Dense Multiway Trees

K. CULIK, II
University of Waterloo, Canada
Th. OTTMANN
Universität Karlsruhe, West Germany
and
D. WOOD
McMaster University, Canada

B-trees of order m are a "balanced" class of m-ary trees, which have applications in the areas of file organization. In fact, they have been the only choice when balanced multiway trees are required. Although they have very simple insertion and deletion algorithms, their storage utilization, that is, the number of keys per page or node, is at worst 50 percent. In the present paper we investigate a new class of balanced m-ary trees, the dense multiway trees, and compare their storage utilization with that of B-trees of order m.

Surprisingly, we are able to demonstrate that weakly dense multiway trees have an $O(\log_2 N)$ insertion algorithm. We also show that inserting $m^h - 1$ keys in ascending order into an initially empty dense multiway tree yields the complete m-ary tree of height h, and that at intermediate steps in the insertion sequence the intermediate trees can also be considered to be as dense as possible. Furthermore, an analysis of the limiting dynamic behavior of the dense m-ary trees under insertion shows that the average storage utilization tends to 1; that is, the trees become as dense as possible. This motivates the use of the term "dense."

Key Words and Phrases: multiway trees, balanced trees, B-trees, search trees, dense trees, storage utilization
CR Categories: 4.33, 4.34, 5.32

1. INTRODUCTION

B-trees [1] are a well-known data structure for organizing dynamic files using backup store. They can be considered as multiway trees of order $m \geq 3$ where each internal node has k sons and $k - 1$ keys for $\lceil m/2 \rceil \leq k \leq m$. Each node

This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grants A-7403 and A-7700 and The Deutsche Forschungsgemeinschaft (DFG).
Some of the results in this paper were presented at the 8th Symposium on Mathematical Foundations of Computer Science, Olomouč, ČSSR, September 1979.
Authors' addresses: K. Culik, II, Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada; Th. Ottmann, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Universität Karlsruhe, 7500 Karlsruhe, West Germany; D. Wood, Unit for Computer Science, McMaster University, Hamilton, Ontario L8S 4K1, Canada.

corresponds to a page of fixed size which can hold up to $m - 1$ keys (and records) and m pointers. But pages may be only partially filled. Information is transferred between main memory and backup store in units of pages. Insertion of new keys and deletion of unwanted keys is quite simple for B-trees. The insertion of a new key ensues by the recursive strategy of splitting an "overfull" node (or page) having m keys into two and then moving one of its keys upward. It can easily be seen that in the worst case each page is at least half filled. The average storage utilization is much better. Yao [15] has shown that after a sequence of random insertions into an initially empty B-tree of high order, the storage utilization is approximately $\ln 2 = 69$ percent. However, if the keys are inserted in ascending order into an initially empty B-tree, the worst case lower bound of 50 percent storage utilization becomes significant. An attempt to solve this "sparsity problem" for B-trees is the main idea of "overflow" introduced by Bayer and McCreight [1], which leads to the notion of B*-trees (see Knuth [2, p. 477 ff]): Instead of splitting an overfull node, look first at its left (or right) brother. Say the immediate right brother has only j keys and $j + 1$ sons where $j < m - 1$. Then the overfull node "flows over" into its brother node. If the brother node is already full, then split both nodes and create three new nodes, each about two-thirds full. The result of this modification is an increase in storage utilization: At least two-thirds of the available space is utilized.

We might even look at both immediate left and right brothers before splitting nodes. However, this also does not yield trees whose nodes are maximally filled when keys are inserted in ascending order.

In this paper we introduce (in Section 3) an insertion strategy which looks at all brothers of a node before splitting it. We demonstrate that looking at brothers allows us to drop the requirement that nodes must be at least binary in order to prevent degenerate trees. The insertion procedure immediately gives rise to a dynamic definition of a class of dense m-ary trees. A tree is dense if it is either the empty tree (of height 1 with no key) or is obtained by one further insertion into a dense tree. The "density" of these dynamically defined trees is underlined by two observations. First, we prove (in Section 4) that iterative insertions of keys in ascending order into the initially empty tree produces complete m-ary trees. Second, we consider trees which are made by iterative insertions of randomly chosen sequences of keys. Because a precise analysis of the average structure of these latter trees raises serious difficulties, we make various assumptions which are both plausible and simplify the analysis. Thus we derive an estimation for the average storage utilization of "random" m-ary trees which is much better than for B-trees of order m. As a preliminary step we give (in Section 2) a static definition of classes of "r-dense m-ary trees" (where $r = 1, \ldots, m - 1$) which extends to multiway trees the idea of brother trees [8, 10], which are binary.

It is easy to see that every dynamically defined dense m-ary tree is at least 1-dense, but not conversely. However, the precise relationship of the dynamic class with the various static classes is still an open problem. Our insertion procedure is designed in such a uniform way that it becomes applicable to arbitrary r-dense m-ary trees. Though we look at all brothers before splitting nodes, it turns out that in the "weakly dense" case, that is, in the 1-dense case, the amount of work to be done is proportional to $\log N$, where N is the number of stored keys.

In Section 5 we discuss the kinds of trees that are obtained by a sequence of N random insertions into the initially empty tree. It turns out that Yao's [15] analysis cannot be directly transferred to compute the average storage utilization of random dense m-ary trees. However, Yao's method is applicable to a related insertion technique, which we claim gives a good estimate for the original insertion algorithm.

Finally, in Section 6 we present some concluding remarks and observations. Furthermore, we raise a number of open problems.

2. BASIC DEFINITIONS AND STATIC STRUCTURE OF DENSE MULTIWAY TREES

In this section we define classes of multiway, in fact, m-ary trees, where $m \geq 2$ is a fixed natural number. We assume some familiarity with notions concerning trees. A node p with only one son σp is called *unary*; a node with two sons is called *binary*; a node which has the maximal allowable number m of sons or which is a leaf is said to be *saturated*, otherwise it is said to be *unsaturated*. The *depth* of a node p is its distance from the root, i.e., the number of edges in the path from the root to p. The *height* of p is the number of edges in the longest path from p to a leaf. The height of a tree T is the height of its root.

An m-ary tree T is said to be *r-dense*, where r is a natural number with $1 \leq r \leq m - 1$ iff (2.1) ... (2.3) hold:

(2.1) The root of T is at least binary.
(2.2) Each unsaturated node different from the root has either only saturated brothers and at least one such brother or at least r saturated brothers.
(2.3) All leaves have the same depth.

We shall make frequent use of the following immediate consequences of the definition of r-dense m-ary trees:

If a node p is the only son of its father φp, then p must be saturated.

If a node p has k sons, where $2 \leq k \leq m$, then there is at most one unsaturated node among the sons of p if $k \leq r$, but there are at least r saturated sons of p if $k \geq r + 1$. A class of m-ary trees is called *dense* if it is a class of r-dense m-ary trees for some r. In particular, we speak of *weakly dense m-ary trees* and *strongly dense m-ary* trees, respectively, if we have in mind the classes of 1-dense and $(m - 1)$-dense m-ary trees, respectively.

Observe that there is only one class of dense binary trees. This class coincides with the class of brother trees [8, 10].

There are two classes of dense ternary trees. Figure 1 shows an example of a strongly dense ternary tree and an example of a weakly dense ternary tree, each with 11 leaves.

When disregarding the root, the 2-3 brother trees of Kriegel, Vaishnavi, and Wood [3] may be considered as special kinds of weakly dense ternary trees. (See [7] for a further explanation of the relationship between 2-3 brother trees and dense ternary trees.) The class of B-trees of order m is not contained in any class of dense m-ary trees, and vice versa.

It is clear that for the given height h the complete m-ary tree (where each internal node has m sons) has the maximal number of leaves of all r-dense m-ary trees of height h.

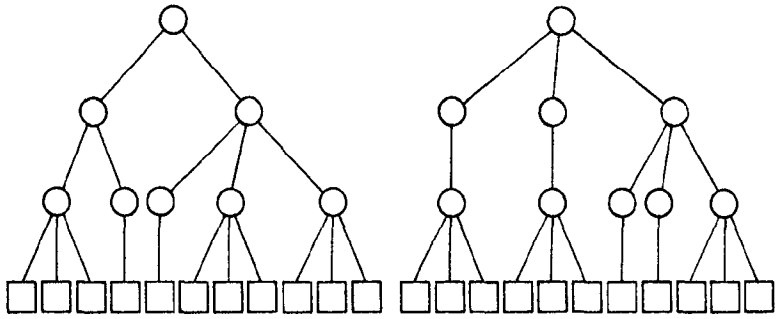


Figure 1

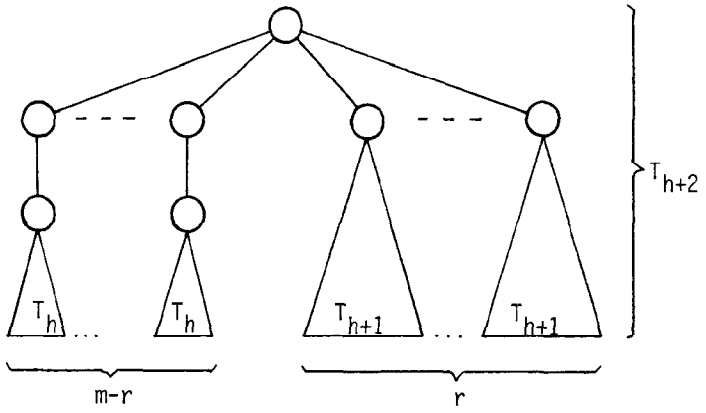


Figure 2

As far as r-dense m-ary trees with the minimal number of leaves are concerned, let us denote by T_h a tree of this kind with height h which has a saturated root. Then it is clear that T_{h+2} is obtained (up to symmetry) as shown in Figure 2 from $(m - r)$ copies of T_h and r copies of T_{h+1}.

Further, $T_0 = \square$ (the tree of height 0 with only one node), and T_1 is as shown in Figure 3. This gives the following recurrence relation for the number $l_{\min}(r, m, h)$ of leaves of the r-dense m-ary tree T_h:

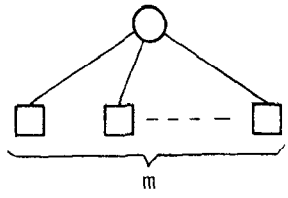
$$\text{(2.4)} \qquad \begin{cases} l_{\min}(r, m, 0) = 1, \quad l_{\min}(r, m, 1) = m, \\ l_{\min}(r, m, h + 2) = (m - r)l_{\min}(r, m, h) + rl_{\min}(r, m, h + 1). \end{cases}$$

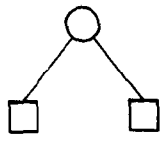
Since the root of an r-dense m-ary tree must be at least binary, the r-dense m-ary tree with the minimal possible number of leaves and with height h is (up to symmetry) of the form shown in Figure 4.

Therefore, an r-dense m-ary tree of height $h \geq 1$ has at least $L_{\min}(r, m, h)$ leaves with

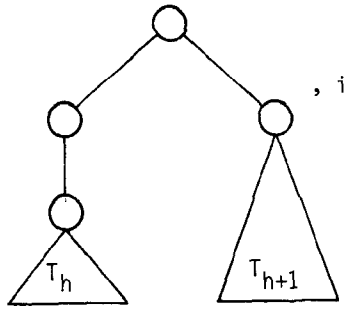
$$L_{\min}(r, m, h) = \begin{cases} 2, & \text{if } h = 1, \\ l_{\min}(r, m, h - 1) + l_{\min}(r, m, h - 2), & \text{if } h > 1. \end{cases}$$

Figure 3





, if h=1, and



, if h > 1.

Figure 4

In order to solve the recurrence (2.4), first observe that

$$l_{\min}(r, m, h) = a_{h+1},$$

with

$$\text{(2.5)} \qquad \begin{cases} a_0 = 1, \quad a_1 = 1, \\ a_n = (m - r) \cdot a_{n-2} + r \cdot a_{n-1}, \quad \text{if} \quad n \geq 2. \end{cases}$$

By using well-known methods (cf., for example, Liu [5]), we obtain as the solution of the recurrence (2.5)

$$a_n = A_1 \cdot \alpha_1^n + A_2 \cdot \alpha_2^n,$$

where

$$\alpha_1 = \frac{1}{2}(r + \sqrt{4(m - r) + r^2}),$$

$$\alpha_2 = \frac{1}{2}(r - \sqrt{4(m - r) + r^2}),$$

$$A_1 = \frac{1 - \alpha_2}{\alpha_1 - \alpha_2}, \qquad A_2 = \frac{\alpha_1 - 1}{\alpha_1 - \alpha_2}.$$

Table I shows some values of α_i and A_i for several values of m and r.
For all $h \geq 1$ we have

$$L_{\min}(r, m, h) = a_h + a_{h-1}$$
$$= \left(A_1 + \frac{A_1}{\alpha_1}\right)\alpha_1^h + \left(A_2 + \frac{A_2}{\alpha_2}\right)\alpha_2^h.$$

This gives in the special case of binary trees ($m = 2$, $r = 1$) the result [8] that the minimal number of leaves of brother trees of height h is about $1.171(1.618)^h$.

Table I

m	r	α_1	α_2	A_1	A_2
2	1	$\frac{1+\sqrt{5}}{2}$	$\frac{1-\sqrt{5}}{2}$	$\frac{1}{\sqrt{5}}\frac{1+\sqrt{5}}{2}$	$-\frac{1}{\sqrt{5}}\frac{1-\sqrt{5}}{2}$
3	1	2	−1	2/3	1/3
	2	$1+\sqrt{2}$	$1-\sqrt{2}$	1/2	1/2
111	1	11	−10	11/21	10/21
	55	56	−1	2/57	55/57
	110	110.009	−0.009	0.009	0.989

Further, a weakly dense ternary tree of height h has at least 2^h leaves, and the minimal number of leaves of a strongly dense ternary tree is

$$L_{\min}(2, 3, h) = \frac{\sqrt{2}}{2(1+\sqrt{2})^h} - \frac{\sqrt{2}}{2(1-\sqrt{2})^h},$$

which is (up to a constant factor) the same as for 2-3 brother trees (cf. [3]).

Finally, for the case $m = 111$ we obtain, for example,

$$L_{\min}(1,111, h) = \frac{1}{21} \times (12 \times 11^h + 9 \times (-10)^h),$$

$$L_{\min}(110,111, h) \approx 0.009 \times 110.009^h.$$

Observe that the number of leaves of a strongly dense m-ary tree of height h is proportional to $(m-1)^h$. For in this case $r = m - 1$, hence $|\alpha_2| < 1$ and α_1 is approximately $(m-1)$ (for large m).

The obvious way of storing keys in a dense m-ary tree is analogous to the method of storing keys in 1-2 brother trees [9], 2, 3-trees [15], and B-trees [1]:

(2.6) Each node with $i+1$ sons holds i keys.

(2.7) The i keys $k_1, \ldots, k_i$ of a node p with $i+1$ sons $\sigma_1 p, \ldots, \sigma_{i+1} p$ are ordered such that for all j, $1 \leq j \leq i$, the following holds: The keys in the subtree with root $\sigma_j p$ are less than k_j, which in turn is less than the keys in the subtree with root $\sigma_{j+1} p$.

Condition (2.6) in particular says that unary nodes and leaves of dense m-ary trees contain no keys. Condition (2.7) enables us to retrieve a key x by an algorithm which is a slight modification of the retrieval algorithm for B-trees (cf. [1]), taking care of the fact that unary nodes with no keys may occur.

Every dense m-ary tree with N keys has $N+1$ leaves. One can easily show by induction on h using the recurrence formula (2.5) that for every r-dense m-ary tree of height h

$$L_{\min}(r, m, h) \geq a_h \geq m^{\lfloor h/2 \rfloor}.$$

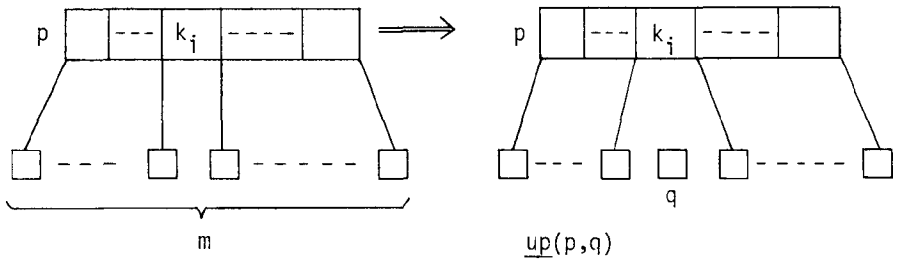


Figure 5

Hence, every r-dense m-ary tree with N keys and $N + 1$ leaves is of height $h \leq 2 \cdot \log_m (N + 1) + 1$. This shows that r-dense m-ary trees cannot degenerate into linear lists. (But the above estimation is very rough.)

It should be kept in mind that the above analysis of r-dense m-ary trees is completely static. We have obtained an overview of the worst possible trees compatible with the purely static conditions (2.1), ..., (2.3). This gives us no more than a rough idea of what happens if we carry out iterative insertions using the insertion procedure explained in the next section. Thus, it is for example easy to see that the weakly dense m-ary trees of height h with $L_{\min}(1, m, h)$ leaves cannot be generated by iterative insertions into the initially empty tree for any $h \geqq 2$.

3. SINGLE KEY INSERTION

In order to insert a new key x into a dense m-ary tree T we first search for the key x in T. When the search is unsuccessful, we end up at a leaf. Let p be the father of this leaf.

Case 1 [p is unsaturated]. Then insert x as an additional key of node p at the expected position, create a new leaf, make it an additional son of p, and FINISH.

Case 2 [p is saturated]. In this case p already has $m - 1$ keys $k_1 < \cdots < k_{m-1}$. We may assume without loss of generality that $k_{i-1} < x < k_i$ holds for some i, $1 \leq i \leq m - 1$, if $m \geq 2$, where $k_0 = -\infty$. (We can replace some key of p by x, if necessary.) Create a new leaf q in between the ith and $(i + 1)$st son of p and call $up(p, q)$ (see Figure 5).

The upward restructuring procedure *up* will be designed in a uniform way for all r-dense m-ary trees, where r is arbitrary in the allowed interval. *Up* may eventually call the subroutines *right-shift* or *left-shift* which we explain first:

right-shift(p, q). On entry q is a right brother of p or $p = q$; p is at least binary.

Case 1 [$p \neq q$]. Let p be the ith son of its father φp and p' the $(i + 1)$st son of φp. Let r be the rightmost son of p. Make r the leftmost son of p' and call right-shift (p', q).

Case 2 [$p = q$]. FINISH.

left-shift is defined similarly.

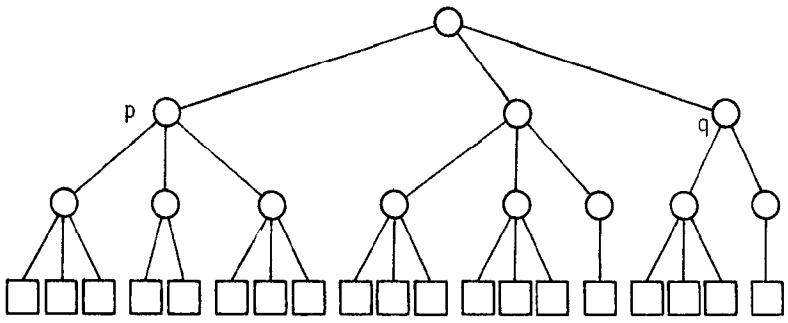


Figure 6

Here, and also in all the procedures designated below, we pass over the question of how to transfer keys locally from one node to the other in order to maintain condition (2.7). The method is similar to the "overflow" technique described in Knuth [2, p. 478].

However, we mention the crucial point in the whole insertion procedure for dense m-ary trees: The shifting of nodes may destroy the property of being an r-dense tree. Consider, for example, the strongly dense ternary tree shown in Figure 6.

The result of performing *right-shift*(p, q) is no longer a strongly dense ternary tree, since q obtains two unary sons. Therefore additional work has to be done in order to keep the tree in strongly dense form. We will return to this problem later and assume for the moment that *right-shift* and *left-shift* always yield r-dense trees when applied to those trees.

Let us call a node a $\otimes$-*node* if it is either the saturated root of an r-dense m-ary tree or a leaf.

up is a recursive procedure with the following invariant condition: Whenever *up*(p, q) is called, then

(1) p and all of p's sons are $\otimes$-nodes,
(2) q is either a leaf or has a single son σq which is a $\otimes$-node,
(3) q lies to the right of the leftmost son of p and to the left of the rightmost son of p.

up(p, q):

Case 1 [p has an unsaturated brother]. Let βp be an unsaturated brother of p which is nearest to p. (That means that all brothers occurring in between p and βp are saturated.) Make q an additional son of p and call *right-shift*$(p, \beta p)$, if p is a right brother of p, and call *left-shift* otherwise; FINISH. (See Figure 7.)

Case 2 [p has only saturated brothers].

Case 2.1 [p has an unsaturated father]. Make q an additional son of p; remove the leftmost son l of p and make it the only son of a newly created node u; make u an additional son of the father φp of p and FINISH. (See Figure 8.)

Case 2.2 [p has no father, i.e., p is the root]. Then create a new root φp with p as its only son and proceed as in Case 2.1.

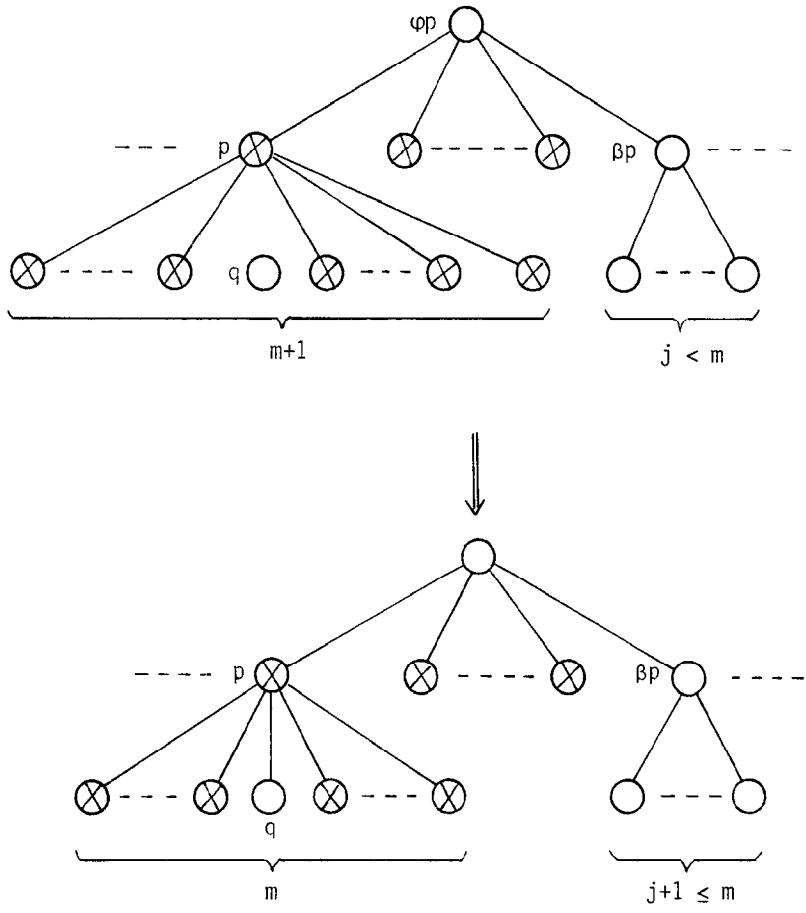


Figure 7

Case 2.3 [the father φp of p is saturated]. Assume first that p is not the rightmost son of its father φp. Create a new immediate right brother q' of p which obtains as its only son the rightmost son of p; make q an additional son of p; call $up(\varphi p, q')$.

If p is the rightmost son of φp, proceed analogously by creating a new immediate left brother of p. (See Figure 9.)

Observe that the invariant condition is maintained.

end of up

It is clear that in the worst case h (recursive) calls of *up* can occur where h is the height of the tree before insertion. Thus the performance time of *up* is dominated by h and the time to carry out *right-shift* (*left-shift*, respectively). If the shift procedures are of the above specified form where no further restructuring is required, then their performance time is constant. Simple shifting without further restructuring is sufficient for the case of binary and weakly dense ternary trees. For in these cases any call of *left-shift* or *right-shift* again yields a dense

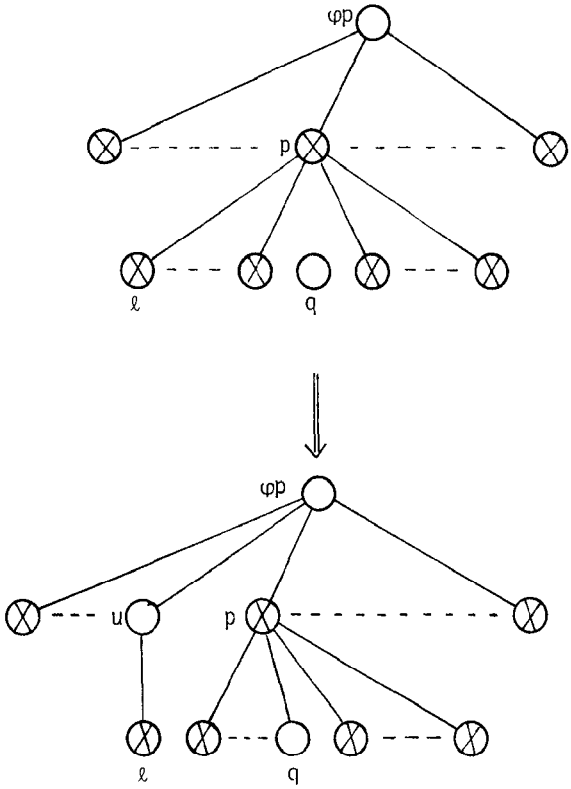


Figure 8

binary and a weakly dense ternary tree, respectively. Thus for these two classes total performance time of *up* is $O(h) = O(\log N)$, where N is the number of leaves (or stored keys). We will show that the same holds for any class of weakly dense m-ary trees.

We first observe that any call of *right-shift* (and *left-shift*) in the main procedure *up* concerns nodes p and q where p is "overfilled" (i.e., has $m + 1$ sons), q is unsaturated, and all brother nodes $p_1, \ldots, p_j$ occurring in between p and q are saturated. Further, all sons of p, except for one (in particular the leftmost and rightmost son of p), must be saturated. Finally, because we are dealing with weakly dense trees, all nodes $p_1, \ldots, p_j$ and q must have at least one saturated son.

Let us assume that q is a right brother of p. Then the situation can be depicted as shown in Figure 10.

The result of performing *right-shift*(p, q) is shown in Figure 11.

Clearly, p, p_1, and q are roots of weakly dense m-ary trees. But some of $p_2, \ldots, p_j$ may no longer be roots of weakly dense subtrees because they may have lost their only saturated son by the shifting. Hence we are faced with the following *transformation problem*:

> Given an m-ary tree with a saturated root p which is weakly dense except for the root (that is, all sons $\sigma_1 p, \ldots, \sigma_m p$ of the root are unsaturated but the

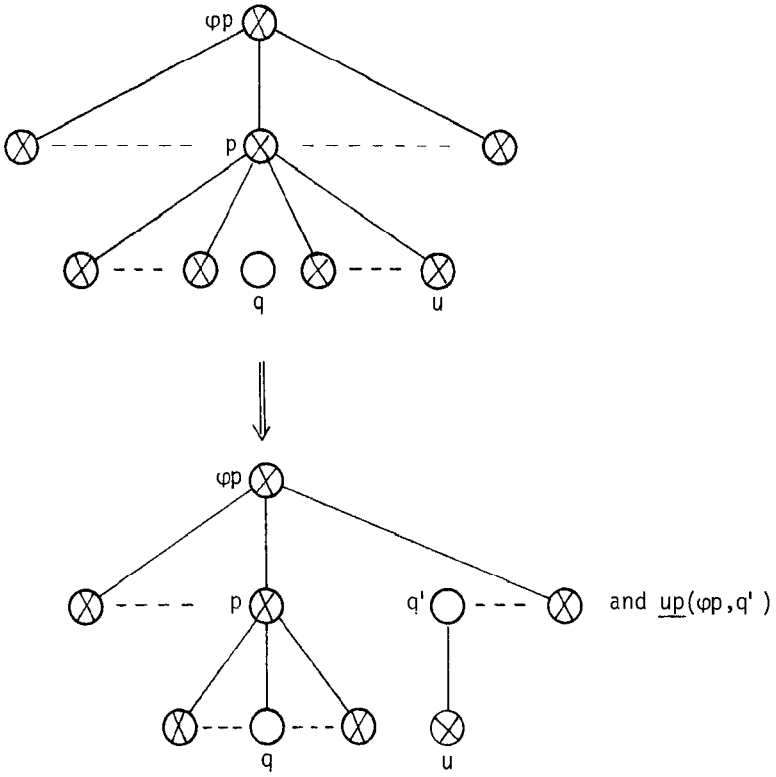


Figure 9

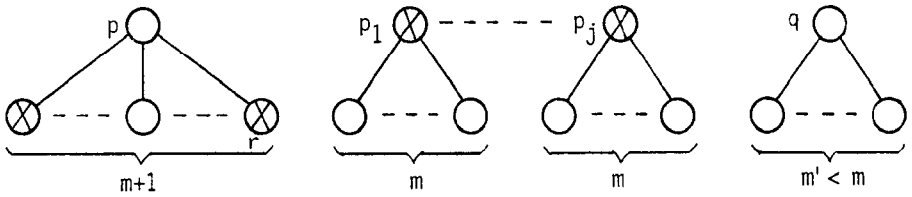


Figure 10

tree is weakly dense everywhere else), then transform the tree as shown in Figure 12, such that the whole tree becomes a weakly dense m-ary tree with the same number of leaves.

To solve this problem, we consider the sequence of grandsons of p. This sequence contains at least m ⊗-nodes and has at most $m(m-1)$ elements. We extend the sequence by attaching two "improper" ⊗-nodes at its ends, one at the left end and the other one at the right end.

Given the sequence of grandsons of p together with the two improper ⊗-nodes, consider a subsequence of length m having the minimal nonzero number of (proper) ⊗-nodes in it. Let i be this number, and now extend the subsequence on both sides up to but excluding the next ⊗-nodes. We call this extended subsequence an *i-interval.*

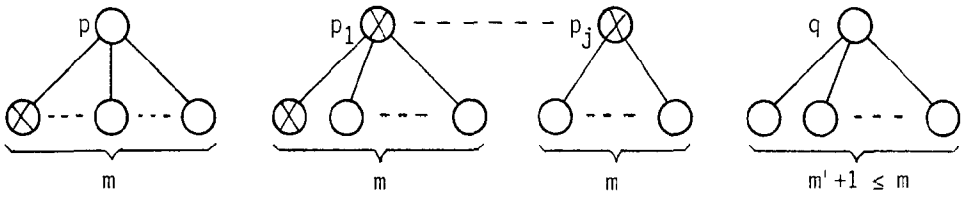
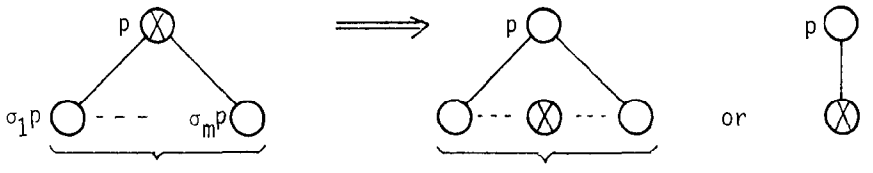


Figure 11



m unsaturated nodes

$m' < m$ nodes at least one of which is saturated

Figure 12

The *length of an i-interval* is the number of its nodes. An example is given in Figure 13.

We now solve the transformation problem described above by ensuring that one of the new sons of p is not only saturated but also takes its sons from the chosen i-interval. We then prove that this can be done in such a way that the remaining grandsons can always be allocated to possibly new sons of p such that the resulting tree is weakly dense.

Case 1: $i = 1$. Let the jth son of p, $\sigma_j p$, be the father of the only ⊗-node in the 1-interval. Since $\sigma_j p$ is unsaturated by assumption, the 1-interval must also contain some other nodes. A possible configuration is depicted in Figure 14.

Of course, it might be that $\sigma_j p$ has either no left or no right brother; further, either the ⊗-node q_l' at the left end of the 1-interval or the ⊗-node q_r' at the right end of the 1-interval may be also a son of $\sigma_j p$.

But in any case we can remove any number of sons of $\sigma_{j-1} p$ and/or $\sigma_{j+1} p$ occurring in the 1-interval and make them additional sons of $\sigma_j p$ such that $\sigma_j p$ becomes saturated. Observe that $\sigma_{j-1} p$ and $\sigma_{j+1} p$ keep at least one ⊗-node while p has now got a saturated son. That is, p becomes the root of a weakly dense m-ary tree.

Case 2: $i \geq 2$. Let $\sigma_j p$ be the father of the leftmost ⊗-node q_l and $\sigma_k p$ be the father of the rightmost ⊗-node q_r in the i-interval. (Clearly, $j = k$ is possible.)

Case 2.1. The total number of sons of $\sigma_j p$, $\sigma_{j+1} p$, . . . , $\sigma_k p$ is less than m.

It follows as in Case 1 that the i-interval must contain some other nodes as well. Thus, we can combine all sons of $\sigma_j p$, . . . , $\sigma_k p$ with some (immediately adjacent) nodes from the i-interval until we have exactly m nodes, make them all sons of the same father, say, q, and make q a son of p instead of $\sigma_j p$, . . . , $\sigma_k p$.

As in Case 1, it follows that p becomes the root of a weakly dense m-ary tree.

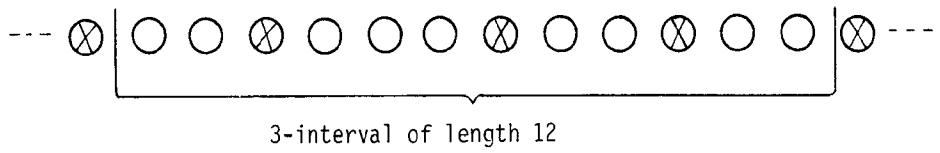


Figure 13

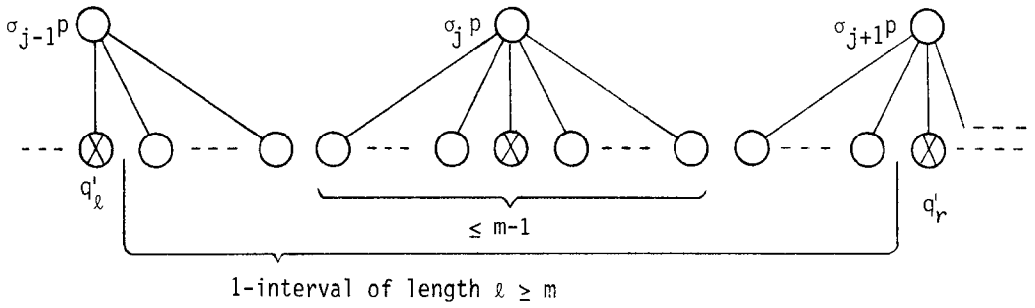


Figure 14

Case 2.2. The total number of sons of $\sigma_j p, \sigma_{j+1} p, \ldots, \sigma_k p$ is $\geq m$.

In this case we have to remove some of the sons of $\sigma_j p$ and $\sigma_k p$ in order to obtain exactly m nodes.

Let there be a nodes in the i-interval to the left of q_l, b nodes between q_l and q_r, and c nodes to the right of q_r. Let q'_l and q'_r denote the (perhaps improper) $\otimes$-nodes at the left and right end of the considered i-interval, respectively, as shown in Figure 15.

First observe that $j \neq k$ holds since $\sigma_j p, \ldots, \sigma_k p$ are all unsaturated, and thus we need at least two nodes to generate (a sequence of) $\geq m$ adjacent sons.

Second, in order to obtain exactly m adjacent nodes by removing some of the sons of $\sigma_j p$ and/or $\sigma_k p$, it suffices either to remove some of the at most c sons of $\sigma_k p$ which lie to the right of q_r in the i-interval or to remove some of the at most a sons of $\sigma_j p$ which lie in the i-interval to the left f q_l. For the sequence of nodes between q'_l and q_r is an $(i-1)$ interval and therefore has a length of at most $m-1$. (This means $a + 1 + b \leq m - 1$.) Similarly, the sequence of nodes between q_l and q'_r is an $(i-1)$ interval, which gives $b + 1 + c \leq m - 1$.

Let us now distinguish two subcases:

Case 2.2.1. At least one of q'_l or q'_r, say, q'_l, is a proper $\otimes$-node; this means that q'_l does not constitute the left end of the sequence of grandsons of p.

Then there must be a (proper or improper) next $\otimes$-node q''_l to the left of q'_l, as shown in Figure 16. The nodes between q''_l and q_l constitute a 1-interval, which has a length of at most $\leq m - 1$ (because $i \geq 2$, by assumption). Now remove as many sons of $\sigma_j p$ to the left of q_l as are necessary to get exactly m sons of $\sigma_j p, \ldots, \sigma_k p$ all together. Make these m nodes all sons of the same node which becomes the son of p instead of $\sigma_j p, \ldots, \sigma_k p$.

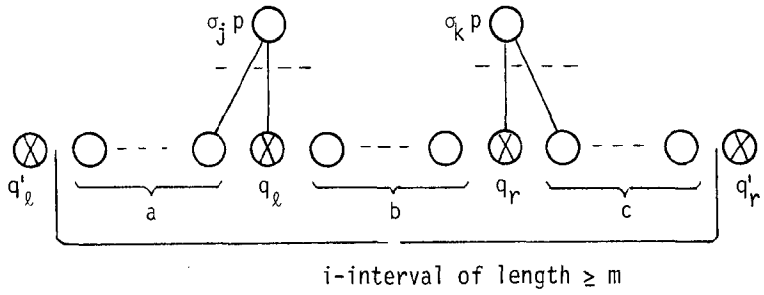


Figure 15

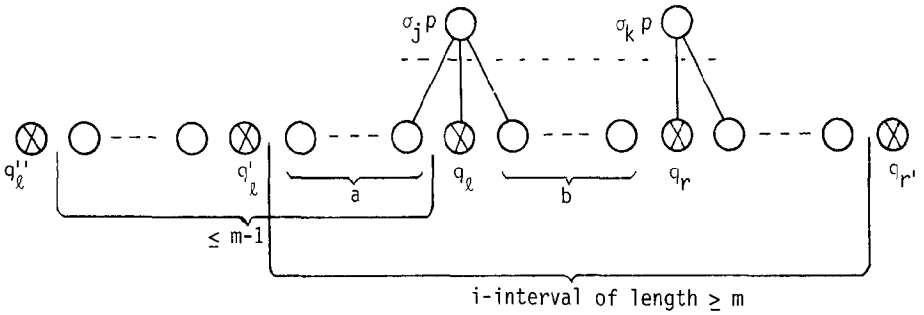


Figure 16

The removed surplus sons of $\sigma_j p$ and all nodes to the left of them but to the right of q_l'' (including q_l') are made sons of one further new node which also becomes a son of p. Thus, we have ultimately replaced at least two nodes (namely, $\sigma_j p, \ldots, \sigma_k p$) by two new ones, one of which is saturated and both of which have at least one saturated son.

Case 2.2.2. q_l' and q_r' are both improper ⊗-nodes, i.e., they constitute the ends of the sequence of grandsons of p.

Then our assumptions imply $b + 2 \geq m$; hence, $i = m$, for the sequence of nodes between q_l and q_r, including q_l and q_r, must contain at least the m saturated grandsons of p. It is now easy to see that the only possibility which is compatible with the assumed minimality of i is $a = b = 0$, $b = m - 2$, i.e., all sons of p have only one son (see Figure 17). Replace this configuration by that in Figure 18.

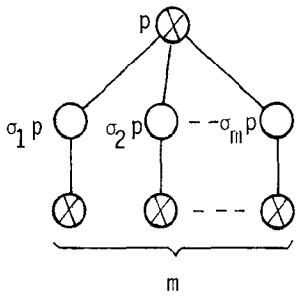
This completes the solution of the above transformation problem.

The whole restructuring obviously can be carried out in time $O(m^3)$, which is a constant for fixed m. Thus we have proved

THEOREM 1. *It is possible to insert a new key into an N-key weakly dense m-ary tree in time at most* $O(m^3 + \log N)$. *Moreover, the insertion procedure maintains the order of keys and retains the weakly dense tree structure.*

We do not know exactly for which other classes of dense m-ary trees insertion can be carried out in time $O(\log N)$ if $m \geq 3$. However, we briefly sketch how trees can be maintained in strongly dense form after insertion of new keys.

Figure 17



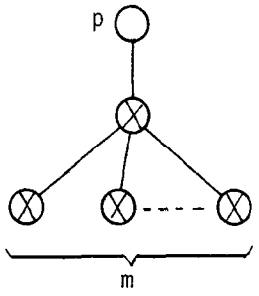


Figure 18

Clearly, it suffices to explain what kind of additional transformation ensures that the shifting procedures called by the recursive procedure *up* do not destroy the strongly dense tree structure. Obviously, the shifting of a node is harmless if the shifted node is a ⊗-node. This can be achieved by calling a (recursive) procedure *rightsat*(p) (*leftsat*(p), respectively) which makes the rightmost (leftmost, respectively) son of p saturated.

rightsat(p). On entry p is at least binary and all sons of p except for at most one are saturated.

Case 1 [The rightmost son of p is saturated]. FINISH.

Case 2 [The rightmost son q of p is unary]. Then the only son σq of q must be saturated. Let βq be the immediate left brother of q (which must be saturated). Call *leftsat*(βq) and make the $(m-1)$ right sons of βp additional sons of q (see Figure 19).

Case 3 [The rightmost son q of p is k-ary where $2 \leq k \leq m-1$]. Let βq denote the (saturated) immediate left brother of q.

for $i := 1$ *step* 1 *until* $(m-k)$ **do**
Call *rightsat*(βq) and make the respective rightmost son of βq the new leftmost son of q (see Figure 20).
end of *rightsat*

leftsat is defined analogously.

If *rightsat*(p) or *leftsat*(p) is called for the root p of a given strongly dense m-ary tree, the resulting tree will be again a strongly dense m-ary tree with the same number of nodes on each level as the given tree.

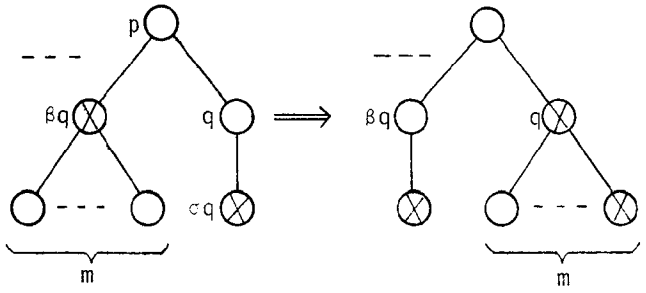


Figure 19

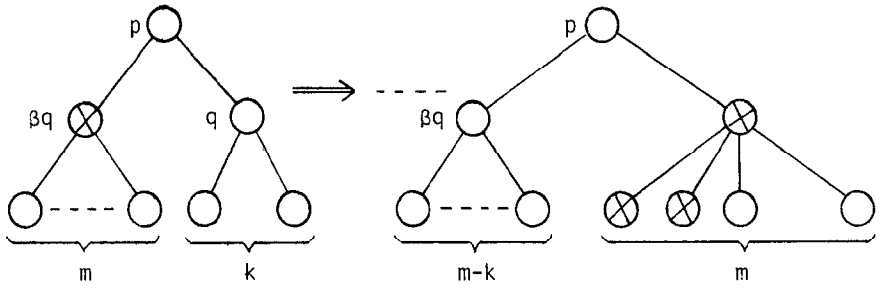
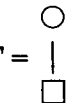


Figure 20

In the worst case a call of *rightsat*(p) may lead to at most ($m - 2$) recursive calls of *rightsat* for nodes on the level below p (or one call of *leftsat*). Hence, *rightsat* can be carried out in time at most $O((\log N)^{m-2})$. This immediately implies that insertion of a new key into a strongly dense ternary tree can be carried out in time $O(\log N)$. In general, however, almost the whole subtree of p has to be restructured.

4. ITERATIVE INSERTION OF ORDERED KEYS

We show that iterative insertion of N keys using the insertion procedure of Section 3 and beginning with the tree



containing no keys yields complete m-ary trees if the keys are inserted in increasing order and $N + 1$ is a power of m. This result holds and will be proved uniformly for every class of dense m-ary trees. Furthermore, the "critical" shifting of nodes to unsaturated brothers which are not adjacent to the given node never happens when inserting keys in ascending order.

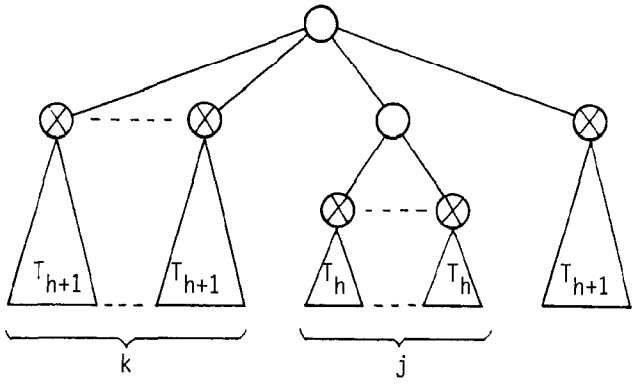


Figure 21

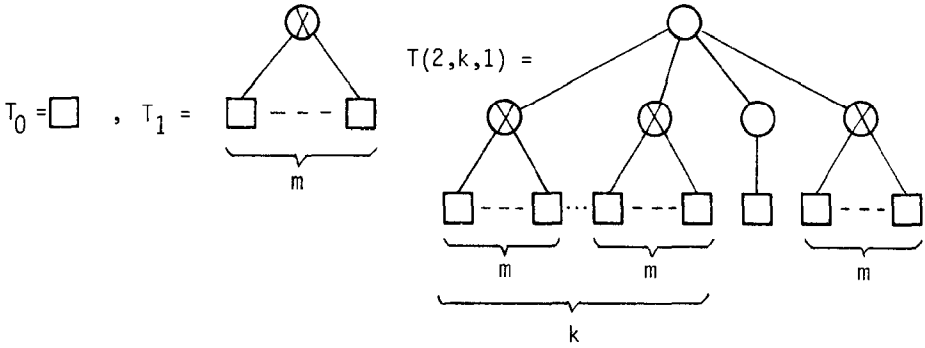


Figure 22

Let us denote by T_h the complete m-ary tree of height h (where each internal node is m-ary) with m^h leaves and $m^h - 1$ keys. Let us further denote by $T(h + 2, k, j)$ the tree shown in Figure 21, where $0 \leq j \leq m$, $k + 2 \leq m$.

LEMMA 1. *Insertion of m^h keys in increasing order into $T(h + 2, k, j)$, where $j \leq m - 1$, yields $T(h + 2, k, j + 1)$.*

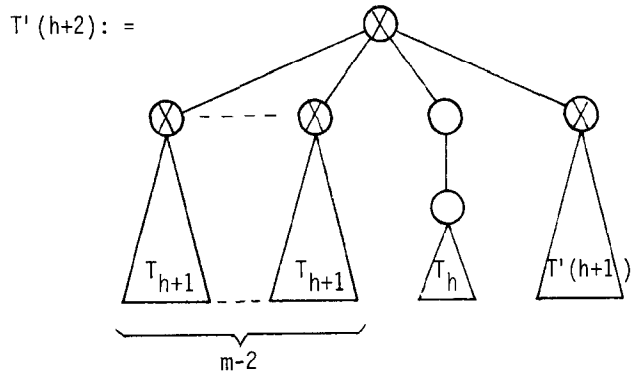
PROOF. We show that the lemma holds for arbitrary pairs (h, j) by induction on the lexicographic ordering of such pairs. First observe that shown in Figure 22.

Insertion of a new maximum into $T(2, k, 0)$ yields $T(2, k, 1)$, since Case 2.1 of the upward restructuring procedure *up* applies. Further insertion of a new maximum into $T(2, k, 1)$ yields $T(2, k, 2)$, since now Case 1 of *up* applies, where the first procedure parameter node has an unsaturated brother.

Assume now (as inductive hypothesis) that the Lemma holds for all pairs (h', j') where either $h' < h$ or else $(h' = h$ and $j' < j)$.

Insertion of a new maximum into T_{h+1} leads to a recursive call of $up(p, q)$, where p is the root of T_{h+1} and q is a unary node which has as its only son the root of a subtree T_h. This follows from the fact that insertion of a new key into a complete m-ary tree leads to a recursive call of *up* on each level, since Case 2.3

Figure 23



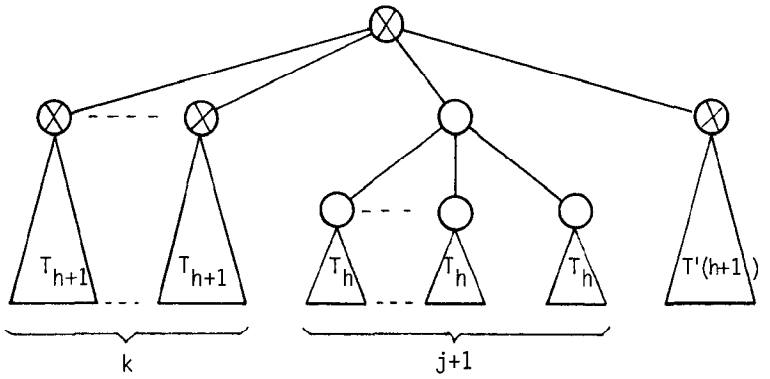


Figure 24

of *up* always applies where the father and all brothers of the first procedure parameter node are saturated. To be more precise, let us denote by $T'(h)$ the tree

$$T'(1) := T_1,$$
$$T'(2) := T(2, m-2, 1),$$

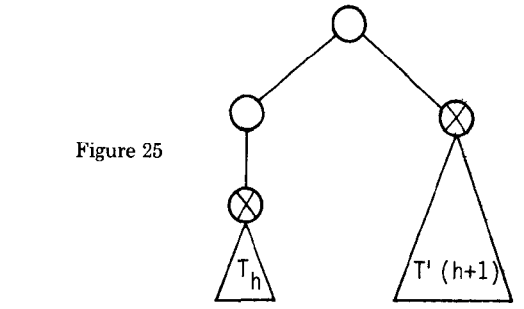
and, for $h \geq 1$, the tree shown in Figure 23. (One can easily show by induction on h that $T'(h)$ has $m^h - m^{h-1} + 1$ leaves.)

Then insertion of *one* new maximum into $T(h+2, k, j)$ yields the tree shown in Figure 24. (Observe that *up* will terminate, since ultimately either Case 2.1 applies, if $j = 0$, or Case 1 applies, if $j \geq 1$. In the latter case, the involved shifting is uncritical, since the only shifted node is saturated (in fact, the root of T_h) and is shifted to its immediate left brother.)

Let us denote by $T''(h+1, l)$, where $1 \leq l \leq h+1$, the tree which results from $T'(h+1)$ by replacing the rightmost subtree of height l (which is just $T'(l)$) by the complete m-ary tree of height l, T_l.

Clearly, $T'(h+1) = T''(h+1, 1)$ and $T''(h+1, h+1) = T_{h+1}$. By using the inductive hypothesis $(m-1)$ times we obtain that inserting new maxima $(m-1) \cdot m^{l-1}$ times into the initial tree $T''(h+1, l)$ yields $T''(h+1, l+1)$.

Figure 25



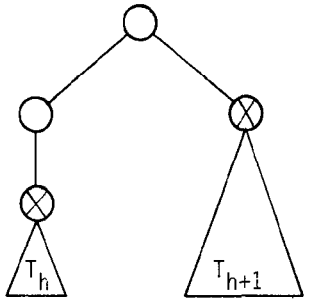


Figure 26

Thus insertion of $\sum_{l=1}^{h} (m-1) \cdot m^{l-1}$ new keys in increasing order starting with $T'(h+1)$ ultimately yields $T''(h+1, h+1)$. Since $T''(h+1, h+1) = T_{h+1}$ and

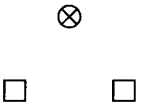
$$1 + \sum_{l=1}^{h} (m-1) \cdot m^{l-1} = m^h,$$

we have proved that iterative insertion of new maxima m^h times into $T(h+2, k, j)$ yields $T(h+2, k, j+1)$ as desired. □

THEOREM 2. *Iterative insertion of* $(m^h - 1)$ *keys in strictly ascending order beginning with the one-leaf tree of height* 1 *yields the complete m-ary tree* T_h.

PROOF. Clearly, insertion of $(m-1)$ keys into

yields

$T_1 =$ 

and insertion of a further $(m-1) \cdot m$ keys yields T_2.

Assume first that one new maximum is inserted into T_{h+1}, $h \geq 0$. Then the upward restructuring procedure generates the tree shown in Figure 25. Further insertion of $\sum_{l=1}^{h} (m-1) \cdot m^{l-1}$ new maxima yields (by Lemma 1) the tree shown in Figure 26.

By using Lemma 1 $(m-1)$ times we infer that $T(h+2, 0, m)$ is generated after a further $(m-1) \cdot m^h$ new maxima are inserted. Observe that $T(h+2, 0, m) = T(h+2, 1, 0)$. By using similar arguments we obtain the results shown in Table

Table II

Insertion of this many new maxima	Beginning with	Yields
$1 + \sum_{l=1}^{h} (m-1)\cdot m^{l-1} = m^h$	T_{h+1}	$T(h+2, 0, 1)$
$(m-1)\cdot m^h$	$T(h+2, 0, 1)$	$T(h+2, 0, m)$ $= T(h+2, 1, 0)$
$1 + \sum_{l=1}^{h} (m-1)\cdot m^{l-1} = m^h$	$T(h+2, 1, 0)$	$T(h+2, 1, 1)$
$(m-1)\cdot m^h$	$T(h+2, 1, 1)$	$T(h+2, 1, m)$ $= T(h+2, 2, 0)$
$1 + \sum_{l=1}^{h} (m-1)\cdot m^{l-1} = m^h$	$T(h+2, m-2, 0)$	$T(h+2, m-2, 1)$
$(m-1)\cdot m^h$	$T(h+2, m-2, 1)$	$T(h+2, m-2, m)$

II. Observe that $T(h + 2, m - 2, m) = T_{h+2}$. Thus we have shown that a total number of $(m - 1)[m^h + (m - 1)m^h] = m^{h+2} - m^{h+1}$ insertion of new maxima into T_{h+1} yields T_{h+2}.

Theorem 2 now follows by induction on h. □

By symmetry it should be clear that a corresponding result also holds for iterative insertion of keys in decreasing order.

We finally point to a further consequence which may be inferred from the proof of Theorem 2. When iteratively inserting N keys (in ascending order) where $N + 1$ is not a power of m, the resulting tree contains no root-to-leaf path with more than one unary node on it. This means that contracting unary nodes and their only sons into one node yields trees whose leaves appear on two adjacent levels. Trees of this kind may be considered to be optimal.

We believe that presorted or at least partially presorted sequences of keys occur quite often in practice. Hence, it is important to know how an insertion scheme behaves when applied iteratively to those sequences. Theorem 2 above is one step in this direction. We conjecture that it can be strengthened as follows: The tree which results by iteratively inserting a given sequence of keys approximates the complete m-ary tree the more closely the more presorted the sequence is.

5. ITERATIVE INSERTION OF KEYS IN RANDOM ORDER

Our insertion strategy for dense m-ary trees is to pass on the overflow problem from a node p to its father φp only when all brothers of p and the father φp of p are saturated. The B-tree splitting procedure, on the other hand, solves the overflow problem by only looking upward. Hence, it is at least intuitively plausible that the iterative application of our insertion procedure to a randomly chosen sequence of N keys generates a tree which (on the average) has fewer nodes for storing the N keys than a B-tree obtained by inserting the same (random) sequence of keys. Unfortunately, we are unable to prove this conjecture. Yao [15] has introduced a method for computing the average numbers of all subtrees of different kinds with a given height h in random N-key B-trees. These values give rise to a good estimation of the average number of nodes required to store N keys in random B-trees. An analysis of random r-dense m-ary trees along the lines of Yao's analysis for B-trees leads to serious difficulties. These difficulties and a suggestion for a way out of them are studied in detail in [9], with the main

emphasis laid on the case of binary trees. In this section we will apply the ideas of [9] to dense m-ary trees in order to estimate the average number of nodes in random dense m-ary trees built up by N iterative (random) insertions.

Instead of studying the average behavior of the insertion procedure of Section 3 for dense m-ary trees, we study a closely related but different procedure, called 2-insertion, along Yao's lines. Roughly speaking, 2-insertion differs from the insertion procedure of Section 3 by introducing "B-tree splitting" at level 2 above the leaves. More precisely, 2-insertion is obtained from the insertion procedure of Section 3 by using 2-*up* instead of *up*: We shall restrict ourselves to the case of even m, i.e., $m = 2k$. (The case of odd m can be treated analogously.)

We define the splitting of ovefilled nodes at level 2 such that the analysis of the 2-insertion procedure can be reduced to Yao's analysis [15] for B-trees.

2-*up*(p, q):

Case A [p occurs at a level $1 \neq 2$]. Then 2-*up*(p, q) is defined as *up*(p, q) (cf. Section 3).

Case B [p occurs at level 2]. We may assume that the invariant condition of *up* is fulfilled. This, in particular, means that p is saturated and q lies to the right of the leftmost son of p and to the left of the rightmost son of p.

Case B.1 [q lies between the 1st and kth son of p]. Create a new node p', make the $(k + 1)$st, $(k + 2)$nd, ..., $2k$th son of p the 1st, 2nd, ..., kth son of p', make q an additional son of p. Solve the insertion problem of the new node p' on level 2 by eventually calling *up*$(\varphi p, p')$ (or first making p' a son of φp and removing p and then calling *up*$(\varphi p, p)$ if p was the leftmost son of its father).

An example is shown in Figure 27.

We explicitly assume that no further restructuring affects the subtrees dangling from the nodes on level 2 after we have once passed level 2 on our way up to the root. This means any eventually initiated downward restructuring (e.g., by calling *leftsat* or *rightsat* for some nodes on levels above level 2) is terminated at level 2.

We will call a tree T a 2*Br*-m-ary tree if T is made by iterative insertions using 2-insertion into the initially empty tree (i.e., into the one-leaf tree of height 1). For sufficiently large N, an N-key 2*Br*-m-ary tree enjoys the following properties:

(1) All subtrees of height 2 dangling from nodes on level 2 and $(m - 1)$ dense m-ary trees.
(2) Each node on level 2 has at least $k = m/2$ sons.
(3) Cutting off all subtrees of height 2 and replacing them by leaves yields a dense m-ary tree.

Every 2*Br*-m-ary tree can be characterized by the numbers of different kinds of subtrees of height 2 as follows: We say an $(m - 1)$ dense m-ary tree T of height 2 is of type j, where $(m^2/2) - 1 \leqq j \leqq m^2 - 1$, iff the root of T has $\lfloor j/m \rfloor$ keys, exactly one son of the root has $\text{mod}(j, m) = m - \lfloor j/m \rfloor$ keys, and all other $\lfloor j/m \rfloor$ sons of the root have $m - 1$ keys. Thus every tree of type j has j keys and $j + 1$ leaves. A 2*Br*-m-ary tree T is said to be of class $(y_{(m^2/2)-1}, y_{m^2/2}, \ldots, y_{m^2-1})$ iff there are y_i height 2 subtrees of type i for every i. A 2*Br*-m-ary tree is said to be

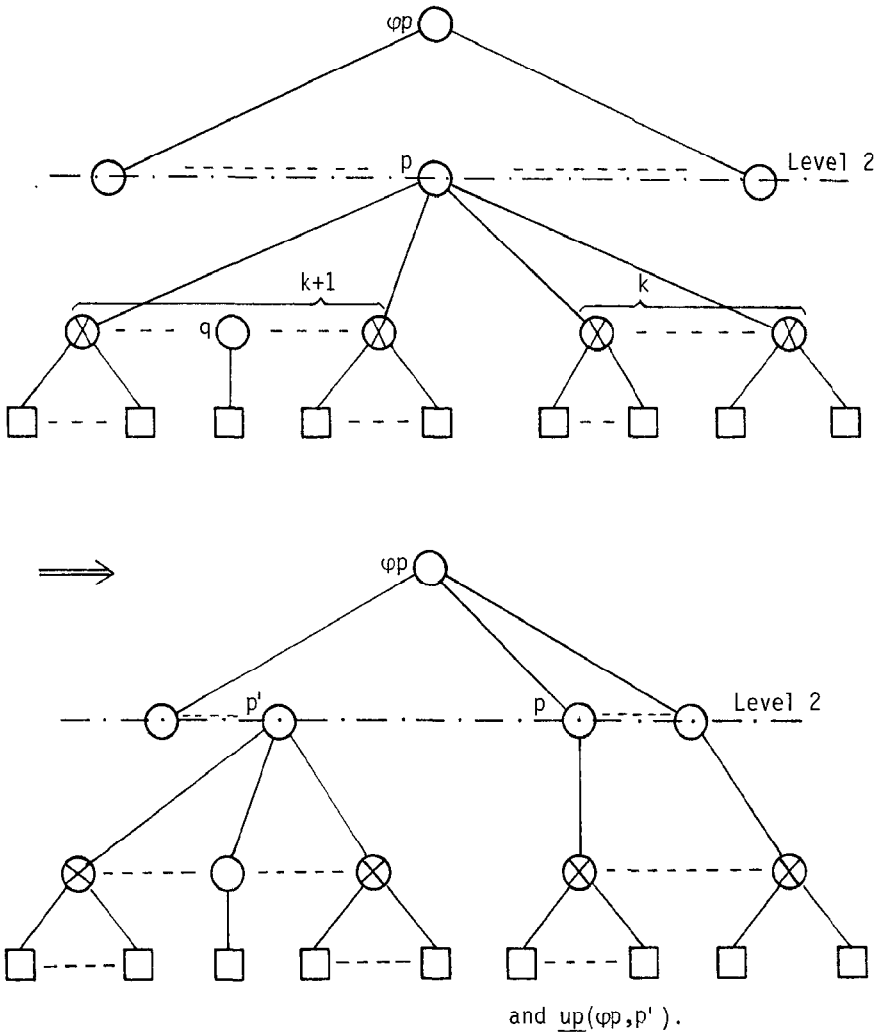


Figure 27

random if it is made by iterative random insertions into the initially empty tree. (Cf. Yao [15] for this notion.)

Let T be an $(N-1)$-key, N-leaf, $2Br$-m-ary tree of class $(y_{(m^2/2)-1}, \ldots, y_{m^2-1})$. The Nth random insertion has probability $i \cdot y_{i-1}/N$ of falling into a subtree of type $i-1$ (for each $i = m^2/2, \ldots, m^2$). The effect of the Nth random insertion using 2-insertion is obviously this: T may either become a $2Br$-m-ary tree of class $(y_{(m^2/2)-1}, \ldots, y_{i-1} - 1, y_i + 1, \ldots, y_{m^2-1})$ with probability $i \cdot y_{i-1}/N$, for each $i = m^2/2, \ldots, m^2 - 1$, or become a $2Br$-m-ary tree of class $(y_{(m^2/2)-1} + 1, y_{m^2/2} + 1, y_{(m^2/2)+1}, \ldots, y_{m^2-1} - 1)$ with probability $m^2 \cdot y_{m^2-1}/N$.

Denoting by $A_i(N)$ the average number of type i subtrees of N random insertions (for each $i = (m^2/2) - 1, \ldots, m^2 - 1$) we obtain a set of recurrence formulas for the quantities $A_i(N)$ which are of exactly the same form as in Yao's paper [15]. (Replace m by m^2 and p by $m^2/2$ in Yao's paper.)

Thus we obtain the solution

$$A_{(m^2/2)-1}(N) \to \frac{1}{(m^2/2)+1} \cdot \frac{1}{m^2+1} \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right)^{-1} \cdot (N+1)$$

$$A_i(N) \to \frac{1}{i+1} \cdot \frac{1}{i+2} \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right)^{-1} \cdot (N+1)$$

for each $i = m^2/2, \ldots, m^2 - 1$ (and large values of N).

Every subtree of type j has j keys and $\lfloor j/m \rfloor + 2$ nodes. Thus

$$x(N, 2) = \sum_{j=(m^2/2)-1}^{m^2-1} j \cdot A_j(N)$$

is the average number of keys stored at the two lowest levels in N-key random $2Br$-m-ary trees, while

$$y(N, 2) = \sum_{j=(m^2/2)-1}^{m^2-1} (m-1) \cdot (\lfloor j/m \rfloor + 2) \cdot A_j(N)$$

is the average number of storage cells in random $2Br$-m-ary trees at the two lowest levels. (For brevity, we have suppressed the dependency of $x(N, 2)$, $y(N, 2)$ on m.)

Yao [15] computed bounds for the average storage utilization of random B-trees from the corresponding quantities as follows. The average numbers of subtrees of different kinds of a given height h yield precise values of the numbers of keys (respectively, nodes) at the lowest h levels of internal nodes. The average number of keys (respectively, nodes) at levels higher than h is roughly estimated by the worst case static structure of B-trees. By combining these quantities, fairly good bounds for the average storage utilization of random N-key B-trees are obtained.

A major drawback for an analogous approach in the case of random N-key dense m-ary trees is the fact that the estimation of the average structure of random dense m-ary trees by their worst case static structure is very bad. Thus we restrict ourselves to computing the average storage utilization of random N-key $2Br$-m-ary trees at the lowest two levels of internal nodes. This value can be considered as an estimator for the average storage utilization of random N-key dense m-ary trees. (Cf. [9] for a justification of this approach.)

The storage utilization (at the lowest two levels of internal nodes) can be defined in different ways. First, we may compare the average number of stored keys at the lowest two levels of internal nodes with the maximal possible number $x_{\max}$ of keys (i.e., when all nodes are m-ary). Then we may be interested in the average value of the ratio of the number of stored keys over the number of reserved storage cells, where we assume that every node has $(m-1)$ storage cells. The quotient $x(N, 2)/y(N, 2)$ may be considered as an approximation of this ratio.

Finally, we may follow Yao's definition [15] and define the storage utilization of an N-key random $2Br$-m-ary tree at the lowest two levels of internal nodes as

the ratio of the number $(N+1)\cdot(m+1)/m^2$ of internal nodes at the lowest two levels in the optimal tree (i.e., in the complete m-ary tree) over the average number $y(N,2)/(m-1) = \sum_{j=(m^2/2)-1}^{m^2-1} (\lfloor j/m \rfloor + 2)A_j(N)$ of internal nodes at the lowest two levels of internal nodes. This latter ratio is denoted by $\mathrm{STU}(N, m)$. Table III shows a few explicitly calculated values of these quantities.

Finally, we compute $\mathrm{STU}(N, m)$ for large values of N and m:

$$\mathrm{STU}(N,m) = \frac{m+1}{m} \cdot \frac{1}{m \cdot \sum_{j=(m^2/2)-1}^{m^2-1} (\lfloor j/m \rfloor + 2)A_j(N)/(N+1)}$$

Now

$$m \cdot \sum_{j=(m^2/2)-1}^{m^2-1} \frac{(\lfloor j/m \rfloor + 2)A_j(N)}{N+1} \leqq \sum_{j=(m^2/2)-1}^{m^2-1} \frac{j \cdot A_j(N)}{N+1+2m} \sum_{j=(m^2/2)-1}^{m^2-1} \frac{A_j(N)}{N+1}.$$

We approximate the nth harmonic number $H(n)$ by $H(n) \approx \ln n + \gamma$ (for large n, where γ is Euler's constant). Then we first obtain for large values of N and m

$$\sum_{j=(m^2/2)-1}^{m^2-1} \frac{j \cdot A_j(N)}{N+1}$$

$$\to \left[\frac{(m^2/2)-1}{((m^2/2)+1)\cdot(m^2+1)} + \sum_{j=m^2/2}^{m^2-1} \frac{j}{(j+1)(j+2)} \right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right)^{-1}$$

$$= \left[\frac{2}{(m^2/2)+1} - \frac{3}{m^2+1} + \sum_{j=m^2/2}^{m^2-1} \left(\frac{2}{j+2} - \frac{1}{j+1} \right) \right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right)^{-1}$$

$$= \left[\frac{2}{(m^2/2)+1} - \frac{3}{m^2+1} + 2 \cdot \left(H(m^2+1) - H\left(\frac{m^2}{2}+1\right) \right) - \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right) \right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right) \right)^{-1}$$

$$\to \left[2 \cdot \left(\ln(m^2+1) - \ln\left(\frac{m^2}{2}+1\right) \right) - \left(\ln m^2 - \ln\left(\frac{m^2}{2}\right) \right) \right] \cdot \frac{1}{\ln m^2 - \ln(m^2/2)}$$

Table III

m	$X(N,2)/(N+1)$	$x_{max}/(N+1)$	$y(N,2)/(N+1)$	$x(N,2)/x_{max}$	$x(N,2)/y(N,2)$	STU
2	0.6571	0.7500	0.9143	0.8762	0.7188	0.8203
4	0.9113	0.9375	0.3724	0.9720	0.8156	0.8391
6	0.9602	0.9722	0.2236	0.9877	0.8589	0.8697
8	0.9776	0.9844	0.1576	0.9931	0.8860	0.8922
10	0.9856	0.9920	0.1211	0.9956	0.9045	0.9085
20	0.9964	0.9975	0.0554	0.9989	0.9474	0.9485
30	0.9984	0.9989	0.0357	0.9995	0.9638	0.9642
40	0.9991	0.9994	0.0263	0.9997	0.9724	0.9726
50	0.9994	0.9996	0.0209	0.9998	0.9777	0.9778
60	0.9996	0.9997	0.0173	0.9999	0.9813	0.9814
70	0.9997	0.9998	0.0147	0.9999	0.9839	0.9839
80	0.9998	0.9998	0.0128	0.9999	0.9858	0.9859
90	0.9998	0.9999	0.0114	0.9999	0.9874	0.9874
100	0.9999	0.9999	0.0102	1.0000	0.9886	0.9886

$$= \left[2 \cdot \ln \frac{m^2+1}{(m^2/2)+1} - \ln 2\right] \cdot \frac{1}{\ln 2}$$

$$\to 1.$$

Second, we obtain for large N and m

$$2m \cdot \sum_{j=(m^2/2)-1}^{m^2-1} \frac{A_j(N)}{N+1}$$

$$\to \left[\frac{2m}{((m^2/2)-1)\cdot(m^2+1)} + 2m \cdot \sum_{j=m^2/2}^{m^2-1} \frac{1}{(j+1)} \cdot \frac{1}{(j+2)}\right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right)\right)^{-1}$$

$$= \left[\frac{4m}{(m^2-2)(m^2+1)} + 2m\left(\sum_{j=m^2/2}^{m^2-1} \frac{1}{(j+1)} - \sum_{j=m^2/2}^{m^2-1} \frac{1}{(j+2)}\right)\right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right)\right)^{-1}$$

$$= \left[\frac{4m}{(m^2-2)(m^2+1)} + 2m\left(\frac{1}{(m^2/2)+1} - \frac{1}{(m^2/2)+1}\right)\right] \cdot \left(H(m^2) - H\left(\frac{m^2}{2}\right)\right)^{-1}$$

$$\to 0, \qquad \text{as } m \text{ becomes large.}$$

Thus as N and m become large, the average storage utilization STU(N, m) (at the lowest two levels of internal nodes) in random $2Br$-m-ary trees tends to 1, i.e., it becomes as large as possible, compared with only 0.69 in the case of B-trees (cf. Yao [15]).

Because it is intuitively plausible that the 2-insertion procedure bounds from below the average behavior of the insertion strategy of Section 3, we infer that N-key random dense m-ary trees also become as dense as possible (for large enough values of N and m).

6. CONCLUDING REMARKS

In this paper we have presented a new class of balanced multiway trees, the dense multiway trees which we consider to be an alternative to the ubiquitous B-trees. Whether they are a viable alternative remains to be seen. However, some strong evidence in favor of this new class has been presented. We have compared the storage utilization of the two classes for insertions. This has indicated that the dense multiway trees do indeed have a better storage utilization than B-trees. In particular, we have proved that carrying out insertions of keys in ascending order leads to complete m-ary trees in the dense class, but this is not true for B-trees. Moreover, we have shown, using some plausible assumptions, that the average storage utilization tends to 1 as m and N tend to the limit. This also is not true for B-trees. This motivates our choice of the term "dense" for this new class of trees.

It should be noted that these analyses are "dynamic" as compared with the "static" analyses for B-trees of [13] and [14]. Mehlhorn [6] has in fact calculated that the static worst case storage utilization tends to 0 in the limit! These widely differing results indicate the differences between the two kinds of analyses. It is our opinion that dynamic analysis based on random insertions gives a better understanding of the average behavior of classes of trees than the corresponding static analysis. As already pointed out, Yao's approach [15] combines both methods of analysis to give bounds for the dynamic behavior of B-trees. Clearly, this should also be modified to give a fully dynamic approach. Such analyses for B-trees and B*-trees remain as topics for further investigation, as does the problem of dealing with both random insertions and random deletions in such an analysis. It should be mentioned that Larson and Walden [4] have made some inroads into the analysis of B*-trees.

There is obviously some kind of a trade-off between the amount of work for insertion and the density of trees obtained by performing iterative insertions.

Not looking at brothers at all (as in the insertion scheme for B-trees) is very simple but generates quite sparse trees. On the other hand, looking at all brothers (as the insertion scheme for dense m-ary trees does) could be very messy and time consuming (because many nodes and keys have to be shifted). However, this generates very dense trees. Thus we may consider these two classes of trees as the two endpoints of a whole variety of trees and insertion schemes in between: They are obtained by looking at more and more brothers.

Whether or not a scheme is appropriate depends on the actual environment. This is determined mainly by the number of search operations compared with the number of updates which have to be carried out and, furthermore, by the time which is necessary to settle a backup storage request. We believe that there are now at least some analytical tools available for evaluating different insertion schemes.

We close this discussion by mentioning some remaining open problems for dense m-ary trees.

First and foremost is the development of a deletion procedure. In the binary case, deletion can be carried out in time $O(\log N)$ (see [10]), and in the case of strongly dense ternary trees an $O(\log^2 N)$ deletion procedure is available (cf. [12]). But the general case remains open. It is not difficult to see how the general approach given in Section 3 for insertion can be adapted for deletion. This gives at worst an $O(N)$ algorithm and at best an $O(\log^{m-1} N)$ algorithm. However, even for weakly dense m-ary trees, $m > 2$, no improvement is known.

Second, the existence of an $O(\log N)$ insertion procedure for strongly dense m-ary trees remains a tantalizing open question. We conjecture that no $O(\log N)$ procedure exists, but this conjecture seems difficult to prove. However, a related open problem is perhaps more tractable, namely, is the average insertion time $O(\log N)$?

Finally, the relationship of the classes of dynamically defined dense m-ary trees with the classes of (statically defined) dense m-ary trees needs to be clarified. This question is, of course, applicable to other classes of trees as well.

REFERENCES

1. BAYER, R., AND MCCREIGHT, E. Organization and maintenance of large ordered indexes. *Acta Inf. 1* (1972), 173–189.
2. KNUTH, D. *The Art of Computer Programming: Sorting and Searching*, vol. 3. Addison-Wesley, Reading, Mass., 1978.
3. KRIEGEL, H.P., VAISHNAVI, V.K., AND WOOD, D. 2-3 brother trees. *BIT 18* (1978), 425–435.
4. LARSON, J.A., AND WALDEN, W.E. Comparing insertion schemes used to update 3-2 trees. Tech. Rep., Univ. New Mexico, Albuquerque, N. Mex., 1979.
5. LIU, C.L. *Introduction to Combinatorial Mathematics*. McGraw-Hill, New York, 1968.
6. MEHLHORN, K. Private communication, 1979.
7. OLIVIÉ, H.J. On the relationship between 2-3 brother trees and dense ternary trees. *Int. J. Comput. Math. 8A* (A80), 233–245.
8. OTTMANN, TH., AND SIX, H.W. Eine neue Klasse von ausgeglichenen Binärbäumen. *Angew. Inf. 18, 9* (1976), 395–400.
9. OTTMANN, TH., AND STUCKY, W. Higher order analysis of random 1-2 brother trees. *BIT 20* (1980), 302–314.
10. OTTMANN, TH., AND WOOD, D. 1-2 brother trees or AVL trees revisited. *Comput. J. 23* (1980), 248–255.
11. OTTMANN, TH., AND WOOD, D. A uniform approach to balanced binary and multiway trees. *Mathematical Foundations of Computer Science*, Springer-Verlag Lecture Notes in Computer Science, vol. 74, Springer-Verlag, New York, 1979, pp. 398–407.
12. PATZAK, E. Dichte 3-Weg Bäume, Master's thesis, Karlsruhe, 1978.
13. ROSENBERG, A.L., AND SNYDER, L. Minimal comparison 2, 3 trees. *SIAM J. Comput. 7* (1978), 465–480.
14. ROSENBERG, A.L., AND SNYDER, L. Compact B-trees. Rep. RC 7343, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.
15. YAO, A. On random 2, 3, trees. *Acta Inf. 9* (1978), 159–170.

Received November 1978; revised November 1979; accepted December 1980