Data Structures & Algorithms

in Python

Michael T. Goodrich • Roberto Tamassia • Michael H. Goldwasser

Data Structures and Algorithms in Python

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

WILEY

VP & PUBLISHER	Don Fowley
EXECUTIVE EDITOR	Beth Lang Golub
EDITORIAL PROGRAM ASSISTANT	Katherine Willis
MARKETING MANAGER	Christopher Ruel
DESIGNER	Kenji Ngieng
SENIOR PRODUCTION MANAGER	Janis Soo
ASSOCIATE PRODUCTION MANAGER	Joyce Poh

This book was set in LaTeX by the authors. Printed and bound by Courier Westford.
The cover was printed by Courier Westford.

This book is printed on acid free paper.

Founded in 1807, John Wiley & Sons, Inc. has been a valued source of knowledge and understanding for more than 200 years, helping people around the world meet their needs and fulfill their aspirations. Our company is built on a foundation of principles that include responsibility to the communities we serve and where we live and work. In 2008, we launched a Corporate Citizenship Initiative, a global effort to address the environmental, social, economic, and ethical challenges we face in our business. Among the issues we are addressing are carbon impact, paper specifications and procurement, ethical conduct within our business and among our vendors, and community and charitable support. For more information, please visit our website: www.wiley.com/go/citizenship.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Deletion

Deleting an item from a binary search tree T is a bit more complex than inserting a new item because the location of the deletion might be anywhere in the tree. (In contrast, insertions are always enacted at the bottom of a path.) To delete an item with key k, we begin by calling TreeSearch(T, T.root(), k) to find the position p of T storing an item with key equal to k. If the search is successful, we distinguish between two cases (of increasing difficulty):

- If p has at most one child, the deletion of the node at position p is easily implemented. When introducing update methods for the LinkedBinaryTree class in Section 8.3.1, we declared a nonpublic utility, _delete(p), that deletes a node at position p and replaces it with its child (if any), presuming that p has at most one child. That is precisely the desired behavior. It removes the item with key k from the map while maintaining all other ancestor-descendant relationships in the tree, thereby assuring the upkeep of the binary search tree property. (See Figure 11.5.)
- If position p has two children, we cannot simply remove the node from T since this would create a "hole" and two orphaned children. Instead, we proceed as follows (see Figure 11.6):
  - We locate position r containing the item having the greatest key that is strictly less than that of position p, that is, r = before(p) by the notation of Section 11.1.1. Because p has two children, its predecessor is the rightmost position of the left subtree of p.
  - We use r's item as a replacement for the one being deleted at position p. Because r has the immediately preceding key in the map, any items in p's right subtree will have keys greater than r and any other items in p's left subtree will have keys less than r. Therefore, the binary search tree property is satisfied after the replacement.
  - Having used r's as a replacement for p, we instead delete the node at position r from the tree. Fortunately, since r was located as the rightmost position in a subtree, r does not have a right child. Therefore, its deletion can be performed using the first (and simpler) approach.

As with searching and insertion, this algorithm for a deletion involves the traversal of a single path downward from the root, possibly moving an item between two positions of this path, and removing a node from that path and promoting its child. Therefore, it executes in time $O(h)$ where h is the height of the tree.

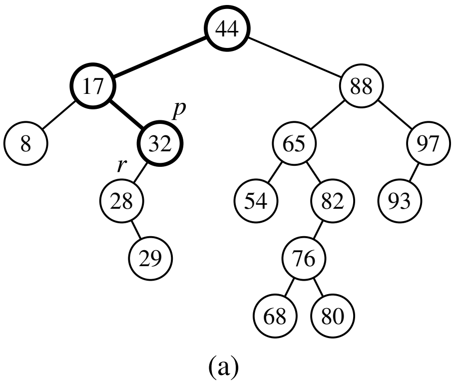

Figure 11.5: Deletion from the binary search tree of Figure 11.4b, where the item to delete (with key 32) is stored at a position p with one child r: (a) before the deletion; (b) after the deletion.

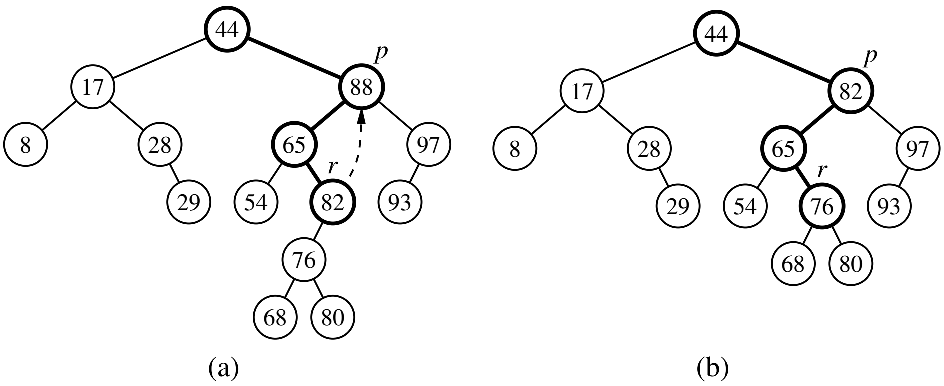

Figure 11.6: Deletion from the binary search tree of Figure 11.5b, where the item to delete (with key 88) is stored at a position p with two children, and replaced by its predecessor r: (a) before the deletion; (b) after the deletion.