Alcom-FT Technical Report Series
ALCOMFT-TR-01-78

On the Existence and Construction of Non-Extreme (a,b)-Trees *

Lars Jacobsen Kim S. Larsen Morten N. Nielsen

University of Southern Denmark, Odense †

Abstract

In amortized analysis of data structures, it is standard to assume that initially the structure is empty. Usually, results cannot be established otherwise. In this paper, we investigate the possibilities of establishing such results for initially non-empty multi-way trees.

Introduction

An (a, b)-tree [8] is a search tree, where each node, except the root, has at least a and at most b children. Here, a and b are integers such that $a \geq 2$ and $b \geq 2a - 1$.

B-trees [3] is a very well known special case of an (a, b)-tree, where usually $b = 2a - 1$ or $b = 2a$, and b is chosen such that each node fits into a block, where the block size is often defined by a disk. The importance of B-trees in database systems and other applications where external memory efficiency is a high priority is well known.

* Supported in part by the Danish Natural Sciences Research Council (SNF) and in part by the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

† Department of Mathematics and Computer Science, University of Southern Denmark, Main campus: Odense University, Campusvej 55, DK-5230 Odense M, Denmark. E-mail: {eljay,kslarsen,nyhave}@imada.sdu.dk.

What is less commonly known is that (a, b)-trees are of very significant importance in the theoretical community as well. The results concerning $o(\log n)$ searching and $o(n \log n)$ sorting on word RAM's [4], for instance, depend heavily on (a, b)-trees; see [1], for example. Also new results in the usual comparison-based model utilize (a, b)-trees, as in [9], for example, which considers time-space trade-offs. See [4] for additional references.

One of the primary reasons for the efficiency of (a, b)-trees in practical applications is the low height which can be guaranteed due to the high out-degree of the nodes. This means that very few nodes must be fetched during a search. This is very important in external memory applications, and it is becoming increasingly important in internal memory because of the increasing difference in speed between cache and main memory.

Other than searching, significant time in search trees is spend on rebalancing the structure after updates; in the case of (a, b)-trees when the number of children would otherwise decrease below a or beyond b. In the worst-case, rebalancing is more expensive than searching because two or three nodes are involved in the process on each layer, as opposed to only one node for searching. However, by allowing some flexibility regarding the number of children of nodes, the hope is that rebalancing is infrequent. By requirering that $b \geq 2a$, a formal guarantee can be obtained in that rebalancing can be proven to be amortized $O(1)$ [5]. Thus, k updates will give rise to at most $O(k)$ rebalancing operations when starting with an initially empty tree.

Amortized results are usually established using a potential function, which is a way of registering that time has been saved at some earlier point, which again means that more time can be spend later without violating the claim that k updates can be processed in time $O(k)$. Most often in data structure results, there is a very direct interpretation of the potential function, where the value of the function at a given point in time can be accounted for by specific nodes in the structure which are in some sense critical with regards to the balance conditions. In fact, quite often, the potential function is defined as the sum of contributions coming from the individual nodes in the structure. This means that not alone does the amortization proof assume that the starting structure is empty, the results would simply usually not hold otherwise.

However, there are certain guidelines for setting up amortization results, which are partly due to mathematical necessecity. One is that potential is usually accumulated gradually as a structure deteriorates. For (a, b)-trees, this means that one would expect that the potential would be high for nodes of extreme degrees, and low, or possibly zero, for other nodes. This is what we aim to take advantage of. If it is possible to create an initial tree where all nodes (except maybe a constant number) have (sufficiently) non-extreme degrees, then it may have zero (or constant) potential, and this would mean that the amortization results which hold when starting with an empty structure would also hold when starting with this type of initial tree.

This makes it interesting to determine for which a and b, a number of keys can be arranged in an (a, b)-tree such that almost no nodes have extreme degrees, and to be able to specify the construction. We solve this problem, and give examples of applications.

We assume familiarity with (a, b)-trees, their rebalancing operations, and terminology [8].

Trees with Nodes of Non-Extreme Degrees

An (a, b)-tree for constants $a \geq 2$ and $b \geq 2a - 1$ is a multi-way search tree, where the following holds:

- The root has between 2 and b children.
- Every non-root node has between a and b children.
- The leaves are all stored at the same depth.

We consider building (a, b)-trees using nodes with degrees in a small sub-range $D = \{d_1, \ldots, d_2\}$ of the possible degrees $\{a, \ldots, b\}$, where $a < d_1$ and $d_2 < b$. In particular, we focus on bounding the number of *extreme* nodes, i.e., nodes with degrees outside the sub-range D.

We distinguish between two types of trees. *Leaf-oriented* trees store all items in the leaves while all keys in internal nodes are routers used only to guide searches. In particular, the routers in internal nodes need not be copies of keys present in the leaves.

As opposed to leaf-oriented trees, we consider *internal* trees. In an internal tree all keys are items, and thus only a fraction of the items are stored in the leaves. Usually, leaf-oriented trees are preferred since deletions are easier to handle and branching in internal nodes is maximized.

The degree of a node is the number of children for internal nodes and the number of items stored for leaves. To simplify the presentation, we let the term *pointers* denote children when we are referring to internal nodes, and items when we are referring to leaves, respectively.

The following results all apply to leaf-oriented (a, b)-trees. At the end of this section, we discuss how to extend the results to apply to internal trees as well.

We define the *size* of an (a, b)-tree to be the number of items n stored in the tree. Let T_n be an (a, b)-tree of size n and height h. Let u be a node in T_n and let $d(u)$ denote the degree of u. Let ℓ_i denote the set of nodes on level i, i.e., at a distance i from the root. Let n_i denote the number of nodes on level i. Thus, $n_0 = 1$ and $n_i = \sum_{u \in \ell_{i-1}} d(u)$ for $i \geq 1$. With regards to the total number of keys, $n_h = n$, since T_n is leaf-oriented. Finally, let $E(T_n) = \{u \in T_n \mid d(u) \notin D\}$, i.e., $E(T_n)$ is the set of extreme nodes.

Positive results

Theorem 1 If $|D| \geq 2$, then for all constants a and b satisfying $a \geq 2$ and $b \geq 2a-1$ and for all n, there exists an (a, b)-tree T_n such that $|E(T_n)| \in O(1)$.

Proof The proof is by induction. Let $d, d+1 \in D$.

Assume $n < d^2$. Since a and b are constants, so are d and d^2. However, any legal (a, b)-tree representing at most a constant number of pointers on the lowest level uses at most a constant number of nodes and the result follows.

Now assume $n \geq d^2$. We show how to represent n pointers on the lowest level. Let $k_1 = n \mod d$. Now using k_1 nodes with degree $d+1$ and $k_2 = \frac{n-(d+1)\cdot k_1}{d}$ nodes with degree d, we obtain a representation of n pointers using no extreme nodes. Let n' denote the number of pointers to be represented on the level above, then $n' = k_1 + k_2 < n$ and the result follows by induction. □

Since we use a constant number of nodes with degree $d+1$ on each layer in the induction step, we immediately have the following corollary.

Corollary 1 If $D = \{d\}$, then for all constants a and b satisfying $a \geq 2$ and $b \geq 2a - 1$ and for all n there exists an (a,b)-tree T_n such that $|E(T_n)| \in O(\log n)$.

Negative results

Theorem 2 If $|D| = 1$, then for all constants a and b satisfying $a \geq 2$ and $b \geq 2a - 1$, there exists infinitely many n such that for all (a,b)-trees, T_n, $|E(T_n)| \in \Omega(\frac{\log n}{\log \log n})$.

Proof For a fixed height h, we can choose to have every extreme node on at most h different levels. Each extreme node can have at most $b - a$ different degrees. Therefore, at most $(h(b-a))^f$ different trees can be represented using at most f extreme nodes. A tree of size N or smaller has height at most $\log_a N$. The number of different trees of height at most $\log_a N$ and with at most f extreme nodes is at most

$$\sum_{h=1}^{\log_a N} (h(b-a))^f \leq \log_a N(\log_a N(b-a))^f = \log_a^{f+1} N \cdot (b-a)^f \leq (c \log_a N)^{f+1},$$

where $c = b - a$. Given $N \in \mathbb{N}$, to represent all trees with sizes $n \in \{0, \ldots, N\}$, we must have $(c \log_a N)^{f+1} \geq N$, leading to $f \in \Omega(\frac{\log n}{\log \log n})$. $\square$

Consider the applications of potential functions for showing (a,b)-trees to have amortized constant time behavior. Such potential functions typically have their minimum for degrees in the range from $2a - 1$ to $\frac{b}{2}$, since nodes of degree approximately $\frac{b}{2}$ and $2a - 1$ are the result after splits and fusions, respectively. If $b \geq 4a$, then the ranges $[a; 2a-1]$ and $[\frac{b}{2}; b]$ do not overlap and we would expect to have all degrees from $2a - 1$ to $\frac{b}{2}$ available to construct non-extreme trees, making Theorem 1 applicable.

When $b < 4a$, there might be exactly one degree available to construct non-extreme trees, since the potential is usually build up gradually from there towards a and b. Under these conditions we show that when $D = \{d\}$ and $b < 2d - 1$, there exist infinitely many n such that the bound in Corollary 1 is the asymptotically best possible.

Let $N_i = d^{i+1} - \sum_{j=0}^{i} d^j$, $i \geq 0$. Some simple properties of these numbers N_i are expressed below:

Proposition 1 The numbers N_i have the properties that $N_i = dN_{i-1} - 1$, for $i \geq 1$, and $N_i \equiv_d -1$, for $i \geq 0$. $\square$

We define a *minimum extreme tree* as a tree T_n having a minimum number of extreme nodes among all trees of size n. Note that such a tree cannot have extreme nodes of degrees less than d and greater than d on the same level.

For a tree T_n, we define the number of *extreme pointers* p_i on level ℓ_i as the total number of pointers missing or in excess in extreme nodes. Formally, if $D = \{d\}$:

$$p_i = \sum_{u \in E(T_n) \cap \ell_i} |d - d(u)| \, .$$

Theorem 3 Assume that $D = \{d\}$ and that we have constants a and b satisfying $a \geq 2$ and $b \geq 2a - 1$. If $b \leq 2d - 1$, then for all N there exists an $n > N$ such that $E(T_n) \in \Omega(\log n)$ for all (a, b)-trees T_n.

Proof Let T_n be a minimum extreme tree. If $E(T_n) \cap \ell_i = \ell_i$ for any $i \geq \log\log n$, we are done, since then $E(T_n) \in \Omega(\log n)$. In the following, we assume that $E(T_n) \cap \ell_i \neq \ell_i$, when $i \geq \log\log n$.

The proof is in two parts. First we show that trees of a certain class contain as few extreme nodes as any minimum extreme tree. Then we show that there exists $n > N$ for all N, such that trees in this class have at least one extreme node on every level, except maybe for the first $\log\log n$ levels.

We define a *canonical extreme tree* to be an extreme tree having $p_i \leq d - 1$ for all $i > \log\log n$.

> *Claim*: Given a T_n, there exists a canonical extreme tree T'_n such that $|E(T'_n)| \leq |E(T_n)|$.

If T_n is a canonical extreme tree, the claim follows by letting $T'_n = T_n$. Otherwise, we construct T'_n by repeatedly applying the following exchange argument:

Consider a level i with i greater than $\log\log n$ in T_n and $p_i \geq d$. Assume that the extreme nodes on this level all have degrees greater than d. Since, by the restriction on b, any extreme node contains at most $d-1$ extreme pointers, we can reduce the number of extreme nodes on this level by at least one by adding a non-extreme node of degree d. When a node is added to level i, n_{i-1} is increased by one. However, this can increase the number of extreme nodes on level $i-1$ by at most one. Thus, $|E(T'_n)| \leq |E(T_n)|$. The case where all extreme nodes have degrees less than d is analogous.

In the following, we restrict our attention to canonical extreme trees, since they are as good as any minimum extreme tree by the argument above.

Claim: If $n_i = N_i - c_1$, then level i must contain at least one extreme node and $n_{i-1} = N_{i-1} - c_2$ for $c_1, c_2 \in \{0, 1\}$.

The first part follows immediately since $N_i \equiv_d -1$. Recall from the definition of $D = \{d_1, \ldots, d_2\}$ that $d_1 > a$, so $d > 2$. For the second part, we consider two cases. First, if level i contains nodes with degrees less than d, we have $1+c_1$ extreme pointers in at least one and at most two extreme nodes. Thus, $n_{i-1} = \frac{N_i - c_1 + (1+c_1)}{d} = \frac{N_i+1}{d} = N_{i-1}$.
Second, if level i contains nodes with degrees greater than d, we have $d-1-c_1$ extreme pointers in at least one and at most $d-1-c_1$ extreme nodes. Thus, $n_{i-1} = \frac{N_i - c_1 - (d-1-c_1)}{d} = \frac{N_i+1-d}{d} = N_{i-1} - 1$.
Since this is true for every level, except maybe the first $\log\log n$, we have that $|E(T_n)| \in \Omega(\log n)$. □

Internal (a, b)-trees

Until now the trees considered have been leaf-oriented. We now verify the results for internal trees. Observe that for internal trees, the number of items stored in the internal nodes is exactly the number of leaves minus one, since these routers are used to distinguish between the leaves. Thus, in this case, $n = n_h + n_{h-1} - 1$.

While the reader should have no difficulty modifying Theorems 1 and 2 to apply to internal trees as well, Theorem 3 deserves a short comment.

The definition and results concerning canonical extreme trees is true for internal trees. Thus, if one chooses an n of the form $n = N_h - 1 + d \cdot N_h$, then having at most $d-1$ extreme pointers on the bottommost level requires that the number of leaves is N_h, and the proof of Theorem 3 is easily modified to cover internal trees as well.

Example Applications

In this section, we give a few examples of where our results give new possibilities. The number of applications of (a, b)-trees is enormous, so we are well aware that this collection of examples is not exhaustive. We also find it very likely that many more (a, b)-tree based structures, capable of exploiting these results, will be designed in the future.

Standard (a, b)-trees

Standard (a, b)-trees with $b \geq 2a$ were proven to have amortized constant rebalancing in [5, 8]. However, a simpler proof can be given using the potential function

$$\Phi(T) = \min(T) + 2\max(T),$$

where $\min(T)$ and $\max(T)$ are the number of nodes in the tree T with degree a and b, respectively.

Using this potential function, it is easily seen that rebalancing is amortized constant. We only need to consider splits and fusions, since the remaining possible operations are applied at most once per update.

Since $b \geq 2a$, splitting a node with $b+1$ children yields at most one node with a children and incurs a potential drop of one. Similarly, a fusion between two nodes, one with $a-1$ children and one with a children, decreases the potential by one as well.

Clearly, a tree with at most a constant number of nodes of extreme degrees would have constant potential, and the amortization results would hold also from that starting point.

Several similar potential functions and results, to which the results of this paper can be applied, can be found in [6].

Multi-way trees with group updates

In [7], results are obtained which primarily show that group insertions into a so-called relaxed (a, b)-tree can be performed efficiently in the amortized sense. Again, the assumption is that the start state is the empty tree. The potential function is a sum of contributions from each node in the tree. The contribution from a node u is defined as

$$\Phi(u) = \begin{cases} 3, & t(u) = 0,\ c(u) < a \\ 1, & t(u) = 0,\ c(u) = a \\ 2, & t(u) = 0,\ c(u) = b \\ 3, & t(u) = -1,\ c(u) \leq 2 \\ 5, & t(u) = -1,\ c(u) > 2 \\ 12(|t(u)| - 1) + 5, & t(u) < -1 \\ 0, & \text{otherwise } (t(u) = 0,\ a < c(u) < b) \end{cases}$$

where $c(u)$ denotes the degree of u. If u is the root, $\Phi(u)$ is defined similarly, except that the constant 2 is substituted for a. A non-zero value of $t(u)$ is used to denote that the descendants of the node u are not at the correct level, so $t(u) = 0$ denotes the normal situation.

Thus, building an initial tree with non-extreme degrees would provide a tree, where $t(u) = 0$ for all nodes, and we would fall into the last case in the definition above.

Buffer trees

In [2], the buffer tree is developed, aimed at minimizing I/O traffic for algorithms in general. A buffer tree has an (a, b)-tree as its basic structure, but in addition, all nodes are equipped with a buffer, which is used to postpone updates until many can be carried out at the same time. Again, amortized results are obtained, assuming that the initial tree is empty. Formulating the proof from [2] in terms of a potential function, it can be viewed as a sum of contributions from the individual nodes, and the contribution from the nodes can be viewed as being composed of a contribution to pay for later standard (a, b)-tree rebalancing and a contribution to pay for emptying buffers. Thus, by building an initial tree with non-extreme degree nodes and empty buffers, a tree with potential zero is obtained.

References

[1] Arne Andersson. Sublogarithmic Searching without Multiplications. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science*, pages 655–663. IEEE Computer Society Press, 1995.

[2] Lars Arge. The Buffer Tree: A New Technique for Optimal I/O-Algorithms. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 334–345. Springer-Verlag, 1995.

[3] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica*, 1:173–189, 1972.

[4] Torben Hagerup. Sorting and Searching on the Word RAM. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *Lecture Notes in Computer Science*, pages 366–398. Springer-Verlag, 1998.

[5] Scott Huddleston and Kurt Mehlhorn. A New Data Structure for Representing Sorted Lists. *Acta Informatica*, 17:157–184, 1982.

[6] Lars Jacobsen and Kim S. Larsen. Variants of (a, b)-Trees with Relaxed Balance. *International Journal of Foundations of Computer Science*. To appear.

[7] Kim S. Larsen. Relaxed Multi-Way Trees with Group Updates. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001. To appear.

[8] Kurt Mehlhorn. *Sorting and Searching*, volume 1 of *Data Structures and Algorithms*. Springer-Verlag, 1984.

[9] Jakob Pagter and Theis Rauhe. Optimal Time-Space Trade-Offs for Sorting. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 264–268. IEEE Computer Society, 1998.