Information Processing Letters 16 (1983) 253–257
North-Holland

10 June 1983

UPDATING A BALANCED SEARCH TREE IN O(1) ROTATIONS

Robert Endre TARJAN

Bell Laboratories, Murray Hill, NJ 07974, U.S.A.

Communicated by David Gries
Received 19 November 1982
Revised 28 January 1983

Olivié has recently introduced the class of 'half-balanced' binary search trees, which have O(log n) access time but require only a constant number of single rotations for rebalancing after an insertion or a deletion. In this paper we show that a well-known class of balanced binary trees, the 'symmetric binary B-trees' of Bayer, have the same properties. This is not surprising, for Bayer's class and Olivié's class contain exactly the same binary trees.

Keywords: Balanced search trees, symmetric binary B-trees, red-black trees, half-balanced binary trees

In this paper we assume some familiarity with the properties and uses of balanced search trees, as described for instance in [5].

Standard classes of balanced search trees require O(log n) time for accessing an item in a tree, inserting a new item, or deleting an old item, where n is the number of items in the tree. Recently Olivié [8,9,10] has introduced the class of *half-balanced binary trees*, which in addition to having O(log n) access/insertion/deletion time require only a constant number of structural changes per insertion or deletion (at most two single rotations per insertion, at most three single rotations per deletion). We shall show that a well-known class of trees first defined by Bayer [1] (see also Wirth's book [11]) has the same property; indeed, Bayer's class contains exactly the same trees as Olivié's class.

The trees we shall study are full binary trees: every node is either an *internal node*, having two children, or an *external node*, having no children. We shall ignore the issue of how items are stored in the nodes of a search tree, since we are concerned primarily with the structural changes needed after an insertion or a deletion. (Usually the items are stored one per internal node or one per external node in symmetric order; see the references cited above.) When discussing insertion and deletion we shall assume that the node to be deleted is internal; this entails no loss of generality.

A *half-balanced binary tree* is a full binary tree such that, for every node v, the length (number of edges) of the longest path from v to an external node, denoted by *long*(v), is at most twice the length of the shortest path from v to an external node, denoted by *short*(v).

A *balanced binary tree* is a full binary tree each of whose nodes v has an integer rank, denoted by *rank*(v), such that the ranks have the following properties, where p(v) denotes the parent of node v in the tree:

(i) If v is any node with a parent,

$$rank(v) \leqslant rank(p(v)) \leqslant rank(v) + 1.$$

(ii) If v is any node with a grandparent, then

$$rank(v) < rank(p(p(v))).$$

(iii) If v is an external node, $rank(v) = 0$ and $rank(p(v)) = 1$ if v has a parent.

After Guibas and Sedgewick [2] we call an edge

(p(v), v) *red* if $rank(p(v)) = rank(v)$ and *black* if $rank(p(v)) = rank(v) + 1$; property (i) implies that $rank(p(v)) - rank(v)$ is either zero or one. Bayer [1] first defined balanced binary trees, calling them *symmetric binary B-trees;* he called red edges *horizontal* and black edges *vertical.* He proved that such trees have O(log n) height, which implies O(log n) access time, and he gave O(log n)-time algorithms for insertion and deletion.

Bayer's update algorithms make O(log n) rather than O(1) structural changes to the tree, but there are alternative algorithms that do better. Guibas and Sedgewick [2] extensively studied balanced binary trees and related classes, collectively calling them *red-black trees.* They described a series of update algorithms for such trees, looking almost exclusively at insertion. Their first insertion algorithm for balanced binary trees is a simple bottom-up method that makes only O(1) structural changes. For completeness we review it here.

The atomic operations necessary for an insertion are a *promotion,* which increases the rank of a node by one (called a *color flip* by Guibas and Sedgewick), a *single rotation,* and a *double rotation* (which consists of two single rotations) (see Fig. 1). To perform an insertion, we replace the appropriate external node by an internal node with two external nodes as children. This has the effect of increasing the rank of one of the nodes in the tree by one, and may create a configuration with three consecutive nodes of the same rank on a path, which violates property (ii). If the topmost such node has two children of the same rank, we promote it and proceed to its grandparent, looking for a new violation of the same kind. Otherwise we perform a single or a double rotation as appropriate, which restores balance and terminates the insertion (see Fig. 2). With this method an insertion requires a sequence of promotions followed by at most two single rotations.

Guibas and Sedgewick described a top-down deletion method for balanced binary trees that unfortunately makes O(log n) structural changes. However, there is a bottom-up method, a little more complicated than bottom-up insertion, that makes only O(1) structural changes. Instead of promotion, we need to use *demotion,* which decreases the rank of a node by one (see Fig. 1).

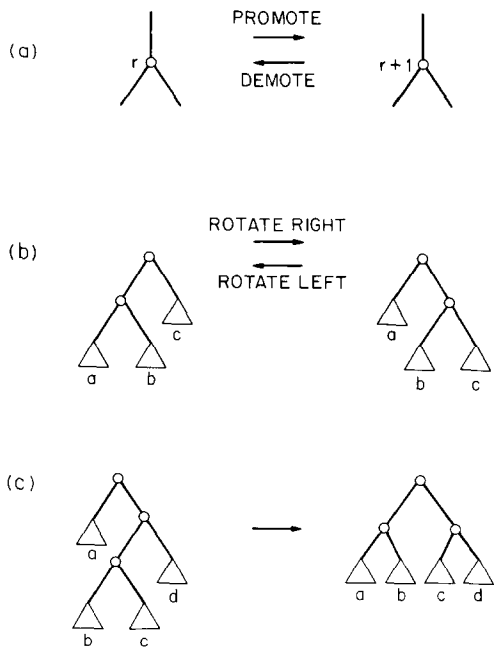

Fig. 1. Atomic operations for insertion and deletion. (a) Promotion/demotion. Circle denotes an internal node of rank indicated. (b) Single rotation. Triangles denote subtrees. (c) Double rotation. There is a symmetric variant.

Without loss of generality we can assume that the node to be deleted has only external children [2]. We replace this node by an external node, which has the effect of decreasing the rank of one of the nodes in the tree by one. The rank of this node may be two less than that of its parent, which violates property (i). Let v be the node whose rank has decreased, w its sibling, and u their common parent; let $r = rank(u)$. Fig. 3 illustrates the possible cases.

Case 1 ($rank(w) = r - 1$, i.e., edge (u, w) is black).

Case 1a (both children of w have rank $r - 2$). Demote u and proceed to the parent of u looking for a new violation.

Case 1b (the child of w farthest from v has rank $r - 1$). Perform a single rotation at u and terminate.

Case 1c (the child of w nearest v has rank $r - 1$ and the other child of w has rank $r - 2$). Perform a double rotation at u and terminate.

Case 2 ($rank(w) = r$, i.e., edge (u, w) is red).

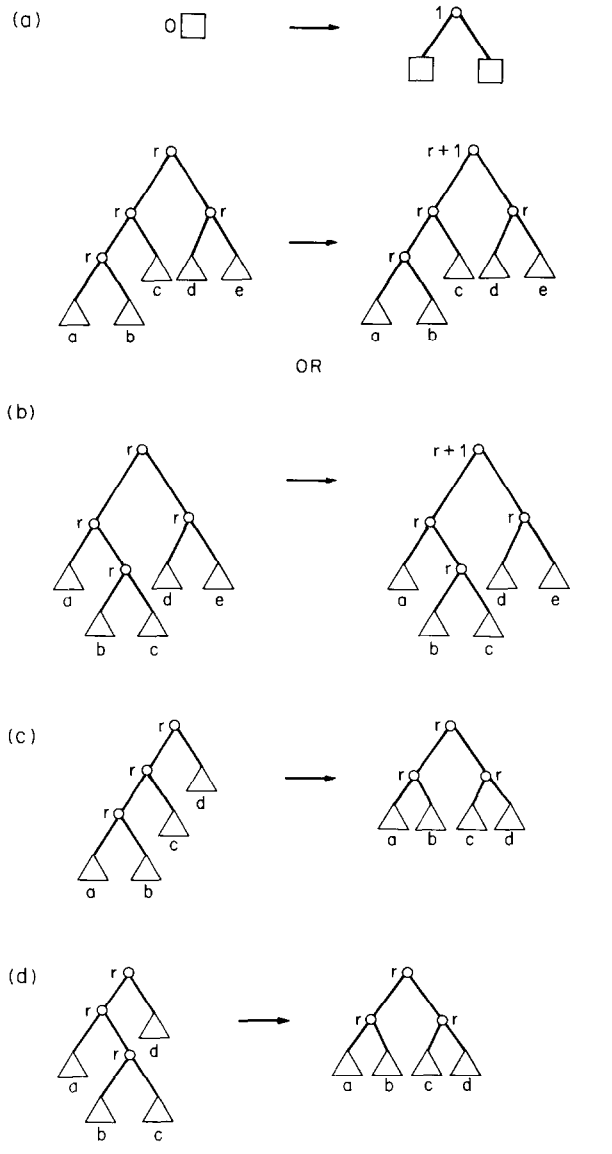

Fig. 2. Insertion steps. Parts (b)–(d) each have a symmetric variant. (a) Replacement of an external node by an internal node with rank one. (b) Promotion. Circles denote internal nodes of indicated ranks. Black edges join roots of subtrees denoted by triangles to their parents. (c) Single rotation. (d) Double rotation.

Both children of w must have rank r − 1 by property (i). Perform a single rotation at u and proceed as in Case 1. In this situation Case la cannot cause

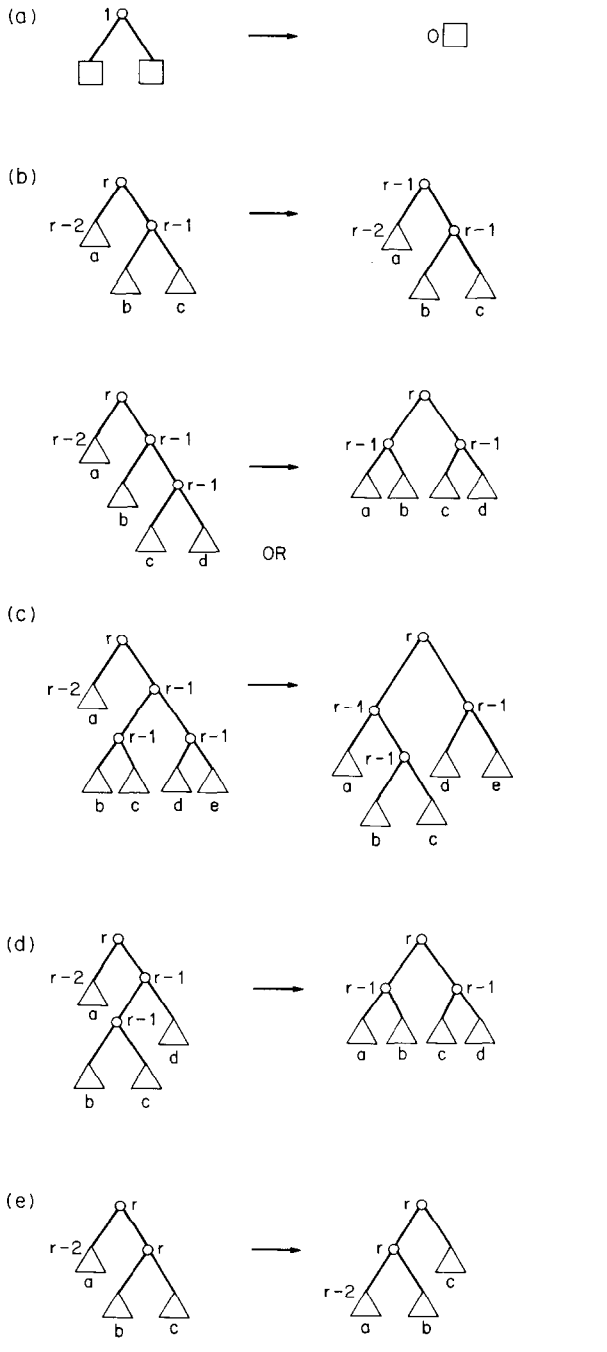

Fig. 3. Deletion steps. Parts (b)–(e) each have a symmetric variant. (a) Replacement of an internal node of rank one by an external node. (b) Case 1a: demotion. (c) Case 1b: single rotation. (d) Case 1c: double rotation. (e) Case 2: single rotation to produce terminating Case 1. Demotion of bottom node of rank r does not cause a new violation of property (i).

a new violation of property (i); thus all subcases are terminal.

With this method a deletion requires a sequence of demotions followed by at most three single rotations.

We have obtained the same result for balanced binary trees that Olivié obtained for half-balanced trees. The following theorem of Olivié [8,9] shows that this is not a coincidence.

Theorem 1. *A full binary tree is balanced if and only if it is half-balanced.*

Proof. Let T be a balanced binary tree, and let v be any node in T. Property (iii) and the definition of black edges imply that the number of black edges on a path from v to an external node is the same for all such paths; namely, it is the rank of v. Let this number be b. Properties (ii) and (iii) imply that at most half the edges on such a path are red. Thus

$$b \leqslant short(v) \leqslant long(v) \leqslant 2b,$$

which implies that T is half-balanced.

Conversely, let T be a half-balanced binary tree, let u be the root of T, and let r be any integer in the range

$$\lceil \tfrac{1}{2} long(u) \rceil \leqslant r \leqslant short(u).$$

We shall describe a way to assign ranks to the nodes of T so that properties (i), (ii) and (iii) hold and node u has rank r. The method is recursive and consists of the following steps:

Step 1. Assign $rank(u) = r$. If u is external, stop.

Step 2. Let v and w be the children of u. Apply the algorithm recursively to the subtree rooted at v, assigning to v the rank $\max\{r - 1, \lceil \frac{1}{2} long(v) \rceil\}$. Apply the algorithm recursively to the subtree rooted at w, assigning to w the rank $\max\{r - 1, \lceil \frac{1}{2} long(w) \rceil\}$.

We can prove the correctness of this method by induction on the number of nodes in T. If u is external,

$$short(u) = long(u) = r = 0,$$

and properties (i), (ii) and (iii) hold. Suppose u is internal. Since $short(v) \geqslant short(u) - 1$,

$$r - 1 \leqslant short(v)$$

and

$$\lceil \tfrac{1}{2} long(v) \rceil \leqslant \max\{r - 1, \lceil \tfrac{1}{2} \mathrm{long}(v) \rceil\} \leqslant short(v).$$

Thus the assignment of ranks to nodes in the subtree rooted at v is correct by the induction hypothesis. If $r - 1 \geqslant \lceil \frac{1}{2} long(v) \rceil$, then the edge (u, v) is black, and properties (i), (ii) and (iii) hold for v and its children. If $r - 1 < \lceil \frac{1}{2} long(v) \rceil$, then

$$\lceil \tfrac{1}{2} long(u) \rceil \leqslant r \quad \text{and} \quad long(v) \leqslant long(u) - 1$$

imply that

$$long(v) = long(u) - 1,$$

$long(u)$ is even, and

$$r = \lceil \tfrac{1}{2} long(v) \rceil = \lceil \tfrac{1}{2} long(u) \rceil.$$

In this case (u, v) is red. Since $long(v)$ must be odd, the edges from v to its children must be black, and properties (i), (ii) and (iii) hold for v and its children. The same argument applies to w. □

What can we learn from these results? It seems that Olivié rediscovered Bayer's trees using an alternative definition that gave him insight into how to perform insertions and deletions in O(1) rotations. However, we can obtain the same result using the original definition. Olivié proposed to represent half-balanced trees by storing $short(v)$ and $long(v)$ for every node v. However, as is apparent from the original definition, it suffices to store just one bit at each internal node v, namely $rank(p(v)) - rank(v)$. Our deletion algorithm is also simpler than Olivié's. For these reasons we prefer the original definition to Olivié's alternative.

The fact that balanced binary trees need only O(1) rotations for an insertion or a deletion makes them usable in McCreight's balanced priority search tree data structure [7]. In addition, balanced binary trees have other remarkable and useful properties. As Guibas and Sedgewick [2] observed, there are top-down methods for insertion and deletion that, though requiring O(log n)

rotations per update, expedite and simplify concurrent search and update operations. Bottom-up updates, in addition to needing only O(1) rotations in the worst-case, require O(1) total time if the time is amortized over a sequence of operations [3,4,6]. Furthermore, the frequency with which an update operation requiring time t occurs is an exponentially decreasing function of t [3,4]. These properties and the simplicity of balanced binary trees make them seem the best choice in practical situations requiring a balanced search tree in internal storage.

References

[1] R. Bayer, Symmetric binary B-trees: Data structure and maintenance algorithms, Acta Inform. 1 (1972) 290–306.

[2] L.J. Guibas and R. Sedgewick, A dichromatic framework for balanced trees, Proc. 19th Ann. IEEE Symp. on Foundations of Computer Science (1978) 8–21.

[3] S. Huddleston and K. Mehlhorn, Robust balancing in B-trees, Lecture Notes in Computer Science 104 (Springer, Berlin, 1981) pp. 234–244.

[4] S. Huddleston and K. Mehlhorn, A new data structure for representing sorted lists, Acta Inform., to appear.

[5] D.E. Knuth, The Art of Computer Programming Vol. 3: Sorting and Searching (Addison-Wesley, Reading, MA, 1973).

[6] D. Maier and S.C. Salveter, Hysterical B-trees, Inform. Process. Lett. 12 (1981) 199–202.

[7] E.M. McCreight, Priority search trees, Tech. Rept. CSL-81-5, Xerox Palo Alto Research Center, Palo Alto, California, 1982; SIAM J. Comput., to appear.

[8] H. Olivié, A study of balanced binary trees and balanced one–two trees, Ph.D. Thesis, University of Antwerp, U.I.A., Wilrijk, Belgium, 1980.

[9] H.J. Olivié, On half-balanced binary trees, Tech. Rept. 80-03, Industriële Hogeschool IHAM, Antwerp, Belgium, 1980.

[10] H.J. Olivié, A new class of balanced search trees: Half-balanced binary search trees, RAIRO Informatique Théoretique 16 (1982) 51–71.

[11] N. Wirth, Algorithms + Data Structures = Programs (Prentice-Hall, Englewood Cliffs, NJ, 1976) pp. 260–264.