

Self-Adjusting Binary Search Trees

DANIEL DOMINIC SLEATOR AND ROBERT ENDRE TARJAN

AT&T Bell Laboratories, Murray Hill, NJ

Abstract. The *splay* tree, a self-adjusting form of binary search tree, is developed and analyzed. The binary search tree is a data structure for representing tables and lists so that accessing, inserting, and deleting items is easy. On an n -node splay tree, all the standard search tree operations have an amortized time bound of $O(\log n)$ per operation, where by "amortized time" is meant the time per operation averaged over a worst-case sequence of operations. Thus splay trees are as efficient as balanced trees when total running time is the measure of interest. In addition, for sufficiently long access sequences, splay trees are as efficient, to within a constant factor, as static optimum search trees. The efficiency of splay trees comes not from an explicit structural constraint, as with balanced trees, but from applying a simple restructuring heuristic, called *splaying*, whenever the tree is accessed. Extensions of splaying give simplified forms of two other data structures: lexicographic or multidimensional search trees and link/cut trees.

Categories and Subject Descriptors: E.1 [Data]: Data Structures—*trees*; F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—*sorting and searching*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Amortized complexity, balanced trees, multidimensional searching, network optimization, self-organizing data structures

1. Introduction

In this paper we apply the related concepts of *amortized complexity* and *self-adjustment* to binary search trees. We are motivated by the observation that the known kinds of efficient search trees have various drawbacks. Balanced trees, such as height-balanced trees [2, 22], weight-balanced trees [26], and B-trees [6] and their variants [5, 18, 19, 24] have a worst-case time bound of $O(\log n)$ per operation on an n -node tree. However, balanced trees are not as efficient as possible if the access pattern is nonuniform, and they also need extra space for storage of balance information. Optimum search trees [16, 20, 22] guarantee minimum average access time, but only under the assumption of fixed, known access probabilities and no correlation among accesses. Their insertion and deletion costs are also very high. Biased search trees [7, 8, 13] combine the fast average access time of optimum trees with the fast updating of balanced trees but have structural constraints even more complicated and harder to maintain than the constraints of balanced trees. Finger search trees [11, 14, 19, 23, 24] allow fast access in the vicinity of one or more "fingers" but require the storage of extra pointers in each node.

Authors' address: AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1985 ACM 0004-5411/85/0700-0652 \$00.75

