HyperChromatic trees: a fine-grained approach to distributed algorithms on RedBlack trees*

Xavier Messeguer and Borja Valles

Universitat Politècnica de Catalunya. Dep. de Llenguatges i Sistemes Informàtics
Campus Nord–Mòdul C6. c/ Jordi Girona Salgado, 1–3.
08034 Barcelona, Spain

Abstract. We introduce a relaxed version of RedBlack trees. As concurrent algorithms on balanced search trees are nowadays based on local rules, we propose a set of fine-grained local rules that take more advantage of concurrency than previous approaches. Based on them we design a rebalancing concurrent algorithm and prove its correctness. Finally we sketch how to complete this algorithm to include concurrent insertions and deletions.

Keywords: Concurrent algorithms, RedBlack trees, Concurrent rebalancing, Safety and liveness proofs, Local rules.

1 Motivation

RedBlack trees are balanced search trees that have been recognized as an important data structure to deal with Dictionaries and Data Bases [CLR90]. Our aim is to deal with RedBlack trees in a concurrent environment. In this section we survey the previous approaches to this problem and outline our contributions.

In a concurrent approach, algorithms are designed for a shared-memory asynchronous parallel architecture. The first ones were designed with processes that locked an unbounded number of nodes, which prevented other processes from accessing to the involved data. This drawback was overcome with the creation of different types of locks, but these attempts have often resulted in complex descriptions with many subtle details to be mastered, so proving correctness became hardly possible.

The design of concurrent algorithms on trees is nowadays based on processes controlled by *local rules* because its temporal and spatial atomicity allows a very high degree of concurrence. A *local rule* is a set of fixed and small number of assignments and tests (*temporal atomicity*) which access to a fixed and small number of neighbour nodes of the tree (*spatial atomicity*). In this approach, rules are applied asynchronously in any order and to any node they are applicable to. The termination of algorithms is considered in a distributed form: they remain not active when no rule applies.

* This work has been partially supported by ESPRIT LTR Project no. 20244 — ALCOM-IT and CICYT Project TIC97-1475-CE and DGICYT under grant PB95-0787 (project KOALA).

Algorithms following this approach, denoted "on-the-fly", were first proposed by E.W. Dijkstra *et al.* to design a garbage collection algorithm [DLM+78] and, later on, were first applied on balanced search trees by J.L. Kessels [Kes83]. Recent applications of this approach on RedBlack trees can be found in the works of O. Nurmi *et al.* [NSS96] and J. Boyar *et al.* [BFL97] who introduced Chromatic trees, and J. Gabarró *et al.* [GMR97] who introduced HyperRed-Black trees. Both were relaxed versions of RedBlack trees.

In this paper we introduce HyperChromatic trees, a more relaxed structure that supports "on-the-fly" concurrent algorithms on RedBlack trees. The advantages of our approach are the following:

1. Our local rules are an extension of the rules of the sequential algorithms, therefore concurrent algorithms based on these rules emulate their sequential counterparts. Also those algorithms are more fine-grained than the already published ones because our rules emulate those proposed for Chromatic trees and HyperRed-Black trees.
2. It takes more advantage of concurrency than Chromatic trees when many new keys are attached simultaneously because our rules can be applied to any node that violates a red-black property.
3. It improves on HyperRed-Black trees by addressing to the deletions of keys.
4. Deletion algorithms based on our rules remove interior nodes and leaves, meanwhile the deletion algorithms of Chromatic trees, as they are *leaf-oriented*, do not remove interior nodes and only remove the leaves.

We prove the total correctness of the rebalancing concurrent algorithm, but we do not have a tight upper bound of the number of rules needed to rebalance a tree; nevertheless, there are two facts that suggest a good performance. First, the bounds in [BFL97] can be inherited because their rules can be emulated in some sense by our rules. Second, experimental data in [GMR97] suggest that the cost of rebalancing a tree is $O(n)$, being n the size of the tree.

The paper is organized as follows. In section 2 we recall RedBlack trees and we abstract some local rules from their sequential algorithms. In section 3 we introduce HyperChromatic trees and the local rules for the rebalancing algorithm. Section 4 contains the proof of correctness. Finally, section 5 contains the conclusions and some extensions.

2 Introduction

RedBlack trees are binary search trees whose nodes have an extra binary field called **color**. We say that a node n is *black* iff $\mathbf{color}(n) = 1$ and is *red* iff $\mathbf{color}(n) = 0$. The trees satisfy the following red-black properties:

P_1 : Every node is either red or black.
P_2 : Every leaf (NIL) is black.
P_3 : If a node is red then both its children are black. This is equivalent to, no path from the root to a leaf contains two consecutive red nodes.

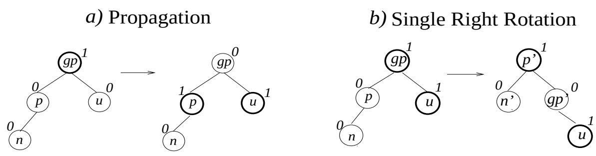

Fig. 1. Rules of RedBlack trees. Red nodes have color 0 and black nodes color 1.

P_4 : Every simple path from a given node to a leaf has the same sum of colors.

In other words, P_4 means that every simple path from a given node to a leaf contains the same number of black nodes.

Sequential algorithms: we consider the insertion algorithm (the deletion algorithm can get a similar treatment). It works in two phases:

1. *Search phase:* The key to be inserted falls down until it is attached to a new red node n at the bottom of the tree. As n is red, properties P_1, P_2 and P_4 are maintained.
2. *Rebalancing phase:* If the *parent* of n is black then P_3 holds and the insertion is over. Otherwise, n and its parent are red and P_3 is false. In this case a bottom-up *rebalancing* part starts (note that in this case the grandparent of n is black by P_3).

Let us describe the rules needed in the rebalancing phase. We denote the involved nodes by their relationship with n: *parent, uncle* and *grandparent*. When necessary we shorten them by p, u and gp. Assume that n and p are red. If node u is red then the redness of n and u is propagated "one step up" to gp redding it (see Figure 1.a), and so on if the parent of gp is red. Otherwise u is black and in this case a rotation around node gp rotates node u down and n up; property P_3 is restored (see Figure 1.b for the single right rotation case). Therefore the rebalancing phase of the sequential algorithm is composed by a sequence of propagations followed, if necessary, by a rotation.

Recall that the rebalancing part of the deletion algorithm also considers rules that propagate one extra black unity located on a black node. We assign color 2 to these nodes and we denote them as *hyper-black* nodes.

Concurrent algorithms: When insertions and deletions are performed concurrently two extreme situations may appear (Figure 2). If many keys are inserted in the same neighborhood then a cluster of red nodes can be created. In the same way, when many keys are deleted in the same area a cluster of hyper-black nodes is created. Clearly, properties P_1 and P_3 do not hold and the resulting tree can be very unbalanced. On the other hand, properties P_2 and P_4 are not violated, then we consider them as invariants of our approach.

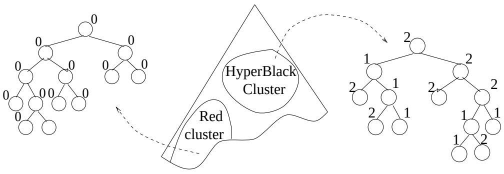

Fig. 2. Red and black clusters in concurrent RedBlack trees.

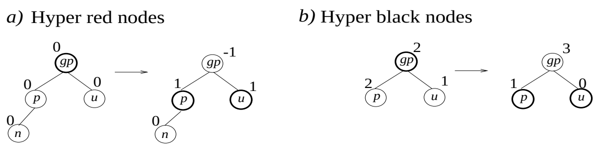

Fig. 3. Propagation rules to deal with clusters.

The question that arises is the following: can we find a set of local rules that turn this kind of trees to RedBlack trees and that can be applied to any node? In previous approaches the answer is negative. In Chromatic trees rules cannot be applied in the middle of a red cluster (their red propagation rule needs a black grandparent, so red clusters only can be dealt with at their top), and HyperRed-Black trees do not consider hyper-black clusters.

We propose a set of rules that give a positive answer to that question. For red nodes we allow the propagation of redness up although the parent is red (see Figure 3.*a*); this fact implies that nodes can be "overweighted" with red color and suggest us to define the *hyper-red* nodes with negative color. For *hyper-black* nodes (*ie.* nodes with color greater than 1) we allow the propagation of blackness up (see Figure 3.*b*). We may even have hyper-red nodes with hyper-black brothers; in this case we propose rotations which rotate hyper-black nodes down and hyper-red nodes up, as in the sequential case.

3 HyperChromatic trees

In this section we introduce HyperChromatic trees and their local rules. We define a *HyperChromatic tree*, HC-tree for short, as a binary search tree whose nodes have an additional integer field called *color* fulfilling these two properties:

Q_1: Every leaf has color 1.
Q_2: Every simple path from a given node to a leaf has the same sum of colors.

If we refer to the nodes whose color is 1 as black nodes and the nodes whose color is 0 as red nodes, we can see HC-trees as a color relaxed version of RedBlack

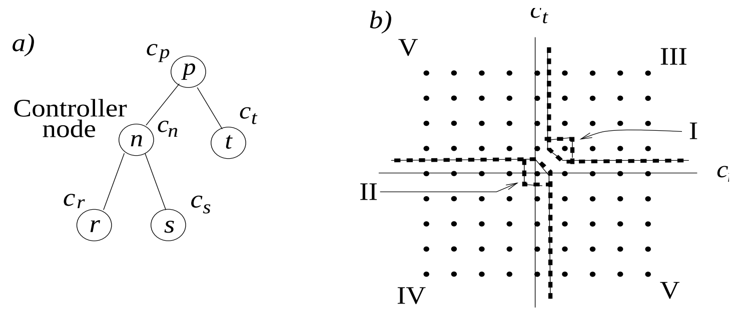

Fig. 4. *a)* Area defined by controller node. *b)* Cases of rule applications.

trees because Q_1 and Q_2 are the red-black properties P_2 and P_4. Moreover, a RedBlack tree is a HC-tree with these two red-black properties:

P_1: All the nodes are red or black.
P_3: If a node is red then both its children are black.

As we can locally test when an HC-tree is a RedBlack tree, the termination of the rebalancing algorithm is distributed.

HC-trees are a color relaxed version of Chromatic trees and HyperRed-Black trees. We sometimes refer to the nodes whose color is bigger than 1 as hyper-black nodes (these are called "overweighted" black nodes in Chromatic trees), and the nodes whose color is smaller than 0 as hyper-red nodes (as in HyperChromatic trees).

Our goal is to show that it is possible to get a RedBlack tree from a HC-tree just by concurrently applying a finite number of local rules. That done, it will be easy to complete a scheme of insertions and deletions for concurrent dictionaries implemented with HC-trees.

Definition of rules: A rule operates on a small cluster of nodes (Figure 4.*a*), comprised by one node n that acts as the controller, its brother t, its parent p and sometimes some of its children r, s and grandchildren. Each node is accompanied by its color.

We define the behavior of the rules:

- **Red Propagation** RP(n): moves up one negative color unit from the controller node and its brother to their father (Figure 5.*a*).
- **Black Propagation** BP(n): moves up one positive color unit from the controller node and its brother to their father (Figure 5.*b*).
- **Single Rotation** SR(n): rebalances the nodes when we suspect the left subtree of n is bigger or equal to the right one (Figure 5.*c*).
- **Double Rotation** DR(n): the same thing when we suspect the left subtree of n is smaller than the right one (Figure 5.*d*).

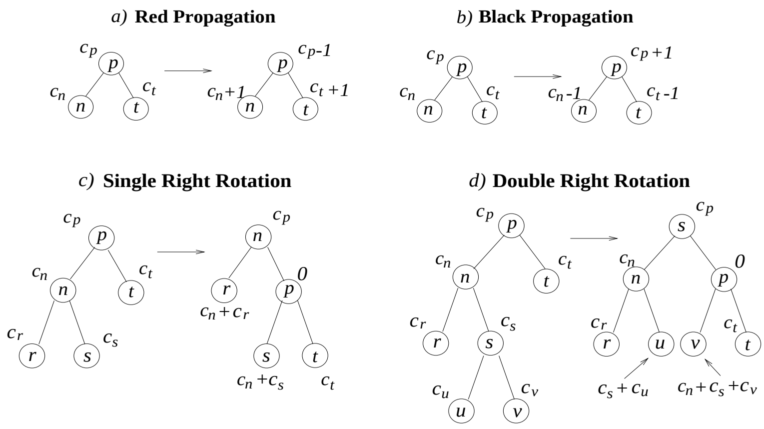

Fig. 5. Rules of HyperChromatic trees.

Application of rules: Each rule is applied with reference to one node. In most cases, but not always, this will be the controller node. Now we study every possible case and select the appropriated rules (Figure 4.b).

I. If $c_n = 1$ and $c_t = 1$ we are in a valid RedBlack tree situation and we do nothing.
II. If $c_n = 0$ and $c_t = 0$ and some child of n or t has zero or negative color we apply a **Red Propagation** on n to avoid the redness accumulation. Otherwise all children of n and t have positive color and we do nothing.
III. If $c_n \geq 1$ and $c_t \geq 1$, excluding case I, we apply a **Black Propagation** on n. Then one unit of blackness of n and t is moved up.
IV. If $c_n \leq 0$ and $c_t \leq 0$, excluding case II, we apply a **Red Propagation** on n. Then one unit of redness of n and t is moved up.
V. If $c_n \leq 0$ and $c_t > 0$ or its symmetric case, $c_n > 0$ and $c_t \leq 0$, we have several subcases. Assume that $c_n \leq 0$. In order to rebalance the tree we need to study the controller's children r and s. The only local information we have is their color so all we can guess is that the node with the smallest color is the root for the biggest subtree (recall that redness suggests insertions and blackness suggests deletions). Then, we need the rule to rotate up the biggest subtree and rotate down the smallest one.
We achieve that by using a **Single Right Rotation** when $c_r \leq c_s$ and a **Double Right Rotation** when $c_r > c_s$. As a special case, if both c_r and c_s are positive and bigger than $|c_n|$ we apply **Black Propagation** on r because some rotations may lead to a loop.

Note that these rules cannot be applied to the root of the tree. Then the color units received from its children remain there. In order to get a proper RedBlack tree, we need one last rule to normalize the root, forcing it to be red or black.

4 Correctness

The correctness is ensured by the *safety* and the *liveness* properties. The *safety* property guarantees that, whenever we start from a HC-tree, any tree obtained through the rules is fine: nothing bad may happen. The *liveness* property ensures that the sequence of rules is finite.

Safety: By induction on the set of rules we obtain the following lemma:

Lemma 1. *The set of rules transforms HC-trees into HC-trees. Moreover, if no rule applies, the tree is a RedBlack tree.*

Liveness: This property is demonstrated by the strict decrease of a function, denoted *variant function*, at each rule application. In this problem, the decrease is difficult to prove because the rotation rules seem to undo previous red and black propagations. Note that both propagation rules move up their respective color meanwhile rotation rules rotate down the blackness and even may move down the redness of some nodes.

In HC-trees the sum of colors of every path from the root to a leaf is the same and by lemma 1 it remains constant after the application of each rule. Assume an initial tree with root of color -1 and two leaves. Then, if a tree has grown by attaching red nodes to that initial tree, such sum is always 0 and the root is always red or hyper-red (unless we normalize it).

To get a better understanding about the meaning of the color of nodes we assume that some nodes have been attached to an initial tree. As in the sequential algorithm, these new nodes are colored red. The rules send their redness up until it disappears or it reaches the root of the tree. But note that when a red node sends its redness up it turns black, therefore this black unit forms a pair with the red unit moved up. By induction, each red unit of an hyper-red node n has a black counterpart at any path from n to its leaves that cancel that redness. Likewise, if n is black or hyper-black, there are as many units of redness into the path from n to the root of the tree as necessary to cancel its blackness.

This distribution of color units along a path can be interpreted as pairs of open-closed brackets: negative red units are open brackets, and black units are closed brackets, and as we are dealing with trees (not with lists) each open bracket has a closed counterpart at any descendent path. Therefore the colors located at any path of the tree can be interpreted as mathematical bracket expressions. Then it is clear that a prefix has more open brackets than closed brackets (see following lemma), so it can be determined a priority (absolute level) between pairs of brackets defining the priority of outermost packets as 0 and so on.

We denote $S(n)$ as the sum of color of nodes belonging to the path from the root to node n, but not including n. As a consequence,

Lemma 2. *It holds $S(n) \leq 0$ for any node and $S(leaf) = -1$.*

Variant function: $\mathcal{V}$ **:** It is a sequence of items composed by the following three functions I_i, $\#P_i$ and C_i ($\#$ means the number of elements of a set). Namely, in a given step t of the algorithm we can consider the following array:

$$\mathcal{V}(t) = (\ [I_0, \#P_0, C_0],\ [I_1, \#P_1, C_1],\ \dots,\ [I_i, \#P_i, C_i],\ \dots\ )$$

Given two arrays $\mathcal{V}$ and $\mathcal{V}'$, we say that $\mathcal{V} < \mathcal{V}'$ if there is an index $i > 0$ such that for all $j < i\ :\ I_j = I'_j,\ \#P_j = \#P'_j,\ C_j = C'_j$ and

- $I_i < I'_i$,
- or $I_i = I'_i$ and $\#P_i < \#P'_i$,
- or $I_i = I'_i$ and $\#P_i = \#P'_i$ and $C_i < C'_i$.

Definition of I_i **:** Given a red or hyper-red node n with color c_n (with $c_n \leq 0$) we split c_n into $|c_n| + 1$ *units of red* having *local labels* $\{0, -1, \dots, c_n\}$. We determine the set of *absolute levels* of a red or hyper-red node n as

$$L_r(n) = \{\ |S(n)|,\ |S(n) - 1|,\ \dots,\ |S(n) + c_n|\}.$$

Any given node n can be considered as the root of a subtree. Then we call $Inside(n)$ the number of nodes in such a tree. We determine the set of nodes of absolute level i that accept rotations

$$R_i = \{n \mid \mathbf{color}(n) \leq 0,\ i \in L_r(n),\ \mathbf{color}(\mathbf{brother}(n)) \geq 1\}.$$

We define

$$I_i = \sum_{n \in R_i} Inside(n)$$

Definition of P_i **:** Given a red or hyper-red node n with color c, we can determine its set of absolute levels $L(n)$. We define the set of red nodes with absolute level i:

$$P_i = \{n \mid \mathbf{color}(n) \leq 0,\ i \in L_r(n)\}.$$

Then a red or hyper-red node n belongs to all the sets P_i with i such that $|S(n)| \leq i \leq |S(n) + c_n|$. If t is the brother of n then $S(n) = S(t)$ and if t is red or hyper-red, then it also belongs, at least, to $P_{|S(n)|}$.

Definition of C_i **:** For black or hyper-black nodes with color c, we split c into c color units $\{1, 2, \dots, c\}$. We determine the set of *absolute levels* of a black or hyper-black node n as

$$L_b(n) = \{|S(n) + 1|,\ \dots,\ |S(n) + c_n|\}.$$

We define

$$W_i = \{n \mid \mathbf{color}(n) > 0,\ i \in L_b(n)\}.$$

Then a black or hyper-black node n belongs to all the sets W_i with i such that $|S(n) + 1| \leq i \leq |S(n) + c_n|$. If t is the brother of n then $S(n) = S(t)$ and if it is black or hyper-black, then it also belongs, at least, to $W_{|S(n)+1|}$.

Now, for each set W_i we denote as C_i the sum of the color of its nodes

$$C_i = \sum_{n \in W_i} \mathbf{color}(n).$$

Lemma 3. *Any application of a propagation or a rotation rule strictly decreases the function $\mathcal{V}(t)$.*

Proof. Let us study each rule separately. Remember the notation of n for the controller node, and p, t, r and s for its parent, brother and children.

• **Red Propagation**: Recall that n and t are red or hyper-red and that if $i = |S(n)|$ then P_i is the set with the smallest index such that n and t belong to.

After the propagation $\mathbf{color}(n)$ and $\mathbf{color}(t)$ increase by one unit, so n and t no longer belong to P_i (they belong at least to P_{i+1}), so $\#P_i$ decreases. The previous components of $\mathcal{V}$ remain unchanged because nodes of levels smaller than i are not involved in the rule, and neither does I_i, because R_i does not gain any element.

• **Black Propagation**: Recall that n and t are black or hyper-black and that if $i = |S(n)|$ then W_i is the set with the greatest index such that n and t belong to. Besides, at least one of them belong to W_{i-1}.

After the propagation $\mathbf{color}(n)$ and $\mathbf{color}(t)$ decrease by one unit, so they no longer belong to W_i and, what is more important they still belong to W_{i-1} but as their colors decrease so does C_{i-1}. The previous components of $\mathcal{V}$ remain unchanged because nodes of levels smaller than $i-1$ are not involved in the rule, and neither do I_{i-1} and $\#P_{i-1}$ because no red nodes of level $i-1$ are involved.

• **Single Rotation**: Recall that $c_n \leq 0$ and $c_t > 0$ (or the symmetric situation). and that then we perform a single rotation only if $c_r \leq c_s$. Finally, suppose that both c_r and c_s are negative (the other cases are simpler than this one).

Assume $j = S(r)$, then P_j is the set with smallest index that r belongs to and if $i = S(n)$ then P_i is the set with smallest index that n belongs to. Besides, as n is the parent of r, we have $j = i + |c_n|$.

After the rotation the color of r is $c_n + c_r$ so now P_i is the set with the smallest index that r belongs to. As p becomes red and so remains s, then s also belongs to P_i. On the other hand, n no longer belongs to P_i (it still is the parent of r). Therefore I_i increases by $Inside(s)$ (its brother t is black), but decreases by $Inside(n)$ which is bigger because n is the parent of s.

The previous components of $\mathcal{V}$ remain unchanged because nodes of levels smaller than i are not involved in the rule.

• **Double Rotation**: Recall that $c_n \leq 0$ and $c_t > 0$ (or the symmetric situation) and that then we perform a double rotation only if $c_r > c_s$.

Assume $i = S(n)$, then P_i is the set with smallest index that n belongs to. In this case we just notice that after the rotation R_i loses n (and maybe s) and at most it gains u and v. As these two nodes are grandchildren of n the sum of their $Inside$ is smaller than $Inside(n)$, so I_i decreases. The previous components of $\mathcal{V}$ remain unchanged because nodes of levels smaller than i are not involved in the rule. □

The normalization of the root does not change the variant function $\mathcal{V}(t)$. However, this variant function can be easily modified to take care of this case. Therefore the proposed algorithm is totally correct.

5 Conclusions and extensions

We have introduced a color relaxed version of RedBlack trees, called HyperChromatic trees, that extends both Chromatic trees [NSS96,BFL97] and HyperRed-Black trees [GMR97]. We allow the color field to have any integer value; in this sense we can say that they are the most color relaxed trees.

We have defined a set of local rules for the concurrent rebalancing algorithm that are an extension of the rules of sequential algorithms. They are more fine-grained than the ones in the previous approaches because they can emulate them.

Our approach takes more advantage of concurrency because for each violation of any red-black property there is a rule that can be applied. Previous approaches do not have rules to be applied to some cases. For instance, in Chromatic trees there are no rules to apply inside clusters of red nodes.

The analysis of rule applications takes into account two nodes; in our case we consider the set $\mathbb{Z} \times \mathbb{Z}$ of possible cases (see Figure 4) meanwhile previous approaches only consider the values of one quadrant, which leads them to more constrained rules.

We have designed a rebalancing concurrent algorithm with its proof of correctness. The extension of this algorithm in order to allow insertions and deletions can be made with a few additional rules (see for instance [BFL97]). For insertions we need a rule that attaches nodes at the bottom of the tree. For deletions, the actions performed by the new rules should be: to mark interior nodes to be removed, to rotate them down and to unattach them. All these rules can be easily designed and the variant function only needs a simple extension.

References

[BFL97] J. Boyar, R. Fagerberg, and K. Larsen. Amortization results for chromatic search trees, with an application to priority queues. *Journal of Computer and System Science*, (55):504–521, 1997.

[CLR90] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. Mc-Graw Hill, MIT, 1990.

[DLM+78] E.W. Dijkstra, L. Lamport, A.J. Martin, C.S. Scholten, and E.F.M. Steffens. On-the fly garbage collection: an exercise in cooperation. *CACM*, 21:966–975, 1978.

[GMR97] J. Gabarró, X. Messeguer, and D Riu. Concurrent rebalancing on HyperRed-Black trees. In IEEE Computer Society, editor, *Proc. of XVII International Conference of the Chilean Computer Society*, pages 93–104, 1997.

[Kes83] J.L.W. Kessels. On-the-fly optimization of data structures. *CACM*, 26(11):895–901, 1983.

[NSS96] O. Nurmi and E. Soisalon-Soininen. Chromatic binary search trees: A structure for concurrent rebalancing. *Acta Informatica*, 33(6):547–557, 1996. Also appeared in 10th ACM PODS, 1991.