Random Binary Search Tree with Equal Elements

Tomi A. Pasanen
Random Binary Search Tree with Equal Elements

Tomi A. Pasanen

Department of Computer Science
P.O. Box 68, FIN-00014 University of Helsinki, Finland
Tomi.Pasanen@cs.helsinki.fi

Helsinki, March 2008, 8 pages

Abstract

We consider random binary search trees when the input consists of a multiset, i.e. a set with multiple occurrences of equal elements, and prove that the randomized insertion and deletion algorithms produce random search trees regardless of multiplicities; even if all the elements are equal during the tree updates, a search tree will maintain its randomness. Thus, equal elements do not degenerate a random search tree and they need not to be handled in any special way. We consider also stability of a search tree with respect to its inorder traversal and prove that the algorithms used produce stable trees. This introduces an implicit indexing of equal elements giving another proof that multiplicities do not pose problems when maintaining random binary search trees.

Computing Reviews (1998) Categories and Subject Descriptors:
E.1 Data Structures — trees
F.2.2 Analysis of Algorithms and Problem Complexity: Nonnumerical Algorithms and Problems — sorting and searching

General Terms:
Binary Search Tree, Equal Elements

Additional Key Words and Phrases:
Multiset, Equal Elements, Monoset
1 Random Binary Search Tree

When a set of size \( n \) is stored into a binary search tree, then all elements are unequal and the left subtree of the root includes elements smaller than the root element and the right subtree of the root includes elements larger than the root (see trees on left in Figure 1). For defining a random binary search tree we use the definition of Martínez and Roura [1].

![Figure 1: Search trees storing set \{1, 2, 3, 4\} and multiset \{7, 7, 7, 7\}.](image)

**Definition 1** Let \( T \) be a binary search tree of size \( n \). We say that \( T \) is a random binary search tree if \( T \) is empty, or \( T \) is not empty and its left subtree \( T_L \) and its right subtree \( T_R \) are independent random binary search trees, and all sizes of \( T_L \) occur with equal probability: for all \( s \in \{0, \ldots, n-1\} \)

\[
\Pr[\text{size}(T_L) = s \mid \text{size}(T) = n] = 1/n.
\]

That is, the conditional probability of \( \text{size}(T_L) \) has a uniform distribution.

When \( T \) includes only unequal elements of a set \( S \), the above equation is equivalent to \( (\forall x \in S): \Pr[x \text{ is at the root of } T \mid \text{size}(T) = n] = 1/n \).

When \( T \) stores a multiset, i.e. a set including multiplicities, the equivalence is not true. The extreme case appears when the tree stores a set consisting only of equal elements (see trees on right in Figure 1). We call this kind of set a *mono-multiset* or *monoset* for short and consider how the randomized algorithms presented by Martínez and Roura [1] manage equal monosets. (See Algorithms 1, 2, and 3 for insertion of an element, and Algorithms 4 and 5 for deletion of an element.) After that we generalize the results for multisets. Finally, we consider stability in trees produced by the algorithms giving another view and proof why the algorithms are able to produce random trees in spite of multiplicities.

**Algorithm 1** Insertion of an element.

```c
bst insert(int x, bst T) {
    if (random(0, T->size) == 0)
      return insert_at_root(x, T);
    if (x < T->key)
      T->left = insert(x, T->left);
    else /* x >= T->key */
      T->right = insert(x, T->right);
    return T;
}
```
Algorithm 2 Insertion at the root.
bst insert_at_root(int x, bst T) {
    bst S, G;
    split(x, T, &S, &G);
    T = new_node();
    T->key = x; T->left = S; T->right = G;
    return T;
}

Algorithm 3 Split at the root.
void split(int x, bst T, bst *S, bst *G) {
    if (T == NULL) {
        *S = *G = NULL;
        return;
    }
    if (x < T->key) {
        *G = T;
        split(x, T->left, S, &(*G->left));
    } else { /* x >= T->key */
        *S = T;
        split(x, T->right, &(*S->right), G);
    }
}

Algorithm 4 Deletion of an element.
bst delete(int x, bst T) {
    bst Aux;
    if (T == NULL)
        return NULL;
    if (x < T->key)
        T->left = delete(x, T->left);
    else if (x > T->key
        T->right = delete(x, T->right);
    else { /* x == T->key */
        Aux = join(T->left, T->right);
        free_node(T);
        T = Aux;
    }
    return T;
}
Algorithm 5 Join of two binary search trees.

```c
bst join(bst L, bst R) {
    int m, n;
    m = L->size; n = R->size;
    if (m + n = 0) return NULL;
    if (random(0, m + n - 1) < m) {
        L->right = join(L->right, R);
        return L
    } else {
        R->left = join(L, R->left);
        return R;
    }
}
```

2 Monoset

Lemma 2 If a random binary search tree $T$ stores a monoset $H$ and $x$ is an element that is equal to all elements of $H$, then subroutine `split()` always produces two independent random binary search trees $S'$ and $G'$ storing monosets $H$ and $\emptyset$, respectively.

Proof. Code inspection reveals that `split()` always produces the original tree $T$ and an empty tree as trees $S'$ and $G'$, respectively, which both are random binary search trees by assumption and definition, and independent too. □

Lemma 3 If a random binary search tree $T$ stores a monoset $H$ and $x$ is an element that is equal to all elements of $H$, then `insert()` produces a new random binary search tree $T'$ storing the monoset union of $H$ and $\{x\}$.

Proof. The proof is by induction on the size $n$ of the tree $T$. When $n = 0$, the claim is true because `insert()` returns a new random binary search tree $T'$ consisting only of a root node containing $x$. Because the left and right subtrees of $T'$ are empty trees, and therefore independent, $T'$ is a random binary search tree.

Assume now that $n > 1$ and the claim is true for all sizes less than $n$. With probability $n/(n+1)$, $x$ is not placed at the root of $T'$ and it will be inserted into the right subtree $T_R$ of $T$, leaving the left subtree $T_L$ of $T$ untouched, i.e., $T'_L = T_L$. Because $T$ was a random binary search tree, the size of $T_L$ is equal to $s$, for any $s \in \{0, 1, \ldots, n-1\}$ with probability $1/n$. Thus,

$$\Pr[size(T'_L) = s \mid size(T') = n + 1] = (1/n)(n/(n+1)) = 1/(n+1),$$

and inserting $x$ into the right subtree $T_R$ will produce a random binary search tree $T'_R$ by the induction hypothesis. Because $T$ is a random binary search tree, $T_L$ is independent of $T_R$, and thus inserting $x$ into $T_R$ leaves $T_L$ and $T_R$ independent. With probability $1/(n+1)$, $x$ is placed at the root of $T'$ having $T'_L = T$ as the left subtree and an empty tree as the right subtree, both of which are independent random binary search trees based on Lemma 2. Because

$$\Pr[size(T'_L) = n \mid size(T') = n + 1] = 1/(n+1),$$

we see that the conditional probability has a uniform distribution. □
Lemma 4 Let $L$ and $R$ be two independent random binary search trees such that the largest element of $L$ is not larger than the smallest element of $R$. Subroutine \texttt{join(} with parameters $L$ and $R$ will produce a new random binary search tree $T'$ containing the elements of $L$ and $R$.

Proof. The lemma is proved by induction on sizes of $L$ and $R$, that is, on $m$ and $n$, respectively. When $L$ or $R$ is empty (or both), the lemma trivially holds: $T'$ will be an empty tree or copy of $L$ respectively. When $L$ is not larger than $R$ and $R$ is empty, the root of $T'$ will be the root of $R$ and $T'$ will be independent of $L$ or $R$ but not both.

Assume now that $m > 0$ and $n > 0$ and that the claim is true if the size of $L$ is less than $m$ or the size of $R$ is less than $n$. The root of $L$ will be chosen as the root of $T'$ with probability $m/(m+n)$, and then the left subtree $T'_L$ of $T'$ will be the left subtree $L_L$ of $L$ and the right subtree of $T'$, will be the join of the right subtree $L_R$ of $L$ with $R$ producing a random binary search tree based on induction hypothesis. Moreover, because $L$ is a binary search tree independent of $R$, then joining $L_R$ and $R$ results in an independent subtree. Now for all $s \in \{0, \ldots, m-1\}$

$$\Pr[\text{size}(T'_L) = s \mid \text{size}(T') = m + n] = (1/m)(m/(m+n)) = 1/(m+n).$$

With probability $n/(m+n)$ the root of $R$ will be chosen as the root of $T'$, and then the right subtree of $T'$ will be the right subtree of $R$ and the left subtree $T'_L$ of $T'$ will be the join of $L$ with the left subtree $R_L$ of $R$ producing a random binary search tree based on the induction hypothesis. Because $R$ was independent of $L$, then joining $L$ and $R_L$ results in an independent subtree. Applying the Definition 1 for $R_L$ and considering the size of $T'_L$ we see that for all $s \in \{0, \ldots, n-1\}$

$$\Pr[\text{size}(T'_L) = m + s \mid \text{size}(T') = m + n] = (1/n)(n/(m+n)) = 1/(m+n).$$

Thus, the conditional probability has a uniform distribution. \(\square\)

Lemma 5 If $T$ is a random binary search tree storing a monoset $H$, then \texttt{delete(} with parameters $x$ and $T$ produces a new random binary search tree $T'$ containing elements of multiset difference $H \setminus \{x\}$.

Proof. If $x$ is not equal to the root element of $T$, then it is not in the tree at all and the theorem holds. When $x$ is in the tree, it is always found at the root and the new tree $T'$ is formed by joining $T_L$ and $T_R$. Based on Lemma 4 the resulting tree $T'$ is always a random binary search tree. \(\square\)

3 Multiset

Lemma 6 If a random binary search tree $T$ stores a multiset $M$ and $x$ is an element that is equal to some elements of $M$, then executing subroutine \texttt{split(} always produces two new independent random binary search trees $S'$ and $G'$ storing multisets $\{y \in M \mid y \leq x\}$ and $\{y \in M \mid y > x\}$, respectively.

Proof. The proof is by induction on the size $n$ of $T$. When $n = 0$, the lemma trivially holds because $S'$ and $G'$ will be independent empty binary search trees.

Let now $n > 0$ and assume that the lemma is true for values smaller than $n$. The multiset $M$ consists of two multisets $M_s = \{y \in M \mid y \leq x\}$, and $M_g = \{y \in M \mid y > x\}$ having sizes $n_s$ and $n_g$, respectively. Let $r$ be the root element of $T$. 

4
If \( x \geq r \), the left subtree \( T_L \) will be left untouched and \( S'_L = T_L \). Splitting \( T_R \) produce two trees \( S'_R \) and \( G' \), which are independent random binary search tree by induction hypothesis: \( S'_R \) includes elements \( \{ y \in M \mid r \leq y \leq x \} \) and \( G' \) elements \( \{ y \in M \mid y > x \} \). Subtree \( S'_L \) will be independent of \( S'_R \) because \( T \) is a random binary search tree, and therefore \( T_L \) is independent of \( T_R \). Based on induction hypothesis the conditional probability of size(\( T_L \)) has a uniform distribution and the size of \( T_L \) is now at most \( n_s - 1 \). Therefore, we have that the size of \( S'_L \) is equal to \( s \), for any \( s \in \{0, 1, \ldots, n_s - 1\} \) with probability \( 1/n_s \). Thus,

\[
\Pr[\text{size}(S'_L) = s \mid \text{size}(S') = n_s] = 1/n_s.
\]

If \( x < r \), the right subtree \( T_R \) will be left untouched and \( G'_R = T_R \). Splitting \( T_L \) produce two trees \( S' \) and \( G'_L \), which are independent random binary search tree by the induction hypothesis: \( S' \) includes elements \( \{ y \in M \mid y \leq x \} \) and \( G'_L \) elements \( \{ y \in M \mid x < y \leq r \} \). Subtree \( G'_R \) will be independent of \( G'_L \) because \( T \) is a random binary search tree, and therefore \( T_R \) is independent of \( T_L \). Based on induction hypothesis the conditional probability of size(\( T_R \)) has a uniform distribution and the size of \( T_R \) is now at most \( n_g - 1 \). Therefore, we have that the size of \( G'_R \) is equal to \( s \), for any \( s \in \{0, 1, \ldots, n_g - 1\} \) with probability \( 1/n_g \). Considering whole \( G \), we have

\[
\Pr[\text{size}(G'_L) = s \mid \text{size}(G') = n_g] = 1/n_g.
\]

In both cases new trees \( S' \) and \( G' \) are independent random binary search trees storing multisets \( \{ y \in M \mid y \leq x \} \) and \( \{ y \in M \mid y > x \} \), respectively. \( \square \)

**Lemma 7** If a random binary search tree \( T \) stores a multiset \( M \) and \( x \) is an element that is equal to some elements in \( M \), then \( \text{insert}(x) \) produces a random binary search tree \( T' \) storing the multiset union of \( M \) and \( \{x\} \).

**Proof.** In a tree \( T \) of size \( n \) storing a multiset \( M \) having \( k \) different element, the \( i \)th smallest element appears \( n_i \) times, and thus \( \sum_{i=1}^{k} n_i = n \). Note that \( n \) cannot be smaller than \( k \). The proof is induction on two variables: first on the number of different elements \( k \), and second on the size \( n \) of \( T \). The induction base \( k = 1 \) and \( n \geq 1 \) is proved in Lemma 3.

The induction hypothesis is that the theorem is true for all values smaller than \( k \) with all permitted values of \( n \). The multiset \( M \) consists of two multisets \( M_s = \{ y \in M \mid y \leq x \} \), and \( M_g = \{ y \in M \mid y > x \} \) having sizes \( n_s \) and \( n_g \), respectively. To prove the theorem with a value \( k \) we set \( n = k \).

With probability \( 1/(n+1) \), \( x \) is placed at the root of \( T' \) and \( T'_L \) will contain all smaller and equal elements; thus

\[
\Pr[\text{size}(T'_L) = n_s \mid \text{size}(T') = n + 1] = 1/(n+1)
\]

or

\[
\Pr[\text{size}(T'_R) = n_g \mid \text{size}(T') = n + 1] = 1/(n+1).
\]

By Lemma 6 the resulting trees \( T'_L \) and \( T'_R \) will be independent random search trees, and, therefore, \( T' \) is also a random binary search tree.

Let \( r \) be the root element of \( T' \). With probability \( n/(n+1) \), \( x \) is not placed at the root of \( T' \) and it will be inserted into the right or left subtree of \( T \) depending whether it is smaller or not than \( r \).
When \( x \geq r \), \( x \) will be inserted into the right subtree of \( T \), leaving the left subtree \( T_L \) of \( T \) untouched, i.e., \( T'_L = T_L \). Based on induction hypothesis the conditional probability of size(\( T'_L \)) has a uniform distribution and the size of \( T_L \) is now at most \( n_s - 1 \) because \( x \geq r \). Therefore, for any \( s \in \{0, 1, \ldots, n_s - 1\} \)

\[
\Pr[\text{size}(T'_L) = s \mid \text{size}(T') = n + 1] = (1/n)(n/(n + 1)) = 1/(n + 1).
\]

By induction hypothesis insertion of \( x \) into the right subtree \( T_R \) having less different keys than \( k \) produces a random binary search tree \( T'_R \) which is independent of \( T'_L \) based in induction hypothesis.

When \( x < r \), \( x \) will be inserted into the left subtree of \( T \) leaving the right subtree \( T_R \) untouched, i.e., \( T'_R = T_R \). Based on induction hypothesis of the conditional probability of the left subtree, the right subtree has also a conditional probability having a uniform distribution and it is now at most \( n_g - 1 \) because \( x < r \). Therefore, for any \( s \in \{0, 1, \ldots, n_g - 1\} \)

\[
\Pr[\text{size}(T'_R) = s \mid \text{size}(T') = n + 1] = (1/n)(n/(n + 1)) = 1/(n + 1).
\]

Combining this with Eq.(1) shows that the size of the right subtree \( T'_R \) is equal to \( s \), for any \( s \in \{0, 1, \ldots, n_g\} \) with probability \( 1/(n + 1) \). Considering whole \( T' \), we see that for all \( s \in \{n_s, n_s + 1, \ldots, n\} \)

\[
\Pr[\text{size}(T'_L) = s \mid \text{size}(T') = n + 1] = (1/n)(n/(n + 1)) = 1/(n + 1).
\]

By induction hypothesis insertion of \( x \) into the left subtree \( T_L \) having less different keys than \( k \) produces a random binary search tree \( T'_L \) which is independent of \( T'_R \) based in induction hypothesis.

Joining both cases shows that when \( n = k \), the size of the left subtree \( T'_L \) of \( T' \) is equal to \( s \), for any \( s \in \{0, \ldots, n\} \) with probability \( 1/(n + 1) \). Fixing \( k \) and continuing induction on \( n \) while keeping \( n = k \) as the new induction base, we can finally prove the theorem as a whole. □

**Lemma 8** If \( T \) is a random binary search tree storing a multiset \( M \), then \texttt{delete()} with parameters \( x \) and \( T \) produces a new random binary search tree \( T' \) containing elements of multiset difference \( M \setminus \{x\} \).

**Proof.** The proof is by induction on the size \( n \) of \( T \). When \( n = 0 \), the theorem holds because \( x \) cannot be in the tree at all and the resulting tree \( T' \) equals to \( T \). Let \( n > 0 \), and let us suppose that the theorem is true for all values smaller than \( n \). If \( x \) is equal to the element at the root, then the new tree \( T' \) is formed by joining \( T_L \) and \( T_R \). Based on Lemma 4 the resulting tree \( T' \) is always a random binary search tree. If \( x \) is not equal to the element at the root, then \texttt{delete()} procedure is continued in a subtree producing a new independent random binary tree as a subtree based on the induction hypothesis. □

**Corollary 9** Regardless of multiplicities of input elements, the order of input elements, and the order of insertion and deletion requests, randomized algorithms \texttt{insert()} and \texttt{delete()} always produce a random binary search tree as a result if a random binary search tree is given as parameter to them.
4 Stable Tree

Let us couple an implicit time stamp to each equal element inserted into a ran search tree (see Figure 2). The stamps are used for recording the relative age between equal elements: the higher, the older. We use normal integers as time stamps running from 1; for an element \( x \) we use notation \( t(x) \) for its time stamp. If an inorder traversal \( \text{In}(T) = \langle x_1, x_2, \ldots \rangle \) of a tree \( T \) produces an increasing time stamp sequence for equal elements, then we say that \( T \) preserves the relative order of equal elements, i.e., it is a stable tree.

![Figure 2: Insertion of fourth equal element 7_4 into a monoset tree. Near each tree is a small fraction which denotes the probability of producing the tree by algorithm insert().](image)

**Definition 10** A tree \( T \) is a stable tree if and only if

\[
(\forall i < j)(\forall x_i, x_j \in \text{In}(T) = \langle x_1, x_2, \ldots \rangle): x_i = x_j \Rightarrow t(x_i) < t(x_j)
\]

**Lemma 11** If a stable tree \( T \) is given as input for insert(), the resulting tree \( T' \) is a stable tree.

**Proof.** When the initial tree is empty, the inserted element \( x \) appears at the root and the claim is true. When the tree is non-empty \( x \) will be inserted either at the root or recursively into a subtree of the root. If \( x \) is placed at the root, \text{split()} moves all equal elements into the left subtree which can be interpreted as those having a smaller time stamp; moreover, the relative order of equal elements already in the tree is preserved by subroutine \text{split()}. When \( x \) is to be inserted into a subtree and the element at the root
Lemma 12 If a stable tree is given as input for delete(), the resulting tree is a stable tree.

Proof. Code inspection reveals that delete() preserves stability because inorder is maintained during execution of join(). □

Corollary 13 Having a monoset of size $n$ as input for insert() and delete() and giving each element a time stamp is conceptually the same as having integer sequence $1, 2, \ldots, n$ as input for insert() and delete().

Corollary 14 Having a multiset of size $n$ as input for insert() and delete() and giving each element a time stamp is conceptually the same as having a permutation of integer sequence $1, 2, \ldots, n$ as input for insert() and delete().

Proof. The algorithms compare different element as usual while equal elements are compared by using time stamps based on Corollary 13. For example, an input sequence $\langle 2, 3, 4, 3, 1, 5, 2 \rangle$ is conceptually equal to the sequence $\langle 2, 4, 6, 5, 1, 7, 3 \rangle$. □

References