Amortized Analysis of (a, b)-trees

Rolf Fagerberg

October 12, 2009

We consider (a, b)-trees for $b \geq 2a-1$ and $a \geq 2$. Recall that the rebalancing operations work as follows:

> *Split:* A node with $b+1$ children are split into two nodes having $\lfloor (b+1)/2 \rfloor$ and $\lceil (b+1)/2 \rceil$ children, respectively.
>
> *Fuse:* A node with $a-1$ children, whose neighboring siblings (one or two exist) all have a children each, fuses with one of these siblings into a single node with $2a-1$ children.
>
> *Share:* A node with $a-1$ children, and at least one neighboring sibling having $x \geq a+1$ children, first fuses with such a sibling, and then splits (into two nodes having $\lfloor (x+a-1)/2 \rfloor$ and $\lceil (x+a-1)/2 \rceil$ children, respectively).

The following points expand on and complement the description of (a, b)-trees in the main notes:

- In all of the rebalancing operations, we end up with legal nodes (check this) at the current level.
- Splits [fuses] increase [decrease] by one the number of children of the parent. These operations may therefore propagate all the way to the root. Sharing does not propagate, as the number of children of the parent is unchanged.
- Rebalancing at the root are special cases. Splitting the root gives rise to a new root (with just two children) one level higher. We will view this as the introduction of a new unary root before the split, such that a splitting node always has a parent. Fuses just below the root may lead to a root having a single child, which is superfluous and hence is removed after the fuse. Thus, by a *root operation* we mean the introduction or removal of a unary root.
- Let the *share threshold* be the minimal number of children after a fuse for which we go on to split (resulting in a share operation). Above, the

threshold is $2a$. This value may be defined differently, while still leading to legal nodes. The minimum value possible is $2a$, the maximum value possible is $b+1$.

- In *leaf-oriented* search trees, the leaves contain all the elements stored, and internal nodes contain only key values to guide the search. B-trees, widely used in database systems for indexes, are usually leaf-oriented (a,b)-trees, with $b = 2a-1$ or $b = 2a$, and b equal to the block size B. Often, the leaf level has a separate set (a', b') of values for a and b, as leaves contain elements, while internal nodes contain keys (normally just a part of elements) and pointers, making the maximal number fitting in a block differ between the leaves and the internal nodes. We in this note use leaf oriented trees, and for simplicity of statements and proofs assume $a' = a$ and $b' = b$. Adapting to the general case is straight-forward.

- The above statement of the rebalancing operations apply for internal nodes. Leaf nodes have corresponding operations with"number of children" changed into "number of elements in node".

- In leaf-oriented trees, the handling of keys in internal nodes differ between the leaf level and the higher levels. When splitting [fusing] at a leaf, a *copy* of an appropriate leaf key is inserted into [deleted from] the next higher level. When splitting [fusing] at an internal node, all required keys are present, and one is moved one level up [down] during the operation. For shares, the handling of keys follows from the definition of a share as a fuse and a split combined.

Now for the main aim of this note, which is to give bounds on the amortized number of rebalancing operations in (a,b)-trees. The bounds turn out to depend on the relation between a and b, as stated in the following theorem.

For the theorem to be true, the share threshold cannot be chosen entirely freely: In the first statement in the theorem, the minimal value $2a$ and maximal value $b+1$ possible are actually the same. The proof of the second statement can be seen to hold for share threshold values from $2a$ to b, i.e., the maximal value $b+1$ must be disallowed (the below proof is carried out using the specific value $2a$ from the description of the rebalancing operations above). For the third statement to be true, the share threshold value should be chosen even more restrictively (actually deviating from the description of the rebalancing operations above). One possible choice is setting the

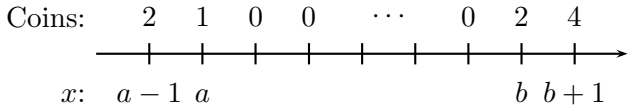
share threshold value for rebalancing at the leaf level to the specific value $2a + 2\Delta/3$, where Δ is defined by $b = 2a + \Delta$, while for the remaining levels just disallowing the maximal value $b + 1$.

Theorem 1 *Starting with an empty tree, the amortized number of rebalancing operations in an (a, b)-tree is:*

1. $\Theta(\log_a n)$ *when* $b = 2a - 1$.
2. $\Theta(1)$ *when* $b = 2a$.
3. $\Theta(1/(\epsilon a))$ *when* $b = (2 + \epsilon)a$, *for* $\epsilon > 0$.

Proof: The first statement is seen by considering a tree where all nodes on the left-most path have b children (and the leaf has b elements). When inserting a new smallest key and deleting it again, each update creates rebalancing which propagates all the way to the root, *and* we end up with the same tree as we started with. As this can be repeated indefinitely, we get the stated lower bound on the amortized number of rebalancing operations. The upper bound follows from the general worst case upper bound.

To prove the upper bound of the second statement, note that each update can at most generate one share (shares do not propagate). Similarly, adding or removing a unary root happens only once per update. Thus, we only need to bound the number of splits and fuses. We use a coin-based amortization argument where one coin is able to pay for one split or one fuse. We maintain the invariant that all non-root nodes has a number of coins based on its number x of children (for leaves: its number x of elements) as follows:



For the root, the invariant is similar, except it is zero for all $x \leq b - 1$.

The initial empty tree has zero coins. *Inserts* and *deletes* of elements at the leaves (not counting any ensuing rebalancing) may need to introduce up to two new coins to maintain the invariant.

When a node is about to do a *split*, it by the invariant has 4 coins. After the split, one of the resulting nodes has $x = \lfloor (b + 1)/2 \rfloor$, which is at least a (as $b + 1 \geq 2a$). This node needs at most one coin. The other resulting node has

$x = \lceil (b+1)/2 \rceil$, which is at least $a+1$ (as $b+1 > 2a$ and a is an integer) and at most $b-1$ (as $(b+1)/2 = b/2+1/2 = b-b/2+1/2 \leq b-2+1/2$), where the inequality follows from $b \geq 2a \geq 2 \cdot 2$). This node needs no coins. For the parent, x increases by one, hence it may need at most two new coins. This leaves at least one free coin, one of which pays for the split (the rest, if any, are thrown away).

When a node is about to do a *fuse*, it has $x = a - 1$, and the sibling fused with has $x = a$. In total, they have three coins. The resulting nodes has $x = 2a - 1$, which is at at least $a + 1$ (as $a \geq 2$) and most $b - 1$ (as $b \geq 2a$). Hence, it needs no coins. For the parent, x may decrease by one, hence it needs at most one new coin. This leaves at least two free coins, one of which pays for the fuse (the rest are thrown away).

When a node is about to do a *share*, it has $x = a-1$, and the sibling shared with has $a + 1 \leq x \leq b$. In total, they have at least two coins. After the share, one of the resulting nodes has $x = \lfloor y/2 \rfloor$ and the other $x = \lceil y/2 \rceil$, for a y between $2a$ and $b + a - 1$. Hence, for both nodes, we have an x of at least a and at most $b - 1$ (the latter because $b + a - 1 \leq 2b - 3$, as $b - a \geq a \geq 2$). For the parent, x is unchanged. Hence, we can always maintain the invariant by reusing existing coins. No coin is needed to pay the share, as shares are already accounted for.

A *root operation*, i.e., the introduction or removal of a unary root, does not require any new coins to maintain the invariant. No coin is needed to pay the operation, as root operations are already accounted for.

Summing up: The tree initially has zero coins. New coins are only added to the tree by updates, and each update adds at most two coins. Each split and fuse is payed by one coin, which is removed from the tree. It follows that at any given point in time, the number of splits and fuses can only be twice the number of updates performed until now. We have previously argued that the number of shares and root operations can at most be the number of updates performed until now. This proves the upper bound of the second statement.

The proof of the lower bound of the second statement is left as an exercise.

The proof of the third statement is also left as an exercise (hint: allow fractional coins, and change the invariant for leaf nodes such that updates only need to pay $O(1/a)$ of a coin). □

The following stronger versions of the amortized bounds above actually hold (exercise: argue they are stronger by using them to prove the statements

in the theorem above). The restrictions on the share threshold value for the previous theorem apply, with the addition that for the third statement, we set the share threshold to $2a + 2\Delta/3$ for the rebalancing of nodes on *all* levels, not just for the leaf nodes.

By the *level* of a rebalancing operation, we mean the height of the node splitting, fusing, or sharing (or in the case of root operations, the height of the unary root added or removed). The height of leaves is defined as one, and the height of an internal is node defined as one larger than the height of its children.

Theorem 2 *Starting with an empty tree, the amortized number of rebalancing operations taking place at a given level i of the tree is*

1. $\Theta(1)$ *when* $b = 2a - 1$.
2. $O((2/3)^i)$ *when* $b = 2a$.
3. $O((\Theta(1/\epsilon a))^i)$ *when* $b = (2 + \epsilon)a$.

Proof: The first statement follows directly from the proof of the corresponding statement of the previous theorem.

The second statement also follows from the proof of the corresponding statement of the previous theorem by noting that coins are only moved upwards during rebalancing. In particular, one coin on level i is used by a split or fuse for each moving of at most two coins to the next level. This means that only two thirds of the coins reaching any given level can reach the level above it. This shows the statement for split and fuse operations. For shares and root operations, note that for any such operation on a given level, there must have been a split or fuse on the level below (or, if the given level is the leaf level, there must have been an insert or delete). This shows the theorem for shares and root operations.

The proof of the third statement is left as an exercise (hint: allow fractional coins, and change the invariant like you did when proving the third statement in the previous theorem, but now for all nodes (not just the one on the leaf level), then follow the reasoning from the proof of the second statement in the current theorem). □