References
1. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.
2. Andrews, G.R. COPS—A protection mechanism for computer Systems. Ph.D. Th. and Tech. Rep. 74-07-12, Computer Sci. Program, U. of Washington, Seattle, Wash., July, 1974.
3. Bell, D.E., and LaPadula, L.J. Secure Computer Systems, Vol. I: Mathematical Foundations and Vol. II: A Mathematical Model. MITRE Corp. Tech. Rep. MTR-2547, 1973.
4. Dennis, J.B., and Van Horn, E.C. Programming semantics for multiprogrammed computations. *Comm. ACM 9,* 3 (March 1966), 143–155.
5. Graham, R.M. Protection in an information processing utility. *Comm. ACM 11,* 5 (May 1968), 365–369.
6. Graham, G.S., and Denning, P.J. Protection—principles and practice. AFIPS Conf. Proc., 1972 SJCC, Vol. 40, AFIPS Press, Montvale, N.J., 1972, pp. 417–429.
7. Hopcroft, J.E., and Ullman, J.D. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, Mass. 1969.
8. Jones, A.K. Protection in programmed systems. Ph.D. Th., Dep. of Computer Sci., Carnegie-Mellon U., Pittsburgh, Pa., June 1973.
9. Jones, A.K., and Wulf, W. Towards the design of secure systems. In *Protection in Operating Systems,* Colloques IRIA, Rocquencourt, France, 1974, pp. 121–136.
10. Lampson, B.W. Protection, Proc. Fifth Princeton Symp. on Information Sciences and Systems, Princeton University, March 1971, pp. 437–443. Reprinted in *Operating Systems Rev. 8,* 1 (Jan. 1974), 18–24.
11. Lampson, B.W. A note on the confinement problem. *Comm. ACM 16,* 10 (Oct. 1973), 613–615.
12. Needham, R.M. Protection systems and protection implementations. AFIPS Conf. Proc., 1972 FJCC, Vol. 41, AFIPS Press, Montvale, N.J., 1972, pp. 571–578.
13. Popek, G.J. Correctness in access control. Proc. ACM Nat. Computer Conf., 1974, pp. 236–241.
14. Ritchie, D.M., and Thompson, K. The UNIX time sharing system. *Comm. ACM 17,* 7 (July 1974), 365–375.
15. Saltzer, J.H. Protection and the control of information sharing in MULTICS. *Comm. ACM 17,* 7 (July 1974), 388–402.

Programming Techniques — C. Manacher, S. L. Graham, Editors

An Insertion Technique for One-Sided Height-Balanced Trees

D. S. Hirschberg
Princeton University

A restriction on height-balanced binary trees is presented. It is seen that this restriction reduces the extra memory requirements by half (from two extra bits per node to one) and maintains fast search capabilities at a cost of increased time requirements for inserting new nodes.

Key Words and Phrases: balanced, binary, search, trees
CR Categories: 3.73, 3.74, 4.34, 5.25, 5.31

Binary search trees are a data structure in common use. To keep search time relatively small, the method of balancing binary trees was introduced by Adel'son-Vel'skii and Landis [1]. These height-balanced binary trees (also called AVL trees) require two extra bits per node and require only $O(\log N)$ operations to search and/or insert an item, where N is the number of nodes in the tree. Each node in the tree has a *height* which is defined to be the length of the longest path from that node down the tree. The heights of the two sons of a node may differ by at most one.

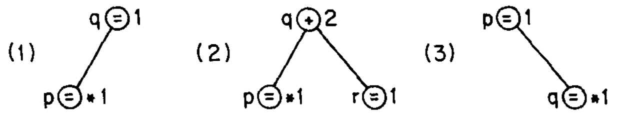
Knuth [2] suggests considering the case where the tree is further restricted so that the right son never has smaller height than the left son. We call such a tree a *one-sided height-balanced* (OSHB) *tree.* In this case, only one extra bit is required per node. The saving of one bit per node is significant if that bit would have re-

Research supported by an IBM fellowship. Author's present address: Department of Electrical Engineering, Rice University, Houston, TX. 77001.

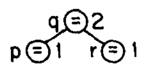
quired the use of an extra word for each node. To search for an item requires $O(\log N)$ time (the same as for AVL trees) but insertion of an item is not as simple. We present a possible solution to this problem and show that an insertion can be made using $O(\log^2 N)$ operations.

We shall use the following notation to describe the *balance factor* at a node (defined to be the difference in height between the right and left sons). ⊖ indicates that both sons are of equal height, ⊕ indicates that the height of the right son is one more than the height of the left son. The number to the right of a node (as, ⊖$n + 2$) indicates the height (level) of that node. An asterisk preceding the number (as, ⊕$*n$) indicates that the level is consistent with all nodes in that node's subtree but that there is an inconsistency between that node's level and the level and/or balance factor of that node's father. In such cases, restructuring of the tree is necessary. The name (or label) of a node may appear to its left.

We wish to make an insertion into an OSHB tree, all of whose leaves are (by definition) at level one and therefore designated by ⊖1. We can find where this new item belongs in $O(\log N)$ comparisons. Let the new item be designated by ⊖$*1$. One of the following must occur:



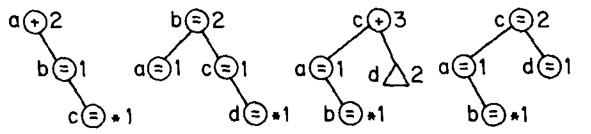
Since (1) can be changed into (3) straightforwardly and since (2) can be changed into



we need only consider (3).

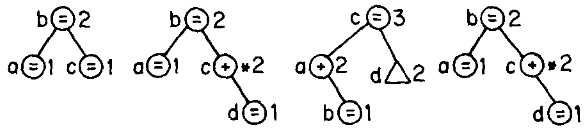
An OSHB tree having one starred node, ⊕$*n$, which is the right son of ⊖n, is defined to be in *normal form of level n*. We shall show how to restructure an OSHB tree in normal form of level n into an OSHB tree having either no starred nodes or one starred node at level $n + 1$ (i.e. ⊕$*n + 1$) which is the right son of ⊖$n + 1$ (i.e., is in normal form of level $n + 1$). Hence, at most $O(\log N)$ such restructurings will be required to result in an OSHB tree having no starred nodes.

Before doing this, we must change (3) into normal form ((3) has a ⊖$*$ node while the normal form has a ⊕$*$ node). One of the following forms of (3) must occur:



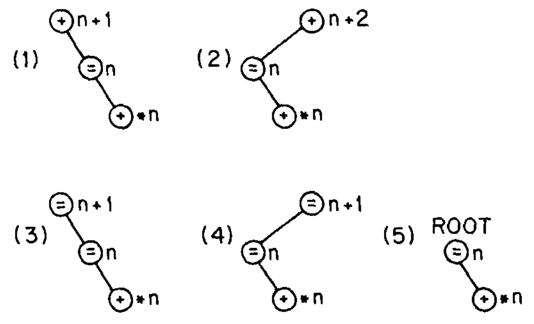
(a "△" indicates a subtree headed by the node whose

and these can be changed to the following:
label appears next to the "△"):

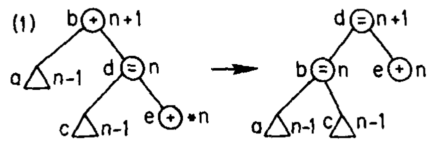


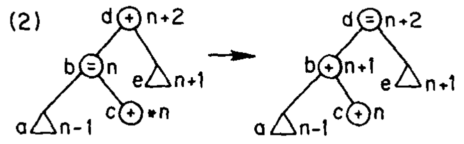
all of which are either in normal or final form.

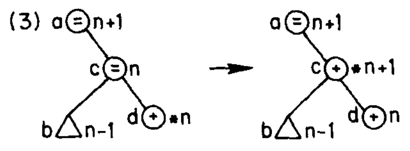
An OSHB tree in normal form of level n must be in one of the following five forms:

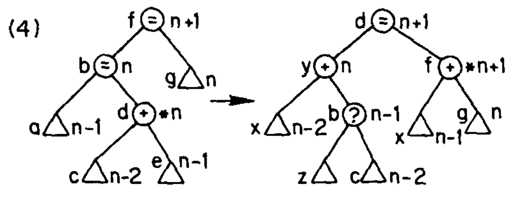


These forms can be changed into a normal form of level $n + 1$ (requiring more changes) or a final form (having no starred nodes) as follows:

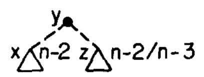




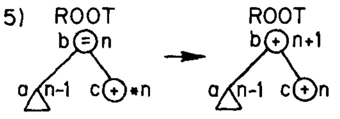


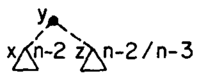


The level of node z must be either $n - 2$ or $n - 3$ (written $n - 2/n - 3$) because its brother (node c) has level $n - 2$; the balance factor of node b is correspondingly either ⊖ or ⊕, and

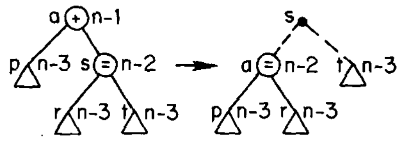


is the result of a restructuring of $a\triangle n - 1$ as shown (recursively) below.

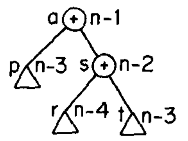
(5)


Restructuring aΔn − 1 into


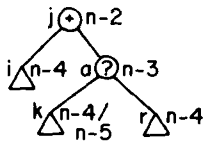
If the balance factor of node a is ⊖ then the subtree headed by node a is in the form we want; otherwise the balance factor at node a is ⊕. We first consider the case where a's right son has a balance factor of ⊖. This case is shown below to be easily restructured into the desired form:



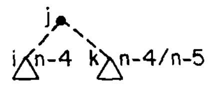
and x is the subtree headed by node a, y is node s, z is the subtree headed by node t. The only other case is that a's right son has balance factor ⊕. In this case, the subtree headed by a is in the following form:



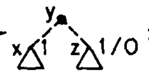
From this structure we can get the following: z is the subtree headed by node t, y is node s , and x is



where $p\triangle n - 3$ has been restructured into

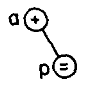


in a similar manner. This is done recursively. We need only show how to restructure subtree a for levels 2 and 3:

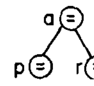
For level 2, $a\triangle 2 \rightarrow$


:

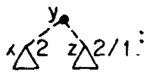
subtree a can be in two forms:

$a\triangle 2$ is


→ x is node a, y is node p, z is null;

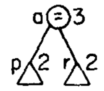
$a\triangle 2$ is


→ x is node p, y is node a, z is node r;

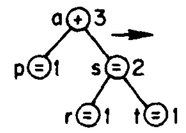
For level 3, $a\triangle 3 \rightarrow$


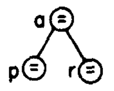
:

subtree a can be in three forms:

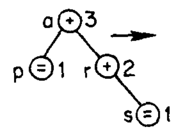
$a\triangle 3$ is


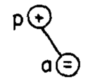
→ x is subtree p, y is node a, z is subtree r;

$a\triangle 3$ is


→ x is


, y is node s, z is node t;

$a\triangle 3$ is


→ x is


, y is node r, z is node s.

We note that restructurings in cases 1, 2, 3, and 5 involve a single restructuring, whereas that of case 4 can involve $O(\log N)$ restructurings. A restructuring of case 4 may result in a starred node at one level higher of case 4. There are $O(\log N)$ levels, each of which may require $O(\log N)$ restructurings to arrive at the next higher level. In the worst case, therefore, $O(\log^2 N)$ restructurings are required for an insertion.

We leave, as an open problem, the question of how to efficiently delete a node from an OSHB tree.

Received June 1975; revised October 1975

References

[References 3 and 4 are not cited in the text.]

1. Adel'son-Vel'skii, G.M., and Landis, E.M. *Doklady Akademia Nauk SSSR 146* (1962), 263–266; English translation in *Sov. Math. 6* (1963), 1259–1263.
2. Knuth, D.E. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, Reading, Mass., 1973, pp. 451–457, 471.
3. Nievergelt, J. Binary Search Trees and File Organization. *Computing Surveys 6*, 3 (Sept. 1974), 195–207.
4. Karlton, P.L., Fuller, S.H., Scroggs, R.E., and Kaehler, E.B. Performance of height-balanced trees. *Comm. ACM 19*, 1 (Jan. 1976), 23–28.