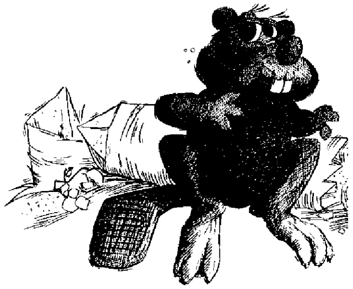
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT
UNIVERSITY OF WATERLOO COMPUTER SCIENCE DEPARTMENT



Balancing Binary Trees by Internal Path Reduction

Gaston H. Gonnet

CS-82-07

February, 1982

Balancing Binary Trees by Internal Path Reduction

by

Gaston H. Gonnet
Department of Computer Science
University of Waterloo

Abstract

We present an algorithm for balancing binary search trees. In this algorithm single or double rotations are performed when they decrease the internal path of the total tree. The idea is very simple and the complete balancing algorithm can be coded in a few statements. It is shown that the worst internal path on such trees is never more than 5% worse than the optimal and that its height is never more than 44% taller than the optimal. This compares very favourably with the AVL trees whose internal path may be 28% worse than optimal and the same worst height, and with the weight-balanced trees which may be 15% and 100% worse than optimal respectively.

Key Words and phrases Binary search trees, balanced trees, AVL trees, Weight balanced trees, searching, analysis of algorithms, rotations, internal path.

CR Categories 3.74, 4.34, 5.25.

1. Introduction

Balanced binary trees are very important in computer science. In theory they provide an excellent dictionary data structure with guaranteed logarithmic cost for various operations. In practice they prevent search trees from degenerating into lists under (a rather usual circumstance) ordered insertions.

Height balanced trees [1] and weight balanced trees [17] are the most popular methods of balancing trees. In the first case we require that the height of two subtrees rooted at the same node to be "about equal" and in the latter we require the number of nodes of each subtree to be "balanced". Balanced trees have received a lot of attention in the literature [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,18,19,20,21,22,23,24,25,26,27].

In the above cases, whenever we perform an update in the tree that alters the balance, we perform "rotations" that cleverly restore the balance conditions. A rotation is a transformation operation on trees that changes the shape of the tree without altering its lexicographical order.

For example

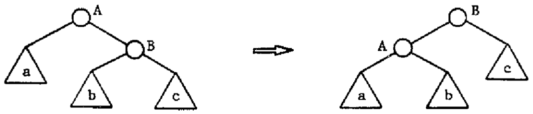


Figure 1: Single left rotation

The literature often describes single, double and sometimes triple rotations. It is important to recognize that in most cases complicated rotations can be effectively described by more than one single rotation.

If we consider these rotations as the basic operations on our trees, the AVL algorithm can be viewed as a rule which performs a single or a double rotation whenever this rotation can reduce the height of a subtree. This point of view is quite important since it describes in a simple way the rotations and emphasizes the fact that the AVL rotations bound the worst case (given by the height) of operations in the tree.

With the above in mind the new algorithm can be described as a counterpart of the AVL trees: perform single or double rotations whenever these rotations can reduce the total internal path of the subtree. To do this balancing, as we will see in the next section, the only extra information that we need to store in each node of the tree, is the number of nodes in its subtree. Intuitively these trees should have smaller internal path than AVL trees. Baer [4] derives a criteria for rotations using weighted paths that could be translated into a similar algorithm but the algorithm and properties of the resulting trees were not discussed.

The internal path of a tree is directly related to the cost of an average successful search and also to the external path and consequently to the average unsuccessful search. Let C_n and C_n' be the average number of node inspections required for a successful and unsuccessful search respectively. Let I_n be the internal path and E_n the external path, then it is known that [9,12]

$$\begin{aligned} C_n &= 1+\frac{I_n}{n}, \\ E_n &= I_n+2n \\ C_n' &= \frac{E_n}{n+1} \end{aligned}$$

By reducing the internal path we reduce the average successful and unsuccessful search complexity.

The following table shows two of the most important measures of complexity for various balanced trees.

	AVL	BB(α)	Our
height	$\leq 1.4402 \ldots \log_2 n$	$\leq 2 \log_2 n$	$\leq 1.4402 \ldots \log_2 n$
worst internal path	$\geq 1.2793 \ldots n \log_2 n$	$\geq 1.1462 \ldots n \log_2 n$	$1.0515 \ldots n \log_2 n$

Although the above are important measures, one could easily come with other interesting measures. Depending on the implementation, use, constraints, etc. any combination of

$$\left\{\begin{array}{c}\text{worst case}\\ \text{average}\\ \text{amortized w.c.}\end{array}\right\}\text{number of}\left\{\begin{array}{c}\text{rotations}\\ \text{node inspections}\end{array}\right\}\text{to}\left\{\begin{array}{c}\text{search}\\ \text{insert}\\ \text{delete}\end{array}\right\}\text{a key in}\left\{\begin{array}{c}\text{AVL}\\ \text{BB}(\alpha)\\ \text{our}\end{array}\right\}\text{trees}$$

is an interesting measure of complexity. (The amortized worst case is the cost of the worst possible sequence of n operations divided by n.)

Several of the above measures are still open questions. In the next section we will find upper bounds on the height and on the internal path of our trees. These measures will answer most of the worst case questions and will give good bounds on the averages.

2. The balancing algorithm.

In this section we will derive the conditions on which a rotation will reduce the internal path of a subtree.

The *internal path* of a binary tree is defined as the sum of the depths of all its nodes. The *depth* of a node is its distance to the root. Consequently the root is at depth 0. For example, a balanced tree with 3 nodes will have internal path 2.

A single rotation, as described in figure 1, will reduce the total internal path by $n_c - n_a$, where n_a is the number of nodes in the subtree a and similarly for n_c. This follows by simply taking the difference of the internal paths before and after the rotations

$$I_a + n_a + I_b + 2n_b + I_c + 2n_c + 3 - (I_a + 2n_a + I_b + 2n_b + I_c + n_c + 3) = n_c - n_a$$

where I_a denotes the internal path of the subtree labelled a and so on.

A double rotation is a transformation described by the following figure:

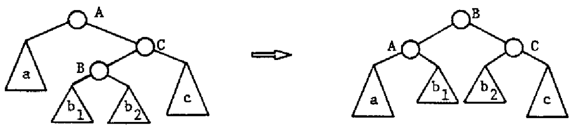


Figure 2: Double rotation

Consequently we see that the total internal path in the subtree will change due to the subtrees b_1 and b_2 that will be one level closer to the root, the subtree a which will be one level farther from the root and by one due to the reorganization of the nodes A B and C. The reduction in total internal path due to a double rotation is

$$n_{b_1}+n_{b_2}+1-n_a$$

or using our previous notation where $n_b=n_{b_1}+n_{b_2}+1$,

$$n_b-n_a.$$

The criteria for single or double rotation is exceedingly simple: if $n_c-n_a>0$ a single rotation will reduce the total internal path of the tree and we should do it, or if $n_b-n_a>0$ a double rotation will reduce the internal path.

In a balanced tree, rotations are needed only when we change the shape of the tree, i.e. when we insert or delete a node. In case of ambiguity, i.e. we may perform a single or a double rotation, we may decide to rotate according to the biggest gain in internal path – the largest of n_b or n_c. If the gains are equal we should clearly use single rotations since, the way we code them, they require less work than double rotations.

The complete code for an insertion with rebalancing is simpler and easier to understand that those for AVL or BB(α) trees. In pseudo Algol 68 the insertion code is:

Balanced Tree Insertion

```
proc insert = ( ref ref tree t, typekey key) void:

if t:=:nil then # insert new node here #
                t := .......
elif k of t = key then  # error: element already in tree #
                action-for-error
else if k of t < key then insert( right of t, key )
                else insert( left of t, key ) fi;
        n of t := n of t + 1;
        checkrot( t ) fi;

proc checkrot = ( ref ref tree t ) void:
# Procedure to check for possible rotations #

if size( left of t) < size( right of t) then
        # Check for left rotations #
        if size( right of right of t ) > size( left of t ) then
                lrot( t ); checkrot( left of t )
        elif size( left of right of t ) > size( left of t ) then
                rrot( right of t ); lrot(t); checkrot( left of t ); checkrot( right of t )
        fi
elif size( left of t ) > size( right of t ) then
        # Same as above exchanging left and right # fi;

proc size = (ref tree t ) int :
if t :=: nil then 0 else n of t fi;
```

Single Left Rotation

```
proc lrot = ( ref ref tree t ) void :
t := right of t :=:= left of right of t :=:= t;
n of t := n of left of t;
n of left of t := size( left of left of t ) + size( right of left of t ) + 1;

proc rrot = (ref ref tree t ) void:
        # Same as lrot exchanging left and right #
```

The operator :=:= is the displacement operator; e.g. a := b :=:= c :=:= a rotates left the values of a, b and c.

Note that a double rotation can be expressed as a sequence of two single rotations, and since a single rotation reconstructs all the information in the nodes, this is all we need to do. For example the double rotation in Figure 2 is achieved by

rrot(right **of** A); lrot(A).

3. Analysis of the new Tree.

In this section we will give upper bounds on the height and on the total internal path of trees constructed with this algorithm.

The worst height will be achieved by trees (or their mirror images) with the following characteristics:

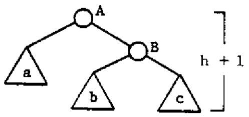


Figure 3: a worst-height tree

(1) The tree rooted at A has height $h+1$

(2) There is no tree with the same height and less nodes.

We can conclude that the subtree rooted at B is also a worst-height tree, and a has the least possible number of nodes. This means that $n_a = \max(n_b, n_c)$. Hence a has less nodes than the subtree rooted at B. But the largest of b or c is the subtree in B that gives the worst possible height, so it is also a tree of the same class and height $h-1$. Let $n(h)$ be the number of nodes of a worst-height tree of height h, then

$$n(h+1) = 1+n(h)+n(h-1),$$

$$n(1) = 1, \qquad n(2) = 2.$$

This recurrence relation is the same as the one for the tallest AVL trees, and $n(h)$ can be expressed in terms of the Fibonacci numbers:

$$\begin{aligned} n(h) &= F_{h+2}-1 \\ &= k_1\varphi^n - 1 + O(\varphi^{-n}) \end{aligned}$$

where $\varphi=(1+\sqrt{5})/2$ is the "golden ratio", $F_1=F_2=1$, $F_{n+1}=F_n+F_{n-1}$ and $F_n = \frac{\varphi^n-(-\varphi)^{-n}}{\sqrt{5}}$.

The inverse of this function gives the height of the tallest tree with n nodes, consequently

Theorem 1. *The height of a tree with n nodes balanced to minimize internal path is bounded by*

$$height \le \log_\varphi 2 \log_2 n - 0.32772\ldots = 1.44042\ldots \log_2 n - 0.32772\ldots$$

To find the worst internal path in these new trees we will use a similar method. Let A be the root of a worst-internal-path tree with n nodes. Then if $n \ge 2$ we can represent the tree (albeit symmetries) as

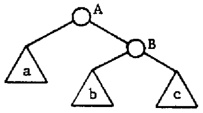


Figure 4: Worst internal path tree.

Let I_n be the worst internal path of any tree with n nodes, then

$$I_n = \max_{n_a, n_b}(1 + I_{n_a} + n_a + I_{n_b} + 2n_b + I_{n_c} + 2n_c)$$

where $n = n_a + n_b + n_c + 2$ and the maximum is taken over all possible values of n_a and n_b that satisfy the balancing requirements. It should be noted that the equation is symmetric in n_c and n_b, so without loss of generality we can assume that $n_c \ge n_b$ and $n_a \le n_b + n_c + 1$. Due to the balancing we further deduce that

$$n_c \le n_a, \qquad n_c \le 2n_b + 1.$$

Let $n_a = \alpha n$ and $n_b = \beta n$. Then when n is large, ignoring $O(1)$ terms, the recurrence is translated into

$$I_n = (2-\alpha)n + I_{\alpha n} + I_{\beta n} + I_{(1-\alpha-\beta)n} \tag{1}$$

where α and β are in the region delimited by

$$\beta \le \frac{1-\alpha}{2}; \quad 1-2\alpha \le \beta; \quad \frac{1-\alpha}{3} \le \beta; \quad \alpha \le ½$$

which are derived from the balancing requirements.

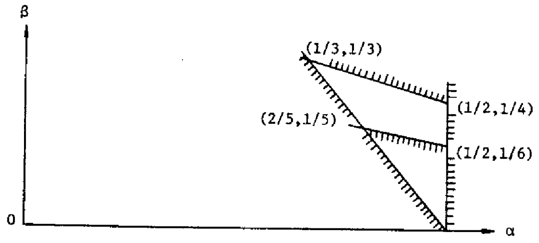


Figure 5: Valid range for α and β

From the discussion on the height we know that the total internal path is bound by $1.44\ldots n\log_2 n$. Since it is obviously bound below by the internal path of a perfectly balanced tree, the internal path is $\Theta(n\log_2 n)$. The following discussion is simplified significantly by assuming that the worst internal path of a tree is

$$I_n = k_1 n\log_2 n + O(n)$$

Taking partial derivatives of the right hand side of equation (1) with respect to α and β we find that there is one minimum at $\alpha=1/2$, $\beta=1/4$, and a few other minima on the perimeter. The maximum in the region occurs for $\alpha=\beta=1/3$. Under this condition

$$k_1 = \frac{5}{3}\log_3 2 = 1.05155\ldots$$

or finally

Theorem 2. *The internal path of a tree with n nodes, balanced to minimize internal path is bounded by*

$$I_n \le 1.05155\ldots n\log_2 n + O(n)$$

This result shows that this new balancing algorithm yields trees that under all circumstances have an internal path at most 5% off optimal.

It is in order to compare this worst internal path with the other balancing schemes.

Lemma. *There are weight balanced trees with n nodes which have internal path*

$$I_n = \frac{1}{H(\alpha)} n\log_2 n + O(n)$$

where $H(\alpha) = -\alpha\log_2\alpha - (1-\alpha)\log_2(1-\alpha)$. This is easily proved by constructing the most unbalanced BB(α) tree. Then

$$I_n = n - 1 + I_{\alpha n} + I_{(1-\alpha)n}$$

For the suggested value of α, $\alpha = 1-\sqrt{2}/2$

$$I_n = 1.146224...n\log_2 n + O(n)$$

Lemma. *There are AVL trees with n nodes which have an internal path*

$$I_n = (1-\sqrt{5}/20)\log_\varphi 2\, n\log_2 n + O(n) = 1.279376...n\log_2 n + O(n).$$

This follows from the construction of a tree as illustrated below

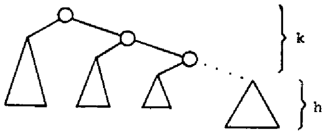


Figure 6: A bad internal-path AVL tree

where the k left subtrees are Fibonacci trees and the rightmost tree is a full tree of height h. Selecting k close to $(\log_\varphi 2-1)h$ and doing a lot of manipulations we obtain the internal path mentioned above.

It should be noted that a Fibonacci tree, although it gives the worst height of an AVL tree, does not give a bad internal path. A Fibonacci tree with n nodes has internal path

$$I_n = \frac{2+\varphi}{5}\log_\varphi 2\, n\log_2 n + O(n) = 1.042298...n\log_2 n + O(n).$$

4. Rotations

First we should note that in these trees each rotation improves the tree (by reducing the internal path), consequently we may be willing to do as many rotations as possible.

The number of rotations is a very important measure of complexity for updates. An insertion or deletion is typically preceeded by a search and then we have to reconstruct the balancing of the tree. The cost of an update is then the cost of a search plus the rotations needed. Only the nodes from the root to the inserted/deleted element need to be checked for an altered balance. If a node becomes out of balance we will perform rotations to restructure this imbalance. Next we can show that two rotations cannot happen at consecutive levels. To prove this we have to analyze a few cases like

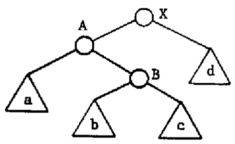


Figure 7

and its symmetrics. The proof is based on the fact that if the tree rooted at X is balanced, then among other restrictions,

$$n_b + n_c + 1 \le n_d; \quad n_c \le n_a.$$

If an insertion in c is to cause two consecutive rotations then

$$n_c + 1 > n_a, \quad and \quad n_a + n_b + 1 > n_d.$$

But this is not consistent with the previous inequalities, hence two consecutive rotations cannot happen.

An upper bound for insertions and deletions can be obtained by the following argument: the internal path of any binary tree is $n \log_2 n + O(n) \le I_n \le 1.05155...n \log_2 n + O(n)$. Each insertion contributes at most $1.44...\log_2 n$ to the internal path (the maximum height) and each rotation decreases the internal path by at least 1. Consequently

Lemma. *The amortized worst case number of rotations for each insertion is*

$$1.44042...\log_2 n + O(1) - \log_2 n = 0.44042 \log_2 n + O(1).$$

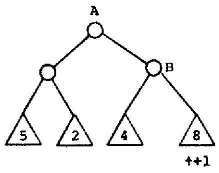
For deletions a similar result follows from the observation that the leaf closest to the root is at depth at least $\log_3(n+1)$. Consequently

Lemma. *The amortized worst case number of rotations for each deletion is*

$$1.05155...\log_2 n + O(1) - \log_3(n+1) = 0.42062...\log_2 n + O(1).$$

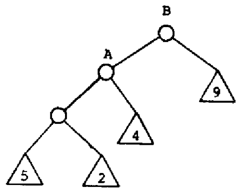
These upper bounds appear to be pessimistic since we were not able to produce even small examples with these properties. Intuitively the reason is that an insertion can be done at the deepest possible node of a tree, but then a rotation shrinks the height and we require several other insertions to restore its maximum height. On top of this the tree will not go in few steps into an optimal state either.

Unfortunately, although the average number of rotations is bounded, there are some situations in which a rotation may propagate other rotations down the tree. The simplest example of such a case is a tree of size 22



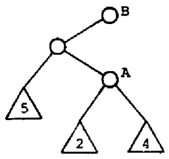
Propagation with a single rotation

where the numbers indicate the size of the subtrees. The above tree is in balance, but if we add a node to the rightmost tree, a single rotation is produced, resulting in the tree



Tree after single rotation

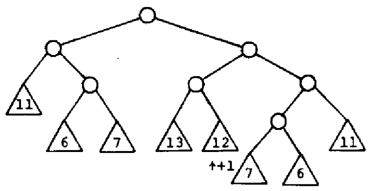
which now requires another single rotation at node A to produce, finally



Final tree

Double rotations can also propagate rotations below and furthermore they can propagate them in two directions. This flaw is the saving grace for both AVL and BB(α) trees which would, otherwise, loose all interest in favour of these trees.

The worst tree that we are able to construct is one on which an insertion causes $\frac{4n-37}{41}$ rotations. An example of size 81 follows.



Bad case of propagation of rotations

The reader can verify that an insertion where indicated will unchain 7 rotations.

5. Extensions

There are two natural extensions to this balancing scheme.

(a) *k-balancing*. The single rotation is performed only if $n_c - n_a \geq k$, and the double rotation is performed only if $n_b - n_a \geq k$. The 1-balanced trees are the ones just described.

Surprisingly, for this balancing scheme the height is still of the same order:

$$height \leq 1.44042...\log_2(n-k+2)+k-1.32772...$$

The recurrence relation for the thinnest tree with height h is

$$n(h+1) = n(h)+n(h-1)+2-k$$

with $n(k)=k$ and $n(k+1)=k+1$; or

$$n(h) = F_{h-k+3}+k-2.$$

The internal path is also bounded by

$$I_n \leq 1.051...n\log_2 n + O(n).$$

The amortized worst case number of rotations is bounded by $0.44042.../k\log_2 n$ for insertions and by $0.42062.../k\log_2 n$ for deletions.

(b) *ε-balancing*. Single or double rotation are performed whenever the gain in internal path is significant compared to the total number of nodes in the subtree, e.g. we perform a single rotation when

$$\frac{n_c - n_a}{n_a + n_b + n_c + 2} > \varepsilon$$

Our trees are the 0%-balanced trees.

Theorem. *The height of an ε-balanced tree is bounded by*

$$height \leq \frac{1}{\log_2(1+\sqrt{5+4\varepsilon})-1-\log_2(1+\varepsilon)}\log_2 n + O(1).$$

For $\varepsilon=5\%$, $height \leq 1.5540\ldots\log_2 n+O(1)$, and for $\varepsilon=10\%$, $height \leq 1.6797\ldots\log_2 n+O(1)$.

Theorem. *The internal path in an $\varepsilon-balanced$ tree can be as large as*

$$I_n \leq -\frac{5+2\varepsilon}{(1-2\varepsilon)\log_2(1-2\varepsilon)+2(1+\varepsilon)\log_2(1+\varepsilon)-3\log_2 3} n \log_2 n+O(n).$$

For $\varepsilon=5\%$ the constant multiplying $n \log_2 n$ is 1.0750... and for $\varepsilon=10\%$ it is 1.1040... .

The *k-balanced* trees have the nice property of preserving the same asymptotic behaviour for both the height and the internal path while, intuitively, reducing the number of rotations by a factor of k.

The *ε-balanced* trees although may have a slightly worse height and internal path, perform rotations only when the gain is proportionally significant. When the rotations require significant cost (i.e. related to the size of the subtree) this variant should be preferred.

6. Conclusions.

We presented a balancing algorithm for binary trees. This balancing requires to store at each node, the number of nodes in its subtree. In this respect the storage requirements are the same as for the weight balanced trees. The performance of such trees in height and internal path is superior to the weight balanced trees in both measures and superior in internal path, sharing the same worst case height, compared to AVL trees. To further complement the above advantages, the algorithm is very easy to understand and simple to code.

Both extensions, the *k-balanced* trees and the *ε-balanced* trees may provide the necessary tailoring for particular real applications.

7. Acknowledgement. The author wishes to acknowledge the help and criticisms from Doron Rotem and Nivio Ziviani.

8. References

[1] Adel'son-Vel'skii, G.M. and Landis, E.M.: An Algorithm for the organization of information; Doklady Akademia Nauk USSR, 146(2):263-266, (1962).

[2] Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: *The Design and Analysis of Computer Algorithms*; Addison Wesley, Reading Mass, (1974)

[3] Baer, J.L. and Schwab, B.: A Comparison of Tree-Balancing Algorithms; C.ACM, 20(5):322-330, (May 1977).

[4] Baer, J.L.: Weight-Balanced Trees; Proceedings of the NCC, 44:467-472 (1975).

[5] Brown, M.R.: A Partial Analysis of Random Height-Balanced Trees; SIAM J on Computing, 8(1):33-41, (Feb 1979).

[6] Brown, M.R.: A Storage Scheme for Height-Balanced Trees; Inf. Processing Letters, 7(5):231-232, (Aug 1978).

[7] Foster, C.C.: A Generalization of AVL Trees; C.ACM, 19(1):23-28, (Jan 1976).

[8] Foster, C.C.: Information Storage and Retrieval Using AVL Trees; Proceedings ACM-NCC, :192-205, (1965).

[9] Gonnet, G.H.: A Handbook of Algorithms and Data Structures; CS-80-23, University of Waterloo, (May 1981).

[10] Hirschberg, D.S.: An Insertion Technique for One-Sided Height-Balanced Trees; C.ACM, 19(8):471-473, (Aug 1976).

[11] Karlton, P.L., Fuller, S.H., Scroggs, R.E. and Kaehler, E.B.: Performance of Height-Balanced Trees; C.ACM, 19(1):23-28, (Jan 1976).

[12] Knuth, D.E.: *The Art of Computer Programming, vol III: Sorting and Searching;* Addison-Wesley, Reading Mass., (1973).

[13] Kosaraju, S.R.: Insertions and Deletions in One-Sided Height-Balanced Trees; C.ACM, 21(3):226-227, (Mar 1978).

[14] Luccio, F. and Pagli, L.: Comment on Generalized AVL Trees; C.ACM, 23(7):394-395, (July 1980).

[15] Luccio, F. and Pagli, L.: Power Trees; C.ACM, 21(11):941-947, (Nov 1978).

[16] Mehlhorn, K.: A Partial Analysis of Height-Balanced Trees Under Random Insertions and Deletions; Report A-79/21, U. des Saarlandes, (Oct 1979).

[17] Nievergelt, J. and Reingold, E.M.: Binary Search Trees of Bounded Balance; SIAM J. Computing, 2(1):33-43, (1973).

[18] Ottmann, Th. and Wood, D.: 1-2 Brother Trees or AVL Trees Revisited; Computer Journal, 23(3):248-255, (1980).

[19] Ottmann, Th., Six, H.W. and Wood, D.: On the Correspondence Between AVL Trees and Brother Trees; Computing, 23(1):43-54, (1979).

[20] Ottmann, Th., Six, H.W. and Wood, D.: One-Sided k-Height-Balanced Trees; Computing, 22(4):283-290, (1979).

[21] Raiha, K.J. and Zweben, S.H.: An Optimal Insertion Algorithm for One-Sided Height-Balanced Binary Search Trees; C.ACM, 22(9):508-512, (Sep 1979).

[22] Standish, T.A.: *Data Structure Techniques;* Addison-Wesley, Reading Mass. (1980).

[23] Unterauer, K.: Dynamic Weighted Binary Search Trees; Acta Inf. 11(4):341-362, (1979).

[24] Wirth, N.: *Algorithms + Data Structures = Programs;* Prentice-Hall, Englewood Cliffs NJ, (1976).

[25] Zaki, A. and Baer, J.L.: A Comparison of Query Costs in AVL and 2-3 Trees; Report 78-02-01 (Aug 1979) Dept. Computer Science, U. of Washington.

[26] Ziviani, N. and Tompa, F.W.: A Look at Symmetric Binary B-trees; to appear in Infor.

[27] Zweben, S.H. and McDonald, M.A.: An Optimal Method for Deletions in One-Sided Height-Balanced Trees; C.ACM, 21(6):441-445, (June 1978).