RAIRO
INFORMATIQUE THÉORIQUE

KURT MEHLHORN

Arbitrary weight changes in dynamic trees

RAIRO – Informatique théorique, tome 15, n° 3 (1981), p. 183-211.
<http://www.numdam.org/item?id=ITA_1981__15_3_183_0>

R.A.I.R.O. Informatique théorique/Theoretical Informatics
(vol. 15, n° 3, 1981, p. 183 à 211)

ARBITRARY WEIGHT CHANGES IN DYNAMIC TREES (*)

by Kurt MEHLHORN (1)

Communicated by J. BERSTEL

Abstract. — *We describe an implementation of dynamic weighted trees, called D-trees. Given a set* $\{B_0, \ldots, B_n\}$ *of objects and access frequencies* $q_0, q_1, \ldots, q_n$ *one wants to store the objects in a binary tree such that average access is nearly optimal and changes of the access frequencies require only small changes of the tree. In D-trees the changes are always limited to the path of search and hence update time is at most proportional to search time.*

Résumé. — *Nous décrivons une implimentation d'arbres pondérés dynamiques appelés D-arbres. Étant donnés un ensemble* $\{B_0, \ldots, B_n\}$ *d'objets et des fréquences d'accès* $q_0, q_1, \ldots, q_n$ *on désire stocker les objets dans un arbre binaire de telle manière que le temps d'accès moyen est presque optimal et que des changements des fréquences d'accès ne requièrent que de petites modifications de l'arbre. Dans un D-arbre, les modifications sont toujours limitées au chemin de recherche et par conséquent le temps de mise à jour est au plus proportionnel au temps de recherche.*

1. INTRODUCTION

One of the popular methods for retrieving information by its 'name' is to store the names in a binary tree. In this paper we treat dynamic weighted binary search trees.

Given a subset $\{B_0, B_1, \ldots, B_n\}$ from an ordered universe U and access frequencies $q_0, q_1, \ldots, q_n \in \mathbb{N}$, the problem is to store the objects $B_0, B_1, \ldots, B_n$ in a binary tree such that:

(*) Received April 1979, revised April 1980.

(1) Fachbereich 10, Angewandte, Mathematik und Informatik, Universität des Saarlandes, D-6600 Saarbrücken.

R.A.I.R.O. Informatique théorique/Theoretical Informatics, 0399-0540/1981/183/$ 5.00

1) The weighted path length (and hence average search time):

$$\sum_{j=0}^{n} q_j a_j \Big/ \sum_{j=0}^{n} q_j,$$

is (nearly) minimal. Here a_j denotes the depth of B_j in the tree.

2) Changing the access frequency q_j of B_j by an arbitrary amount $d \in \mathbb{Z}$ requires only small changes of the tree. In particular, it should be possible to insert new objects into the tree ($q_j = 0$) and to delete objects from the tree ($d = -q_j$).

The above problem comes up in many contexts. Consider for example a library system. The objects would be books. Every request for a book would increase its frequency count by 1. Retirement of a book corresponds to the deletion of an object ($d = -q_j$). Furthermore, acquisition of a new book corresponds to the insertion of an object, i. e. q_j was zero and will be increased to some positive level. It is conceivable that a librarian might want to make an initial guess at the popularity of a book and set d to an appropriate value; this corresponds to the insertion with arbitrary positive d. Furthermore, one might want to update the weight of objects not after every single request, but sum up the requests separately, and increase the weight q_j by d at one blow, say whenever the weight has doubled.

In this paper we introduce D-trees which provide us with a solution to the above problem which is optimal up to a constant factor:

1) The average search time (average weighted path length) is always $\leqq 2 \cdot$ search time in an optimal tree.

2) The cost of updating the structure after an arbitrary weight change is at most proportional to search time. This is achieved by restricting the changes of the tree structure to the path of search.

A solution to the above problem is called a dynamic weighted tree; weighted because of 1) and dynamic because of 2). An immense amount of knowledge is available about weighted trees (access frequencies are static and no insertions and deletions take place) and dynamic trees (access frequencies are 1, but insertions and deletions are allowed). In particular, the weighted path length of a binary tree for access frequencies $q_0, q_1, \ldots, q_n$ is at least:

$$\overline{H}(q_0, q_1, \ldots, q_n)/\log 3 = \sum_{i=0}^{n} \frac{q_i}{W} \log \frac{W}{q_i} \Big/ \log 3,$$

where $W = \sum_{i=0}^{n} q_i$ (*cf.* Mehlhorn, 1977). Implementations of dynamic trees are known which allow insertions and deletions in $O(\log n)$ units of time.

Several kinds of dynamic weighted trees were already proposed. Baer proposed the first solution, however he gave no theoretical analysis of it. Allen and Munro describe and analyse a probabilistic approach. Unterauer introduced $B_{1/3}$ trees. The weighted path length of $B_{1/3}$ trees is always nearly optimal and the expected update time after the insertion of a new key is proportional to the length of the path of search. The underlying assumptions about the distribution of access frequencies are reasonable. However, the update time may be exponential in the size of the tree in the worst case. *D*-trees were introduced in Mehlhorn, 1979, *see* also Mehlhorn, 1977. In *D*-trees the frequency changes are restricted to ± 1. *D*-tree exhibit the following behavior:

1) The weighted path length of a *D*-tree is always nearly optimal. In particular $a_j = O(\log W/q_j)$ where a_j is the depth of object B_j in the *D*-tree.

2) Update time is at most proportional to search time in the worst case.

In this paper we generalize *D*-trees and prove the following theorem.

THEOREM: *Let* $(q_0, q_1, \ldots, q_n)$ *be a frequency distribution and let* $W := q_0 + q_1 + \ldots + q_n$. *Let T be a D-tree for this frequency distribution:*

1) *Searching for object* B_i *(which has frequency* q_i*) takes time* $O(\log W/q_i)$. *In particular, the depth of object* B_i *in the tree T is bounded by* $2 \log W/q_i + 3$. *Average weighted path length is bounded by* $2 \cdot \overline{H} + 3 \leqq 2 \cdot \sqrt{3} \cdot P_{\text{opt}} + 3$ *where* P_{opt} *is the weighted path length of an optimal search tree.*

2) *Updating the tree structure after increasing* q_j *by d takes time:*

$$O(\log(W/\max(1, q_j)) + \log(\max(1, d/W))).$$

3) *Update time after decreasing* q_j *by d is:*

$$O(\log(W/\max(q_j - d, 1))). \quad \square$$

Note that the factor $\log \max(1, d/W)$ is usually negligible and hence update time is proportional to search time. Since the search time is within a constant factor of optimality we conclude that *D*-trees provide a realization of dynamic weighted trees which is optimal up to a constant factor. Hence they generalize the behavior of balanced trees (*AVL*-trees, 2-3 trees) from the unweighted to the weighted case.

D-trees are based on weight-balanced trees (Nievergelt and Reingold). As a byproduct of our analysis we obtain that weight-balanced trees support the full repertoire of Concatenable Queue Operations (Insert, Delete, Member, Concatenate, Split) with logarithmic execution time per operation.

In section 3 we review weight-balanced trees and introduce *D*-trees. In section 2 we show how to support concatenable queues by weight-balanced

trees, in section 4 we deal with weight increases and in section 5 with weight decreases. Section 3 is mainly intended as a warm-up.

Knowledge of Mehlhorn, 1979 is helpful but not required.

2. PRELIMINARIES: *D*-TREES

D-trees (Mehlhorn, 1977 or 1979) are an extension of weight-balanced trees (Nievergelt and Reingold). Weight-balanced trees are a special case of binary trees. In a binary tree a node has either two sons or no son. Nodes with no sons are called leaves.

DEFINITION: Let T be a binary tree. If T is a single leaf then the root-balance $\rho(T)$ is $1/2$, otherwise we define $\rho(T) = |T_l| / |T|$, where $|T_l|$ is the number of leaves in the left subtree of T and $|T|$ is the number of leaves in tree T.

DEFINITION: A binary tree T is said to be of bounded balance α, or in the set $BB[\alpha]$, for $0 \leqq \alpha \leqq 1/2$, if and only if:

1. $\alpha \leqq \rho(T) \leqq 1-\alpha$.
2. T is a single leaf or both subtrees are of bounded balance α.

Remarks: a) The definition of root-balance is apparently unsymmetric with respect to left and right. But note that $|T| = |T_l| + |T_r|$ where $|T_r|$ is the number of leaves in the right subtree and thus $|T_r|/|T| = 1 - |T_l|/|T|$. This shows that the unsymmetry is inessential.

b) If T is in class $BB[\alpha]$, then $|T_l| \leqq (1-\alpha).|T|$, $|T_l| \geqq \alpha.|T|$, $|T_l| \leqq [(1-\alpha)/\alpha].|T_r|$ and $|T_l| \geqq [\alpha/(1-\alpha)].|T_r|$. As an immediate consequence we infer that the depth of a $BB[\alpha]$ tree is $O(\log|T|)$.

We add a leaf to a tree T by replacing a leaf by a tree consisting of one node and two leaves. "If upon the addition of a leaf to a tree in $BB[\alpha]$ the tree becomes unbalanced relative to α, that is, some subtree of T has root-balance outside the range $[\alpha, 1-\alpha]$ then that subtree can be rebalanced by a rotation or a double rotation. In figure 1 we have used squares to represent nodes, and triangles to represent subtrees; the root-balance is given beside each node".

Symmetrical variantes of the operations exist.

If we denote by $x_1, x_2, \ldots$ the number of leaves in the respective subtrees show in figure 1 then the root-balance of B after the rotation is $(x_1+x_2)/(x_1+x_2+x_3)$. Using $\beta_2 = x_2/(x_2+x_3)$ and $\beta_1 = x_1/(x_1+x_2+x_3)$ this is easily seen to be equal to $\beta_1 + (1-\beta_1)\beta_2$. The expressions for the other root-balances are verified similarly.

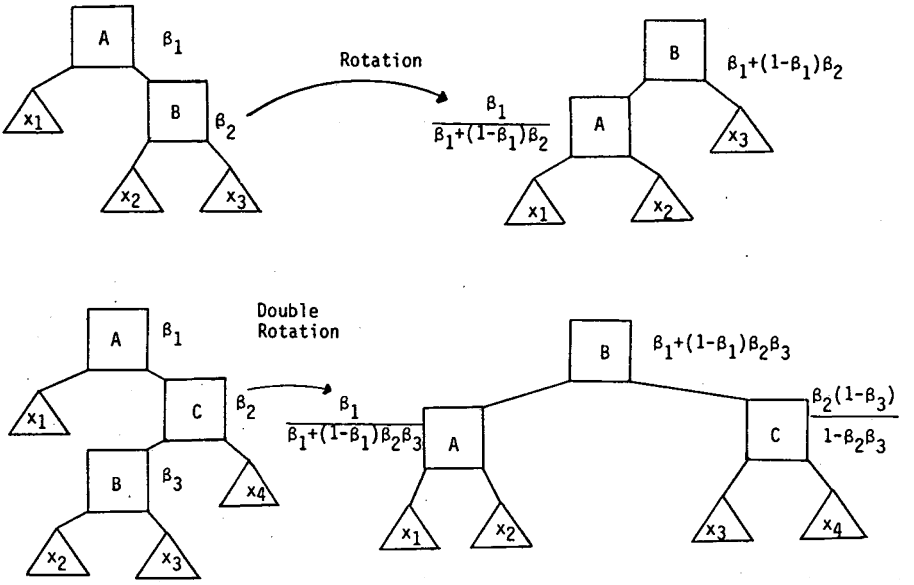


Figure 1

For the sequel, α is a fixed real number, $2/11 \leqq \alpha \leqq 1-\sqrt{2}/2$

Nievergelt and Reingold state in their paper (without proof) that rotations and double-rotations suffice to rebalance a tree after the insertion or deletion of a leaf, provided that α is restricted to the range $2/11 \leqq \alpha \leqq 1-\sqrt{2}/2$. In Blum and Mehlhorn an rigorous proof may be found. They also show that a constant number of rebalancing operations suffices on the average provided that $\alpha < 1-\sqrt{2}/2$, i. e. they show that the total number of rotations and double-rotations needed to process an arbitrary sequence of n insertions and deletions starting with an empty tree is $O(n)$. Here, we need a more detailed outlook at the effect of rotations and double-rotations in weight-balanced trees.

LEMMA 1: *Let $0 < \alpha \leqq 1-\sqrt{2}/2$. Let T be a binary tree with left (right) subtree $T_l(T_r)$ such that:*

1) *T_l and T_r are in $BB[\alpha]$.*

2) $\alpha(1-\alpha) \leqq \rho(T) < \alpha$.

Then a rotation about the root of T will produce a tree in $BB[\alpha]$ if $\rho(T_r) \leqq (1-2\alpha)/(1-\alpha)$ and a double rotation otherwise.

Proof: Compute the balance parameters of the trees obtained by rotation and double rotation and show that they are in the interval $[\alpha, 1-\alpha]$. We give one example and leave the rest to the reader.

Suppose we perform a rotation. Then the balance parameter of the root is $\beta_1+(1-\beta_1)\beta_2$.

By assumption:

$$\alpha(1-\alpha)\leqq\beta_1\leqq\alpha \qquad \text{and} \qquad \alpha\leqq\beta_2\leqq(1-2\alpha)/(1-\alpha).$$

Since $\beta_1+(1-\beta_1)\beta_2$ is increasing in both arguments:

$$\beta_1+(1-\beta_1)\beta_2\leqq\alpha+(1-\alpha)(1-2\alpha)/(1-\alpha)=1-\alpha$$

and:

$$\beta_1+(1-\beta_1)\beta_2\geqq\alpha(1-\alpha)+(1-\alpha(1-\alpha)).\alpha=\alpha(1-\alpha+1-\alpha+\alpha^2)\geqq\alpha$$

if $2-2\alpha+\alpha^2\geqq 1$ if $(\alpha-1)^2\geqq 0$. □

A symmetrical variant of lemma 1 exists. Together they show that rotations and double-rotations suffice to rebalance a *BB* [α]-tree as long as the root-balances are in the range $[\alpha(1-\alpha),\ 1-\alpha(1-\alpha)]$.

DEFINITION: A node v in a binary tree is *balancable* if the balance $\rho(v)$ of v is in $[\alpha(1-\alpha),\ 1-\alpha\ (1-\alpha)]$. A pair (a, b) of real numbers is balancable if $b/a\in[\alpha\ (1-\alpha),\ 1-\alpha\ (1-\alpha)]$.

D-trees are an extension of *BB*[α] trees. Given objects $B_0, B_1, \ldots, B_n$ and access frequencies $q_0, q_1, \ldots, q_n$ let T be a *BB*[α] tree with $W=q_0+q_1+\ \ldots\ +q_n$ leaves. We label the leaves of T according to the following rule. The left-most q_0 leaves are labelled by B_0, the next q_1 leaves are labelled by $B_1, \ldots$

DEFINITION: *a*) A leaf labelled by B_j is a j-leaf.

b) A node v of T is a j-node iff all leaves in the subtree with root v are j-leaves and v's father does not have this property.

c) A node v of T is the j-joint iff all j-leaves are descendants of v and neither of v's sons has this property.

d) Consider the j-joint v. q'_j j-leaves are to the left of v and q''_j j-leaves are to the right of v. If $q'_j\geqq q''_j$ then the j-node of minimal depth to the left of v is active, otherwise the j-node of minimal depth to the right of v is active.

e) The thickness $th(v)$ of a node v is the number of leaves in the subtree with root v.

A *D*-tree is finally obtained from the *BB* [α]-tree T by:

1) Pruning all proper descendants of j-nodes.

2) Storing in each node.

a) a query of the form "*if* $X\leqq B_i$ *then* go left *else* go right";

b) the type of the node: joint node, *j*-node or neither of above;

c) its thickness;

d) in the case of the *j*-joint the number of *j*-leaves in its left and right subtree.

It was shown in Mehlhorn, 1979 that:

a) The depth a_j of the active *j*-node is $O(\log W/q_j)$, more precisely $a_j \leqq c_1 \log W/q_j + c_2$ where:

$$c_1 = 1/\log(1/(1-\alpha)) \qquad \text{and} \qquad c_2 = 1 + c_1.$$

b) Changes of the tree structure after increasing (decreasing) access frequency q_j by 1 are limited to the path from the root to the active *j*-node and hence take time $O(\log W/q_j)$.

Compact *D*-trees were also introduced. They give the same access time and update time bound, but use less space.

3. CONCATENABLE QUEUES BASED ON WEIGHT-BALANCED TREES

Aho, Hopcroft and Ullman introduced the concept of concatenable queues. A Concatenable queue is a family of subsets of some ordered universe U together with the operations INSERT, DELETE, MIN, MEMBER, CONCATENATE and SPLIT where:

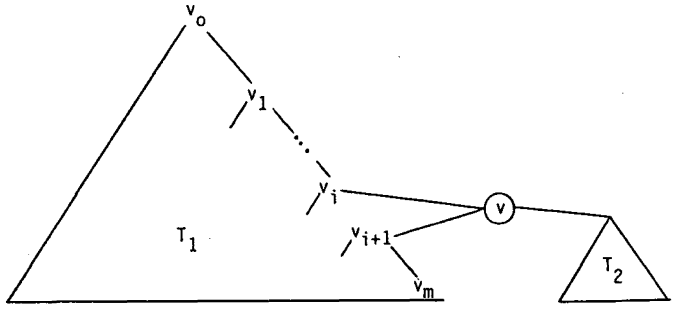
INSERT (a, S)	: $S \leftarrow S \cup \{a\}$
DELETE (a, S)	: $S \leftarrow S - \{a\}$
MIN (S)	: $\min\{a;\ a \in S\}$
MEMBER (a, S)	: the predicate $a \in S$
CONCATENATE (S_1, S_2, S_3)	: $S_1 \leftarrow S_2 \cup S_3$
SPLIT (a, S, S_1, S_2)	: $S_1 \leftarrow \{x;\ x \leqq a \text{ and } x \in S\}$ and
	: $S_2 \leftarrow \{x;\ x > a \text{ and } x \in S\}$

The operation CONCATENATE is only applicable if $\max S_2 < \min S_3$. The sets S_2, S_3 (the set S) cease to exist after an application of CONCATENATE (, S_2, S_3) (SPLIT (, S, ,)).

Various implementations of Concatenable queues exist (*cf.* e. g. Aho, Hopcroft and Ullman, Mehlhorn, 1977). All of them are based on some sort of height-balanced trees (2-3 trees, *HB*-trees) and require $O(\log n)$ time units per operation.

In this section we show that weight-balanced trees also support the full repertoire of concatenable queue operations. A set S of size n is represented by a $BB[\alpha]$-tree with n leaves. The leaves are labelled from left to right by the elements of S in increasing order. An (interior) node is labelled by the label of the rightmost leaf in the subtree rooted at v. In order to search for an element X in the tree with root v we only have to compare X with the label of the left son of v. If X is not greater than we continue the search process in the left subtree, otherwise we proceed to the right subtree. It is well known that the operations INSERT, MIN, DELETE, MEMBER can be performed in $O(\log|S|)$ time units (Reingold and Nievergelt, Mehlhorn, 1977).

CONCATENATE : Let sets S_1, S_2 be represented by $BB[\alpha]$-trees T_1 and T_2, $\max S_1 < \min S_2$. Assume w. l. o g. that $|S_1| \geqq |S_2|$. Let $v_0, v_1, \ldots, v_m$ be the right spine of T_1; i. e. v_0 is the root, v_{i+1} is the right son of v_i for $0 \leqq i < m$, and v_m is a leaf. We will construct the following tree.



In order to make that construction work we only need to show that there exists some i such that $v_0, \ldots, v_i$ and v are balancable in the new tree. This follows from the following lemma.

DEFINITION : A sequence $w_0, w_1, w_2, \ldots$ of positive reals is α-admissible if:

$$w_{i+1}/w_i \in [\alpha, 1-\alpha],$$

for all i.

REMARK : Let $v_0, v_1, v_2, \ldots$ be a path through a $BB[\alpha]$-tree, v_0 being the root. Let $w_i = th(v_i)$ be the thickness of node v. Then $w_0, w_1, w_2, \ldots$ is α-admissible.

REMARK : In the following estimations we will often use the fact that for $b > a$ the function $f(x) = (x+a)/(x+b)$ is strictly increasing in x and $g(x) = (x-a)/(x-b)$ is strictly decreasing in x.

SPINE-LEMMA: *Let* $w_0, w_1, w_2, \ldots, w_n$ *be an* α*-admissable sequence and let* $d \in \mathbb{R}_+$. *If:*

$$d/(w_0+d) \leqq 1-\alpha(1-\alpha),$$

then there exists some i *namely:*

$$i = \begin{cases} -1 & \text{if } d/(d+w_0) \geqq \alpha, \\ \max\{j;\, d/(d+w_j) < \alpha\}, & \text{otherwise,} \end{cases}$$

such that:

1) $(w_{i+1}+d, d)$ *is balancable or* $i=n$;
2) $(d+w_j, d+w_{j+1})$ *is balancable for* $j \leqq i$;
3) $i \leqq \max(c_1 \log(w_0/d)+c_2, -1)$ *where:*

$$c_1 = 1/\log(1/(1-\alpha)) \quad \text{and} \quad c_2 = 1+c_1 \log \alpha.$$

Proof: If $d/(w_0+d) \geqq \alpha$ then put $i := -1$.

Otherwise let i be maximal such that:

$$d/(w_i+d) < \alpha.$$

Then $d/(w_{i+1}+d) \geqq \alpha$ or $i=n$, $d < \alpha\, w_i/(1-\alpha)$ and $d \geqq \alpha\, w_{i+1}/(1-\alpha)$.

1) We have to show: If $i<n$ then $(w_{i+1}+d, d)$ is balancable. Since $d/(d+w_{i+1}) \geqq \alpha$ by definition, it remains to show that:

$$d(d+w_{i+1}) \leqq 1-\alpha(1-\alpha).$$

For $i=-1$ this is true by assumption, for $i \geqq 0$ we even show:

$$d/(d+w_{i+1}) \leqq 1-\alpha.$$

Since $d < \alpha\, w_i/(1-\alpha)$, $w_{i+1} \geqq \alpha\, w_i$ and $\alpha < 1/3$:

$$d/(d+w_{i+1}) \leqq \frac{\alpha/(1-\alpha)}{\alpha/(1-\alpha)+\alpha} = \frac{\alpha}{\alpha+\alpha(1-\alpha)} \leqq \frac{\alpha}{5\alpha/3} \leqq 3/5 \leqq 2/3 < 1-\alpha.$$

2) $(d+w_j, d+w_{j+1})$ is balancable for $j \leqq i$.

Certainly:

$$\frac{d+w_{j+1}}{d+w_j} \geqq \frac{w_{j+1}}{w_j} \geqq \alpha.$$

Also $d<\alpha w_i/(1-\alpha)\leqq\alpha w_j/(1-\alpha)$ and $w_{j+1}\leqq(1-\alpha)w_j$. Hence:

$$\frac{d+w_{j+1}}{d+w_j}\leqq\frac{\alpha/(1-\alpha)+(1-\alpha)}{\alpha/(1-\alpha)+1}\leqq\alpha+(1-\alpha)^2=\alpha+1-2\alpha+\alpha^2=1-\alpha(1-\alpha).$$

3) Since $w_k\leqq(1-\alpha)^k w_0$:

$$d/(d+w_k)\geqq\frac{d}{d+(1-\alpha)^k w_0}=\frac{d/w_0}{d/w_0+(1-\alpha)^k}\geqq\alpha$$

if $d/w_0\geqq\alpha(1-\alpha)^{k-1}$, i.e. $k-1\geqq\log(d/\alpha w_0)/\log(1-\alpha)$.
Since i is chosen such that $d/(d+w_i)<\alpha$ we cannot have:

$$i-1\geqq\log(\alpha w_0/d)/\log(1/(1-\alpha)).$$

Hence:

$$i<\log(\alpha w_0/d)/\log(1/(1-\alpha))+1.$$

This proves 3) in the case $i\geqq 0$. For $i=-1$ there is nothing to show. □

Let $w_j=th(v_j)$ for $0\leqq j\leqq m$. Then $w_0=|S_1|$ and $w_m=1$. Let $d=|S_2|\geqq 1$. Then:

$$d/(w_0+d)\leqq 1/2\leqq 1-\alpha(1-\alpha)$$

and hence the spine lemma applies. Let i be defined as in the spine lemma. Since $d/(w_m+d)=d/(1+d)\geqq 1/2\geqq\alpha$ we have $i<m$. We construct a new node v, make v_{i+1} the left son of v, make the root of T_2 the right son of v and finally make v the right son of v_i. The label of v is the same as the label of the root of T_2. The balance of node v is $w_{i+1}/(|S_2|+w_{i+1})$, the balance of $v_j (j\leqq i)$ is $1-[(w_{j+1}+d)/(w_j+d)]$. By 1) and 2) of the spine lemma $v, v_i, \ldots, v_0$ are balancable. Hence we only have to walk back to the root and restore balance by rotations and double-rotations. Finally:

$$i=O(\log w_0/d)=O(\log|S_1|-\log|S_2|).$$

This proves:

LEMMA: *Concatenate* $(\ ,S_1,S_2)$ *takes* $O(|\log|S_1|-\log|S_2||)$ *units of time. Here* $|\ \ |$ *denotes absolute value.*

Split: Let the set S be represented by $BB[\alpha]$-tree T and let a be an arbitrary element of the universe. We first search for a in tree T. This takes $O(\log|S|)$ units of time. Then we delete all nodes on the path of search and collect the left and right subtrees of that path in two sets $\mathscr{F}_l$ and $\mathscr{F}_r$ respectively. $\mathscr{F}_l$ is an ordered forest of $BB[\alpha]$ trees $T_1, T_2, \ldots, T_q$ for some $q\leqq\log|S|$.

Let t_i be the thickness of T_i, $1 \leqq i \leqq q$.

Trees $T_{i+1}, \ldots, T_q$ are subtrees of the right brother tree of T_i. Hence:

$$t_{i+1} + \ldots + t_q \leqq [(1-\alpha)/\alpha] . t_i$$

by the remark following the definition of $BB[\alpha]$-trees. The tree $T_1, \ldots, T_q$ represent sets $S_1, \ldots, S_q$. We execute Concatenate $(\tilde{S}_{q-1}, S_{q-1}, S_q)$, Concatenate $(\tilde{S}_{q-2}, S_{q-2}, \tilde{S}_{q-1})$..., Concatenate $(\tilde{S}_1, S_1, \tilde{S}_2)$ and obtain a tree T which represents the first set obtained in the split Split $(a, S, \ , \ )$. Executing the above sequence of $q-1$ Concatenate-Operations take:

$$\sum_{i=1}^{q-1} O(|\log|S_i| - \log|S_{i+1} \cup \ldots \cup S_q|),$$

units of time. Here $|\log \ldots|$ denotes absolute value. Since:

$$\frac{|S_{i+1}| + \ldots + |S_q|}{|S_i|} = \frac{t_{i+1} + \ldots + t_q}{t_i} \leqq \frac{1-\alpha}{\alpha},$$

we have:

$$\left|\log \frac{|S_i|}{|S_{i+1}| + \ldots + |S_q|}\right|$$

$$\leqq \begin{cases} \log \dfrac{|S_i|}{|S_{i+1}| + \ldots + |S_q|} & \text{if} \quad |S_i| \geqq |S_{i+1}| + \ldots + |S_q|, \\ \log(1-\alpha)/\alpha, & \text{otherwise} \end{cases}$$

$$\leqq 2\log\frac{1-\alpha}{\alpha} + \log \frac{|S_i|}{|S_{i+1}| + \ldots + |S_q|},$$

and hence:

$$\sum_{i=1}^{q-1} O(|\log(|S_i|/(|S_{i+1}| + \ldots + |S_q|))|)$$

$$= \sum_{i=1}^{q-1} O(2\log(1-\alpha)/\alpha + \log|S_i|/(|S_{i+1}| + \ldots + |S_q|))$$

$$= O(q) + O\left(\sum_{i=1}^{q-1} \log|S_i|/(|S_{i+1}| + \ldots + |S_q|)\right),$$

since:

$$\sum_{i=1}^{q} O(1 + f(x_i)) = O(q) + O\left(\sum_{i=1}^{q} f(x_i)\right),$$

for every function f and arbitrary $x_i's$:

$$= O(q) + O\left(\log \prod_{i=1}^{q-1} |S_i|/(|S_{i+1}| + \ldots + |S_q|)\right)$$

$$= O(q) + O(\log |S_1|/|S_q|) = \quad O(q + \log |S_1|) = O(\log |S|).$$

This shows that Splits can also be executed in time $O(\log S)$. Note further that non-trivial bounds for q and S_q would allow us to improve the time bound. This fact will be used in the discussion of case 1 in section weight decreases.

THEOREM 1: *Weight-Balanced Trees support the full repertoire of Concatenable Queue operations with a performance bound of* $O(\log n)$ *per operation.*

4. WEIGHT INCREASES IN A *D*-TREE

We now return to D-trees. In this section we treat weight increases, in the next section weight decreases. Let T be a D-tree for weights $q_0, q_1, \ldots, q_n$. Suppose we want to increase q_j by d. If $d=1$ then the problem was treated already in Mehlhorn, 1979. If d is small with respect to q_j (precisely $d<(\alpha/(1-\alpha))q_j$) then the spine lemma is almost the answer. This is worked out in 4.1. If d is large with respect to q_j then we need an extension of the spine lemma, the path lemma (*see* 4.2).

4.1. Small weight increases

In this section we show how to deal with small weight increases. Theorem 2 is almost a direct consequence of the spine lemma.

THEOREM 2: *Let T be a D-tree of total thickness $W=q_0+q_1+\ldots+q_n$. Increasing q_j by d can be done in time $O(\log W/q_j)$ provided that $d<(\alpha/(1-\alpha))q_j$ or $d=1$.*

Proof: The case $d=1$ is treated in Mehlhorn, 1979.

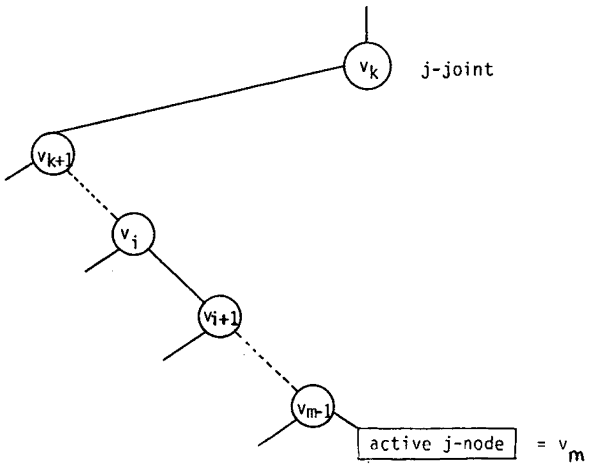
Suppose $d<(\alpha/(1-\alpha))\,q_j$. We first access the active j-node. This takes time $O(\log W/q_j)$. Let $v_0, v_1, \ldots, v_k, \ldots, v_m$ be the path from the root of T to the active j-node; v_k is the j-joint. It is possible that $k=m$. In this case there is exactly one j-node.

Let $w_j=th(v_j)$ for $0\leqq j\leqq m$. Then $w_k\geqq q_j>((1-\alpha)/\alpha)\,d$ and hence $d/(d+w_k)<\alpha$. Let i be defined as in the spine lemma. Then $i\geqq k$.

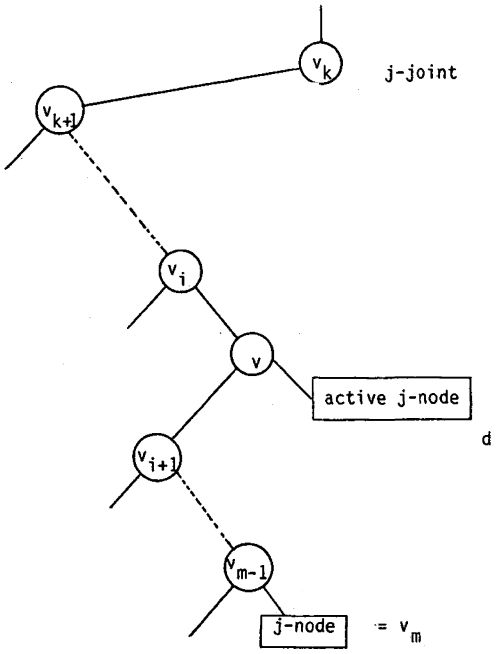
Case 1: $i\geqq m-1$. (This case will certainly apply if the active j-node is the j-joint.) Then we increase the thickness of the active *j-node* by d, i. e. we increase its

thickness from v_m to $v_m + d$. By part 2) of the spine lemma the nodes $v_0, \ldots, v_{m-1}$ remain balancable. So we only have to walk back to the root and restore balance by rotations and double-rotations as described in Mehlhorn, 1979.

Case 2: $i \leqq m-2$. The relative position of j-joint and active j-node is as shown in the following figure. (We assume w.l.o.g. that the active j-node is a left descendant of the j-joint.)



We change the tree into.



v is a new node. Its right son is the new active j-node of thickness d. By the spine lemma $v, v_i, v_{i-1}, \ldots, v_0$ are balancable. Hence we only have to walk back to the root and restore balance by rotations and double rotations as described in Mehlhorn, 1979.

In either case $O(\log W/q_j)$ time units suffice to restore the D-tree property. □

If $\alpha = 1/4$ then theorem 1 solves the problem as long as weights are never increased by more than 33% in a single step. Iterating this process gives us a solution to the general problem with time bound $O(\max(1, \log d/q_j) . \log W/q_j)$. Namely write $d = d_1 + d_2 + \ldots + d_k$ where:

$$d_i = \frac{\alpha}{2(1-\alpha)}(q_j + d_1 + \ldots + d_{i-1}) \qquad \text{for} \quad i < k$$

and:

$$d_k < \frac{\alpha}{1-\alpha}(q_j + d_1 + \ldots + d_{k-1}).$$

Then $k = O(\max(1, \log d/q_j))$. Increase q_j by d_1, then by d_2,... Since:

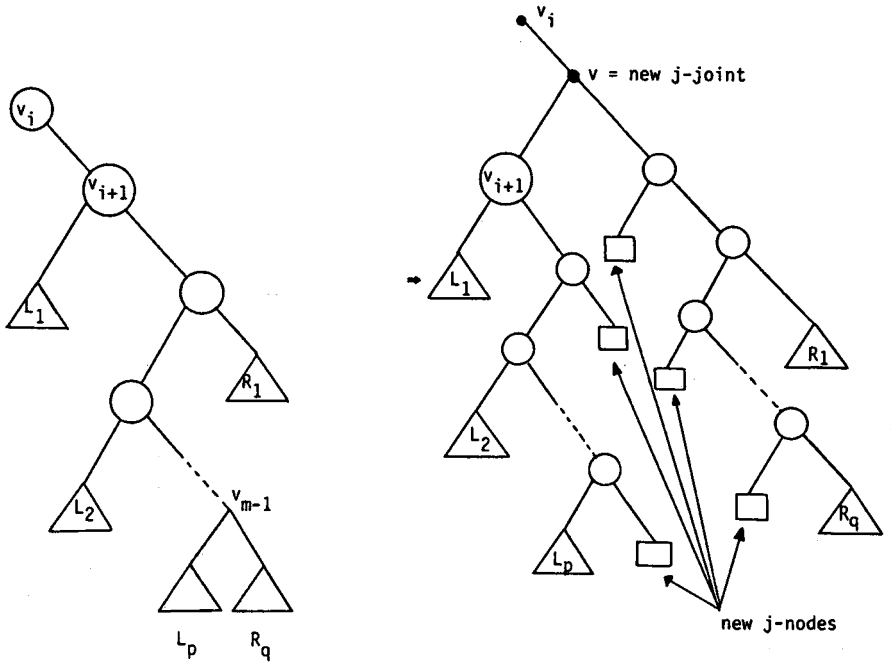
$$\frac{W + d_1 + d_2 + \ldots + d_i}{q_j + d_1 + d_2 + \ldots + d_i} \leqq \frac{W}{q_j}$$

we obtain the above time bound. We show next that we can turn the multiplicative factor $\max(1, \log d/q_j)$ into an additive factor.

4.2. Arbitrary weight increases

We want to improve upon the procedure described at the end of the previous section. Suppose we want to increase q_j by d. Let $v_0, v_1, \ldots, v_m$ be the path from the root to the active j-node. As above we want to identify a node v_i such that we can leave the total weight increase below v_i without destroying the balance above v_i too much. However, it will not be possible to leave the total weight increase d in one additional j-node. Rather we will build two copies of the subtree rooted at v_{i+1}. In one copy we replace the left subtrees along the path from v_{i+1} to the active j-node by new j-nodes of the appropriate weight, in the other copy we replace the right subtrees. Then we make these copies the sons of a new node v. v is the new j-joint. Finally v will take the position of v_{i+1} as a son of v_i.

In order to show that this strategy works we need to prove a lemma similar to the spine lemma. Before stating the lemma we need to discuss one of the



assumptions in that lemma. Let's revisite the proof of theorem 2 again. Let $v_0, v_1, \ldots, v_m$ be the path from the root to the active j-node and let i be defined as in the spine lemma, namely $i = \max\{j;\, d/(d+w_j) < \alpha\}$. If $i \geqq m-1$ then case 1 of the proof applies. In that case we did not make use of the fact that $i \geqq k$, i. e. v_i is a descendant of the j-joint. In other words, if $d/(d+w_{m-1}) < \alpha$ then we solved the problem already.

PATH-LEMMA: *Let $w_0, w_1, \ldots, w_{m-1}$ be an α-admissible sequence and let $d \in \mathbb{R}_+$. If:*

$$(d-w_0)/d < \alpha \qquad \text{and} \qquad d/(d+w_{m-1}) \geqq \alpha,$$

then there exists an i namely:

$$i = \min[\{j; (d-w_{j+2})/d \geqq \alpha\} \cup \{m-2\}]$$

such that:

1) $(d, d-w_{i+2})$ *is balancable or* $i+2=m$.
2) $(w_{i+1}+d, d)$ *is balancable.*
3) $(d+w_j, d+w_{j+1})$ *is balancable for:*

$$j \leqq i - \log\alpha/\log(1-\alpha) + 1.$$

Proof: If $i<m-2$ then $(d-w_{i+2})/d \geqq \alpha$ but $(d-w_{i+1})/d<\alpha$ and hence $d \geqq w_{i+2}/(1-\alpha)$ and $d<w_{i+1}/(1-\alpha)$. If $i=m-2$ then $(d-w_{m-1})/d<\alpha$ and hence $d<w_{m-1}/(1-\alpha) \leqq w_{i+1}/(1-\alpha)$. Finally $w_{i+2} \geqq \alpha w_{i+1}$.

1) If $i+2<m$ then $(d-w_{i+2})/d \geqq \alpha$ is true by definition of i. Furthermore:

$$(d-w_{i+2})/d \leqq 1-w_{i+2}/d \leqq 1-\alpha w_{i+1}/(w_{i+1}/(1-\alpha))=1-\alpha(1-\alpha).$$

This proves condition 1).

2) We have to show:

$$\alpha(1-\alpha) \underset{2a}{\leqq} d/(w_{i+1}+d) \underset{2b}{\leqq} 1-\alpha(1-\alpha).$$

We first show 2*b*. Since $d<w_{i+1}/(1-\alpha)$:

$$d/(w_{i+1}+d) \leqq \frac{1/(1-\alpha)}{1+1/(1-\alpha)} \leqq 1/(2-\alpha) \leqq 1-\alpha,$$

iff $1 \leqq (2-\alpha)(1-\alpha)=2-3\alpha+\alpha^2$;
iff $\alpha^2-3\alpha+1 \geqq 0$;
iff $(\alpha-3/2)^2 \geqq 5/4$;
if $\alpha \leqq (3-\sqrt{5})/2 \approx 0.382$.

This shows 2*b*. Next we prove 2*a*.

If $i=m-2$ then there is nothing to show.

Otherwise $d \geqq w_{i+2}/(1-\alpha)$ and $w_{i+2} \geqq \alpha w_{i+1}$ and hence $d \geqq \alpha w_{i+1}/(1-\alpha)$. Thus:

$$d/(d+w_{i+1}) \geqq \frac{\alpha/(1-\alpha)}{\alpha/(1-\alpha)+1} \geqq \frac{\alpha}{\alpha+1-\alpha}=\alpha,$$

This shows 2*a*.

3) We have to show:

$$\alpha.(1-\alpha) \underset{3a}{\leqq} \frac{d+w_{j+1}}{d+w_j} \underset{3b}{\leqq} 1-\alpha(1-\alpha),$$

for all $j \leqq i-\log \alpha/\log(1-\alpha)+1$:

$$\frac{d+w_{j+1}}{d+w_j} \geqq \frac{w_{j+1}}{w_j} \geqq \alpha.$$

This shows 3 *a*. Furthermore $w_j \geqq w_{j+1}/(1-\alpha)$ for all j and hence:

$$w_{i-k+1} \geqq w_i/(1-\alpha)^{k-1} \geqq w_{i+1}/(1-\alpha)^k \geqq d.(1-\alpha)/(1-\alpha)^k \geqq d/(1-\alpha)^{k-1}.$$

Thus:

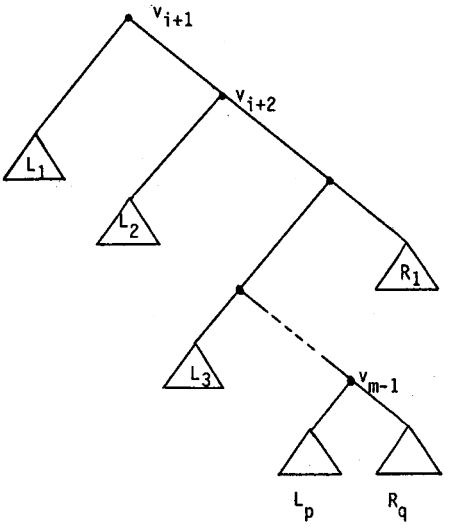
$$\frac{d+w_{i-k+1}}{d+w_{i-k}} \leqq \frac{(1-\alpha)^{k-1}+1}{(1-\alpha)^{k-1}+(1/(1-\alpha))} = \frac{(1-\alpha)^k+1-\alpha}{(1-\alpha)^k+1} \leqq 1-\alpha(1-\alpha),$$

iff $1/((1-\alpha)^k+1) \geqq (1-\alpha)$;
iff $1 \geqq (1-\alpha)^{k+1}+(1-\alpha)$;
iff $(1-\alpha)^{k+1} \leqq \alpha$;
iff $k+1 \geqq \log \alpha/\log(1-\alpha)$.

This proves 3 *b* for all j with $i-j=k \geqq \log \alpha/\log(1-\alpha)-1$. □

We are now ready to present the solution to the general problem. Suppose we want to increase q_j by d. Let $v_0, v_1, \ldots, v_m$ be the path from the root to the active j-node. Let $w_j = th(v_j)$. By the discussion proceeding the path lemma we may as well assume that $d/(d+w_{m-1}) \geqq \alpha$.

Case 1: $(d-w_0)/d < \alpha$. Then the path lemma applies. Let i be defined as in the path lemma. Then $i \leqq m-2$. We may assume w. l. o. g. that v_{i+2} is the right son of v_{i+1}. Consider the path from v_{i+1} to v_{m-1} (both end points included). Let $L_1, \ldots, L_p (R_1, \ldots, R_q)$ be the left (right) subtrees along that path. L_p and R_q are the two sons of v_{m-1}. One of them is the active j-node.



Let $l_j(r_j)$ be the thickness of $L_j(R_j)$. Then:

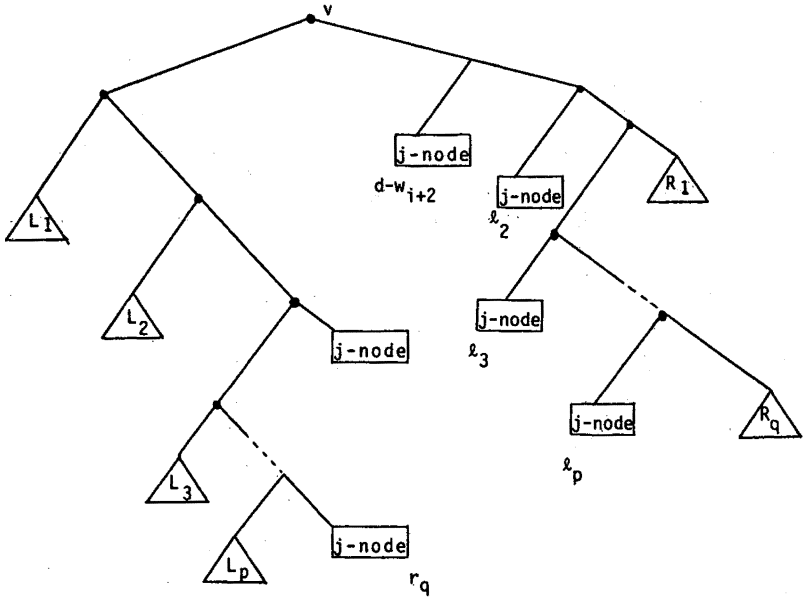
$$w_{i+1} = l_1 + l_2 + \ldots + l_p + r_1 + \ldots + r_q,$$

$$w_{i+2} = \quad l_2 + \ldots + l_p + r_1 + \ldots + r_q.$$

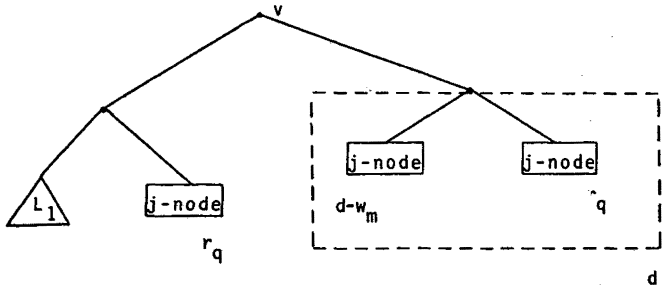
Construct two copies of the tree T rooted at v_{i+1}.

In the first copy, call it T_1, replace the trees $R_1, \ldots, R_q$ by j-nodes of thickness $r_1, \ldots, r_q$ respectively, in the second copy, call it T_2, replace the trees $L_1, \ldots, L_p$ by j-nodes of thickness $d - w_{i+2}, l_2, \ldots, l_p$ respectively.

Finally make T_1 (T_2) the left (right) subtree of a new node v, and let v replace v_{i+1} as a son of v_i.



REMARK: Note that on either side it may be possible to combine j-nodes into larger nodes. This is easily done by checking if the brothers of the newly



constructed j-nodes are j-nodes. We assume for the sequel that these combinations are done. In particular, if $i=m-2$ and v_m is the right son of v_{m-1} then the right son of v is a j-node of thickness d. (Note that $w_{m+2}=r_q$.)

T_1 is certainly a tree in $BB[\alpha]$, as is the right subtree of T_2. The right subtree of T_2 has thickness w_{i+2}, its left subtree is a j-node of thickness $d-w_{i+2}$. If $i=m-2$ then we can combine both nodes to a single j-node of thickness d, *cf.* the preceeding remark. If $i<m-2$ then the root of T_2 is balancable by condition 1) of the path lemma. Furthermore v is balancable by condition 2) of the path lemma.

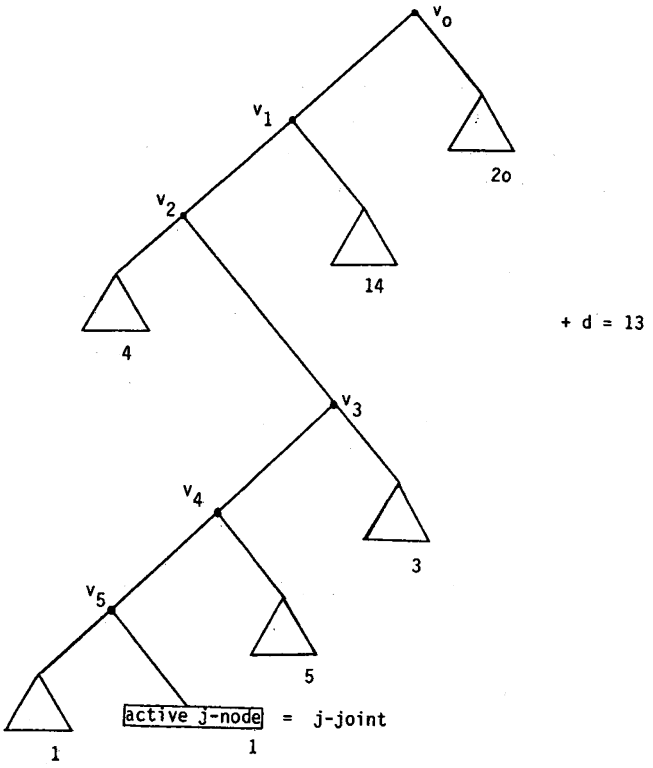
Next we need to show that the j-leaves still form a contiguous segment of the leaves of the underlying $BB[\alpha]$ tree, that we can determine the queries assigned to the new nodes efficiently, and that we can determine the type of each of the new nodes. The first problem is resolved by the following observation. Either L_p or R_q is the active j-node and hence we insert the new j-leaves immediately adjacent to some already existing j-leaves. Hence the j-leaves still form a contiguous segment of leaves. The assignment of queries to the fathers of the new j-nodes of thickness $r_1, \ldots, r_q$ is also easy. The active j-node has to be to the right of them and hence they receive the query "*if* $X \leqq B_{j-1}$ *then* go left *else* go right". Analogously the query "*if* $X \leqq B_j$ *then* left *else* right" is assigned to the fathers of the new j-nodes of thickness $d-w_{i+2}, l_2, \ldots, l_p$ respectively. It remains to consider node v. If v is not the j-joint then one of its sons is a j-node and we assign the query as described above. Suppose now, that v is the j-joint. The distribution of j-leaves with respect to v is easily computed from the distribution with respect to the old j-joint and the numbers $r_1, \ldots, r_q, d-w_{i+2}, l_2, \ldots, l_p$. Note that the old j-joint has to be one of the nodes $v_{i+1}, v_{i+2}, \ldots, v_{m-1}$ in this case.

Let q_1 (q_2) be the number of j-leaves to the left (right) of it in the D-tree before the insertion. Then $q_1+r_1+\ldots+r_q(q_2+d-w_{i+2}+l_2+\ldots+l_p)$ j-leaves are to the left (right) of v.

It remains to show how to determine the type of the new nodes. This was done already in the case of v. Consider any of the new nodes in T_1. If such a node has an L_i as its left son then it is an "xyz"-joint if the corresponding node in T was a an 'xyz'-joint. If it does not have an L_i as its left son then it is of no special type. An analogous statement holds for T_2.

This shows that we still have a D-tree after the weight increase of q_j by d except that some of the nodes $v_0, v_1, \ldots, v_i, v$ root of T_2 may be out of balance. Also $O(\log(W/q_j))$ time units were spent up to this point.

Example: $\alpha = 1/4$, we want to increase:



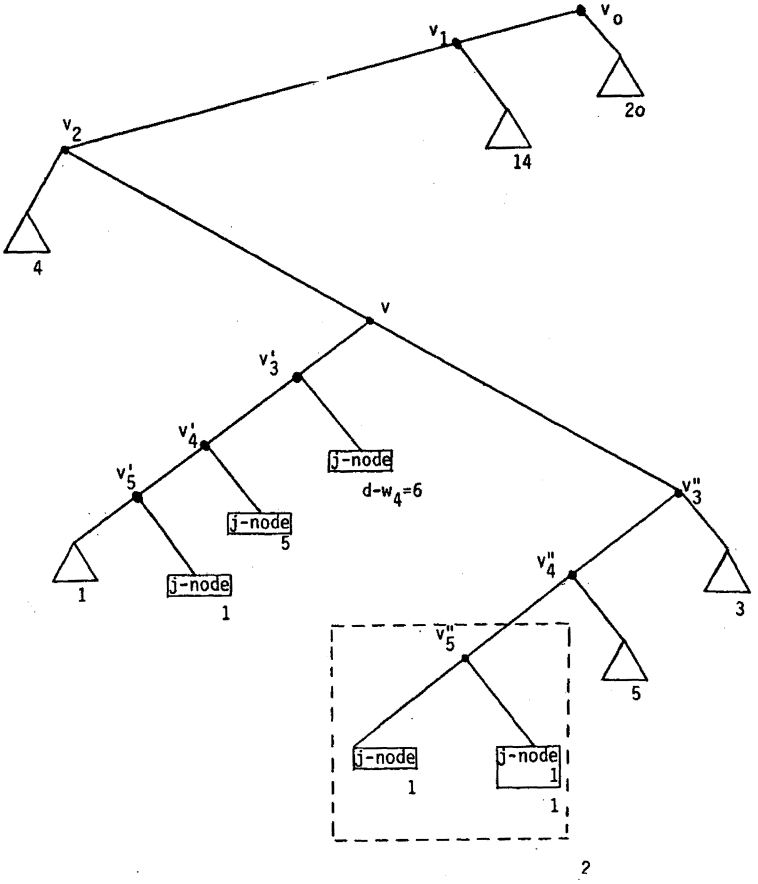
q_j from 1 by 13. The path from the root to the active j-node defines the following α-admissible sequence 48, 28, 14, 10, 7, 2, 1.

In the path lemma we have $i=2$. We construct:

Node v is the new j-joint. Among the nodes v_0, v_1, v_2, v, v_3' only v_2 is out of balance. It's balance is $4/27 < 1/4(1-1/4)$. Hence v_2 is not even balancable. This is in accordance with claim 3) of the path lemma.

It remains to show how to rebalance nodes v_0, v_1, ..., v_i, v, root of T_2. By claims 1 and 2 of the path lemma nodes v, root of T_2 are balancable. Hence we can use rotations and double rotations as described in Mehlhorn, 1977 *a*.

Furthermore, by claim 3 there is some $p \leqq \log\alpha/\log(1-\alpha)-1$ such that $(d+w_j, d+w_{j+1})$ is balancable for all $j \leqq i-p$ and either $(d+w_{i-p+1}, d+w_{i-p+2})$ is not balancable or $i-p+1=i+1$, i.e. $p=0$. If $p=0$ works then we only have to walk back to the root and restore balance by means of rotations and double-rotations. Suppose $p>0$. (In our example $i=2$ and



$p=1$.) Consider the path from v_{i-p+1} to the root v of the newly constructed tree T of thickness $d+w_{i+1}$.

Let $L_1, L_2, \ldots, L_r$ and $R_1, \ldots, R_q$ be the left and right subtrees along that path. Let $l_s(r_s)$ be the thickness of $L_s(R_s)$. So we are left with an ordered forest $\{L_1, L_2, \ldots, L_r$, tree T with root v, $R_q, \ldots, R_1\}$ of D-trees. This forest contains $p+1$ trees. Consider any left subtree L_s. Its thickness l_s is equal to $w_j - w_{j+1}$ for some j, $i-p+1 \leqq j \leqq i$.

Hence:

$$w_j - w_{j+1} \geqq \alpha w_j \geqq \alpha w_{i+1}/(1-\alpha) \geqq \alpha d,$$

by the proof of the path lemma. Also:

$$1-\alpha(1-\alpha) < \frac{w_{j+1}+d}{w_j+d} \leqq \frac{((1-\alpha)/\alpha)(w_j-w_{j+1})+d}{(w_j-w_{j+1})/\alpha+d}.$$

Thus:

$$\left[\frac{1-\alpha(1-\alpha)}{\alpha}-\frac{1-\alpha}{\alpha}\right](w_j-w_{j+1})\leqq\alpha(1-\alpha)d$$

and:

$$w_j-w_{j+1}\leqq(1-\alpha)d.$$

We want to use the spine lemma to insert $L_r, \ldots, L_1$ (in that order) into the left spine of T and $R_q, \ldots, R_1$ (in that order) into the right spine of T. T has thickness $d+w_{i+1}$. Hence:

$$\frac{w_j-w_{j+1}}{w_j-w_{j+1}+d+w_{i+1}}\leqq\frac{(1-\alpha)d}{(1-\alpha)d+d}-\frac{1-\alpha}{2-\alpha}\leqq\frac{1}{2}$$

and the spine lemma applies (w_j-w_{j+1} plays the role of d and $d+w_{i+1}$ the role of w_0 in that lemma). From the proof of the path lemma we know $w_{i+2}\leqq(1-\alpha)d$ and $w_{i+1}\leqq(1/\alpha)w_{i+2}$. Hence $w_{i+1}\leqq((1-\alpha)/\alpha)d$. After inserting the first $(p-1)$ trees into the left and right spine of T its thickness has grown to at most:

$$d+w_{i+1}+(p-1).(1-\alpha)d\leqq[1/\alpha+(p-1)(1-\alpha)]d.$$

Hence the i of the spine lemma is in:

$$O\left(\log\frac{[1/\alpha+(p-1)(1-\alpha)]d}{\alpha d}\right)=O(1).$$

This shows that the trees $L_r, \ldots, L_1, R_q, \ldots, R_1$ will be inserted above some constant depth in T and hence these insertions take time $O(1)$. It is easy to see how to update the additional D-tree information during the insertion process. Finally, we use rotations and double-rotations to restore balance above v_{i-p}. This shows that increasing q_j by d can be done in time $O(\log W/q_j)$ provided that $(d-w_0)/d<\alpha$, i.e. $d<w_0/(1-\alpha)$.

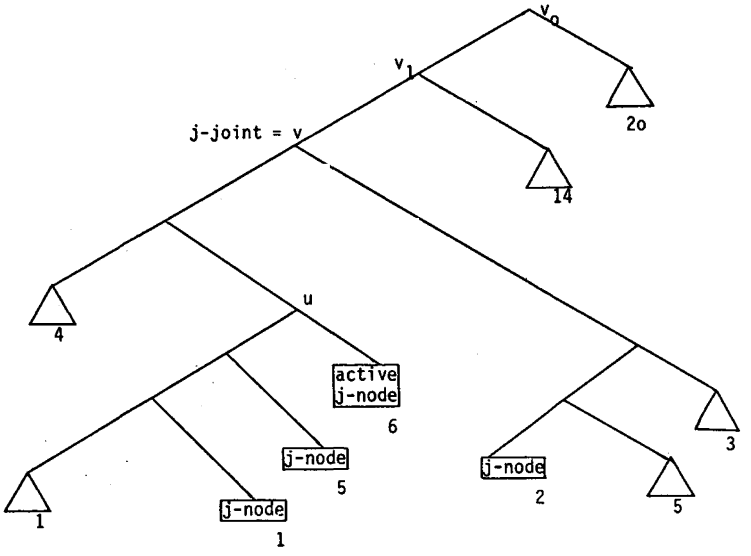
Example continued: In our example we have $p=1$, i.e. we need to insert the left subtree of v_2 in the left spine of the tree with root v. We obtain:

Case 2: $(d-w_0)/d\geqq\alpha$, i.e. $d\geqq w_0/(1-\alpha)$.

Choose any $d'<d$ such that $(d'-w_0)/d'<\alpha$ and $(d'-w_1)d'>\alpha$.

Then go through the above with d' instead of d.

Case 1 applies with $i=-1$ and hence the tree shown in the discussion of case 1 will be the entire D-tree after increasing q_j by d'. Now the root of the D-tree is the j-joint and hence we may apply theorem 2 repeatedly in order to increase q_j by additional $d-d'$ units. The discussion following theorem 1 shows that $O(\log(d/w_0))$ iterations will suffice each of which costs $O(1)$ units of time.



THEOREM 3: *Let T be a D-tree of total thickness $W=q_0+q_1+\ldots+q_n$. Increasing q_j by d can be done in time $O(\log W/q_j+\log(\max(1, d/W)))$.*

So we can increase access frequencies by an arbitrary amount with hardly paying any penalty [only $O(\log d/W)$ time units in addition to the access cost]. It is left as an exercise to the reader that the penalty can be bounded by a constant in compact D-trees, i. e. weight increases in compact trees take time $O(\log W/q_j)$.

5. WEIGHT DECREASES

In this section we will show how to decrease the access frequency q_j by $d \in \mathbb{N}$. We will assume $0 \leqq d \leqq q_j$. The solution will rely heavily upon the spine lemma.

Let $v_0, v_1, \ldots, v_n$ be the path from the root to the active j-node, let v_k be the j-joint, $k \leqq n$. Remember that the number of j-leaves to the left and right of the j-joint are stored in the j-joint and that the thickness w_{n-1} of the father v_{n-1} of the active j-node is at least $q_j/2$. This follows from the fact that all j-leaves which are on the same side of the j-joint as the active j-node are descendants of v_{n-1}. Hence the thickness w_n of the active j-node is at least $\alpha q_j/2$.

Suppose $d > w_n$ first. The following figure shows the relative position of j-joint and j-nodes.

Let x_1 be the thickness of the active j-node and let q'_j, q''_j be the distribution of j-leaves with respect to the j-joint, $q_j = q'_j + q''_j$. Then $x_1 \geqq \alpha q_j/2$.

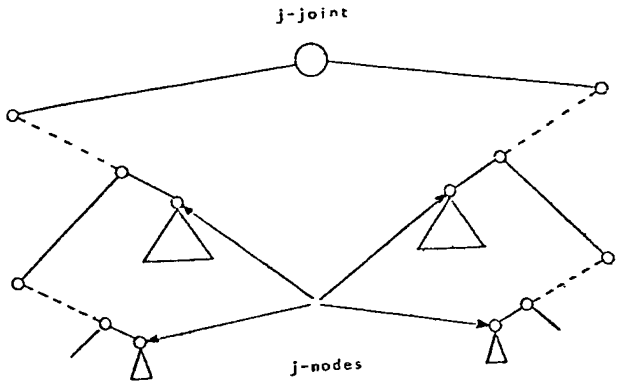


Figure 2. – Dotted lines ... denote zero or more tree edges.

Suppose w.l.o.g. that the active j-node is a left descendant of the j-joint. We delete the active j-node of thickness x_1 and update the distribution numbers $\tilde{q}'_j \leftarrow q'_j - x_1, \tilde{q}''_j \leftarrow q''_j$ in the j-joint. If $\tilde{q}'_j > q''_j$ then we consider next the j-node of minimal depth to the left of the j-joint. Let its thickness be x_2. As above (in the first paragraph of this section) one shows:

$$x_2 \geqq \alpha \tilde{q}'_j \geqq \alpha(\tilde{q}'_j + \tilde{q}''_j)/2 \geqq \alpha(q_j - x_1)/2.$$

We delete the j-node of thickness x_2. Similarly, if $\tilde{q}'_j \leqq \tilde{q}''_j$ then we consider next the j-node of minimal depth to the right of the j-joint. In this fashion we delete j-nodes of thickness $x_1, \ldots, x_{r-1}$ until $x_1 + x_2 + \ldots + x_r \geqq d$. It is easy to see that $x_i \geqq \alpha(q_j - x_1 - \ldots - x_{i-1})/2$ for all i.

If we keep a pointer to the j-joint and to the fathers of the j-nodes on either side of the j-joint which were deleted last then the process above takes time $O(r + \max_{1 \leqq i \leqq r} \text{depth}(x_i))$. Here and in the sequel we will misuse notation and use x_i also for the j-node of thickness x_i. We need a bound on r and depth (x_i).

LEMMA: *Let* $y_1, y_2, \ldots, y_q \in \mathbb{N}$ *with:*

$$y_i \geqq \alpha/2.(y_i + y_{i+1} + \ldots + y_q) \quad \textit{for} \quad 1 \leqq i \leqq q.$$

Let $0 < d \leqq y_1 + y_2 + \ldots + y_q = Y$ *and:*

$$y_1 + \ldots + y_{r-1} < d \leqq y_1 + y_2 + \ldots + y_r.$$

Then:

a) $r = O(\log \min(Y, d/(Y-d)))$.

b) $y_i = \Omega(\max(Y-d, 1))$ *for all* $i \leqq r$.

Proof: Let $i \leqq r$. Then:

$$y_i \geqq \alpha/2(y_i + y_{i+1} + \ldots + y_q) \geqq \alpha/2(y_r + y_{r+1} + \ldots + y_q) \geqq \alpha/2(Y-d).$$

This proves *b*). Define:

$$Y_{i,j} = y_i + y_{i+1} + \ldots + y_j \quad \text{for} \quad i \leqq j.$$

Then $y_i \geqq \alpha/2 . Y_{i,j}$ and hence:

$$Y_{i+1,j} \leqq (1-\alpha/2)\, Y_{i,j}$$

and further:

$$Y_{1,j} \geqq [1/(1-\alpha/2)]^{j-1}\, Y_{j,j} = [2/(2-\alpha)]^{j-1}\, y_j.$$

For $j = r-1$ we obtain:

$$d \geqq Y_{1,r-1} \geqq [2/(2-\alpha)]^{r-2}\, y_{r-1} \geqq \frac{\alpha}{2}\left[\frac{2}{2-\alpha}\right]^{r-2}(Y-d).$$

This proves *a*) for $d < Y$. If $d = Y$ then we only have to observe that:

$$Y = Y_{1,q} \geqq [2/(2-\alpha)]^{q-1}\, y_q \geqq [2/(2-\alpha)]^{q-1}. \quad \square$$

Let $x_1, x_2, \ldots, x_q$ be the thickness of j-nodes in the order in which they would be deleted if we wanted to delete them all. Then $q_j = x_1 + x_2 + \ldots + x_q$ and:

$$x_i \geqq \alpha(q_j - x_1 - \ldots - x_{i-1})/2 = \alpha/2(x_i + \ldots + x_q).$$

Hence the lemma applies and we have:

$$r = O(\log \min(q_j, d/(q_j - d)))$$

and:

$$\text{depth}(x_i) = O(\log W/x_i) = O(\log \min(W/(q_j - d), W)).$$

This shows that up to now only $O(\log \min(W/(q_j - d), W))$ time units are spent.

At this point we are left with the following problem. We are currently working on a j-node of thickness x_r with $x_1 + \ldots + x_{r-1} < d \leqq x_1 + \ldots + x_r$ and $x_r = \Omega(q_j - d)$, we deleted j-nodes of thickness $x_1, \ldots, x_{r-1}$ and thus created many unbalanced nodes. If $d \leqq w_n$ then $r = 1$ and no j-node was deleted so far. Next we distinguish cases: whether the thickness x_r of the currently considered j-node has to be reduced considerably or not, i.e. whether $x_1 + \ldots + x_r - d$ is small or not.

Case 1: $x_1 + \ldots + x_r - d \leqq (q_j - d)/2$, i.e. the thickness x_r has to be reduced considerably. In this case we also delete the j-node of thickness x_r completely and in a second pass increase the j-th access frequency by $(x_1 + \ldots + x_r) - d$. At this point we deleted some j-nodes to the left of the j-joint and some j-nodes to the right of the j-joint. Consider the situation to the left of the j-joint first. Let u_0 be the father of the j-node of thickness x_1 and let u_m be the father of the j-node of maximal depth which was deleted to the left of the j-joint. Let $u_0, u_1, \ldots, u_m$ be the path from u_0 to u_m. Then the deleted j-nodes were right sons of some of the u_i's. In particular, the j-node of thickness x_1 was the right son of u_0. Deleting $u_0, \ldots, u_m$ leaves us with an ordered forest consisting of the left subtrees of those u_i which are not father of a deleted j-node plus the left subtree of u_m. We want to concatenate these subtrees as described in section 3 on concatenable queues. The situation here corresponds exactly to the SPLIT operation. Let $t_1, \ldots, t_q$ be the thickness of the trees in the ordered forest. Then $t_1 \leqq t_1 + \ldots + t_q \leqq ((1-\alpha)/\alpha)\, x_1$ since the thickness of the left (right) subtree of u_0 is less then $t_1 + \ldots + t_q$ (equal to x_1).

Furthermore $t_q \geqq (\alpha/(1-\alpha))\, x_r$ since t_q is the thickness of the left subtree of u_m and the thickness of the right subtree of u_m is at least x_r. The analysis of the SPLIT operation [remark immediately preceeding the statement of theorem 1. Note that $q \leqq \text{depth}(x_r)$ and $|S_1| = t_1$, $|S_q| = t_q$] shows that:

$$O\left(\text{depth}(x_r) + \log \frac{t_1}{t_q}\right) = O(\log \min(W,\ W/(q_j - d)))$$

time units suffice to concatenate this ordered forest. [Note that t_q is the brother of x_r and hence $t_1/t_q \leqq W/t_q = O(W/x_r)$. Furthermore $\text{depth}(x_r) = O(\log W/x_r)$. Finally observe that $x_r = \Omega(q_j - d)$ and that $x_r \geqq 1$.]

An analogous statement holds for the right side of the j-joint. Let us summarize what we achieved so far. We reorganized the tree below the fathers of the j-nodes of minimal depth on either side of the j-joint. Next we need to organize above these nodes. We concentrate on the left side first. Let v_{n-1} be the father of the j-node of thickness x_1. By the reorganization described so far the subtree rooted at v_{n-1} was replaced by a subtree of smaller thickness [at least thickness $t_q = \Omega(q_j - d)$]. This reduction in thickness unbalances $v_{n-2}, v_{n-3}, \ldots$ However v_j will remain balancable for $j \leqq n - p$. We will show that p can be bounded by a constant.

LEMMA: *Let* $w_0, w_1, \ldots, w_n$ *be an* α*-admissible sequence. Then* $(w_i - w_n, w_{i+1} - w_n)$ *is balancable for all:*

$$j \leqq n - \lceil \log(\alpha^2/(1-\alpha+\alpha^2))/\log(1-\alpha) \rceil .$$

Proof: From $w_{n-k} \geqq w_n/(1-\alpha)^k$ we infer:

$$\frac{w_{n-k+1}-w_n}{w_{n-k}-w_n} = \frac{w_{n-k+1}/w_{n-k}-w_n/w_{n-k}}{1-w_n/w_{n-k}} \geqq \frac{\alpha-(1-\alpha)^k}{1-(1-\alpha)^k} \geqq \alpha(1-\alpha),$$

if:

$$\alpha^2 \geqq (1-\alpha+\alpha^2)(1-\alpha)^k,$$

if:

$$k \geqq \frac{\log(\alpha^2/(1-\alpha+\alpha^2))}{\log(1-\alpha)}.$$

This proves the lemma. □

The lemma shows that p can be bounded by a constant even if we replace v_{n-1} by a node of thickness 0. (Use $n-1$ instead of n in the lemma.) Since we replace v_{n-1} by a tree of non-zero thickness this is even more true.

Hence we only need to consider the ordered forest of subtrees along the path from v_{n-p} to v_{n-1}. These subtrees have thicknesses $w_{n-p}-w_{n-p+1}, \ldots, w_{n-2}-w_{n-1}$, new thickness of v_{n-1}. We merge this subtrees by means of the spine lemma, say by choosing the thickest one and then merging the other ones into its left and right spine. All except one of these merges can be performed in constant time. The single exception is the merge with the new subtree with root v_{n-1}. However, this subtree has thickness $\Omega(q_j-d)$, and the other trees certainly have thickness $\leqq W$. Hence the time bound $O(\log \min(W/(q_j-d), W))$ also holds. The nodes above v_{n-p} and below the j-joint are balancable. Here rotations and double rotations suffice to rebalance them. An analogous statement holds for the right side of the j-joint.

Finally we need to balance the j-joint and the nodes above it. By arguments quite similar to the ones above one can show that the same time bound $O(\log \min(W/(q_j-d), W))$ again holds.

Altogether we have shown that the tree can be rebalanced in $O(\log \min(W/(q_j-d), W))$ time units after deleting j-nodes of thickness $x_1, \ldots, x_r$. The j-th acess frequency now has the value:

$$q_j-(x_1+x_2+\ldots+x_r)=(q_j-d)-(x_1+x_2+\ldots+x_r-d) \geqq (q_j-d)/2.$$

In a second pass we increase the j-th access frequency by $x_1+\ldots+x_r-d$. By theorem 3 this does not destroy the time bound stated above.

Case 2: $x_1+\ldots+x_r-d>(q_j-d)/2$.

In this case we do not delete the j-node of thickness x_r. Rather we decrease its thickness to $x_r-(d-x_1-x_2-\ldots-x_{r-1})$ and include its remnants into the ordered forests considered above. Note that the remnants have thickness $>(q_j-d)/2$ (this is on the order of the bound we had for x_r and t_q above) and hence the time bounds developed in case 1 are still valid.

We summarize:

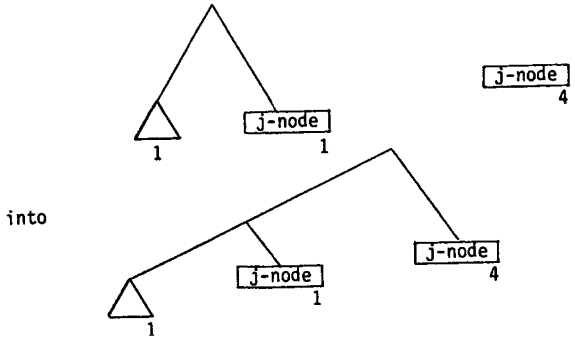
THEOREM 4: *Let T be a D-tree of total thickness $W=q_0+q_1+\ldots+q_n$. Decreasing q_j by d can be done in time $O(\log\min(W, W/(q_j-d)))$.*

So, the time needed to restructure the tree is at most proportional to the new access time.

Example: We continue our example of the previous section. Suppose we want to decrease q_j by 7. This forces us to delete the active j-node of thickness 6. Since:

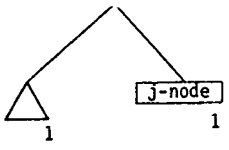
$$(6+5)-7>(14-7)/2$$

we decrease the thickness of the j-node of thickness 5 to 4. Then we reassemble the forest consisting of the two trees:



and replace the tree rooted at u by the tree above. No other changes are required.

If we wanted to decrease q_j by 8 then case 1 would apply. In this case the subtree rooted at u would be replaced by the tree:



No other changes are required. In a second pass we would increase q_j by 3.

REFERENCES

AHO, HOPCROFT and ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

B. ALLAN and I. MUNRO, *Self-Organizing Binary Search Tress*, J. Assoc. Comput. Mach., Vol. 25, 1978, pp. 526-535.

J. L. BAER, *Weight-Balanced Trees*, Proc. A.F.I.P.S., Vol. 44, 1975, pp. 467-472.

N. BLUM and K. MEHLHORN, *On the Average Number of Balancing Operations in Weight-Balanced Trees*, Theoretical Computer Science II, 1980, pp. 303-320.

K. MEHLHORN, [79], *Dynamic Binary Search*, S.I.A.M. J. Comput., Vol. 8, No. 2, 1979, pp. 175-198.

K. Mehlhorn, *Effiziente Algorithmen*, Teubner Verlag, Studienbücher Informatik, 1977.

J. NIEVERGELT and E. M. REINGOLD, *Binary Search Trees of Bounded Balance*, S.I.A.M. J. Comput., Vol. 2, No. 1, March 1973, pp. 33-43.

K. UNTERAUER, *Optimierung gewichtetet Binärbäume zur Organisation geordneter dynamischer Dateien*, Doktorarbeit, TU München, 1977.

K. UNTERAUER, *Dynamic weighted Binary Search Trees*, Acta Informatica, 1979, pp. 341-362.