Data Structures and Algorithms in Java™

Sixth Edition

Michael T. Goodrich
Department of Computer Science
University of California, Irvine

Roberto Tamassia
Department of Computer Science
Brown University

Michael H. Goldwasser
Department of Mathematics and Computer Science
Saint Louis University

Study Guide: Hints to Exercises

WILEY

Chapter

11 Search Trees

Hints

Reinforcement

R-11.1) Recall the definition of where we perform an insertion in a binary search tree.

R-11.2) You will need to draw 8 trees, but they are all small.

R-11.3) You can enumerate them with pictures.

R-11.4) Try a few examples of five-entry binary search trees.

R-11.5) Try a few examples of five-entry AVL trees.

R-11.6) Use a loop to express the repetition

R-11.7) There is one of each type. Which one is which?

R-11.8) Mimic the figure in the book.

R-11.9) Mimic the figure in the book.

R-11.10) Think about the data movements needed in an array list representation of a binary tree.

R-11.11) Carefully note the heights of all subtrees before the deletion, and after the deletion but before the restructuring.

R-11.12) Carefully note the heights of all subtrees before the deletion, and after the deletion but before the restructuring.

R-11.13) Carefully trace the potential heights of various subtrees.

R-11.14) Each entry is splayed to the root in increasing order.

R-11.15) Use a pencil with a good eraser.

R-11.16) Perform the splay operation on the lowest entry accessed for each operation.

R-11.17) No. Why not?

R-11.18) It is not k_1. Why?

R-11.19) You will need at list five entries to find a counterexample.

R-11.20) Use the correspondence rules described in the chapter.

R-11.21) Use a pencil with a good eraser.

R-11.22) Use a pencil with a good eraser.

R-11.23) Consider looking at the (2,4) tree and red-black tree definitions again.

R-11.24) Some have $O(\log n)$ worst-case height and some have $O(n)$ worst-case height. Make sure you know which. Also, try to get the constant factors right in this case.

R-11.25) You need to create a node that does not satisfy the AVL balance condition, but would be acceptable in a red-black tree. A good example would be a tree with at least 6 nodes, but no more than 16.

R-11.26) Case 2 is the only one that is repeated.

R-11.27) Case 2 is the only one that might be applied more than once.

Creativity

C-11.28) Recall the definition of a binary search tree, in general.

C-11.29) The method is similar to priority-queue sorting.

C-11.30) Review what it means for splay trees to have $O(\log n)$ amortized time performance.

C-11.31) The goal is to perform a single tree search.

C-11.32) Show that $O(n)$ rotations suffice to convert any binary tree into a ***left chain***, where each internal node has an external right child.

C-11.33) Where might the search path for k diverge from a path to one of the other keys?

C-11.34) Consider the maximum number of times the recursive method is called on a position that is not within the subrange.

C-11.35) Consider a top-down recursive approach.

C-11.36) Make sure that the result is a valid AVL tree.

C-11.37) Note that this method returns a single integer, so it is not necessary to visit all s entries that lie in the range. You will need to extend the tree data structure, adding a new field to each node.

C-11.38) How do the rebalancing actions affect the information stored at each node?

C-11.39) Use triangles to represent subtrees that are not affected by this operation, and think of how to cascade the imbalances up the tree.

C-11.40) Study closely the balance property of an AVL tree and the rebalance operation. Also, make a node high up in tree have its AVL balance depend on the node that just got inserted.

C-11.41) Study closely the balance property of an AVL tree and the rebalance operation.

C-11.42) Think carefully about how the balance of a node and its ancestors changes immediately after an insertion or deletion.

C-11.43) Just consider the operations that could change the leftmost position or who points to it.

C-11.44) How do the rebalancing actions affect the minimum?

C-11.45) These operations will be easier if you know the size of each subtree.

C-11.46) Is it possible for an splay tree to also be a red-black tree?

C-11.47) Find the right place to "splice" one tree into the other to maintain the (2,4) tree property. Also, it is okay to destroy the old versions of T and U.

C-11.48) Search down for k and cut along this path. Now consider how to "glue" the pieces back together in the right order.

C-11.49) You don't need to use induction here.

C-11.50) Think about a way of using the structure of the binary search tree itself to indicate color.

C-11.51) Consider the red and black meaning of the three possible balance factors in an AVL tree.

C-11.52) Since you know the node x will eventually become the root, maintain a tree of nodes to the left of x and a tree of nodes to the right of x, which will eventually become the two children of x.

C-11.53) The analysis in the book works also for half-splay trees, with minor modifications.

C-11.54) If you are having trouble with this problem, you may wish to gain some intuition about splay trees by "playing" with an interactive splay tree program on the Internet.

Projects

P-11.55) One of the biggest challenges is what to do for an unsuccessful tree search.

P-11.56) Consider having an entry instance keep a reference to the node at which it is stored.

P-11.57) We've provided the implementations; you need to develop the experiment.

P-11.58) In this case, you will need a skip list implementation to test.

P-11.59) The order is implicit in the tree, so adding these methods should not be hard.

P-11.60) Use a recursive method to do the conversion.

P-11.61) The most significant challenge is how to handle the insertion of duplicate, given that the original tree search will stop when it finds the existing key.

P-11.62) Review the cases for zig-zag, zig-zig, and zig. Make sure you do splaying right before doing anything else.

P-11.63) First figure out a way that works assuming that all keys in existing mergeable heaps are distinct, and then work out how this is not strictly necessary.