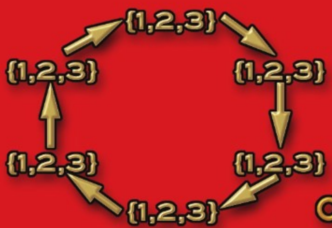
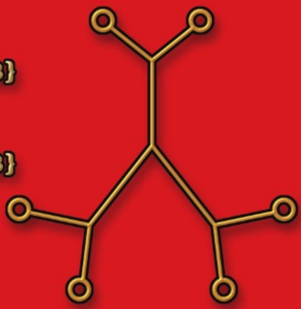
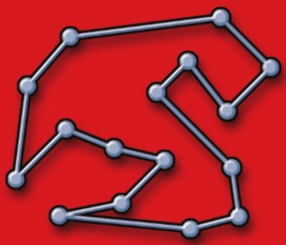
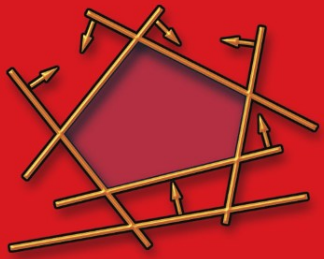
Second Edition

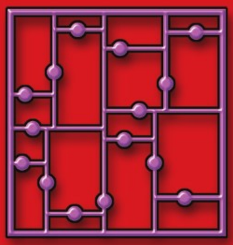
THE Algorithm Design MANUAL

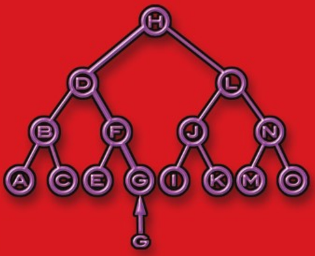












Steven S. Skiena

Springer

The Algorithm Design Manual

Second Edition

Steven S. Skiena

The Algorithm Design Manual

Second Edition

Springer

Steven S. Skiena
Department of Computer Science
State University of New York
at Stony Brook
New York, USA
skiena@cs.sunysb.edu

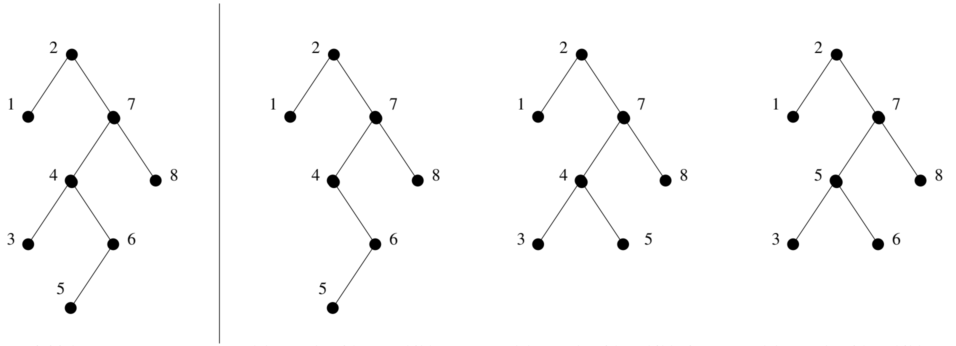
ISBN: 978-1-84800-069-8 e-ISBN: 978-1-84800-070-4
DOI: 10.1007/978-1-84800-070-4

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2008931136

Printed on acid-free paper

Springer Science+Business Media
springer.com



initial tree

delete node with zero children (3)

delete node with 1 child (6)

delete node with 2 children (4)

Figure 3.4: Deleting tree nodes with 0, 1, and 2 children

Deletion from a Tree

Deletion is somewhat trickier than insertion, because removing a node means appropriately linking its two descendant subtrees back into the tree somewhere else. There are three cases, illustrated in Figure 3.4. Leaf nodes have no children, and so may be deleted by simply clearing the pointer to the given node.

The case of the doomed node having one child is also straightforward. There is one parent and one grandchild, and we can link the grandchild directly to the parent without violating the in-order labeling property of the tree.

But what of a to-be-deleted node with two children? Our solution is to relabel this node with the key of its immediate successor in sorted order. This successor must be the smallest value in the right subtree, specifically the leftmost descendant in the right subtree. Moving this to the point of deletion results in a properly-labeled binary search tree, and reduces our deletion problem to physically removing a node with at most one child—a case that has been resolved above.

The full implementation has been omitted here because it looks a little ghastly, but the code follows logically from the description above.

The worst-case complexity analysis is as follows. Every deletion requires the cost of at most two search operations, each taking $O(h)$ time where h is the height of the tree, plus a constant amount of pointer manipulation.

3.4.2 How Good Are Binary Search Trees?

When implemented using binary search trees, all three dictionary operations take $O(h)$ time, where h is the height of the tree. The smallest height we can hope for occurs when the tree is perfectly balanced, where $h = \lceil \log n \rceil$. This is very good, but the tree must be perfectly balanced.