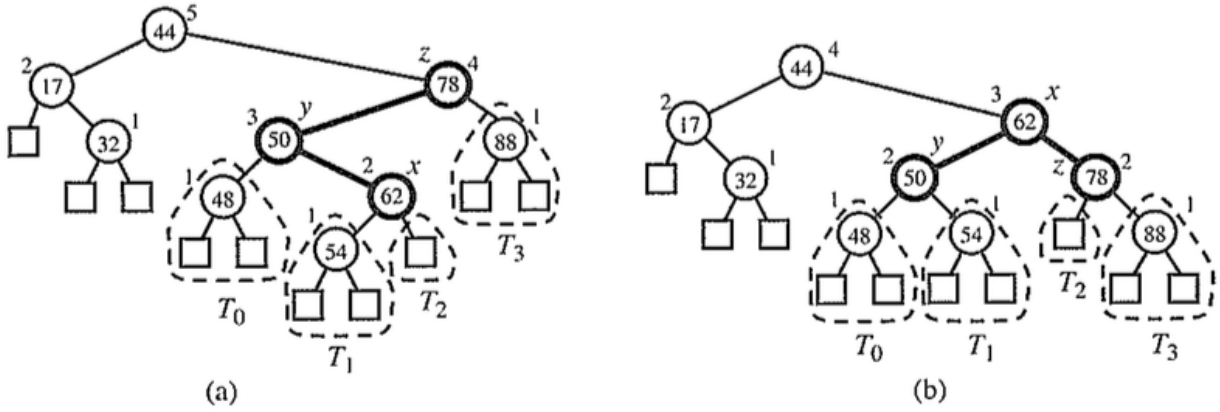

Figure 3.12: An example insertion of an element with key 54 in an AVL tree

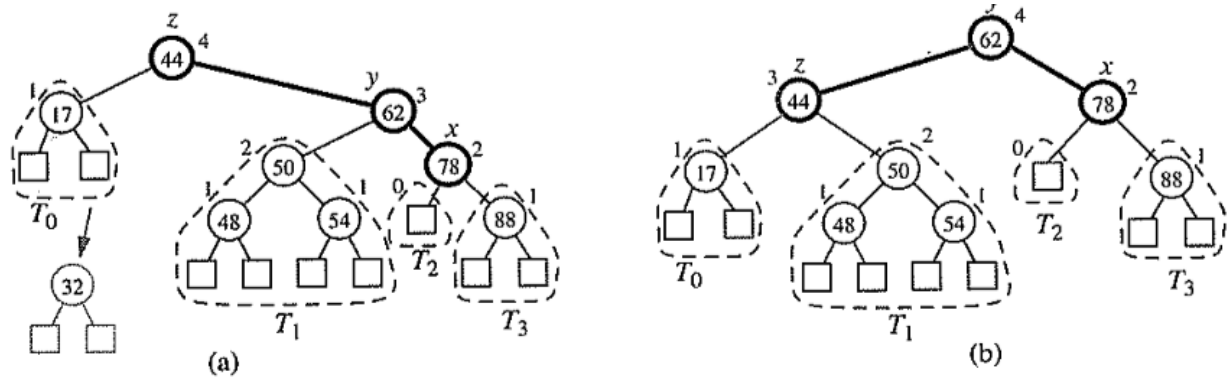

Figure 3.15: Removal of the element with key 32 from the AVL tree

R-3.5. Draw the AVL tree resulting from the insertion of an item with key 52 into the AVL tree of Figure 3.15(b).

R-3.6. Draw the AVL tree resulting from the removal of the item with key 62 from the AVL tree of Figure 3.15(b).

C-3.1. Suppose we are given two ordered dictionaries S and T, each with n items, and suppose that S and T are implemented by means of array-based ordered sequences. Describe an $O(\log(n))$-time algorithm for finding the kth smallest key in the union of the keys from S and T (assuming no duplicates).

C-3.11. Let D be an ordered dictionary with n items implemented with an AVL tree. Show how to implement the following method for D in time $O(\log(n))$:

CountAllInRange(k_1, k_2): Compute and return the number of items in D with key k such that $k_1 \leq k \leq k_2$.

Note that this method returns a single integer.

Hint: You will need to extend the AVL tree data structure, adding a new field to each internal node and ways of maintaining this field during updates.