A New Weight Balanced Binary Search Tree[1]

Seonghun Cho and Sartaj Sahni

Department of Computer and Information Science and Engineering

University of Florida

Gainesville, FL 32611, U.S.A.

Technical Report 96-001

Abstract

We develop a new class of weight balanced binary search trees called β-balanced binary search trees (β-BBSTs). β-BBSTs are designed to have reduced internal path length. As a result, they are expected to exhibit good search time characteristics. Individual search, insert, and delete operations in an n node β-BBST take $O(\log n)$ time for $0 < \beta \leq \sqrt{2}-1$. Experimental results comparing the performance of β-BBSTs, WB(α) trees, AVL-trees, red/black trees, treaps, deterministic skip lists and skip lists are presented. Two simplified versions of β-BBSTs are also developed.

Keywords and Phrases. data structures, weight balanced binary search trees

1 Introduction

A dictionary is a set of elements on which the operations of search, insert, and delete are performed. Many data structures have been proposed for the efficient representation of a dictionary [HORO94]. These include direct addressing schemes such as hash tables and comparison schemes such as binary search trees, AVL-trees, red/black trees [GUIB78], trees of bounded balance [NIEV73], treaps [ARAG89], deterministic skip lists [MUNR92], and skip lists [PUGH90]. Of these schemes, AVL-trees, red/black trees, and trees of bounded balance (WB(α)) are balanced binary search trees. When representing a dictionary with n elements, using one of these schemes, the corresponding binary search tree has height $O(\log n)$ and individual search, insert, and delete operations take $O(\log n)$ time. When (unbalanced)

[1]This research was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111, and by the National Science Foundation under grant MIP91-03379.

binary search trees, treaps, or skip lists are used, each operation has an expected complexity of $O(\log n)$ but the worst case complexity is $O(n)$. When hash tables are used, the expected complexity is $O(1)$ per operation. However, the worst case complexity is $O(n)$. So, in applications where a worst case complexity guarantee is critical, one of the balanced binary search tree schemes is to be performed.

In this paper, we develop a new balanced binary search tree called β-BBST (β-balanced binary search tree). Like $WB(\alpha)$ trees, this achieves balancing by controlling the relative number of nodes in each subtree. However, unlike $WB(\alpha)$ trees, during insert and delete operations, rotations are performed along the search path whenever they reduce the internal path length of the tree (rather than only when a subtree is out of balance). As a result, the constructed trees are expected to have a smaller internal path length than the corresponding $WB(\alpha)$ tree. Since the average search time is closely related to the internal path length, the time need to search in a β-BBST is expected to be less than that in a $WB(\alpha)$ tree.

In Section 2, we define the total search cost of a binary search tree and show that the rebalancing rotations performed in AVL and red/black trees might increase this metric. We also show that while similar rotations in $WB(\alpha)$ trees do not increase this metric, insert and delete operations in $WB(\alpha)$ trees do not avail of all opportunities to reduce the metric. In Section 3, we define β-BBSTs and show their relationship to $WB(\alpha)$ trees. Search, insert, and delete algorithms for β-BBSTs are developed in Section 4. A simplified version of β-BBSTs is developed in Section 5. Search, insert and delete operations for this version also take $O(\log n)$ time each. An even simpler version of β-BBSTs is developed in Section 6. For this version, we show that the average cost of an insert and search operation is $O(\log n)$ provided no deletes are performed.

An experimental evaluation of β-BBSTs and competing schemes for dictionaries (AVL, red/black, skip lists, etc.) was done and the results of this are presented in Section 7. This section also compares the relative performance of β-BBSTs and the two simplified versions of Sections 5 and 6.

2 Balanced Trees and Rotations

Following an insert or delete operation in a balanced binary search tree (e.g., AVL, red/black, WB(α), etc.), it may be necessary to perform rotations to restore balance. The rotations are classified as LL, RR, LR, and RL [HORO94]. LL and RR rotations as well as LR and RL rotations are symmetric. While the conditions under which the rotations are performed vary with the class of balanced tree considered, the node movement patterns are the same. Figure 1 shows the transformation performed by an LL and an LR rotation. In this figure, nodes whose subtrees have changed as a result of the rotation are designated by a prime. So, p' is the original node p however its subtrees are different.

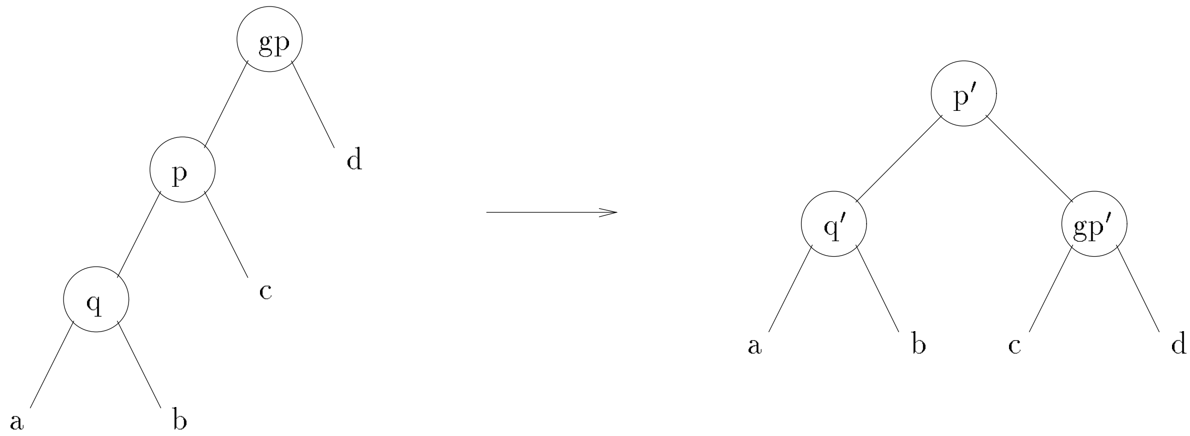
Let $h(x)$ be the height of the subtree with root x. Let $s(x)$ be the number of nodes in this subtree. When searching for an element x, x is compared with one element at each of $l(x)$ levels, where $l(x)$ is the level at which x is present (the root is at level 1). So, one measure of the "goodness" of the binary search tree, T, for search operations (assuming each element is searched for with equal probability) is its total search cost defined as:

$$C(T) = \sum_{x \in T} l(x).$$

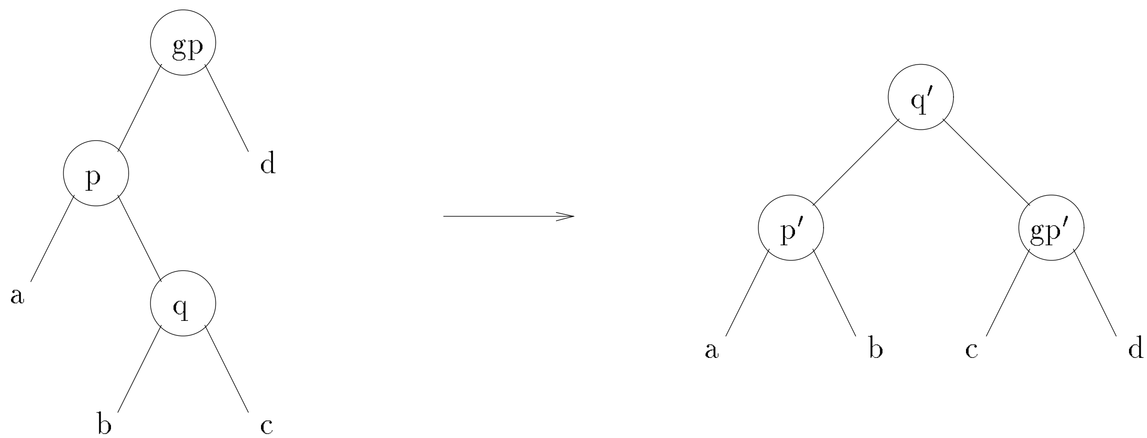
Notice that $C(T) = I(T) + n$ where $I(T)$ is the internal path length of T and n is the number of elements/nodes in T. The cost of unsuccessful searches is equal to the external path length $E(T)$. Since $E(T) = I(T) + 2n$, minimizing $C(T)$ also minimizes $E(T)$.

Total search cost is important as this is the dominant operation in a dictionary (note that insert can be modeled as an unsuccessful search followed by the insertion of a node at the point where the search terminated and deletion can be modeled by a successful search followed by a physical deletion; both operations are then followed by a rebalancing/restructuring step).

Observe that in an actual implementation of the search operation in programming languages such as C++, C, and Pascal, the search for an x at level $l(x)$ will involve upto two comparisons at levels $1, 2, \ldots, l(x)$. If the code first checks $x = e_i$ where e_i is the element



(a) LL rotation



(b) LR rotation

Figure 1: LL and RL rotations

at level i to be compared and then $x < e_i$ to decide whether to move to the left or right subtree, then the number of element comparisons is exactly $2l(x) - 1$. In this case, the total number of element comparisons is

$$NC(T) = 2\sum_{x \in T} l(x) - n = 2C(T) - n$$

and minimizing $C(T)$ also minimizes $NC(T)$. If the code first checks $x < e_i$ and then $x = e_i$ (or $> e_i$), the number of element comparisons done to find x is $l(x)+r(x)+1$ where $r(x)$ is the number of right branches on the path from the root to x. The total number of comparisons is bounded by $2C(T)$. For simplicity, we use $C(T)$ to motivate our data structure.

In an AVL tree, when an LL rotation is performed, $h(q) = h(c) + 1 = h(d) + 1$ (see Figure 1(a)). At this time, the balance factor at gp is $h(p) - h(d) = 2$. The rotation restores height balance which is necessary to guarantee $O(\log n)$ search, insert, delete operations in an n node AVL tree. The rotation may, however, increase the total search cost. To see this, notice that an LL rotation affects the level numbers of only those nodes that are in the subtree with root gp prior to the rotation. We see that $l(q') = l(q) - 1$, $l(p') = l(p) - 1$, $l(gp') = l(gp) + 1$, the total search cost of the subtree with root a is decreased by $s(a)$ as a result of the rotation, etc. Hence, the increase in $C(T)$ due to the rotation is:

$$l(p') - l(p) + l(q') - l(q) + l(gp') - l(gp) - s(a) - s(b) + s(d)$$

$$= -1 - 1 + 1 - s(q) + 1 + s(d) = s(d) - s(q).$$

A similar analysis shows that an LR rotation increases $C(T)$ by $s(d) - s(q)$.

If the LL rotation was triggered by an insertion, $s(q)$ is at least one more than the minimum number of nodes in an AVL tree of height $t = h(q) - 1$. So, $s(q) \geq \phi^{t+2}/\sqrt{5}$ where $\phi = (1 + \sqrt{5})/2$. The maximum value for $s(d)$ is $2^t - 1$. So, an LL rotation has the potential of increasing total search cost by as much as

$$2^t - 1 - \phi^{t+2}/\sqrt{5} \approx 2^t - 1 - 1.62^{t+2}/2.24.$$

This is negative for $t \leq 2$ and positive for $t > 2$. When $t = 10$, for example, an LL rotation may increase total search cost by as much as 877. As t gets larger, the potential increase in search cost gets much greater. This analysis is easily extended to the remaining rotations and also to red/black trees.

Definition (WB(α) [NIEV73]) The balance, $B(p)$, of a node p in a binary tree is the ratio $(s(l) + 1)/(s(p) + 1)$ where l is the left child of p. For $\alpha \in [0, 1/2]$, a binary tree T is in WB(α) iff $\alpha \leq B(p) \leq 1 - \alpha$ for every node p in T. By definition, the empty tree is in WB(α) for all α.

Lemma 1 *(1) The maximum height, hmax(n), of an n node tree in WB(α) is* $\sim \log_{\frac{1}{1-\alpha}}(n+1)$ *[NIEV73]*

(2) Inserts and deletes can be performed in an n node tree in WB(α) in $O(\log n)$ time for $2/11 < \alpha \leq 1 - \sqrt{2}/2$ [BLUM80].

(3) Each search operation in an n node tree in WB(α) takes $O(\log n)$ time [NIEV73].

In the case of weight balanced trees WB(α), an LL rotation is performed when $B(gp) \approx 1-\alpha$ and $B(p) \geq \alpha/(1-\alpha)$ (see Figure 1(a)) [NIEV73]. So,

$$1-\alpha \approx \frac{s(p)+1}{s(gp)+1} = \frac{s(p)+1}{s(p)+s(d)+2}$$

or

$$s(d) \approx s(p)\frac{\alpha}{1-\alpha} + \frac{2\alpha-1}{1-\alpha}$$

and

$$\frac{\alpha}{1-\alpha} \leq B(p) = \frac{s(q)+1}{s(p)+1}$$

or

$$s(q) \geq s(p)\frac{\alpha}{1-\alpha} + \frac{2\alpha-1}{1-\alpha}.$$

So, LL rotations (and also RR) do not increase the search cost. For LR rotations [NIEV73], $B(gp) \approx 1-\alpha$ and $B(p) < \alpha/(1-\alpha)$. So, $s(d) \approx s(p)\frac{\alpha}{1-\alpha} + \frac{2\alpha-1}{1-\alpha}$ and with respect to Figure 1(b),

$$\frac{\alpha}{1-\alpha} > B(p) = \frac{s(p)-s(q)}{s(p)+1}$$

or

$$s(q) > s(p)\frac{1-2\alpha}{1-\alpha} - \frac{\alpha}{1-\alpha}.$$

For $\alpha \leq 1/3$, $s(q) \geq s(d)$ and LR (RL) rotations do not increase search cost. Thus, in the case of WB(α) trees, the rebalancing rotations do not increase search cost. This statement remains true if the conditions for LL and LR rotation are changed to those in [BLUM80].

While rotations do not increase the search cost of WB(α) trees, these trees miss performing some rotations that would reduce search cost. For example, it is possible to have

$\alpha < B(gp) < 1-\alpha$, $B(p) \geq \frac{\alpha}{1-\alpha}$, and $s(q) > s(d)$. Since $B(gp)$ isn't high enough, an LL rotation isn't performed. Yet, performing such a rotation would reduce search cost.

3 β-BBSTs

Definition A *cost optimized search tree* (COST) is a binary search tree whose search cost cannot be reduced by performing a single LL, RR, LR, or RL rotation.

Theorem 1 *If T is a COST with n nodes, its height is at most* $\log_\phi(\sqrt{5}(n+1)) - 2$.

Proof Let N_h be the minimum number of nodes in a COST of height h. Clearly, $N_0 = 0$ and $N_1 = 1$. Consider a COST Q of height $h \geq 2$ having the minimum number of nodes N_h. Q has one subtree R whose height is $h-1$ and another, S, whose height is $\leq h-1$. R must be a minimal COST of height $h-1$ and so has N_{h-1} nodes. R, in return, must have one subtree, U, of height $h-2$ and another, V, of height $\leq h-2$. Both U and V are COSTs as R is a COST. Since R is a minimal COST, U is a minimal COST of height $h-2$ and so has N_{h-2} nodes. Since Q is a COST, $|S| \geq max\{|U|, |V|\}$. We may assume that N_h is a nondecreasing function of h. So, $|S| \geq N_{h-2}$. Since Q is a minimal COST of height h, $|S| = N_{h-2}$. So,

$$N_h = N_{h-1} + N_{h-2} + 1, \;\; h \geq 2$$

$$N_0 = 0, N_1 = 1.$$

This recurrence is the same as that for the minimum number of nodes in an AVL tree of height h. So, $N_h = F_{h+2} - 1$ where F_i is the i'th Fibbonacci number. Consequently, $N_h \approx \phi^{h+2}/\sqrt{5} - 1$ and $h \leq \log_\phi(\sqrt{5}(n+1)) - 2$. □

Corollary 1 *The maximum height of a COST with n nodes is the same as that of an AVL tree with this many nodes.*

Definition Let a and b be the root of two binary trees. a and b are β-balanced, $0 \leq \beta \leq 1$, with respect to one another, denoted β-(a, b), iff

$$\text{(a) } \beta(s(a) - 1) \leq s(b)$$

$$\text{(b) } \beta(s(b) - 1) \leq s(a)$$

A binary tree T is β-balanced iff the children of every node in T are β-balanced.

A full binary tree is 1-balanced and a binary tree whose height equals its size (i.e., number of nodes) is 0-balanced.

Lemma 2 *If the binary tree T is β-balanced, then it is γ-balanced for $0 \leq \gamma \leq \beta$.*

Proof Follows from the definition of balance. □

Lemma 3 *If the binary tree T is β-balanced, $0 \leq \beta \leq 1/2$, then it is in WB(α) for $\alpha = \beta/(1+\beta)$.*

Proof Consider any node p in T. Let l and r be node p's left and right children.

$$B(p) = \frac{s(l) + 1}{s(l) + s(r) + 2} = \frac{1}{1 + \frac{s(r)+1}{s(l)+1}}.$$

Since T is β-balanced, $s(l) - 1 \leq s(r)/\beta$ or $s(l) + 1 \leq s(r)/\beta + 2$. So,

$$\frac{s(l) + 1}{s(r) + 1} \leq 1/\beta + \frac{2\beta - 1}{\beta(s(r) + 1)} \leq 1/\beta$$

or

$$\frac{s(r) + 1}{s(l) + 1} \geq \beta.$$

So, $B(p) \leq 1/(1 + \beta)$. Further, $s(r) - 1 \leq s(l)/\beta$. So,

$$\frac{s(r) + 1}{s(l) + 1} \leq 1/\beta.$$

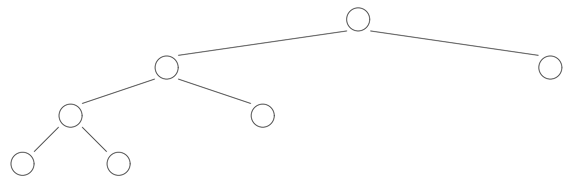


Figure 2: A tree in WB(1/4) that is not $\frac{1}{3}$-balanced

And, $B(p) \geq 1/(1+1/\beta) = \beta/(1+\beta)$. Hence $\beta/(1+\beta) \leq B(p) \leq 1/(1+\beta)$ for every p in T. So, T is in WB(α) for $\alpha = \beta/(1+\beta)$. □

Remark While every β-balanced tree, $0 \leq \beta \leq 1/2$, is in WB(α) for $\alpha = \beta/(1+\beta)$, there are trees in WB(α) that are not β-balanced. Figure 2 shows an example of a tree in WB(1/4) that is not $\frac{1}{3}$-balanced.

Lemma 4 *If T is a COST then T is $\frac{1}{2}$-balanced.*

Proof If T is a COST, then every subtree of T is a COST. Consider any subtree with root p, left child l, and right child r. If neither l nor r exist, then $s(l) = s(r) = 0$ and p is $\frac{1}{2}$-balanced. If $s(l) = 0$ and $s(r) > 1$, then r has a nonempty subtree with root t and $s(t) > s(l)$. So p is not a COST. Hence, $s(r) \leq 1$ and p is $\frac{1}{2}$-balanced. The same is true when $s(r) = 0$. So, assume $s(l) > 0$ and $s(r) > 0$.

If $s(l) = 1$, then $s(r) \leq 3$ as otherwise, one of the subtrees of r has $m \geq 2$ nodes and $m > s(l)$ implies p is not a COST. Since $s(r) \leq 3$, $\frac{1}{2}(s(r)-1) \leq s(l)$ and $\frac{1}{2}(s(l)-1) \leq s(r)$. So, p is $\frac{1}{2}$-balanced. The same proof applies when $s(r) = 1$. When $s(l) > 1$ and $s(r) > 1$, let a and b be the roots of the left and right subtrees of l. Since p is a COST, $s(a) \leq s(r)$ and $s(b) \leq s(r)$. So, $s(l) = s(a) + s(b) + 1 \leq 2s(r) + 1$ and $\frac{1}{2}(s(l)-1) \leq s(r)$. Similarly, $\frac{1}{2}(s(r)-1) \leq s(l)$. So, $\frac{1}{2}$-(l, r). Since this proof applies to every nodes in T, the children of every p are $\frac{1}{2}$-balanced and T is $\frac{1}{2}$-balanced. □

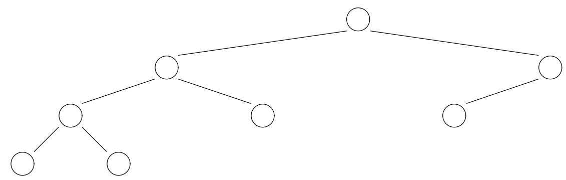


Figure 3: $\frac{1}{2}$-balanced tree that is not a COST

Remark There are $\frac{1}{2}$-balanced trees that are not COSTs (see Figure 3).

While a COST is in WB(1/3) and WB(α) trees can be maintained efficiently only for $2/11 < \alpha \leq 1 - 1/\sqrt{2} \approx 0.293$, a COST is better balanced than WB(α) trees with α in the usable range. Unfortunately, we are unable to develop O($\log n$) insert/delete algorithms for a COST.

In the next section, we develop insert and delete algorithms for β-balanced binary search trees (β-BBST) for $0 < \beta \leq \sqrt{2} - 1$. Note that every ($\sqrt{2} - 1$)-BBST is in WB($\alpha$) for $\alpha = 1 - 1/\sqrt{2}$ which is the largest permissible α. Since our insert and delete algorithms perform rotations along the search path whenever these result in improved search cost, BBSTs are expected to have better search performance than WB(α) trees (for $\alpha = \beta/(1+\beta)$).

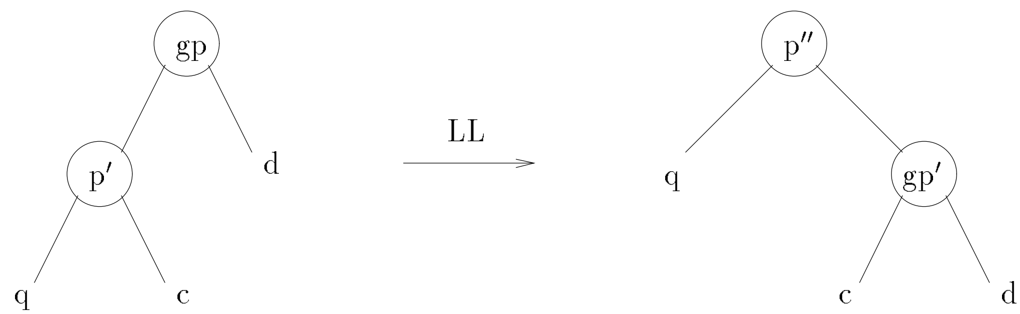
Each node of a β-BBST has the fields LeftChild, Size, Data, and RightChild. Since every β-BBST, $\beta > 0$, is in WB(α), for $\alpha > 0$, β-BBSTs have height that is logarithmic in n, the number of nodes (provided $\beta > 0$).

4 Search, Insert, and Delete in a β-BBST

To reduce notational clutter, in the rest of the paper, we abbreviate $s(a)$ by a (i.e., the node name denotes subtree size).

4.1 Search

This is done exactly as in any binary search tree. Its complexity is O(h) where h is the height of the tree. Notice that since each node has a size field, it is easy to perform a search



(a) before (b) after

Figure 4: LL rotation for insertion

based on index (i.e., find the 10'th smallest key). Similarly, our insert and delete algorithms can be adapted to indexed insert and delete.

4.2 Insertion

To insert a new element x into a β-BBST, we first search for x in the β-BBST. This search is unsuccessful (as x is not in the tree) and terminates by falling off the tree. A new node y containing x is inserted at the point where the search falls off the tree. Let p' be the parent (if any) of the newly inserted node. We now retrace the path from p' to the root performing rebalancing rotations.

There are four kinds of rotations LL, LR, RL, and RR. LL and RR rotations are symmetric and so also are LR and RL rotations. The typical configuration before an LL rotation is performed is given in Figure 4(a). p' denotes the root of a subtree in which the insertion was made. Let p be the (size of the) subtree before the insertion. Then, since the tree was a β-BBST prior to the insertion, β-(p, d). Also, for the LL rotation to be performed, we require that $(q \geq c)$ and $(q > d)$. Note that $q > d$ implies $q \geq 1$. We shall see that β-(q, c) follows from the fact that the insertion is made into a β-BBST and from properties of the rotation. Following an LL rotation, p' is updated to be the node p''.

Lemma 5 *[LL insertion lemma] If [β-$(p, d) \wedge \beta$-$(q, c) \wedge (q \geq c) \wedge (q > d)$] for* $0 \leq \beta \leq 1/2$

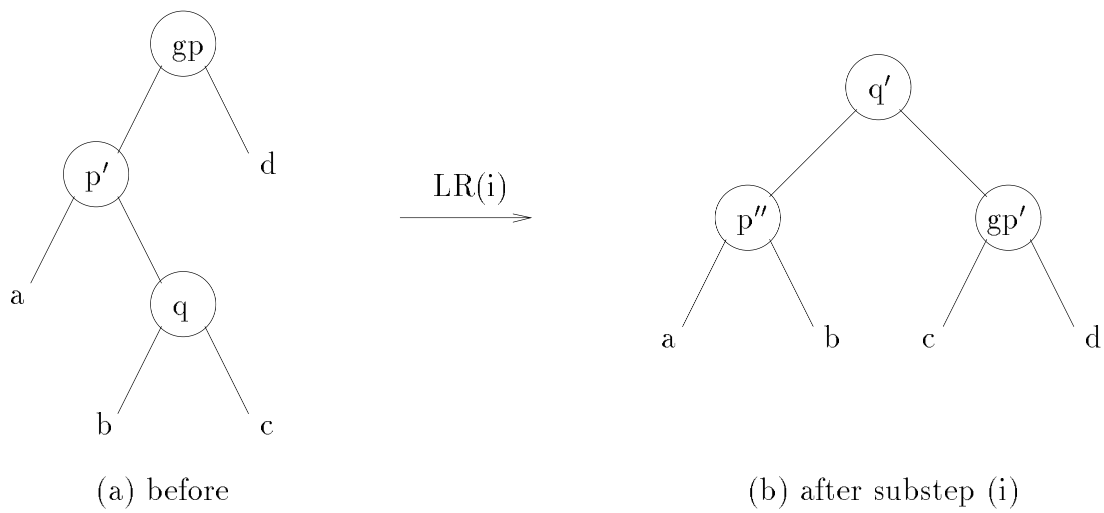


Figure 5: Substep (i) of insertion LR rotation

before the rotation, then β-(q, gp') *and* β-(c, d) *after the rotation.*

Proof Assume the before condition.

(a) $\beta(q-1) \le c$ (as β-(q,c)) $< gp'$. Also, $\beta(gp'-1) = \beta(c+d) \le 2\beta q$ (as $\beta \ge 0, q \ge c$ and $q > d$) $\le q$ (as $\beta \le 1/2$). So, β-(q, gp').

(b) $d < q \Rightarrow d-1 < q-1 \Rightarrow \beta(d-1) \le \beta(q-1) \le c$ (as β-(q,c)). Also, $\beta(c-1) \le \beta(q+c-1) = \beta(p'-2) = \beta(p-1) \le d$ (as β-(p,d)). So, β-(c,d). □

In an LR rotation, the before configuration is as in Figure 4(a). However, this time $q < c$. Figure 4(a) is redrawn in Figure 5(a). In this, the node labeled c in Figure 4(a) has been labeled q and that labeled q in Figure 4(a) has been labeled a. With respect to the labelings of Figure 5(a), rotation LR is applied when

$$[(q > a) \wedge (q > d)].$$

The other conditions that apply when an LR rotation is performed are

$$[\beta\text{-}(p,d) \wedge \beta\text{-}(a,q) \wedge \beta\text{-}(b,c)].$$

Here p denotes the (size of the) left subtree of gp prior to the insertion. An LR rotation is accomplished in two substeps (or two subrotations). The first of these is shown in Figure 5(b). Following an LR rotation, p' is updated to be node q'.

Lemma 6 *[LR substep(i) insertion lemma] If* $[\beta\text{-}(p,d) \wedge \beta\text{-}(a,q) \wedge \beta\text{-}(b,c) \wedge (q > a) \wedge (q > d)]$ *for* $0 \leq \beta \leq 1/2$ *before the subrotation, then* $[\beta\text{-}(p'', gp') \wedge \{(\beta\text{-}(a,b) \wedge \frac{\beta}{1+\beta}\text{-}(c,d)) \vee (\frac{\beta}{1+\beta}\text{-}(a,b) \wedge \beta\text{-}(c,d))\}]$ *after the subrotation.*

Proof Assume the before condition. First, we show that $\beta\text{-}(p'', gp')$ after the rotation. Note that $\beta(p''-1) = \beta(a+b) = \beta(a+b+c+1) - \beta(c+1) = \beta(p'-1) - \beta(c+1) = \beta(p-1) - \beta c \leq d - \beta c \leq d < gp'$. Also, $\beta(gp'-1) = \beta(c+d) \leq b + \beta + \beta d$ (as $\beta\text{-}(b,c)$) $\leq b + \beta q$ (as $q > d$) $\leq b + a + \beta$ (as $\beta\text{-}(a,q)$) $< p''$ (as $\beta \leq 1/2$ and $p'' = a+b+1$). So, $\beta\text{-}(p'', gp')$.

Next, we prove two properties that will be used to complete the proof.

P1: $\beta(b-1) \leq a$.

To see this, note that $\beta(b-1) \leq \beta(q-1) \leq a$ (as $\beta\text{-}(a,q)$).

P2: $\beta(c-1) \leq d$.

For this, observe that $p'-1 = a+q \geq \beta(q-1)+q$ (as $\beta\text{-}(a,q)$) $= (\beta+1)(q-1)+1$. So, $q-1 \leq \frac{p'-2}{\beta+1} = \frac{p-1}{\beta+1}$. Similarly, $q-1 = b+c \geq \beta(c-1)+c$ (as $\beta\text{-}(b,c)$) $= (\beta+1)(c-1)+1$. So, $\beta(c-1) \leq \frac{\beta}{\beta+1}(q-2) \leq \frac{\beta}{\beta+1}(q-1) \leq \frac{\beta(p-1)}{(\beta+1)^2} \leq \frac{d}{(\beta+1)^2}$ (as $\beta\text{-}(p,d)$) $\leq d$.

To complete the proof of the lemma, we need to show

$$\{(\beta\text{-}(a,b) \wedge \frac{\beta}{1+\beta}\text{-}(c,d)) \vee (\frac{\beta}{1+\beta}\text{-}(a,b) \wedge \beta\text{-}(c,d))\}.$$

We do this by considering the two cases $b \geq c$ and $b < c$.

Case $b \geq c$: Since $a < q = b+c+1, \beta(a-1) \leq \beta(b+c) \leq 2\beta b \leq b$. This and P1 imply $\beta\text{-}(a,b)$. Also, $d < q = b+c+1$. So, $\frac{\beta}{\beta+1}(d-1) \leq \frac{\beta}{\beta+1}(b+c-1) = \frac{\beta}{\beta+1}c + \frac{\beta}{\beta+1}(b-1) \leq \frac{\beta}{\beta+1}c + \frac{c}{\beta+1}$ (as $\beta\text{-}(b,c)$) $= c$. This, together with P2 implies $\frac{\beta}{1+\beta}\text{-}(c,d)$. So, $\beta\text{-}(a,b) \wedge \frac{\beta}{1+\beta}\text{-}(c,d)$.

Case $b < c$: Since $a < q = b+c+1, a-1 < b+c$. So, $a-1 \leq b+c-1$ or $\frac{\beta(a-1)}{1+\beta} \leq \frac{\beta b}{1+\beta} + \frac{\beta(c-1)}{1+\beta} \leq \frac{\beta b}{1+\beta} + \frac{b}{1+\beta}$ (as $\beta\text{-}(b,c)$) $= b$. This and P1 imply $\frac{\beta}{1+\beta}\text{-}(a,b)$. Also, $d-1 \leq q-2 = b+c-1$. So, $\beta(d-1) \leq \beta(b+c-1) \leq \beta(2c-1) \leq c$. This, together with

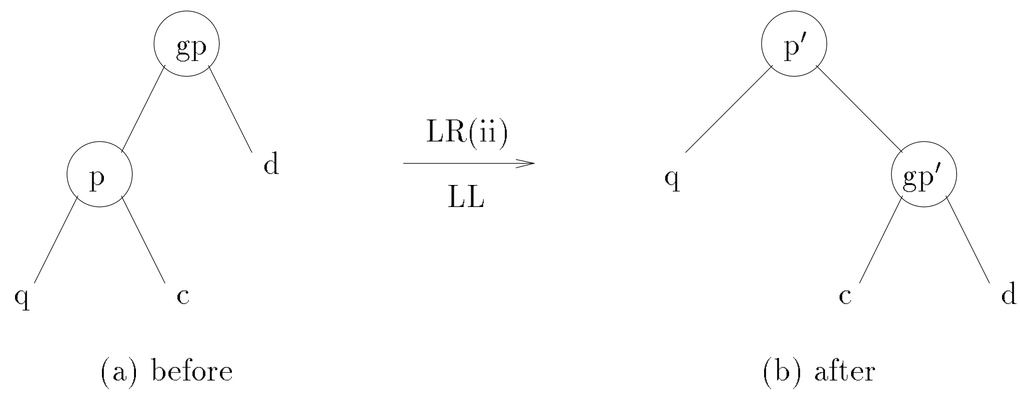


Figure 6: Case LL for LR(ii) rotation

P2 implies β-(c, d). So, $\frac{\beta}{1+\beta}$-$(a, b) \wedge \beta$-(c, d). □

Since an LR(i) rotation can cause the tree to lose its β-balance property, it is necessary to follow this with another rotation that restores the β-balance property. It suffices to consider the two cases of Figures 6 and 7 for this follow up rotation. The remaining cases are symmetric to these. In Figures 6 and 7, p and d denote the nodes that do not satisfy β-(p, d). Note, however, that these nodes do satisfy $\frac{\beta}{1+\beta}$-(p, d).

Since the follow up rotation to LR(i) is done only when

$$\frac{\beta}{1+\beta}\text{-}(p, d) \wedge (\neg\beta\text{-}(p, d)),$$

either $\beta(p-1) > d$ or $\beta(d-1) > p$. When $\beta(p-1) > d$, the second substep rotation is one of the two given in Figures 6 and 7. When $\beta(d-1) > p$, rotations symmetric to these are performed. In the following, we assume $\beta(p-1) > d$. Further, we may assume $d > 0$, as $d = 0$ and $\frac{\beta}{1+\beta}$-(p, d) imply $p \leq 1$. Hence, β-(p, d). Also, $d > 0$ and $\beta(p-1) > d$ imply $p > 1$.

The LR(ii) LL rotation is done when the condition

$$A = (q > d) \wedge (c < (1+\beta)q + (1-\beta)) \wedge B \quad \text{where}$$

$$B = \frac{\beta}{1+\beta}\text{-}(p, d) \wedge (\neg\beta\text{-}(p, d)) \wedge \beta\text{-}(q, c) \wedge (\beta(p-1) > d > 0).$$

Lemma 7 *[Case LR(ii) LL rotation] If A holds before the rotation of Figure 6, then* β*-*(q, gp') *and* β*-*(c, d) *after the rotation provided* $0 < \beta \leq \sqrt{2} - 1$.

Proof (a) β-(q, gp'):

$\beta(q-1) \leq c$ (as β-(q,c)) $< gp'$. Also, $\beta(gp'-1) = \beta(c+d) < \beta((1+\beta)q + (1-\beta) + d) \leq \beta(1+\beta)q + \beta(1-\beta) + \beta(q-1)$ (as $q > d$) $= \beta(2+\beta)q - \beta^2 < q$ (as $\beta(2+\beta) \leq 1$ for $0 < \beta \leq \sqrt{2}-1$). So, β-(q, gp').

(b) β-(c, d):

$\beta(d-1) < \beta(q-1) \leq c$ (as β-(q,c)). And, $\beta(c-1) = \frac{\beta^2}{1+\beta}(c-1) + \frac{\beta}{1+\beta}(c-1) \leq \frac{\beta}{1+\beta}q + \frac{\beta}{1+\beta}(c-1) = \frac{\beta}{1+\beta}(q+c-1) = \frac{\beta}{1+\beta}(p-2) < \frac{\beta}{1+\beta}(p-1) \leq d$ (as $\frac{\beta}{1+\beta}$-(p,d)). So, β-(c,d). □

Lemma 8 *If* $(c < (1+\beta)q + (1-\beta)) \wedge (\beta(p-1) > d)$ *in Figure 6, then* $d \leq q$ *provided* $0 < \beta \leq \sqrt{2} - 1$.

Proof Since $d < \beta(p-1) = \beta(q+c) < \beta(q + (1+\beta)q + 1 - \beta) = \beta(\beta+2)q + \beta(1-\beta) < q+1$ (as $\beta(\beta+2) \leq 1$ and $\beta(1-\beta) < 1$ for $0 < \beta \leq \sqrt{2}-1$). So, $d \leq q$. □

So, the only time an LR(ii) LL rotation is not done is when $C = (C_1 \vee C_2) \wedge B$ holds where

$$C_1 = (q = d) \wedge (c < (1+\beta)q + 1 - \beta)$$

$$C_2 = c \geq (1+\beta)q + (1-\beta).$$

At this time, the LR rotation of Figure 7 is done. In terms of the notation of Figure 7, the condition C becomes $D = (D_1 \vee D_2) \wedge E$ where

$$D_1 = (a = d) \wedge (q < (1+\beta)a + 1 - \beta)$$

$$D_2 = q \geq (1+\beta)a + 1 - \beta$$

$$E = \frac{\beta}{1+\beta}\text{-}(p,d) \wedge \neg\beta\text{-}(p,d) \wedge \beta\text{-}(a,q) \wedge \beta\text{-}(b,c) \wedge (\beta(p-1) > d > 0).$$

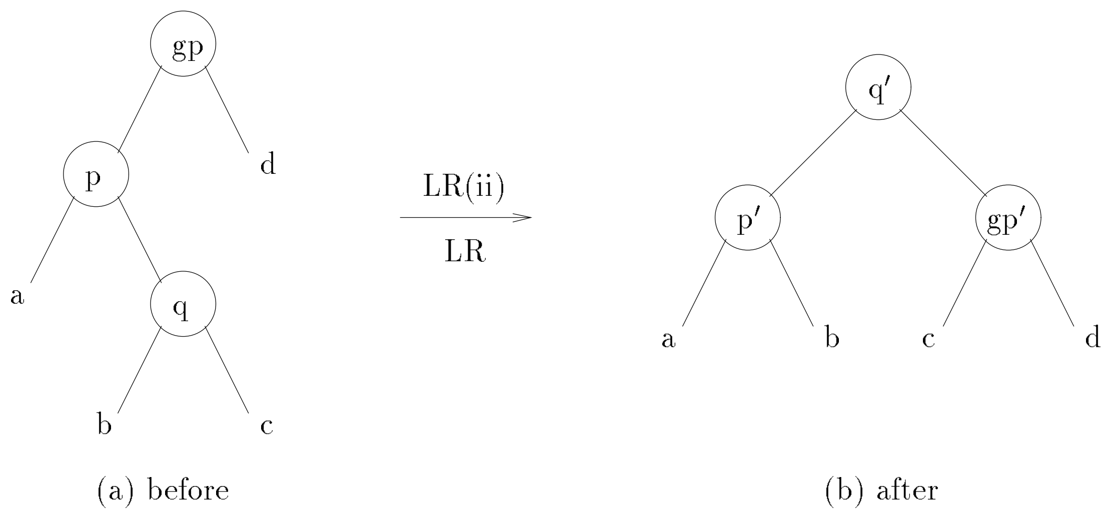


Figure 7: Case LR for LR(ii) rotation

Lemma 9 *When an LR(ii) LR rotation is performed and $\beta \leq \sqrt{2}-1$, $q > d$ and so search cost is reduced.*

Proof If D_1, then since $d < \beta(p-1) = \beta(a+q) = \beta(d+q)$, $q > d/\beta - d > d$ as $\beta \leq \sqrt{2}-1$. If D_2, then $d < \beta(p-1) = \beta(a+q) \leq \beta(\frac{q-1+\beta}{1+\beta} + q) = \frac{\beta(2+\beta)}{1+\beta}q - \frac{\beta(1-\beta)}{1+\beta} < \frac{\beta(2+\beta)}{1+\beta}q \leq q$ (as $\beta \leq \sqrt{2}-1$). □

Lemma 10 *When $(d = a) \wedge \beta\text{-}(b,c) \wedge (\beta(p-1) > d) \wedge (\beta \leq \sqrt{2}-1)$ (see Figure 7), $\beta(a-1) \leq b$ and $\beta(d-1) \leq c$.*

Proof Since $\beta(p-1) > d$ and $d = a, \beta(p-1) > a$ or $\beta(a+q) > a$ or $a(1-\beta) < \beta q$ or $a < \frac{\beta}{1-\beta}q$. So, $\beta(a-1) < \frac{\beta^2}{1-\beta}q - \beta = \frac{\beta^2}{1-\beta}(b+c+1) - \beta$.
If $c \leq \frac{b}{\beta} + \beta$, then

$$\begin{aligned}\beta(a-1) &< \frac{\beta^2}{1-\beta}(b + \frac{b}{\beta} + \beta + 1) - \beta \\ &= \frac{\beta(\beta+1)b}{1-\beta} + \frac{\beta^2(\beta+1)}{1-\beta} - \beta \\ &= \frac{\beta(\beta+1)b}{1-\beta} + \frac{\beta(\beta^2+\beta-1+\beta)}{1-\beta}\end{aligned}$$

$$= \frac{\beta(\beta+1)b}{1-\beta} + \frac{\beta(\beta^2+2\beta-1)}{1-\beta}$$
$$\leq b \ \text{ (as } \beta(\beta+1) \leq 1-\beta \text{ for } \beta \leq \sqrt{2}-1 \text{ and } \beta^2+2\beta-1 \leq 0 \text{ for } \beta \leq \sqrt{2}-1).$$

Since $\beta(c-1) \leq b, c \leq \frac{b}{\beta}+1$. So,

$$\beta(a-1) < \frac{\beta^2}{1-\beta}(b+c+1) \leq \frac{\beta^2}{1-\beta}(b+\frac{b}{\beta}+2) \leq \frac{\beta(\beta+1)b}{1-\beta} + \frac{3\beta^2-\beta}{1-\beta}.$$

So,

$$a-1 < \frac{\beta+1}{1-\beta}b + \frac{3\beta-1}{1-\beta}.$$

However, since $\beta^2+2\beta-1 \leq 0$ for $\beta \leq \sqrt{2}-1$, $(1+\beta)/(1-\beta) \leq \frac{1}{\beta}$ and $(3\beta-1)/(1-\beta) \leq \beta$. So, $a-1 < b/\beta+\beta$. If $a \geq c+1$, then $c \leq a-1 < b/\beta+\beta$. We have already shown that for $c \leq b/\beta+\beta$, $\beta(a-1) \leq b$. So, assume $a < c+1$. Now, $a \leq c$ and $\beta(a-1) \leq \beta(c-1) \leq b$ (as β-(b,c)). So, $\beta(a-1) \leq b$ in all cases. $\beta(a-1) \leq c$ may be shown in a similar way. Since $a = d$, we get $\beta(d-1) \leq c$. □

Lemma 11 *[Case LR(ii) LR rotation] If D holds before the rotation of Figure 7, then β-(p', gp'), β-(a,b), and β-(c,d) following the rotation provided* $0 < \beta \leq \sqrt{2}-1$.

Proof (a) β-(p', gp'):

$\beta(gp'-1) = \beta(c+d) \leq b+\beta+\beta d$ (as β-(b,c)) $\leq b+\beta+\beta q$ (from Lemmas 9 and 10, $q \geq d$) $\leq b+\beta+a+\beta = a+b+2\beta < a+b+1 = p'$. Also, since $\frac{\beta}{1+\beta}$-(p,d) and $q \geq d$, $\beta(p-1) \leq (\beta+1)d$ or $\beta(a+q) \leq (\beta+1)d$ or $a+q \leq (1+\frac{1}{\beta})d$ or $a \leq (1+\frac{1}{\beta})d - q \leq (1+\frac{1}{\beta})d - d = d/\beta$. So, $\beta(p'-1) = \beta(a+b) < d+\beta b \leq d+c+\beta$ (as β-(b,c)) $< d+c+1 = gp'$.

(b) β-(a,b):

Since $b \leq q$ and β-(a,q), $\beta(b-1) \leq \beta(q-1) \leq a$.

When D_1, $\beta(a-1) \leq b$ was proved in Lemma 10. So, β-(a,b).

When D_2, $q \geq a(1+\beta)+1-\beta$. So,

$$a \leq \frac{q}{1+\beta} - \frac{1-\beta}{1+\beta} = \frac{b+c+1}{1+\beta} - \frac{1-\beta}{1+\beta}.$$

So,

$$\beta(a-1) \le \frac{\beta b + \beta c + \beta}{1+\beta} - \frac{1-\beta}{1+\beta}\beta - \beta \le \frac{\beta b + b + 2\beta}{1+\beta} - \frac{1-\beta}{1+\beta}\beta - \beta = b.$$

So, β-(a, b).

(c) β-(c, d):

Note that $\beta(c-1) < \beta(q-1) < \frac{\beta}{1+\beta}(q-1) < \frac{\beta}{1+\beta}(p-1) \le d$.

When D_1, $\beta(d-1) \le c$ was proved in Lemma 10. So, β-(c, d).

When D_2, if $d < b+1$, then $d \le b$ and $\beta(d-1) \le \beta(b-1) \le c$. So, assume $d \ge b+1$. Now, $b \le d-1 < \beta(p-1) - 1$. So,

$$\begin{aligned}
b &< \beta(a+b+c+1) - 1 \\
&\le \beta(\frac{q-1+\beta}{1+\beta} + b + c + 1) - 1 \\
&= \frac{\beta}{1+\beta}(b + c + \beta + (1+\beta)(b+c+1)) - 1 \\
&\le \frac{\beta}{1+\beta}(\frac{c}{\beta} + 1 + c + \beta + (1+\beta)(\frac{c}{\beta} + 1 + c + 1)) - 1 \\
&= \frac{\beta}{1+\beta}(\frac{\beta+1}{\beta}c + (1+\beta) + (1+\beta)(\frac{1+\beta}{\beta}c + 2)) - 1 \\
&= c + \beta + (1+\beta)c + 2\beta - 1 \\
&= (2+\beta)c + 3\beta - 1 < (2+\beta)c + \beta \text{ (as } \beta \le \sqrt{2}-1) \\
&\le \frac{c}{\beta} + \beta \text{ (as } \beta \le \sqrt{2}-1).
\end{aligned}$$

Also, from $d < \beta(p-1)$ and the above derivation, we get

$$\begin{aligned}
d &< \frac{\beta}{1+\beta}(b + c + \beta + (1+\beta)(b+c+1)) \\
&\le \frac{\beta}{1+\beta}(\frac{c}{\beta} + \beta + c + \beta + (1+\beta)(\frac{c}{\beta} + \beta + c + 1)) \\
&= \frac{\beta}{1+\beta}(\frac{\beta+1}{\beta}c) + \frac{2\beta^2}{1+\beta} + \beta(\frac{1+\beta}{\beta}c) + \beta(\beta+1) \\
&= (2+\beta)c + \frac{2\beta^2}{1+\beta} + \beta(\beta+1) \\
&= (2+\beta)c + \frac{2\beta^2 + \beta^2 + \beta^3 + \beta + \beta^2}{1+\beta} \\
&= (2+\beta)c + \frac{\beta^3 + 4\beta^2 + \beta}{1+\beta}
\end{aligned}$$

$$\leq \quad (2+\beta)c+1 \text{ (as } \beta^3+4\beta^2+\beta<1+\beta \text{ for } \beta\leq\sqrt{2}-1).$$

So, $\beta(d-1) \leq \beta(2+\beta)c \leq c$ (as $\beta \leq \sqrt{2}-1$). So, β-(c,d). □

Theorem 2 *If T is β-balanced, $0 \leq \beta \leq \sqrt{2}-1$, prior to insertion, it is so following the insertion.*

Proof First note that since all binary search trees are balanced for $\beta = 0$, the rotations (while unnecessary) preserve 0-balance. So, assume $\beta > 0$. Consider the tree T' just after the new element has been inserted but before the backward restructuring pass begins.

If the newly inserted node, z, has no parent in T', then T was empty and T' is β-balanced. If z has a parent but no grandparent, then T has at most one nonempty subtree X. Since T is β-balanced, $\beta(|X|-1) \leq 0$. So, $|X| \leq 1$. Following the insertion, T' has one subtree with ≤ 1 nodes and one with exactly one. So, T' is β-balanced. We may therefore assume that z has a grandparent in T'.

From the downward insertion path, it follows that all nodes u in T' that have children l and r for which $\neg\beta$-(l,r) must lie on the path from the root to z. During the backward restructuring pass, each node on this path (other than z and its parent) play the role of gp in Figures 4 and 5. The β-property cannot be violated at z as z has no children. It cannot be violated at the parent, s, of z as s satisfied the β-property prior to insertion. As a result its other subtree has ≤ 1 element. So, following the insertion, s satisfies the β-property. As a result, each node in T' that might possibly violate the β-property becomes the gp node during the restructuring pass. Consider one such gp node. It has children in T' denoted by p' and d. Its children in T are p and d. Figures 4 and 5 show the case when d is the right subtree of gp in both T and T'. The cases RR and RL arise when d is the left subtree.

During the restructuring pass, gp begins at the grandparent of z and moves up to the root of T'. If z is at level r in T', (the root being at level 1), then gp takes on $r-2$ values during the restructuring pass. We shall show that at each of these $r-2$ positions either

(a) no rotation is performed and all descendants of gp satisfy the β-property or

(b) a rotation is performed and following this, all descendants of node p'' (Figure 4) or of node q' (Figure 5) satisfy the β-property.

As a result, following the rotation (if any) performed when gp becomes the root of T', the restructured tree is β-balanced. The proof is by induction on r. When $r = 3$ (recall, we assume z has a grandparent), gp begins at the root of T' and its descendants satisfy the β-property.

Without loss of generality, assume that the insertion took place in the left subtree of gp. With respect to Figure 4, we have three cases: (i) $q \geq c$ and $q > d$, (ii) $q < c$ and $c > d$, and (iii) $q \leq d$ and $c \leq d$. In case (i), all conditions for an LL rotation hold and such a rotation is performed. In case (ii), an LR rotation is performed. Following either rotation, T' is β-balanced. In case (iii), $\beta(p' - 1) = \beta(q + c) \leq 2\beta d < d$ (as $\beta \leq \sqrt{2} - 1$). Also, $\beta(d - 1) \leq p < p + 1 = p'$. So, $\beta(d - 1) < p'$. Hence, β-(p', d) and T' is β-balanced.

For the induction hypothesis, assume (a) and (b) whenever $r \leq k$. In the induction step, we show (a) and (b) for trees T with $r = k + 1$. The subtree in which the insertion is done has $r = k$. So, (a) and (b) hold for all gp locations in the subtree. We need to show (a) and (b) only when gp is at the root of T'. This follows from Lemmas 5, 6, 7, and 11.

The theorem now follows. □

Lemma 12 *The time needed to do an insertion in an n node β-BBST is $O(\log n)$ provided $0 < \beta \leq \sqrt{2} - 1$.*

Proof Follows from the fact that insertion takes $O(h)$ time where h is the tree height and $h = O(\log n)$ when $\beta > 0$ (Lemmas 1 and 3). □

4.3 Deletion

To delete element x from a β-BBST, we first use the unbalanced binary search tree deletion algorithm of [HORO94] to delete x and then perform a series of rebalancing rotations. The steps are:

Step 1 [Locate x] Search the β-BBST for the node y that contains x. If there is no such node, terminate.

Step 2 [Delete x] If y is a leaf, set d' to nil, gp to the parent of y, and delete node y. If y has exactly one child, set d' to be this child; change the pointer from the parent (if any) of y to point to the child of y; delete node y; set gp to be the parent of d'. If y has two children, find the node z in the left subtree of y that has largest value; move this value into node y; set $y = z$; go to the start of Step 2. { note that the new y has either 0 or 1 child }

Step 3 [Rebalance] Retrace the path from d' to the root performing rebalancing rotations.

There are four rebalancing rotations LL, LR, RR, and RL. Since LL and RR as well as LR and RL are symmetric rotations, we describe LL and LR only. The discussion is very similar to the case of insertion. The differences in proofs are due to the fact that a deletion reduces the size of encountered subtrees by 1 while an insertion increases it by 1. In an LL rotation, the configuration just before and after the rotation is shown in Figure 8. This rotation is performed when $q \geq c$ and $q > d'$. Following the rotation, d' is updated to the node p'.

Let d denote the size of the right subtree of gp before the deletion. So, $d = d' + 1$. Since prior to the deletion the β-BBST was β-balanced, it follows that β-(p, d) and β-(q, c).

Lemma 13 *[LL deletion lemma] If [*β-$(p, d) \wedge \beta$-$(q, c) \wedge (q \geq c) \wedge (q > d) \wedge (1/3 \leq \beta \leq 1/2)$*] before the rotation, then [*β-$(q, gp') \wedge \beta$-(c, d')*] after the rotation.*

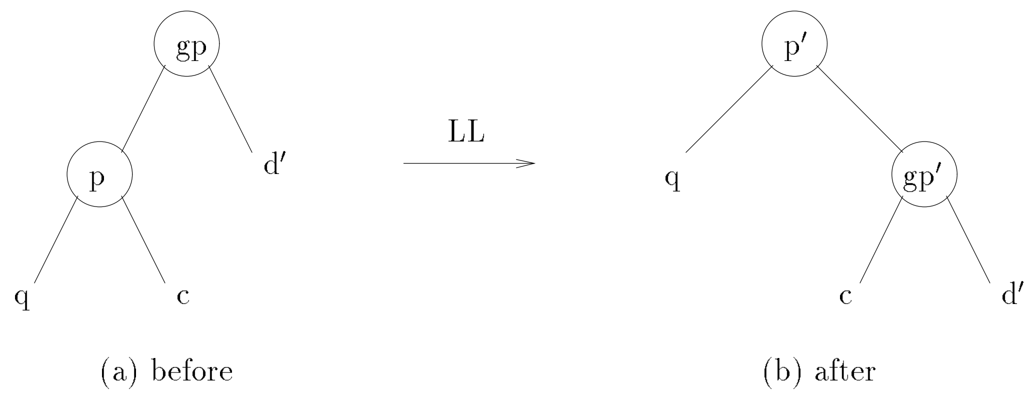


Figure 8: LL rotation for deletion

Proof (a) β-(q, gp'):

$\beta(q-1) \leq c$ (as β-(q,c)) $< gp'$. Also, $\beta(gp'-1) = \beta(c+d') < 2\beta q$ (as $c \leq q$ and $d' < q$) $\leq q$ (as $\beta \leq 1/2$). So, β-(q, gp').

(b) β-(c, d'):

$d' < q \Rightarrow d'-1 < q-1 \Rightarrow \beta(d'-1) < \beta(q-1) \leq c$. Also, when $c \leq 1$, $\beta(c-1) \leq 0 \leq d'$ (as $d' \geq 0$). When $c > 1, q \geq c \Rightarrow q \geq 2$ and $p = q+c+1 \geq c+3$. So, $\beta(c-1) \leq \beta(p-1) - 3\beta \leq d - 3\beta$ (as β-(p,d)) $\leq d-1$ (as $\beta \geq 1/3$) $= d'$. Hence, β-(c, d'). □

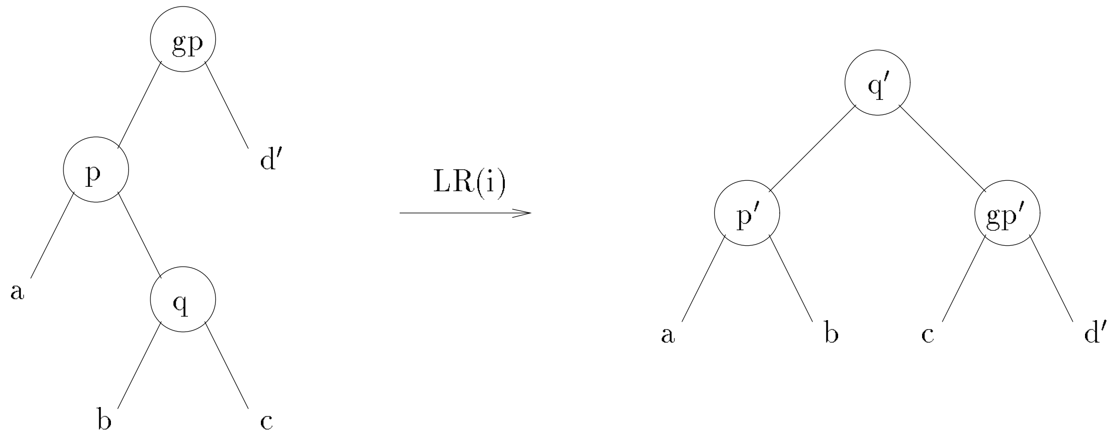
In an LR rotation, the before configuration is as in Figure 8(a). However, this time $q < c$. Figure 8(a) is redrawn in Figure 9(a). In this, the node labeled c in Figure 8(a) has been relabeled q and that labeled q in Figure 8(a) has been relabeled a. With respect to the labelings of Figure 9(a), rotation LR is applied when

$$[(q > a) \wedge (q > d')].$$

The other conditions that apply when an LR rotation is performed are

$$[\beta\text{-}(p,d) \wedge \beta\text{-}(a,q) \wedge \beta\text{-}(b,c)].$$

Here d denotes the (size of) right subtree of gp prior to the deletion. As in the case of insertion, an LR rotation is accomplished in two substeps (or two subrotations). The first



(a) before (b) after substep (i)

Figure 9: LR rotation for deletion

of these is shown in Figure 9. Following an LR rotation, d' is updated to node q'.

Lemma 14 *[LR substep(i) deletion lemma] If* $[\beta\text{-}(p,d) \wedge \beta\text{-}(a,q) \wedge \beta\text{-}(b,c) \wedge (q > a) \wedge (q > d')]$ *before the subrotation LR(i), then* $[\beta\text{-}(p',gp') \wedge \{(\beta\text{-}(a,b) \wedge \frac{\beta}{1+\beta}\text{-}(c,d')) \vee (\frac{\beta}{1+\beta}\text{-}(a,b) \wedge \beta\text{-}(c,d'))\}]$ *after the subrotation provided* $1/3 \leq \beta \leq 1/2$.

Proof Assume the before condition.

(a) If $b = c = 0$, then $q = b + c + 1 = 1$. Furthermore, $(q > a)$ and $(q > d')$ imply $a = d' = 0$. So, $gp' = p' = 1$. Hence, $[\frac{1}{2}\text{-}(p',gp') \wedge \frac{1}{2}\text{-}(a,b) \wedge \frac{1}{2}\text{-}(c,d')]$

(b) If $b = 1$ and $c = 0$, then $q = 2, a \leq 1$, and $d' \leq 1$. So, $1 \leq p' \leq 3$ and $1 \leq gp' \leq 2$. Hence, $[\frac{1}{2}\text{-}(p',gp') \wedge \frac{1}{2}\text{-}(a,b) \wedge \frac{1}{2}\text{-}(c,d')]$

(c) If $b = 0$ and $c = 1$, then $q = 2, a \leq 1$, and $d' \leq 1$. So, $1 \leq p' \leq 2$ and $1 \leq gp' \leq 3$. Hence, $[\frac{1}{2}\text{-}(p',gp') \wedge \frac{1}{2}\text{-}(a,b) \wedge \frac{1}{2}\text{-}(c,d')]$

As a result of (a) – (c), to complete the proof, we may assume that $b \geq 1$ and $c \geq 1$. So, $q \geq 3, a \geq 1$ (as $\beta\text{-}(a,q) \Rightarrow \beta(q-1) \leq a$ or $a \geq 2\beta > 0$), $p = a + q + 1 \geq 5, d \geq 2$ (as $\beta\text{-}(p,d) \Rightarrow \beta(p-1) \leq d$ and $\beta \geq 1/3$), and $d' = d - 1 \geq 1$.

First, we show that $\beta\text{-}(p',gp')$. For this, note that $a + b + c + 1 = p - 1$. From $\beta\text{-}(p,d)$, it follows that $\beta(a+b+c+1) = \beta(p-1) \leq d$. So, $\beta(a+b) \leq d - \beta c - \beta$. From Figure 9(b), we see that $\beta(p'-1) = \beta(a+b)$. Hence, $\beta(p'-1) \le d-\beta c-\beta = d'-\beta c+1-\beta \le d'+1-2\beta < gp'$. Also,

$$\begin{aligned}
\beta(gp'-1) = \beta(c+d') &\le b + \beta + \beta d' \text{ (as } \beta\text{-}(b,c))\\
&< b + \beta q + \beta \text{ (as } q > d')\\
&\le b + a + 2\beta \text{ (as } \beta\text{-}(a,q))\\
&< p'.
\end{aligned}$$

So, β-(p', gp').

Next, we prove two properties that will be used to complete the proof.

P1: $\beta(b-1) \le a$.

To see this, note that $\beta(b-1) < \beta(q-1) \le a$ (as β-(a,q)).

P2: $\beta(c-1) \le d'$.

For this, observe that $\beta(c-1) \le \beta(q-2)$ (as $c \le q-1$) $\le \beta(p-4)$ (as $q = p-a-1$ and $a \ge 1$) $= \beta(p-1) - 3\beta \le d-1$ (as β-(p,d) and $\beta \ge 1/3$) $= d'$.

To complete the proof of the lemma, we need to show

$$\{(\beta\text{-}(a,b) \wedge \frac{\beta}{1+\beta}\text{-}(c,d')) \vee (\frac{\beta}{1+\beta}\text{-}(a,b) \wedge \beta\text{-}(c,d'))\}.$$

For this, consider the two cases $b \ge c$ and $b < c$ (as in Lemma 6).

Case $b \ge c$: Since $a < q = b+c+1, \beta(a-1) < \beta(b+c) \le 2\beta b \le b$. This, together with P1 implies β-(a,b). Also, $d' < q = b+c+1$. So, $\frac{\beta}{\beta+1}(d'-1) \le \frac{\beta}{\beta+1}(b+c-1) = \frac{\beta}{\beta+1}c + \frac{\beta}{\beta+1}(b-1) \le \frac{\beta}{\beta+1}c + \frac{c}{\beta+1} = c$. This, together with P2 implies $\frac{\beta}{1+\beta}$-(c,d'). So, β-$(a,b) \wedge \frac{\beta}{1+\beta}$-$(c,d')$.

Case $b < c$: Since $a < q = b+c+1, a-1 < b+c$. So, $a-1 \le b+c-1$ or $\frac{\beta(a-1)}{1+\beta} \le \frac{\beta b}{1+\beta} + \frac{\beta(c-1)}{1+\beta} \le \frac{\beta b}{1+\beta} + \frac{b}{1+\beta} = b$. This and P1 imply $\frac{\beta}{1+\beta}$-(a,b). Also, $d'-1 \le q-2 = b+c-1$. So, $\beta(d'-1) \le \beta(b+c-1) < \beta(2c-1) < c$. This and P2 imply β-(c,d'). Hence, $\frac{\beta}{1+\beta}$-$(a,b) \wedge \beta$-(c,d'). □

The substep(ii) rotations are the same as for insertion.

Theorem 3 *If T is β-balanced, then following a deletion the resulting tree T' is also β-balanced provided* $1/3 \leq \beta \leq \sqrt{2}-1$.

Proof Similar to that of Theorem 2. □

When $0 < \beta < 1/3$, we need to augment the LL rotation by a transformation for the case $d' = 0$. When $d' = 0, \beta(p-1) \leq d = d'+1 = 1$. So, $p \leq 1/\beta+1$ and $gp = p+d'+1 \leq 1/\beta+2$. To β-balance at gp, the at most $1/\beta + 2$ nodes in gp are rearranged into any β-BBST in constant time (as $1/\beta+2$ is a constant). When $d' > 0$, the proof of Lemma 13 part (b) can be changed to show $\beta(c-1) \leq d'$ for $0 < \beta \leq \sqrt{2}-1$. The new proof is: since $c \leq q, c \leq (p-1)/2$ and $\beta(c-1) \leq \beta(p-1)/2 - \beta \leq d/2 - \beta = d - d/2 - \beta \leq d - 1 - \beta < d'$. The LR rotation needs to be augmented by a transformation for the case $d' = d - 1 < \frac{1}{\beta(2+\beta)} - 1$. At this time, $\beta(p-1) \leq d < \frac{1}{\beta(2+\beta)}$. So, $gp = p + d < \frac{1}{\beta^2(2+\beta)} + 1 + \frac{1}{\beta(2+\beta)}$. To β-balance at gp, we rearrange the fewer than $\frac{1}{\beta^2(2+\beta)} + 1 + \frac{1}{\beta(2+\beta)}$ nodes in the subtree, in constant time, into any β-balanced tree. When $d' \geq \frac{1}{\beta(2+\beta)} - 1$, the proof for $\beta(c-1) \leq d'$ in Lemma 14 needs to be changed to show that the LR substep(i) lemma holds. The new proof is:

$$\begin{aligned} d &\geq \beta(p-1) = \beta(a+b+c+1) \geq \beta(\beta(q-1)+b+c+1) \\ &= \beta(\beta(b+c)+b+c+1) \\ &\geq \beta((1+\beta)\beta(c-1)+(1+\beta)c+1) \\ &= \beta((1+\beta)^2(c-1)+2+\beta). \end{aligned}$$

So, $\beta(c-1) \leq \frac{d-2\beta-\beta^2}{(1+\beta)^2} \leq d-1$ (as $d \geq \frac{1}{\beta(2+\beta)}$) $= d'$.

Also, note that when $\beta = 0$, all trees are β-balanced so the rotations (while not needed) preserve balance.

Theorem 4 *With the special handling of the case $d' = 0$, the tree T' resulting from a deletion in a β-BBST is also β-balanced for* $0 \leq \beta \leq \sqrt{2}-1$.

Lemma 15 *The time needed to delete an element from an n node β-BBST is $O(\log n)$ provided* $0 < \beta \leq \sqrt{2}-1$.

4.4 Enhancements

Since our objective is to create search trees with minimum search cost, the rebalancing rotations may be performed at each positioning of gp during the backward restructuring pass so long as the conditions for the rotation apply rather than only at gp positions where the tree is unbalanced.

Consider Figure 4(a). If $p' < d$, then the conditions of Lemmas 5 and 6 cannot apply as $q < p' < d$. However, it is possible that $e > p'$ where e is the size of either the left or right subtree of d. In this case, an RR or RL rotation would reduce the total search cost. The proofs of Lemmas 5 and 6 are easily extended to show that these rotations would preserve balance even though no insertion was done in the subtree d. The same observation applies to deletion. Hence the backward restructuring pass for the insert and delete operations can determine the need for a rotation at each gp location as below (l and r are, respectively, the left and right children of gp).

if $s(l) > s(r)$ **then** check conditions for an LL and LR rotation

else check conditions for an RR and RL rotation.

The enhanced restructuring procedure used for insertion and deletion is given in Figure 10. In the RR and RL cases, we have used the relation '$\geq$' rather than '$>$' as this results in better observed run time.

Since it can be shown that the rotations preserve balance even when there has been no insert or delete, we may check the rotation conditions during a search operation and perform rotations when these improve total search cost.

Finally, we note that it is possible to use other definitions of β-balance. For example, we could require $\beta(s(a) - 2) < s(b)$ and $\beta(s(b) - 2) < s(a)$ for β-(a, b). One can show that the development of this paper applies to these modifications also. Furthermore, when this new definition is used, the number of comparisons in the second substep of the LR and RL rotations is reduced by one.

```
procedure Restructuring ;
begin
while (gp) do
    begin
    if s(gp.left) > s(gp.right) then { check conditions for an LL and LR rotation }
        begin
        p = gp.left ;
        if (s(p.left) > s(p.right)) then
            begin if (s(p.left) > s(gp.right)) then do LL rotation; end
        else
            begin
            if (s(p.right) > s(gp.right)) then { LR }
                begin
                do LR rotation ;
                { now notations a, b, c, and d follow from figure 1(b) }
                if (β(s(a) − 1) > s(b)) then
                    if ((s(a.right) < (1 + β)s(a.left) + 1 − β) and
                        (s(b) < s(a.left))) then
                        do LL rotation
                    else do LR rotation
                else if (β(s(d) − 1) > s(c)) then
                    if ((s(d.left) < (1 + β)s(d.right) + 1 − β) and
                        (s(c) < s(d.right))) then
                        do RR rotation
                    else do RL rotation ;
                end
            end
        end
    else { check conditions for an RR and RL rotation }
        begin
        p = gp.right ;
        if (s(p.left) > s(p.right)) then
            begin
            if (s(p.left) ≥ s(gp.left)) then { RL }
                do symmetric to the above LR case ;
            end
        else
            begin if (s(p.right) ≥ s(gp.left)) then do RR rotation; end ;
        end ;
    gp = gp.parent ;
    end ;
end ;
```

Figure 10: Restructuring procedure

4.5 Top Down Algorithms

As in the case of red/black and WB(α) trees, it is possible to perform, in O($\log n$) time, inserts and deletes using a single top to bottom pass. The algorithms are similar to those already presented.

5 Simple β-BBSTs

The development of Section 4 was motivated by our desire to construct trees with minimal search cost. If instead, we desire only logarithmic performance per operation, we may simplify the restructuring pass so that rotations are performed only at nodes where the β-balance property is violated. In this case, we may dispense with the LL/RR rotations and the first substep of an LR/RL rotation. Only LR/RL substep (ii) rotations are needed. To see this, observe that Lemmas 7 and 11 show that the second substep rotations rebalance at gp (see Figures 6 and 7) provided $\frac{\beta}{1+\beta}$-(p, d) (The remaining conditions are ensured by the bottom-up nature of restructuring and the fact the tree was β-balanced prior to the insert or delete).

If the operation that resulted in loss of balance at gp was an insert, then $\beta(p-2) \leq d$ (as $p > d$, the insert took place in subtree p and gp was β-balanced prior to the insert) and $\beta(p-1) > d$ (gp is not β-balanced following the insert). For the substep (ii) rotation to restore balance, we need $\beta(p-1) \leq (1+\beta)d$. This is assured if $d + \beta \leq (\beta+1)d$ (as $\beta(p-2) \leq d$). So, we need $d \geq 1$. If $d < 1$, then $d = 0$. Now $\beta(p-2) \leq d$ and $\beta(p-1) > d$ imply $p = 2$. One may verify that when $p = 2$, the LR(ii) rotations restore balance.

If the loss of β-balance at gp is the result of a deletion (say from its right subtree), then $\beta(p-1) \leq d+1$ (as gp was β-balanced prior to the delete). For the substep (ii) rotation to accomplish the rebalancing, we need $\beta(p-1) \leq (\beta+1)d$. This is guaranteed if $d+1 \leq (\beta+1)d$ or $d \geq 1/\beta$. When $d < 1/\beta$ and $\beta \geq 1/3$, $d \leq 2$. Since $\beta(p-1) \leq d+1$ and $\beta \geq 1/3$, when $d = 2$, $p \leq 10$; when $d = 1$, $p \leq 7$; and when $d = 0$, $p \leq 4$. We may verify that for all these cases, the LR(ii) rotations restore balance. Hence, the only problematic case is when $\beta < 1/3$ and $d < 1/\beta$.

```
procedure Restructuring2 ;
begin
while (gp) do
    begin
    if (β(s(gp.left) − 1) > s(gp.right)) then { do an LL or LR rotation }
        begin
        p = gp.left ;
        if ((s(p.right) < (1 + β)s(p.left) + 1 − β) and
            (s(gp.right) < s(p.left))) then
            do LL rotation
        else do LR rotation ;
        end
    else
        do symmetric to the above L case ;
    gp = gp.parent ;
    end ;
end ;
```

Figure 11: Simple restructuring procedure for insertion

When $\beta < 1/3$, an LL rotation fails to restore balance only when $d = 0$ (see discussion following Theorem 3). So we need to rearrange the at most $1/\beta + 2$ nodes in gp into any β-balanced tree when $d = 0$. An LR rotation fails only when $d < \frac{1}{\beta(2+\beta)} - 1$. To see this, note that in the terminology of Lemma 14, d is d'. The proof of P2 is extended to the case $\beta \leq 1/3$ when $d' \geq \frac{1}{\beta(2+\beta)} - 1$. Also, since $d' < 1/\beta$, for the case $b \geq c$, we get $\beta(d'-1) < 1-\beta < c$ (as $c \geq 1$). For the case $b < c$, we need to show $\beta(a-1) \leq b$. Since an LR rotation is done only when condition $D1 \vee D2$ holds, from Lemmas 10 and 11, it follows that $\beta(a-1) \leq b$. So, an LR rotation rebalances when $\beta < 1/3$ provided $d \geq \frac{1}{\beta(2+\beta)} - 1$. For smaller d, the at most $\frac{1}{\beta^2(2+\beta)} + \frac{1}{\beta(2+\beta)} + 1$ nodes in the subtree gp may be directly rearranged into a β-balanced tree.

The restructuring algorithm for simple β-BBSTs is given in Figures 11 and 12. The algorithm of Figure 11 is used following an insert and that of Figure 12 after a delete.

Simple β-BBSTs are expected to have higher search cost than the β-BBSTs of Section 4. However, they are a good alternative to traditional WB(α) trees as they are expected to be "better balanced". To see this, note that from the proof of Lemma 3, the balance, $B(p)$, at

```
procedure Restructuring3 ;
begin
while (gp) do
    begin
    if (β(s(gp.left) − 1) > s(gp.right)) then
        if (β < 1/3) and (s(gp.right) < 1/β(2 + β) − 1) then
            rearrange the subtree rooted at gp into any β-balanced tree
        else { do an LL or LR rotation }
            begin
            p = gp.left ;
            if ((s(p.right) < (1 + β)s(p.left) + 1 − β) and
                (s(gp.right) < s(p.left))) then
                do LL rotation
            else do LR rotation ;
            end
        end
    else
        do symmetric to the above L case ;
    gp = gp.parent ;
    end ;
end ;
```

Figure 12: Simple restructuring procedure for deletion

any node p in a β-balanced tree satisfies

$$\begin{aligned}\frac{1}{B(p)} &= 1+\frac{s(r)+1}{s(l)+1}\\ &\geq 1+\frac{1}{1/\beta+\frac{2\beta-1}{\beta(s(r)+1)}}\\ &= \frac{1+\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(r)+1)}}{\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(r)+1)}}.\end{aligned}$$

So,

$$B(p)\leq 1-\frac{1}{1+\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(r)+1)}}.$$

Also, since $s(r)-1\leq s(l)/\beta$, $s(r)+1\leq s(l)/\beta+2$. Hence, $1+\frac{s(r)+1}{s(l)+1}\leq 1+\frac{s(l)}{\beta(s(l)+1)}+\frac{2}{s(l)+1}$.

So,

$$\begin{aligned}B(p) &\geq \frac{1}{1+\frac{1}{\beta}-\frac{1}{\beta(s(l)+1)}+\frac{2}{s(l)+1}}\\ &= \frac{1}{1+\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(l)+1)}}.\end{aligned}$$

Consequently,

$$\frac{1}{1+\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(l)+1)}} \leq B(p) \leq 1-\frac{1}{1+\frac{1}{\beta}+\frac{2\beta-1}{\beta(s(r)+1)}}.$$

When $\beta = \sqrt{2}-1$,

$$\frac{1}{2+\sqrt{2}+\frac{1-\sqrt{2}}{s(l)+1}} \leq B(p) \leq 1-\frac{1}{2+\sqrt{2}+\frac{1-\sqrt{2}}{s(r)+1}}.$$

If $s(p) \leq 10$, $0.296 \leq B(p) \leq 1-0.296$. So, every β-balanced subtree with 10 or fewer nodes is in WB(α) for $\alpha \approx 0.296$. Similarly, every subtree with 100 or fewer nodes is in WB(α) for $\alpha \approx 0.293$. In fact, for every fixed k, subtrees of size k or less are in WB(α) for α slightly higher than $1-\frac{1}{\sqrt{2}} \approx 0.2929$ which is the largest value of α for which WB(α) trees can be maintained.

6 BBSTs without Deletion

In some applications of a dictionary, we need to support only the insert and search operations. In these applications, we can construct binary search trees with total cost

$$C(T) \leq n\log_{\phi}(\sqrt{5}(n+1))$$

by using the simpler restructuring algorithm of Figure 13.

Theorem 5 *When the only operations are search and insert and restructuring is done as in Figure 13,* $C(T) \leq n\log_{\phi}(\sqrt{5}(n+1))$.

Proof Suppose T currently has $m-1$ elements and a new element is inserted. Let u be the level at which the new element is inserted. Suppose that the restructuring pass performs rotations at $q < u$ of the nodes on the path from the root to the newly inserted node. Then $C(T)$ increases by at most $v = u - q$ as a result of the insertion. The number of nodes on the path from the root to the newly inserted node at which no rotation is performed is also v. Let these nodes be numbered 1 through v bottom to top. Let S_i denote the number of elements in the subtree with root i prior to the restructuring pass. We see that $S_1 \geq 1$ and

```
procedure Restructuring4 ;
begin
while (gp) do
    begin
    if s(gp.left) > s(gp.right) then { check conditions for an LL and LR rotation }
        begin
        p = gp.left ;
        if (s(p.left) > s(p.right)) and (s(p.left) > s(gp.right)) then
            do LL rotation
        else if (s(p.left) ≤ s(p.right)) and (s(p.right) > s(gp.right)) then
            do LR rotation ;
        end
    else { check conditions for an RR and RL rotation }
        do symmetric to the above L case ;
    gp = gp.parent ;
    end ;
end ;
```

Figure 13: Simple restructuring procedure without a β value

$S_2 \geq 2$. For node i, $2 < i \leq v$, one of its subtrees contains node $i-1$. Without loss of generality, let this be the left subtree of i. Let the root of the right subtree of i be d. So,

$$S_i \geq S_{i-1} + s(d) + 1.$$

If $i-1$ is not the left child of i, then since no rotation is done at i, $s(d) \geq S_{i-1}$. If $i-1$ is the left child of i, then consider node $i-2$. This is in one of the subtrees of i. Since no rotation is performed at $i-1$, $s(d) \geq S_{i-2}$. Since $S_{i-1} > S_{i-2}$, we get

$$S_i \geq S_{i-1} + S_{i-2} + 1.$$

Hence, $S_v \geq N_v$ where N_v is the minimum number of elements in a COST of height v. So, $v \leq \log_\phi(\sqrt{5}(m+1))$. So, when an element is inserted into a tree that has $m-1$ elements, its cost $C(T)$ increases by at most $\log_\phi(\sqrt{5}(m+1))$. Starting with an empty tree and inserting n elements results in a tree whose cost is at most $n \log_\phi(\sqrt{5}(n+1))$. □

Corollary 2 *The expected cost of a search or insert in a BBST constructed as above is* $O(\log n)$.

Proof Since $C(T) \leq n \log_\phi(\sqrt{5}(n+1))$, the expected search cost is $C(T)/n \leq \log_\phi(\sqrt{5}(n+1))$. The cost of an insert is the same order as that of a search as each insert follows the corresponding search path twice (top down and bottom up). □

7 Experimental Results

For comparison purposes, we wrote C programs for BBSTs, SBBSTs (simple BBSTs), BBSTDs (BBSTs in which procedure Restructuring4 (Figure 13) is used to restructure following inserts as well as deletes), unbalanced binary search trees (BST), AVL-trees, top-down red-black trees (RB-T), bottom-up red-black trees (RB-B) [TARJ83], weight balanced trees (WB), deterministic skip lists (DSL), treaps (TRP), and skip lists (SKIP). For the BBST and SBBST structures, we used $\beta = 207/500$ while for the WB structure, we used $\alpha = 207/707$. While these are not the highest permissible values of β and α, this choice permitted us to use integer arithmetic rather than the substantially more expensive real arithmetic. For instance, β-(a, b) for $\beta = 207/500$ can be checked using the comparisons $207(s(a) - 1) > 500s(b)$ and $207(s(b) - 1) > 500s(a)$. The randomized structures TRP and SKIP used the same random number generator with the same seed. SKIP was programmed with probability value $p = 1/4$ as in [PUGH90].

To minimize the impact of system call overheads on run time measurements, we programmed all structures using simulated pointers (i.e., an array of nodes with integer pointers [SAHN93]. Skip lists use variable size nodes. This requires more complex storage management than required by the remaining structures which use nodes of the same size. For our experiments, we implemented skip lists using fixed size nodes, each node being of the maximum size. As a result, our run times for skip lists are smaller than if a space efficient implementation had been used. In all our tree structure implementations, null pointers were replaced by a pointer to a tail node whose data field could be set to the search/insert/delete key and thus avoid checking for falling off the tree. Similar tail pointers are part of the defined structure of skip and deterministic skip lists. Each tree also had a head node. WB(α) trees were implemented with a bottom-up restructuring pass. Our codes for SKIP and DSL are based on the codes of [PUGH90] and [PAPA93], respectively. Our AVL and RB-T codes are based on those of [PAPA93] and [SEDG94]. The treap structure was implemented using joins and splits rather than rotations. This results in better performance. Furthermore, AVL, RB-B, WB, and BBST were implemented with parent pointers in addition to left and right child pointers. For BBSTs, the enhancements described in Section 4.4 for insert and delete (see Figure 10) were employed. No rotations were performed during a search when using any of the structures.

For our experiments, we tried two versions of the code. These varied in the order in which the 'equality' and 'less than' or 'greater than' check between x and e (where x is the key being searched/inserted/deleted and e is the key in the current node) is done. In version 1, we conducted an initial experiment to determine if the total comparison count is less using the order L:

if $x < e$ **then** move to left child
else if $x \neq e$ **then** move to right child
  else found

or the order R:

if $x > e$ **then** move to right child
else if $x \neq e$ **then** move to left child
  else found.

Our experiment indicated that doing the 'left child' check first (i.e. order L) worked better for AVL, BBST, BBSTD, and DSL structures while R worked better for the RB-T, RB-B, WB, SBBST, and TRP structures. No significant difference between L and R was observed for BSTs. For skip lists, we do not have the flexibility to change the comparison order. The version 1 codes performed the comparisons in the order determined to be better. For BSTs, the order R was used.

In the version 2 codes the comparisons in each node took the standard form

if $x = e$ **then** found

else if $x < e$ **then** move to left child

else move to right child.

The version 2 restructuring code for BBSTs differed from that of Figure 10 in that the '>' test in the second, third, and forth **if** statements was changed to '≥'. No change was made in the corresponding if statements for RR and RL rotations. While this increased the number of comparisons, it reduced the run time.

We experimented with n = 10,000, 50,000, 100,000, and 200,000. For each n, the following experiments were conducted:

(a) start with an empty structure and perform n inserts;

(b) search for each item in the resulting structure once; items are searched for in the order they were inserted

(c) perform an alternating sequence of n inserts and n deletes; in this, the n elements inserted in (a) are deleted in the order they were inserted and n new elements are inserted

(d) search for each of the remaining n elements in the order they were inserted

(e) delete the n elements in the order they were inserted.

For each n, the above five part experiment was repeated ten times using different random permutations of distinct elements. For each permutation, we measured the total number of element comparisons performed and then averaged these over the ten permutations.

First, we report on the relative performance of SBBSTs, BBSTDs, and BBSTs. For this comparison, we used only version 1 of the code. Table 1 gives the average number of key comparisons performed for each of the five parts of the experiment. The three versions of our proposed data structure are very competitive on this measure. BBSTDs and BBSTs generally performed fewer comparisons than did SBBSTs. All three structures had a comparison count within 2% of one another. However, when we used ordered data rather than random data (Table 2), SBBSTs performed noticeably inferior to BBSTDs and BBSTs; the later two

n	operation	SBBST	BBSTD	BBST
	insert	212495	212223	212111
	search	194661	191599	191578
10,000	ins/del	416125	416967	416862
	search	194957	191666	191676
	delete	168033	166441	166487
	insert	1241080	1236682	1236114
	search	1152137	1135131	1134969
50,000	ins/del	2437918	2438083	2437639
	search	1153821	1134277	1134062
	delete	1018675	1007766	1007688
	insert	2635913	2624829	2623792
	search	2458079	2423988	2423613
100,000	ins/del	5183619	5180383	5179653
	search	2461221	2420282	2419990
	delete	2190798	2168049	2168110
	insert	5580139	5555190	5553256
	search	5223989	5148220	5147698
200,000	ins/del	10981441	10969578	10968053
	search	5229172	5144808	5144148
	delete	4692447	4641349	4641389

Table 1: The number of key comparisons on random inputs (version 1 code)

remained very competitive.

Tables 3 and 4 give the average heights of the trees using random data and using ordered data, respectively. The first number gives the height following part (a) of the experiment and the second following part (c). The numbers are identical for BBSTDs and BBSTs and slightly higher (lower) for SBBSTs using random (ordered) data.

The average number of rotations performed by each of the three structures is given in Tables 5 and 6. A single rotation (i.e., LL or RR) is denoted 'S' and a double rotation (i.e., LR or RL) denoted 'D'. In the case of BBSTs, double rotations have been divided into three categories: D = LR and RL rotations that do not perform a second substep rotation; DS = LR and RL rotations with a second substep rotation of type LL and RR; DD = LR and RL rotations with a second substep rotation of type LR and RL. BBSTDs and BBSTs

n	operation	SBBST	BBSTD	BBST
10,000	insert	170182	150554	150554
	search	188722	185530	185530
	ins/del	425305	315177	314998
	search	191681	184155	184155
	delete	215214	135311	135131
50,000	insert	991526	872967	872967
	search	1117174	1101481	1101481
	ins/del	2472808	1806346	1805439
	search	1116390	1098065	1098065
	delete	1277756	792717	791815
100,000	insert	2103808	1850548	1850548
	search	2384327	2354757	2354757
	ins/del	5249194	3823415	3821594
	search	2382759	2346118	2346128
	delete	2738294	1686397	1684584
200,000	insert	4449143	3903083	3903083
	search	5068632	4946753	4946753
	ins/del	11105525	8051695	8048058
	search	5065496	5001967	5001967
	delete	5842168	3580856	3577223

Table 2: The number of key comparisons on ordered inputs (version 1 code)

n	SBBST	BBSTD	BBST
10,000	17,17	16,16	16,16
50,000	20,20	19,19	19,19
100,000	21,21	20,20	20,20
200,000	22,23	21,21	21,21

Table 3: Height of the trees on random inputs (version 1 code)

n	SBBST	BBSTD	BBST
10,000	16,15	17,17	17,17
50,000	20,20	20,20	20,20
100,000	21,21	21,21	21,21
200,000	22,22	23,22	23,22

Table 4: Height of the trees on ordered inputs (version 1 code)

n	operation	SBBST		BBSTD		BBST			
		S	D	S	D	S	D	DS	DD
10,000	insert	2341	2220	5045	4314	5025	3938	151	93
	ins/del	4269	3216	10158	6311	10104	5849	232	103
	delete	1607	1110	5235	2104	5201	2018	51	28
50,000	insert	11719	11120	25216	21596	25059	19732	754	455
	ins/del	21330	16125	51238	31499	50979	29198	1161	531
	delete	8058	5648	26214	10462	26068	10033	248	131
100,000	insert	23450	22262	50283	43230	50047	39461	1527	920
	ins/del	42780	32203	102218	62967	101836	58491	2275	1046
	delete	16095	11306	52227	21022	51943	20147	496	260
200,000	insert	46934	44525	100664	86605	100205	79013	3054	1840
	ins/del	85283	64417	204459	125960	203568	116940	4593	2059
	delete	32233	22551	104344	41884	103826	40157	990	523

Table 5: The number of rotations on random inputs (version 1 code)

performed a comparable number of rotations on both data sets. However, on random data SBBSTs performed about half as many rotations as did BBSTDs and BBSTs. On ordered data, SBBSTs performed 15 to 20% fewer rotations on part (a), 34% fewer on part (c), and 51% fewer on part (e).

The run-time performance of the structures is significantly influenced by compiler and architectural features as well as the complexity of a key comparison. The results we report are from a SUN SPARC-5 using the UNIX C compiler cc with optimization option. Because of instruction pipelining features, cache replacement policies, etc., the measured run times are not always consistent with the compiler and architecture independent metrics reported in Tables 1 through 6 and later in Tables 11 through 16. For example, since the search codes for all tree based methods are essentially identical, we would expect methods with a smaller comparison count to have a smaller run time for parts (b) and (d) of the experiment. This was not always the case.

Tables 7 and 8 give the run times of the three BBST structures using integer keys and Tables 9 and 10 do this for the case of real (i.e., floating point) keys. The sum of the run

n	operation	SBBST		BBSTD		BBST			
		S	D	S	D	S	D	DS	DD
10,000	insert	9984	0	9985	2387	9985	2387	0	0
	ins/del	14997	0	16567	6130	16644	5797	25	154
	delete	4989	0	6570	3726	6647	3392	26	154
50,000	insert	49980	0	49983	11956	49983	11956	0	0
	ins/del	74996	0	82862	30659	83247	28982	137	770
	delete	24987	0	32859	18686	33242	17018	136	766
100,000	insert	99979	0	99983	23917	99983	23917	0	0
	ins/del	149996	0	165738	61327	166504	57969	280	1540
	delete	49986	0	65733	37392	66505	34040	278	1536
200,000	insert	199978	0	199982	47839	199982	47839	0	0
	ins/del	299996	0	331473	122653	333012	115938	559	3078
	delete	99985	0	131478	74795	133016	68086	557	3076

Table 6: The number of rotations on ordered inputs (version 1 code)

time for parts (a) – (e) of the experiment is graphed in Figure 14. For random data, SBBSTs significantly and consistently outperformed BBSTDs and BBSTs. On ordered data, however, BBSTDs were slightly faster than BBSTs and both were significantly faster than SBBSTs.

Since BBSTs generated trees with the least search cost, we expect BBSTs to outperform SBBSTs and BBSTDs in applications where the comparison cost is very high relative to that of other operations and searches are done with a much higher frequency than inserts and deletes. However, with the mix of operations used in our tests, SBBSTs are the clear choice for random inputs and BBSTDs for ordered inputs.

In comparing with the other structures, our tables repeat the data for BBSTs. The reader may make the comparison with SBBSTs and BBSTDs.

The average number of comparisons for each of the five parts of the experiment are given in Table 11 for the version 1 implementation. On the comparison measure, AVL, RB-B, WB, and BBSTs are the front runners and are quite competitive with one another. On parts (a) (insert n elements) and (c) (insert n and delete n elements), AVL trees performed best while on the two search tests ((b) and (d)) and the deletion test (e), BBSTs performed best.

n	operation	SBBST	BBSTD	BBST
10,000	insert	0.27	0.30	0.34
	search	0.06	0.06	0.07
	ins/del	0.57	0.62	0.70
	search	0.06	0.06	0.06
	delete	0.22	0.25	0.26
50,000	insert	1.48	1.61	1.75
	search	0.35	0.36	0.37
	ins/del	2.90	3.47	3.84
	search	0.36	0.38	0.39
	delete	1.13	1.47	1.62
100,000	insert	3.00	3.57	3.80
	search	0.78	0.83	0.84
	ins/del	6.28	7.78	8.41
	search	0.83	0.87	0.88
	delete	2.54	3.31	3.58
200,000	insert	6.56	7.74	8.37
	search	1.80	1.89	1.89
	ins/del	13.89	17.32	18.57
	search	1.86	1.98	1.98
	delete	5.64	7.41	8.02

Time Unit : *sec*

Table 7: Run time on random inputs using integer keys (version 1 code)

n	operation	SBBST	BBSTD	BBST
10,000	insert	0.32	0.20	0.27
	search	0.05	0.03	0.05
	ins/del	0.58	0.43	0.57
	search	0.07	0.03	0.03
	delete	0.20	0.17	0.23
50,000	insert	1.38	1.20	1.10
	search	0.25	0.20	0.20
	ins/del	2.63	2.18	2.40
	search	0.25	0.20	0.20
	delete	0.95	0.92	1.05
100,000	insert	3.43	2.23	2.53
	search	0.72	0.45	0.42
	ins/del	5.97	4.70	5.13
	search	0.55	0.47	0.42
	delete	2.10	1.98	2.15
200,000	insert	6.65	4.95	5.25
	search	1.20	0.92	0.90
	ins/del	13.13	10.23	10.88
	search	1.17	0.90	0.90
	delete	4.63	4.25	4.58

Time Unit : *sec*

Table 8: Run time on ordered inputs using integer keys (version 1 code)

n	operation	SBBST	BBSTD	BBST
10,000	insert	0.23	0.34	0.36
	search	0.07	0.10	0.10
	ins/del	0.44	0.75	0.79
	search	0.08	0.10	0.10
	delete	0.17	0.29	0.30
50,000	insert	1.43	1.76	1.93
	search	0.47	0.53	0.52
	ins/del	2.76	3.89	4.22
	search	0.50	0.54	0.55
	delete	1.13	1.62	1.76
100,000	insert	2.96	3.94	4.36
	search	1.08	1.17	1.16
	ins/del	6.11	8.58	9.30
	search	1.12	1.20	1.22
	delete	2.50	3.66	3.95
200,000	insert	6.85	8.92	9.33
	search	2.41	2.58	2.57
	ins/del	13.86	19.49	20.46
	search	2.49	2.69	2.66
	delete	5.61	8.25	8.80

Time Unit : *sec*

Table 9: Run time on random real inputs (version 1 code)

n	operation	SBBST	BBSTD	BBST
10,000	insert	0.27	0.23	0.20
	search	0.08	0.07	0.07
	ins/del	0.53	0.50	0.43
	search	0.08	0.07	0.05
	delete	0.18	0.23	0.20
50,000	insert	1.43	1.25	1.12
	search	0.40	0.30	0.30
	ins/del	2.80	2.17	2.37
	search	0.40	0.30	0.30
	delete	1.07	0.90	0.97
100,000	insert	3.28	2.58	2.77
	search	0.90	0.62	0.63
	ins/del	6.15	4.70	5.13
	search	0.87	0.62	0.63
	delete	2.35	1.93	2.10
200,000	insert	7.37	4.55	4.92
	search	1.85	1.32	1.32
	ins/del	13.35	10.03	10.93
	search	1.87	1.33	1.33
	delete	5.08	4.17	4.43

Time Unit : *sec*

Table 10: Run time on ordered real inputs (version 1 code)

n	operation	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	264175	211401	262838	211886	211916	212111	276247	296866	224757
	search	254175	193253	194606	194291	194153	191578	258089	258662	255072
	ins/del	516853	411220	515184	414990	414635	416862	923524	601137	519430
	search	252200	193141	197399	195525	194442	191676	256578	254119	256124
	delete	215555	167312	200218	167455	167531	166487	526242	242743	231745
50,000	insert	1560958	1234911	1550701	1236968	1238628	1236114	1640660	1717037	1357076
	search	1510958	1147273	1150466	1146754	1149970	1134969	1512093	1503452	1537547
	ins/del	3061868	2417733	3058045	2424944	2431281	2437639	5351715	3456045	2996512
	search	1500504	1145808	1173662	1152764	1151578	1134062	1499657	1497081	1501731
	delete	1316917	1013535	1242426	1013144	1015988	1007688	3077266	1451835	1373858
100,000	insert	3329780	2623894	3305332	2626314	2631411	2623792	3513401	3632046	2919371
	search	3229780	2445659	2451137	2446466	2453855	2423613	3244497	3247143	3188621
	ins/del	6537563	5137280	6564352	5154118	5170695	5179653	11545200	7476441	6399463
	search	3208453	2443038	2502098	2457531	2456748	2419990	3229747	3310823	3225343
	delete	2839934	2181327	2692672	2177946	2185213	2168110	6561272	3177135	2981173
200,000	insert	7076132	5553640	7016676	5558174	5571133	5553256	7483199	7682439	6178596
	search	6876132	5191730	5209189	5199786	5215568	5147698	6887196	6797942	6697223
	ins/del	13907058	10862426	13940982	10921880	10956496	10968053	24207106	15543559	13377747
	search	6830718	5186737	5332771	5223154	5220965	5144148	6814733	6916150	6680642
	delete	6095324	4664876	5800203	4664344	4680768	4641389	13811271	6700557	6149268

Table 11: The number of key comparisons on random inputs (version 1 code)

n	operation	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	277234	376228	241383	171017	150554	435199	135989	247129
	search	191917	188246	190106	188722	185530	262423	271087	256706
	ins/del	421032	718040	508810	425843	314998	983676	390899	354566
	search	195133	189494	190090	191681	184155	249694	269031	250538
	delete	104038	276136	218216	214930	135131	468244	193080	84392
50,000	insert	1618930	2233658	1436225	995720	872967	2585557	825390	1422120
	search	1120497	1117001	1120495	1117174	1101481	1509152	1540082	1467217
	ins/del	2418422	4311748	3055100	2475487	1805439	6019215	2194668	1973416
	search	1124001	1168633	1126126	1116390	1098065	1481819	1568903	1449810
	delete	607478	1719212	1323918	1276262	791815	2785792	1181612	486498
100,000	insert	3437858	4767564	3072389	2112201	1850548	5521408	1724473	2925618
	search	2390963	2383979	2390961	2384327	2354757	3218246	3564282	2970715
	ins/del	5111850	9223606	6510188	5254541	3821594	12788447	4438266	4406427
	search	2397971	2487243	2402224	2382759	2346128	3163554	3281308	3277089
	delete	1289954	3737982	2847792	2735270	1684584	5971196	2403622	961283
200,000	insert	7275714	10135418	6544713	4465935	3903083	11743159	3428355	6403207
	search	5081893	5067933	5081891	5068632	4946753	6836428	7174727	6448304
	ins/del	10773706	19647336	13820364	11116226	8048058	27076911	9054078	9062233
	search	5095909	5274461	5104418	5065496	5001967	6727017	7006341	6458321
	delete	2729906	8075474	6095538	5836096	3577223	12741948	5094044	1995215

Table 12: The number of key comparisons on ordered inputs (version 1 code)

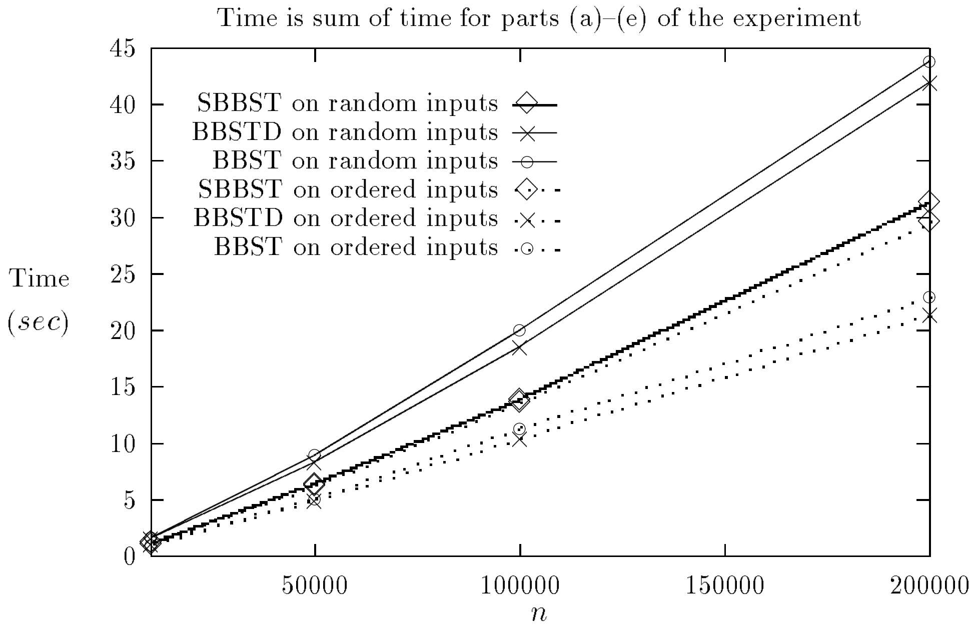


Figure 14: Run time on real inputs (version 1 code)

Table 12 gives the number of comparisons performed when ordered data (i.e., the elements in part (a) are $1, 2, \ldots, n$ and are inserted in this order) and those in part (c) are $n+1, \ldots, 2n$ (in this order) is used instead of random permutations of distinct elements. This experiment attempts to model realistic situations in which the inserted elements are in "nearly sorted order". BSTs were not included in this test as they perform very poorly with ordered data taking $O(n^2)$ time to insert n times. The computer time needed to perform this test on BSTs was determined to be excessive. This test exhibited greater variance in performance. Among the deterministic structures, BBSTs outperformed the others in parts (a) – (d) while AVL trees were ahead in part (e). For part (a), BBSTs performed approximately 45% fewer comparisons than did AVL trees and approximately 12% fewer than WB trees. The randomized structure TRP was the best of the eight structures reported in Table 12 for part (a). It performed approximately 10% fewer comparisons than did BBST trees. However, the BBST remained best overall on parts (b), (c), and (d).

The heights of the trees (number of levels in the case of DSL and SKIP) for the exper-

n	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	31,31	16,16	17,18	16,17	17,17	16,16	12,11	32,31	8,8
50,000	38,38	19,19	20,21	19,20	20,20	19,19	13,12	38,37	9,9
100,000	41,41	20,20	21,22	20,21	21,22	20,20	14,13	41,40	9,9
200,000	44,43	21,21	22,24	21,22	23,23	21,21	15,14	43,44	9,9

Table 13: Height of the trees on random inputs (version 1 code)

n	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	14,14	20,20	24,24	16,15	17,17	14,13	33,34	8,8
50,000	16,16	23,23	29,28	20,20	20,20	16,16	41,41	9,9
100,000	17,17	25,25	31,30	21,21	21,21	17,17	46,41	9,9
200,000	18,18	27,27	33,32	22,22	23,22	18,18	47,46	9,9

Table 14: Height of the trees on ordered inputs (version 1 code)

iments with random and ordered data are given in Tables 13 and 14 respectively. The first number in each table entry is the tree height after part (a) of the experiment and the second, the height after part (c). In all cases, the number of levels using skip lists is fewest. However, among the tree structures, AVL and BBST trees have least height on random data and AVL has least with ordered data.

Tables 15 and 16, respectively, give the number of rotations performed by each of the deterministic tree schemes for experiment parts (a), (c), and (e). Note that none of the schemes performs rotations during a search.

On ordered data, BBSTs perform about 25% more rotations than do the remaining structures. These remaining structures perform about the same number of rotations. On random data, AVL trees, bottom-up red-black trees and WB trees perform a comparable number of rotations. Top-down red-black trees and BBST trees perform a significantly larger number of rotations. In fact, BBSTs perform about twice as many rotations as AVL trees.

The average run times for the random data tests are given in Table 17 and in Table 18 for the ordered data test. Both of these use integer keys. The times using real keys are

n	operation	AVL		RB-T		RB-B		WB		BBST			
		S	D	S	D	S	D	S	D	S	D	DS	DD
10,000	insert	2328	2322	1964	1955	1946	1933	2274	2065	5025	3938	151	93
	ins/del	4343	3224	14773	8213	4053	2591	4256	2978	10104	5849	232	103
	delete	1645	1120	9558	2678	1845	1166	1595	1022	5201	2018	51	28
50,000	insert	11664	11614	9822	9815	9710	9689	11355	10352	25059	19732	754	455
	ins/del	21585	16214	81895	45180	20255	12979	21266	14975	50979	29198	1161	531
	delete	8231	5630	54806	13431	9196	5844	7963	5194	26068	10033	248	131
100,000	insert	23316	23254	19593	19677	19340	19414	22723	20730	50047	39461	1527	920
	ins/del	43243	32361	196769	103835	40618	25919	42567	29898	101836	58491	2275	1046
	delete	16466	11264	119825	26953	18530	11708	16024	10420	51943	20147	496	260
200,000	insert	46631	46518	39290	39291	38797	38793	45458	41480	100205	79013	3054	1840
	ins/del	86218	64712	394187	209941	80892	52030	84927	59911	203568	116940	4593	2059
	delete	33047	22477	247905	54046	37083	23379	31984	20800	103826	40157	990	523

Table 15: The number of rotations on random inputs (version 1 code)

		AVL		RB-T		RB-B		WB		BBST			
n	operation	S	D	S	D	S	D	S	D	S	D	DS	DD
10,000	insert	9986	0	9980	0	9976	0	9984	0	9985	2387	0	0
	ins/del	14996	0	14999	0	14995	0	14997	0	16644	5797	25	154
	delete	4990	0	4983	1	4989	0	4989	0	6647	3392	26	154
50,000	insert	49984	0	49977	0	49971	0	49980	0	49983	11956	0	0
	ins/del	74994	0	75000	0	74994	0	74996	0	83247	28982	137	770
	delete	24988	0	24978	1	24986	0	24987	0	33242	17018	136	766
100,000	insert	99983	0	99975	0	99969	0	99979	0	99983	23917	0	0
	ins/del	149994	0	150000	0	149994	0	149996	0	166504	57969	280	1540
	delete	49987	0	49977	1	49985	0	49986	0	66505	34040	278	1536
200,000	insert	199982	0	199973	0	199967	0	199978	0	199982	47839	0	0
	ins/del	299994	0	300000	0	299994	0	299996	0	333012	115938	559	3078
	delete	99986	0	99976	1	99984	0	99985	0	133016	68086	557	3076

Table 16: The number of rotations on ordered inputs (version 1 code)

n	operation	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.08	0.12	0.15	0.12	0.20	0.34	0.19	0.18	0.24
	search	0.05	0.05	0.05	0.06	0.05	0.07	0.09	0.09	0.18
	ins/del	0.14	0.21	0.36	0.22	0.39	0.70	0.49	0.33	0.45
	search	0.05	0.05	0.05	0.05	0.05	0.06	0.09	0.09	0.18
	delete	0.05	0.08	0.12	0.09	0.16	0.26	0.20	0.08	0.16
50,000	insert	0.65	0.79	0.98	0.73	1.18	1.75	1.10	1.01	1.36
	search	0.40	0.36	0.36	0.36	0.35	0.37	0.58	0.56	1.25
	ins/del	1.04	1.48	2.50	1.26	2.22	3.84	2.77	1.86	2.73
	search	0.40	0.41	0.44	0.36	0.36	0.39	0.57	0.56	1.16
	delete	0.39	0.54	1.01	0.51	0.94	1.62	1.16	0.51	1.10
100,000	insert	1.34	1.57	2.10	1.54	2.54	3.80	2.46	2.23	2.84
	search	0.88	0.80	0.80	0.83	0.78	0.84	1.36	1.30	2.63
	ins/del	2.36	3.21	5.52	2.74	4.86	8.41	6.35	4.10	6.13
	search	0.93	0.94	1.00	0.84	0.83	0.88	1.33	1.29	2.61
	delete	0.88	1.24	2.26	1.14	2.11	3.58	2.64	1.23	2.41
200,000	insert	2.79	3.37	4.41	3.18	5.21	8.37	5.56	4.70	6.25
	search	2.00	1.80	1.81	1.81	1.78	1.89	3.03	2.91	5.85
	ins/del	5.24	6.99	12.51	5.99	10.54	18.57	14.29	8.95	13.29
	search	2.08	2.12	2.25	1.91	1.87	1.98	3.04	2.93	5.81
	delete	2.01	2.69	5.06	2.51	4.55	8.02	5.84	2.76	5.35

Time Unit : *sec*

Table 17: Run time on random inputs using integer keys (version 1 code)

given in Tables 19 and 20. The sum of the run time for parts (b) and (d) of the experiment is graphed in Figure 15 for random data and in Figure 16 for ordered data. The graph of Figure 17 shows only one line MIX for AVL, RB-T, RB-B, WB, and BBST while that of Figure 18 shows MIX for AVL, RB-T, RB-B, and WB as the times for these are very close. With integer keys and random data, unbalanced binary search trees (BSTs) outperformed each of the remaining structures. The next best performance was exhibited by bottom-up red-black trees. They did marginally better than AVL trees. The remaining structures have a noticeably inferior structure. For ordered integer keys, BSTs take more time than we were willing to expend. Of the remaining structures, treaps generally performed best on parts (a), (c), and (e) while BBSTs did best on parts (b) and (d).

n	operation	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.12	0.17	0.12	0.18	0.27	0.23	0.08	0.20
	search	0.05	0.03	0.03	0.07	0.05	0.07	0.05	0.12
	ins/del	0.18	0.32	0.20	0.35	0.57	0.42	0.17	0.20
	search	0.05	0.05	0.05	0.05	0.03	0.07	0.05	0.13
	delete	0.05	0.10	0.07	0.13	0.23	0.15	0.05	0.07
50,000	insert	0.75	1.02	0.92	1.25	1.10	0.98	0.47	0.92
	search	0.32	0.27	0.27	0.28	0.20	0.33	0.32	0.62
	ins/del	1.28	2.17	1.25	2.20	2.40	2.03	0.80	1.07
	search	0.28	0.28	0.27	0.28	0.20	0.30	0.37	0.62
	delete	0.30	0.75	0.37	0.85	1.05	0.65	0.30	0.27
100,000	insert	1.50	2.52	1.70	2.58	2.53	2.58	0.90	1.72
	search	0.70	0.60	0.57	0.70	0.42	0.70	0.63	1.23
	ins/del	2.60	4.68	2.53	4.78	5.13	4.42	1.52	2.43
	search	0.63	0.60	0.55	0.62	0.42	0.70	0.58	1.35
	delete	0.62	1.65	0.78	1.87	2.15	1.42	0.45	0.55
200,000	insert	3.12	4.82	3.38	5.67	5.25	4.72	1.80	3.52
	search	1.38	1.30	1.22	1.33	0.90	1.60	1.25	2.70
	ins/del	5.15	10.40	5.35	10.40	10.88	9.48	3.10	5.13
	search	1.33	1.33	1.18	1.32	0.90	1.50	1.28	2.72
	delete	1.35	3.63	1.68	4.12	4.58	2.98	0.93	1.12

Time Unit : *sec*

Table 18: Run time on ordered inputs using integer keys (version 1 code)

n	operation	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.14	0.15	0.21	0.17	0.23	0.36	0.22	0.23	0.30
	search	0.09	0.07	0.09	0.10	0.08	0.10	0.13	0.13	0.21
	ins/del	0.24	0.27	0.51	0.32	0.38	0.79	0.62	0.41	0.53
	search	0.09	0.08	0.09	0.10	0.08	0.10	0.12	0.12	0.21
	delete	0.09	0.09	0.17	0.14	0.14	0.30	0.28	0.11	0.19
50,000	insert	0.94	0.97	1.22	0.86	1.29	1.93	1.48	1.19	1.67
	search	0.64	0.52	0.50	0.51	0.51	0.52	0.87	0.71	1.44
	ins/del	1.68	1.77	2.74	1.53	2.29	4.22	3.93	2.17	3.15
	search	0.66	0.55	0.56	0.54	0.56	0.55	0.86	0.71	1.33
	delete	0.63	0.67	1.10	0.72	0.92	1.76	1.80	0.69	1.22
100,000	insert	2.06	1.85	2.34	1.90	2.66	4.36	3.05	2.67	3.61
	search	1.43	1.13	1.09	1.13	1.14	1.16	1.84	1.66	3.00
	ins/del	3.63	3.93	6.18	3.33	4.96	9.30	8.45	4.84	7.10
	search	1.45	1.26	1.27	1.17	1.26	1.22	1.83	1.65	3.01
	delete	1.39	1.50	2.51	1.55	2.03	3.95	3.91	1.61	2.75
200,000	insert	4.34	3.95	5.20	3.88	5.56	9.33	6.77	5.81	7.90
	search	3.19	2.49	2.42	2.50	2.45	2.57	4.14	3.67	6.62
	ins/del	8.01	8.25	13.78	7.29	10.65	20.46	18.88	10.48	15.83
	search	3.21	2.83	2.86	2.62	2.74	2.66	4.08	3.73	6.74
	delete	3.11	3.27	5.55	3.41	4.43	8.80	8.56	3.54	6.04

Time Unit : *sec*

Table 19: Run time on random real inputs (version 1 code)

With real keys and random data, BSTs did not outperform the remaining structures. Now, the five balanced binary tree structure became quite competitive with respect to the search operations (i.e., parts (b) and (d)). RB-B generally outperformed the other structures on parts (a), (c), and (e). Using ordered real keys, the treap was the clear winner on parts (a), (c), and (e) while BBSTs handily outperformed the remaining structures on parts (b) and (d).

Some of the experimental results using version 2 of the code are shown in Tables 21– 24. On the comparison measure, with random data (Table 21), skip lists performed best on part (a). Of the deterministic methods, BBSTs slightly outperformed the others on part (a). On parts (b) – (e), AVL, RB-T, RB-B, WB, and BBSTs were quite competitive and

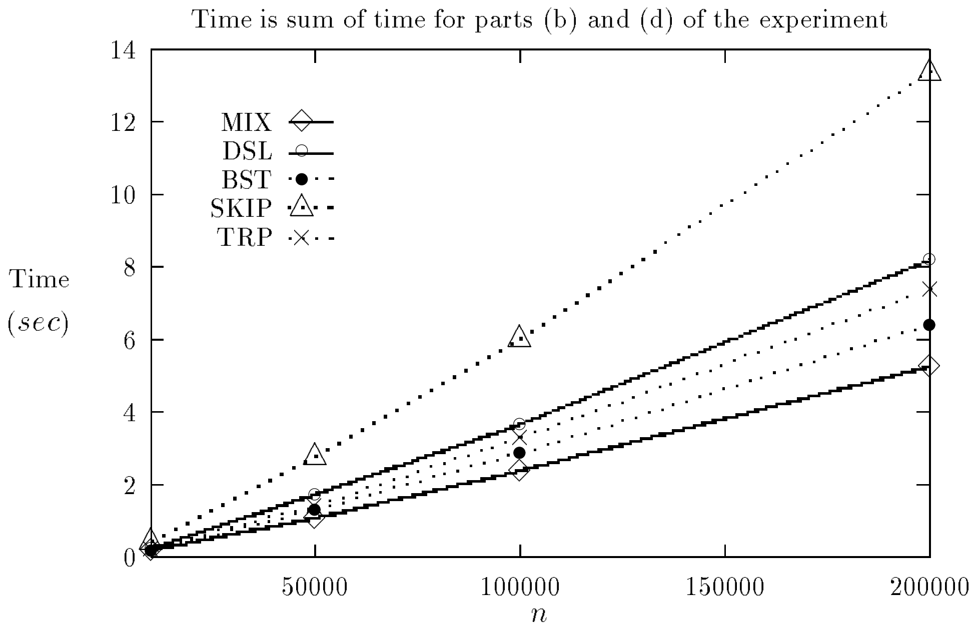


Figure 15: Run time on random real inputs (version 1 code)

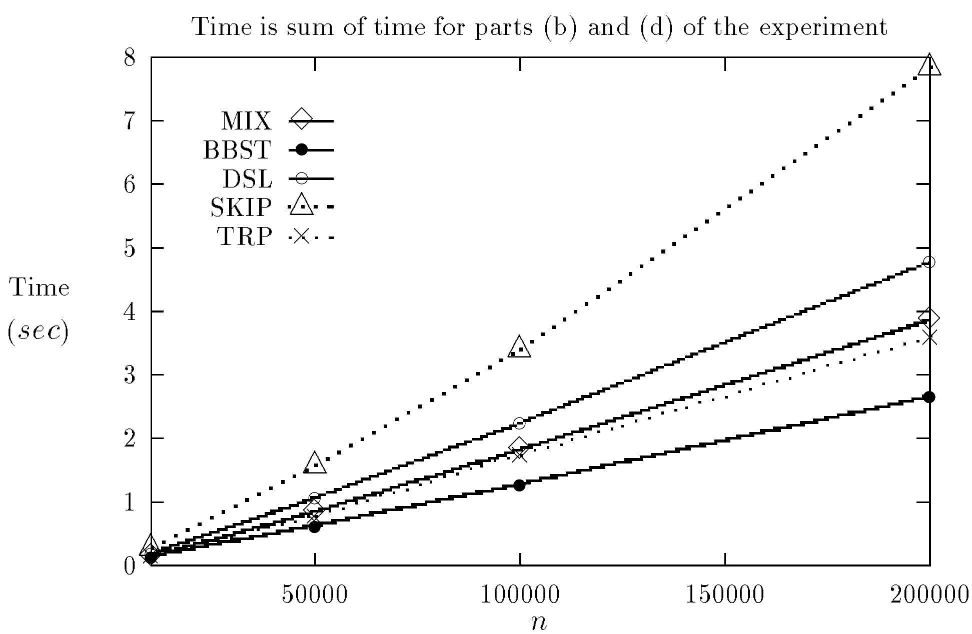


Figure 16: Run time on ordered real inputs (version 1 code)

n	operation	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.13	0.22	0.15	0.25	0.20	0.25	0.12	0.30
	search	0.07	0.08	0.07	0.07	0.07	0.10	0.07	0.15
	ins/del	0.23	0.42	0.27	0.40	0.43	0.47	0.18	0.28
	search	0.07	0.05	0.08	0.08	0.05	0.08	0.08	0.12
	delete	0.07	0.17	0.08	0.15	0.20	0.20	0.05	0.07
50,000	insert	1.15	1.58	1.12	1.85	1.12	1.30	0.67	1.35
	search	0.42	0.42	0.43	0.40	0.30	0.53	0.38	0.82
	ins/del	1.28	2.75	1.57	2.57	2.37	3.02	0.92	1.40
	search	0.40	0.42	0.42	0.48	0.30	0.53	0.40	0.75
	delete	0.38	0.95	0.55	0.93	0.97	1.15	0.33	0.35
100,000	insert	1.77	3.23	2.12	3.35	2.77	3.13	1.17	2.42
	search	0.90	0.87	0.90	0.88	0.63	1.12	0.92	1.70
	ins/del	3.00	6.00	3.42	5.38	5.13	6.32	1.92	3.22
	search	0.97	0.92	0.88	0.98	0.63	1.12	0.82	1.70
	delete	0.87	2.08	1.17	2.05	2.10	2.40	0.70	0.67
200,000	insert	3.92	6.42	4.27	7.25	4.92	6.03	2.58	4.93
	search	1.92	1.87	1.92	1.88	1.32	2.40	1.85	3.87
	ins/del	5.78	13.80	7.33	11.88	10.93	13.72	3.75	6.67
	search	1.90	1.93	1.92	2.13	1.33	2.38	1.75	3.97
	delete	1.67	4.55	2.48	4.45	4.43	5.10	1.40	1.35

Time Unit : *sec*

Table 20: Run time on ordered real inputs (version 1 code)

outperformed BSTs and the randomized schemes. BBSTs performed best on parts (b) and (d), RB-Ts did best on part (e) and RB-B and AVL did best on part (c). In comparing the results of Table 21 to those of Table 11 (using version 1 code), we see that the change to version 2 generally increased the comparison cost of the deterministic tree structures by about 25%. For the DSL, the change in code had mixed results. Notice that for RB-T and DSLs, the comparison count for parts (a), (c), and (e) are the same as for the version 1 code. This is because for inserts and deletes, it is necessary to do the equal check first when using these structures. For SKIPs the count is the same for all five parts as the version 1 and 2 codes are the same.

With ordered data (Table 22), treaps required the fewest comparisons for part (a). Skip lists did best on parts (c) and (e), and AVL trees generally outperformed the other structures on parts (b) and (d). Once again, the comparison counts were generally higher using the version 2 code than using the version 1 code.

Run time data using real keys is given in Tables 23 and 24. The sum of the run time for parts (b) and (d) of the experiment is graphed in Figure 17 for random data and in Figure 18 for ordered data. The graph of Figure 17 shows only one line MIX for AVL, RB-T, RB-B, WB, and BBST while that of Figure 18 shows MIX for AVL, RB-T, RB-B, and WB as the times for these are very close. With random data, RB-B generally performed best on part (a), on parts (b) and (d), the front runner varied among AVL, RB-T, and WB, and on parts (c) and (e) RB-Bs generally did best. On ordered data, TRPs did best on parts (a), (c), and (e) while BBSTs did best on parts (b) and (d).

8 Conclusion

We have developed a new weight balanced data structure called β-BBST. This was developed for the representation of a dictionary. In developing the insert/delete algorithms, we sought to minimize the search cost of the resulting tree. Our experimental results show that BBSTs generally have the best search cost of the structures considered. Furthermore, this translates

n	operation	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	332753	262198	262838	262726	263177	260896	276247	375698	224757
	search	322753	241557	242258	242262	242824	240126	348403	329411	255072
	ins/del	650901	514371	515184	513920	515732	518124	923524	755629	519430
	search	318749	241536	247130	243191	242867	240126	335613	320612	256124
	delete	271004	206558	200218	206721	207622	207210	526242	300619	231745
50,000	insert	1983939	1546988	1550701	1549795	1554520	1539666	1640660	2184066	1357076
	search	1933939	1443879	1447870	1446679	1452920	1435927	2043618	1921255	1537547
	ins/del	3892221	3043090	3058045	3040654	3055092	3061443	5351715	4393520	2996512
	search	1913068	1443837	1476158	1451163	1452625	1435726	1969926	1909919	1501731
	delete	1674128	1267637	1242426	1268881	1275612	1270935	3077266	1815736	1373858
100,000	insert	4245062	3297162	3305332	3302792	3314410	3281959	3513401	4637264	2919371
	search	4145062	3090057	3098143	3096011	3111095	3074661	4387427	4161175	3188621
	ins/del	8336846	6490752	6564352	6486464	6520729	6528606	11545200	9484761	6399463
	search	4102672	3089826	3176862	3105465	3110184	3074305	4270168	4224698	3225343
	delete	3623179	2738267	2692672	2740846	2756006	2744369	6561272	4008111	2981173
200,000	insert	9045367	6999791	7016676	7012317	7040203	6969465	7483199	9834444	6178596
	search	8845367	6584279	6603044	6599643	6633218	6554714	9373163	8752856	6697223
	ins/del	17782478	13790643	13940982	13789492	13862467	13867876	24207106	19825904	13377747
	search	8757433	6585758	6747566	6618833	6630334	6554354	8995685	8889053	6680642
	delete	7800524	5882302	5800203	5889983	5923552	5893982	13811271	8456931	6149268

Table 21: The number of key comparisons on random inputs (version 2 code)

n	operation	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	267234	376228	442766	302034	261108	435199	216958	247129
	search	237262	239442	247706	237298	243110	372444	332060	256706
	ins/del	493028	718040	727620	562808	558770	983676	482499	354566
	search	240910	238834	246330	238320	242736	349730	344982	250538
	delete	178076	276136	208216	204930	239028	468244	183080	84392
50,000	insert	1568930	2233658	2672450	1791440	1545934	2585557	1375770	1422120
	search	1418962	1421560	1459588	1420858	1455936	2159176	1990474	1467217
	ins/del	2881762	4311748	4360200	3301668	3251450	6019215	2742877	1973416
	search	1419154	1444824	1444258	1424442	1452494	2131862	1956194	1449810
	delete	1064956	1719212	1273918	1226262	1427504	2785792	1131612	486498
100,000	insert	3337858	4767564	5744778	3824402	3301096	5521408	2898893	2925618
	search	3037892	3043084	3119128	3041676	3121098	4618272	4538718	2970715
	ins/del	6113530	9223606	9320376	7027676	6930932	12788447	5492066	4406427
	search	3038276	3089612	3088470	3048844	3114012	4563600	4158994	3277089
	delete	2279908	3737982	2747792	2635270	3056908	5971196	2303622	961283
200,000	insert	7075714	10135418	12289426	8131870	7006166	11743159	5756575	6403207
	search	6475750	6486128	6638204	6483310	6646168	9836456	9102954	6448304
	ins/del	12927066	19647336	19840728	14904040	14671602	27076911	11290926	9062233
	search	6476518	6579184	6576890	6497646	6634260	9727066	8918638	6458321
	delete	4859812	8075474	5895538	5636096	6529928	12741948	4894044	1995215

Table 22: The number of key comparisons on ordered inputs (version 2 code)

n	operation	BST	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.15	0.14	0.20	0.18	0.25	0.36	0.23	0.25	0.31
	search	0.10	0.08	0.10	0.11	0.09	0.11	0.13	0.16	0.21
	ins/del	0.27	0.27	0.52	0.34	0.47	0.80	0.64	0.50	0.54
	search	0.10	0.08	0.10	0.11	0.09	0.11	0.13	0.14	0.21
	delete	0.10	0.10	0.20	0.14	0.18	0.32	0.29	0.14	0.19
50,000	insert	1.02	0.98	1.15	0.89	1.46	1.88	1.44	1.34	1.65
	search	0.69	0.55	0.57	0.55	0.57	0.55	0.89	0.83	1.42
	ins/del	1.79	1.80	2.99	1.59	2.93	3.97	3.82	2.44	3.16
	search	0.71	0.60	0.63	0.55	0.57	0.56	0.87	0.79	1.32
	delete	0.67	0.67	1.22	0.66	1.19	1.63	1.80	0.75	1.21
100,000	insert	2.15	2.00	2.58	1.90	3.18	4.01	3.11	2.95	3.69
	search	1.52	1.21	1.24	1.18	1.23	1.23	1.97	1.84	3.04
	ins/del	3.88	3.92	6.74	3.46	6.28	8.73	8.50	5.39	7.18
	search	1.55	1.32	1.45	1.25	1.29	1.27	1.95	1.82	2.98
	delete	1.51	1.49	2.75	1.45	2.57	3.64	3.93	1.73	2.77
200,000	insert	5.04	4.45	5.79	4.28	6.92	9.20	7.05	6.81	8.01
	search	3.43	2.63	2.70	2.64	2.73	2.69	4.43	4.00	6.60
	ins/del	8.92	8.87	15.36	7.88	13.85	19.53	19.55	12.17	16.11
	search	3.43	2.98	3.13	2.73	2.83	2.77	4.37	4.02	6.70
	delete	3.33	3.32	6.08	3.20	5.65	8.24	8.91	3.88	6.04

Time Unit : *sec*

Table 23: Run time on random real inputs (version 2 code)

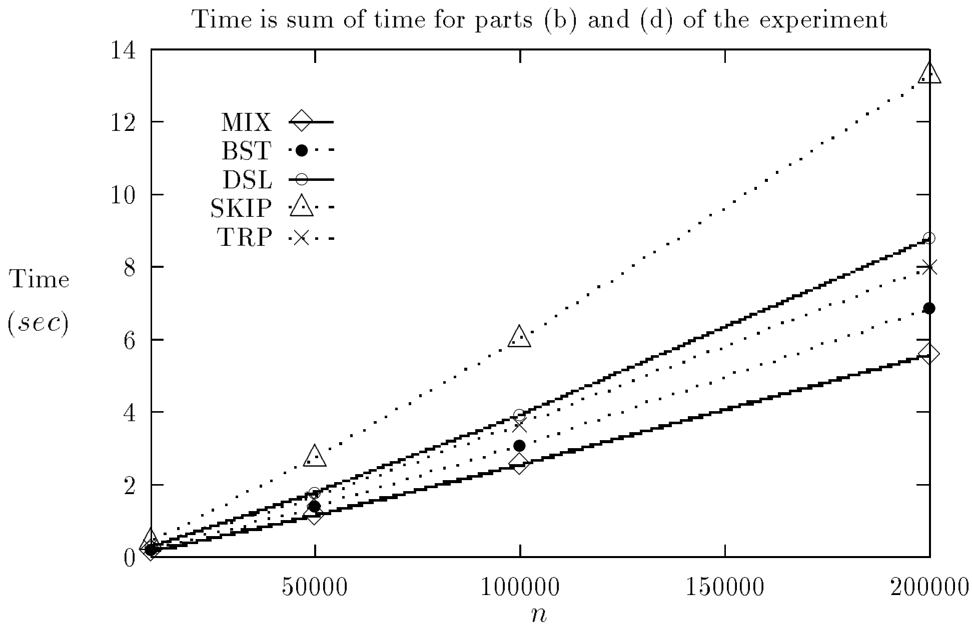


Figure 17: Run time on random real inputs (version 2 code)

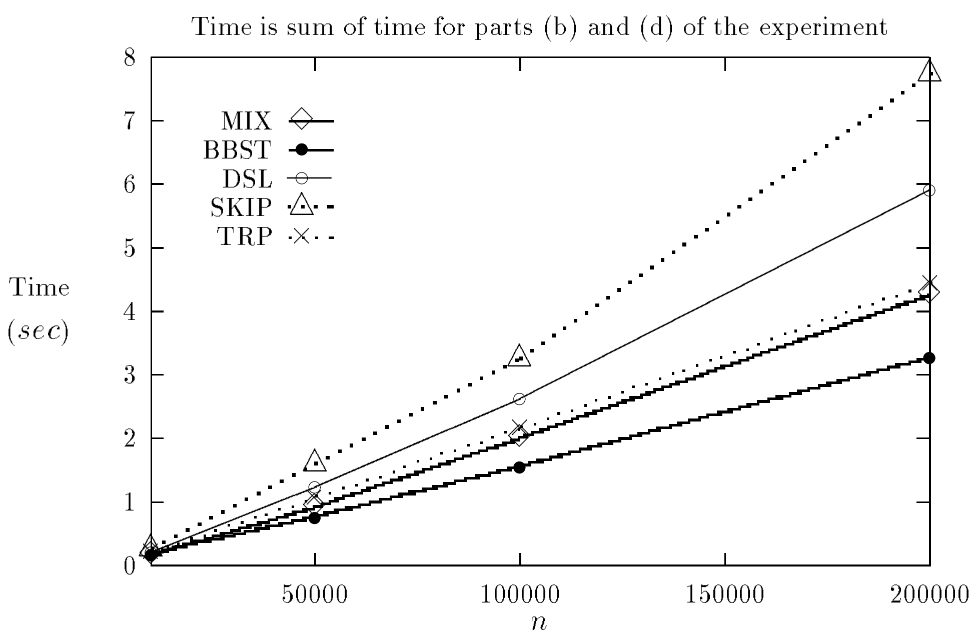


Figure 18: Run time on ordered real inputs (version 2 code)

n	operation	AVL	RB-T	RB-B	WB	BBST	DSL	TRP	SKIP
10,000	insert	0.17	0.23	0.28	0.27	0.30	0.23	0.15	0.30
	search	0.08	0.08	0.12	0.08	0.08	0.12	0.12	0.13
	ins/del	0.23	0.43	0.40	0.47	0.60	0.48	0.17	0.27
	search	0.08	0.08	0.07	0.08	0.08	0.08	0.10	0.13
	delete	0.08	0.15	0.12	0.17	0.20	0.20	0.08	0.05
50,000	insert	0.83	1.45	1.43	1.57	1.37	1.35	0.82	1.18
	search	0.45	0.48	0.48	0.47	0.38	0.60	0.50	0.83
	ins/del	1.35	2.65	1.95	2.75	2.47	3.05	1.05	1.42
	search	0.45	0.47	0.45	0.47	0.37	0.63	0.58	0.77
	delete	0.45	1.05	0.50	1.00	1.03	1.17	0.43	0.33
100,000	insert	1.78	2.75	2.73	3.43	2.63	3.23	1.33	2.18
	search	0.97	0.98	1.00	1.03	0.77	1.30	1.15	1.55
	ins/del	2.85	6.22	3.98	6.00	5.33	6.37	2.02	3.33
	search	0.97	1.10	0.98	1.02	0.77	1.32	1.03	1.70
	delete	0.97	2.18	1.05	2.15	2.22	2.43	0.63	0.67
200,000	insert	3.78	6.08	5.43	7.18	5.37	6.07	2.87	5.23
	search	2.08	2.13	2.13	2.17	1.63	3.10	2.27	3.47
	ins/del	6.13	13.93	8.48	13.42	11.33	13.60	4.10	7.02
	search	2.12	2.15	2.13	2.17	1.63	2.80	2.18	4.27
	delete	2.03	4.75	2.27	4.77	4.72	5.18	1.35	1.35

Time Unit : *sec*

Table 24: Run time on ordered real inputs (version 2 code)

into reduced search time when the key comparison cost is relatively high (e.g., for real keys). The insert and delete algorithms for β-BBSTs are not as efficient as those for other dictionary structures (such as AVL trees). As a result, we recommend β-BBSTs for environments where searches are done with much greater frequency than inserts and/or deletes. Based on our experiments, we conclude that AVL trees remain the best dictionary structure for general applications.

We have also proposed two simplified versions of the BBST called SBBST and BBSTD. The SBBST seeks only to provide logarithmic run time per operation and unlike the general BBST, does not reduce search cost at every opportunity. The SBBST provides slightly better balance than provided by WB(α) trees. The BBSTD does not attempt to maintain β-balance. However it performs rotations to reduce search cost whenever possible. Both versions are very competitive with BBSTs. The SBBST exhibited much better run time performance than BBSTs on random data and the BBSTD slightly outperformed the BBST on ordered data. However, BBSTs generated trees with the lowest search cost (though not by much).

References

[ARAG89] C. R. Aragon and R. G. Seidel, Randomized Search Trees, Proc. 30th Ann. IEEE Symposium on Foundations of Computer Science, pp. 540-545, October 1989.

[BLUM80] N. Blum and K. Mehlhorn, On the Average Number of Rebalancing Operations in Weight-balanced Trees, Theoretical Computer Science, vol 11, pp.303-320, 1980.

[GUIB78] L. J. Guibas and R. Sedgewick, A Dichromatic Framework for Balanced Trees, Proc. 19th FOCS, pp. 8-21, 1978.

[HORO94] E. Horowitz and S. Sahni, Fundatamentals of Data Structures in Pascal, 4th Edition, New York: W. H. Freeman and Company, 1994.

[MUNR92] J. I. Munro, T. Papadakis and R. Sedgewick, Deterministic Skip Lists, 3rd Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 367-375, January 1992.

[NIEV73] J. Nievergelt and E. M. Reingold, Binary Search Trees of Bounded Balance, SIAM J. Computing, Vol. 2, No. 2, pp. 33-43, March 1973.

[PAPA93] T. Papadakis, Skip Lists and Probabilistic Analysis of Algorithms, PhD Dissertation, Univ. of Waterloo, 1993.

[PUGH90] W. Pugh, Skip Lists: a Probabilistic Alternative to Balanced Trees, Communications of the ACM, vol. 33, no. 6, pp.668-676, 1990.

[SAHN93] S. Sahni, Software Development in Pascal, Florida: NSPAN Printing and Publishing Co., 1993.

[SEDG94] R. Sedgewick, Algorithms in C++, Mass.: Addison-Wesley Pub. Co., 1994.

[TARJ83] R. E. Tarjan, Updating a Balanced Search Tree in O(1) Rotations, Information Processing Letters, Vol. 16, pp. 253-257, June 1983.