MODIFIED BINARY SEARCH TREES

by

Alon Itai* and Michael Rodeh**

Technical Report #182
August 1980

* Dept. of Computer Science, Technion-IIT, Haifa, Israel

** IBM Israel Scientific Center, Technion City, Haifa, Israel.
ABSTRACT

A degenerate node in a binary search tree is a node with only one child. It is shown that on the average, an n-node binary search tree has (n+1)/3 degenerate nodes. Suppose the insertion algorithm is modified as follows: Whenever a new node is to be inserted as a grandchild of a degenerate node, the new node, its parent and its grandparent are replaced by the 3-node full binary tree. This heuristic which is very easy to implement has the effect of reducing the average number of degenerate nodes to (n+1)/7 and the average height by a factor of 6/7, to \( \approx 1.188 \log_2 n \) which is only 18.8% more than the optimum.
1. INTRODUCTION

Following Knuth [K1], a binary tree is a set of nodes which is either empty, or consists of a root and two disjoint binary trees, called the left and right subtrees of the root. A leaf is a node both subtrees of which are empty. A node is degenerate if it has exactly one nonempty subtree and is complete if both its subtrees are empty.

Binary trees are very often used as a data structure to implement the operations search, insert and sometimes - delete. Given n keys and an n-node binary tree, each key is associated with a distinct node. The inorder of the nodes corresponds to the natural order of the keys.

To search for a key $k$, it is first compared to the key stored at the root. If they are equal then $k$ is found; if $k$ is smaller, then the search continues in the left subtree, otherwise it turns to the right subtree. If an empty tree is reached then the search is unsuccessful - $k$ does not appear in the tree.

Each empty subtree corresponds to the interval of values between the keys of its inorder predecessor and successor. To insert a new key $k$, first find the empty subtree corresponding to the interval of values to which $k$ belongs. Then replace that subtree by a single node tree whose information field contains $k$. The insertion time is dominated by the search time. In Section 2 several implementations of binary search trees are discussed. A method to delete keys is also given.

In the sequel, random variables are denoted by capital leters. Let the keys to be inserted $K_1, K_2, \ldots, K_n$ be random variables. $T_n$ - the tree obtained after $n$ insertions into an initially empty tree - is also a random variable. The distribution of $T_n$ depends on the $K_i$'s. Several probabilistic models will be discussed in Section 3, where it is shown that all these models yield the same distribution of trees.
As shown in [K2] the average number of nodes visited during a single unsuccessful search of $T_n$ is $2\ln(n+1) - 0.84556 + O(1/n) \approx 1.386 \log_2 n + O(1)$. In Section 4 we present, for completeness, an alternative derivation. It is also shown that both the average number of leaves and the average number of degenerate nodes is equal to $(n+1)/3$.

Consider the following heuristic to reduce the number of degenerate nodes: On insertion, if the parent and grandparent of a new node are degenerate, replace the 3-node subtree rooted at the grandparent by the full 3-node tree. See Figure 1 for an example.

![Figure 1](image)

The trees obtained by modified insertion are still ordinary search trees, only their distribution differs. To see the difference notice that some $n$-node binary trees are not obtained by modified insertions.

Let $M_n$ be a binary search tree obtained by $n$ modified insertions. In Section 5 it is shown that the average number of nodes visited by an unsuccessful search in $M_n$ is reduced by a factor of $6/7$ in comparison to regular binary tree (to $1.188 \log_2(n+1) - 0.54109 + O(1/n)$). This compares very well to the optimum binary search tree whose average search time is $\log_2 n + O(1)$. It is also shown that the average number of degenerate nodes is only $(n+1)/7$ and that the number of leaves is $3(n+1)/7$. 
We see that from a probabilistic point of view, modified insertions are worthwhile both in time and in space complexity.

As for the worst case, the height of modified binary search tree is bounded by \( n/2 \) while the maximum height of regular binary search trees is \( n-1 \). The maximal number of degenerated nodes is shown to be \((n+1)/3\) while in regular binary trees it may be as large as \( n-1 \).

2. IMPLEMENTING BINARY TREES

Binary search trees are dynamic by nature, therefore, they are usually implemented by nodes which are allocated and freed at run time. Each node has three fields: an information field to store the key associated with the node and two pointer fields (lchild and rchild) which point to the left and right subtrees. If a subtree is empty, the corresponding pointer is \( \text{nil} \).

On insertion, a new leaf \( v \) is allocated, it replaces an empty subtree of a node \( p \) whose corresponding pointer is now made to point at \( v \). Deletions are somewhat trickier: let \( v \) be a node to be deleted and let \( p \) be its parent. If \( v \) is leaf, the pointer of \( p \) which pointed to \( v \) is set to \( \text{nil} \). If \( v \) is degenerate, \( p \) is made to point at \( v \)'s child. Finally, to delete a complete node, find the node \( v' \) which immediately succeeds \( v \) in inorder, and exchange the information fields of \( v \) and \( v' \). The node \( v' \) has at most one non-empty subtree, and thus can be deleted as discussed above. In all cases, as a result of deletion, some node is "freed" and may be reused.

If insertions and deletions followed a last-in-first-out discipline then a stack could be used to allocate nodes, thus a deletion will cause the top node to be removed. Note that we must be able to inspect values other than the top node, and change pointer fields.
Even when the last-in-first-out discipline does not hold, the nodes can be allocated on a stack: to remove a node from the middle of the stack, exchange its contents with the top node, this involves finding the parent of the top node and updating its pointer. Thus, the last-in-first-out allocation discipline is retained at the expense of an extra search.

An n-node binary tree contains 2n pointer fields, n+1 of which are null. Denoting the number of leaves by \( \ell \) and the number of degenerate nodes by \( d \), we have:

\[
n + 1 = d + 2\ell.
\]

It is wasteful in space to allocate the same amount of space both to leaves and to internal nodes. Thus, it might be advantageous to allocate two types of nodes: one for internal nodes and one for leaves. Each type of nodes can be made to adhere to the last-in-first-out allocation discipline. Thus, we have two stacks which in turn can be implemented by two consecutive chunks of memory, one starting from low memory and growing upwards and the other starting from high memory and growing downwards. Since the memory allocated to each stack is consecutive, given the address of a node it is easy to decide to which stack it belongs and consequently whether the node is a leaf or an internal node.

3. PROBABILISTIC MODELS FOR ORDERING PROBLEMS

When conducting a probabilistic analysis we must make some assumptions on the distribution of the data. The weakest assumptions (thus the most general model) are those made only on the relative order of the data, not on their specific values.

Many probabilistic analyses assume a uniform distribution model,
i.e., the inputs $X_1, X_2, ...$ are uniformly distributed random variables over some domain, usually the interval $[0,1]$. In this model assumptions are made on the values of the data.

Yao [Y] discusses symmetric models, in which all permutations of the input data are assumed to be equiprobable, thus the density of the pair $(0.5, 0.2)$ is equal to that of $(0.2, 0.5)$. The uniform distribution models are contained in the symmetric models (i.e. they satisfy the requirements for a model to be symmetric). Again, in symmetric models assumptions are made on specific values of the problem domain.

Other data dependent models discussed by Yao [Y] are the random insertion models: Given $n$ distinct inputs $X_1, \ldots, X_n$, let $\pi$ be the permutation such that $X_{\pi_1} < \ldots < X_{\pi_n}$. Then the next input $X_{n+1}$ is chosen such that the following $n+1$ events:

$$X_{n+1} < X_{\pi_1}$$
$$X_{\pi_i} < X_{n+1} < X_{\pi_{i+1}} \quad i = 1, \ldots, n-1$$
$$X_{\pi_n} < X_{n+1}$$

have equal probability. Note that random insertion models are not symmetric: The density of $(0.5, 0.2)$ is 1 while the density of $(0.2, 0.5)$ is 0.8.

We wish to define a more general family of models in which no assumption is made on specific values. To this end, let $(X_1, \ldots, X_n)$ be a sequence of $n$ distinct numbers. The ordering of $(X_1, \ldots, X_n)$ is the unique permutation $\pi$ which sorts the sequence, i.e. $X_{\pi_1} < X_{\pi_2} < \ldots < X_{\pi_n}$. Thus, $\pi = (3,1,2)$ is the ordering of the sequence $(0.7, 0.1, 0.4)$ as well as of the sequence $(0.9, 0.7, 0.8)$. A model is an equiprobable ordering model if for all permutations $\pi$ there is equal probability that $\pi$ is the ordering of a random sequence $(X_1, \ldots, X_n)$. 
The equiprobable ordering models contain the symmetric models and are, therefore, more general. As the following theorem shows, the random insertion models, which are history dependent by definition, also satisfy the equiprobable ordering requirements.

**Theorem 1:** Random insertion models are included in the equiprobable ordering models.

**Proof.** Let $X_1, X_2, \ldots$ be random variables emitted by a source which fulfills the random insertion requirement. Let $P_{\pi, i}$ denote the probability that the ordering of $(\pi_1, \ldots, \pi_i)$ is equal to that of $(X_1, \ldots, X_i)$. We show by induction that $P_{\pi, i} = 1/i!$.

**Base** $i = 1$. The only ordering is the permutation $(1)$, which sorts both $(X_1)$ and $(\pi_1)$. Therefore, $P_{\pi, 1} = 1 = 1/1!$.

**Induction hypothesis** $P_{\pi, i-1} = 1/(i-1)!$.

**Induction step** Let $PC_{\pi, i}$ denote the conditional probability that the orderings of $(X_1, \ldots, X_i)$ and $(\pi_1, \ldots, \pi_i)$ are equal, given that the ordering of $(X_1, \ldots, X_{i-1})$ is equal to that of $(\pi_1, \ldots, \pi_{i-1})$. By the random insertion hypothesis $PC_{\pi, i} = 1/i$. However,

$$P_{\pi, i} = PC_{\pi, i} \cdot P_{\pi, i-1}.$$ 

Using $PC_{\pi, i} = 1/i$, the induction hypothesis yields

$$P_{\pi, i} = (1/i)(1/(i-1)!) = 1/i!.$$ 

The containment relations between the various models are illustrated in Figure 2.
The information contained in the equiprobable ordering models suffices to analyze algorithms whose behavior depends only on the relative order, not on the specific values. This leads us to the following definition: An algorithm is an ordering algorithm if only comparisons of the data are conducted and the specific values are immaterial. Thus, heap-sort and binary-search are ordering algorithms, while bucket-sort and interpolation-search are not.

The behavior of an ordering algorithm depends only on the relative order of the data and on the probability of occurrence of the various orderings, i.e. the probabilistic model. The average behavior of an ordering algorithm is equal in two models which assign the same probabilities to the various orderings. We see that an average behavior analysis carried out in one of the above submodels is valid for all the others. For example, when analyzing the average behavior of binary search trees we may use the random insertion model instead of the uniform distribution model.

This observation generalizes that made by Yao [Y] concerning the distribution of random 2-3 trees.

Figure 2

The information contained in the equiprobable ordering models suffices to analyze algorithms whose behavior depends only on the relative order, not on the specific values. This leads us to the following definition: An algorithm is an ordering algorithm if only comparisons of the data are conducted and the specific values are immaterial. Thus, heap-sort and binary-search are ordering algorithms, while bucket-sort and interpolation-search are not.

The behavior of an ordering algorithm depends only on the relative order of the data and on the probability of occurrence of the various orderings, i.e. the probabilistic model. The average behavior of an ordering algorithm is equal in two models which assign the same probabilities to the various orderings. We see that an average behavior analysis carried out in one of the above submodels is valid for all the others. For example, when analyzing the average behavior of binary search trees we may use the random insertion model instead of the uniform distribution model.

This observation generalizes that made by Yao [Y] concerning the distribution of random 2-3 trees.
4. REGULAR BINARY SEARCH TREES

We first analyze the characteristics of \( T_n \) - the binary search tree obtained by regular insertions. As a matter of notation, if \( f \) is a function of a random variable \( X \) then \( F(X) \) is the corresponding random variable and \( \bar{F}(X) \) is its expected value.

**Theorem 2:** The average number of leaves of a regular insertion binary tree is

\[
\bar{L}(T_n) = \frac{(n+1)}{3} \quad (n > 1).
\]

**Proof:** By induction

**Basis:** \( n = 2 \). \( T_2 \) is either \( o \) or \( o \), so \( \bar{L}(T_2) = L(T_2) = 1 \).

**Induction Step:** Let \( t_n \) be a specific \( n \)-node binary tree, and \( t_{n+1} \) the tree obtained by a random insertion into \( t_n \), i.e. by replacing a null pointer. The tree \( t_n \) has \( n+1 \) null pointers, \( 2L(t_n) \) of which belong to leaves. If the new node is a child of a former leaf, the number of leaves does not change. This happens with probability \( 2L(t_n)/(n+1) \). Otherwise, the number of leaves increases by one. Therefore,

\[
\bar{L}(t_{n+1}) = \frac{2L(t_n)}{n+1} \bar{L}(t_n) + \left(1 - \frac{2L(t_n)}{n+1}\right)(\bar{L}(t_n) + 1)
\]

\[
= \frac{1}{n+1} (2L(t_n)^2 + (n+1)L(t_n) + (n+1) - 2L(t_n)^2 - 2L(t_n))
\]

\[
= 1 + \frac{n-1}{n+1} \bar{L}(t_n).
\]

Averaging over all binary trees yields:

\[
\bar{L}(T_{n+1}) = 1 + \frac{n-1}{n+1} \bar{L}(T_n).
\]
The desired result is obtained by substituting the induction hypothesis:

\[ L(T_{n+1}) = 1 + \frac{n-1}{n+1} \frac{n+1}{3} = \frac{n+2}{3}. \]

Since \( n+1 = d+2 \ell \), the average number of degenerate nodes is

\[ D(T_n) = n+1 - 2L(T_n) = \frac{(n+1)}{3}. \]

Consequently, if leaves are implemented by pointerless nodes, there remain \( \frac{(n+1)}{3} \) null pointers on the average.

To analyze the average time, replace every null pointer of a binary tree by a new external node. In Figure 3 external nodes are illustrated as square boxes.

\[ \text{Figure 3} \]

Let \( \text{ext}(t) \), the external path length, be the sum of the distances from the external nodes to the root of a binary tree \( t \).

**Lemma 1:** The average external path length of a regular binary search tree satisfies:

\[ \overline{\text{EXT}}(T_n) = 2 + \left(1 + \frac{1}{n}\right)\overline{\text{EXT}}(T_{n-1}). \]

**Proof:** Let \( T_{n-1} \) be an \( (n-1) \)-node binary search tree and let \( k \) be a new key, the insertion of which replaces an external node at distance \( r \) from the root. The external path length of the new tree \( T_n \) is
\[ \text{ext}(t_n) = 2 + r + \text{ext}(t_{n-1}). \]

The average value of \( r \) is \( \text{ext}(t_{n-1})/n \). Thus, we have:

\[ \bar{\text{ext}}(t_n) = 2 + (1 + 1/n)\text{ext}(t_{n-1}). \]

Averaging over all binary search trees yields the desired formula. \( \square \)

Let \( H_n = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \) denote the harmonic series of length \( n \). Thus, \( H_n = \ln n + \gamma + O(1/n) \) where \( \gamma \approx 0.57722 \) is Euler's constant.

**Theorem 3:** [K3, pp.427]. The average number of nodes visited in an unsuccessful search of \( T_n \) is:

\[ \frac{\text{EXT}(T_n)}{n+1} = 2(H_{n+1} - 1) = c_1 \log_2 (n+1) - c_2 + O(1/n) \]

where \( c_1 \approx 1.386 \) and \( c_2 \approx 0.84556. \)

**Proof:** Lemma 1 implies the following recursive equation

\[ \frac{\text{EXT}(T_n)}{n+1} = \frac{2}{n+1} + \frac{\text{EXT}(T_{n-1})}{n-1}. \]

The desired formula is the solution to this equation. \( \square \)

5. **MODIFIED BINARY SEARCH TREES**

The possible values of \( M_n \) and their distribution for \( 1 \leq n \leq 6 \) are depicted in Figure 4. From each symmetry class only one tree is drawn.

**Lemma 2:** In any modified binary search the degenerate nodes occur only as parents of leaves.

**Proof:** By induction.

**Basis:** For \( n = 1 \) the lemma is vacuously true.
Figure 4

Induction Step: An n-node modified binary tree $m_n$ is obtained by inserting a new key $k$ into some $(n-1)$-node modified binary search tree $m_{n-1}$. In $m_{n-1}$ the lemma holds by the induction hypothesis. The insertion of $k$ may replace a degenerate node by a complete one, or create a new degenerate node with a single child which is a leaf containing $k$. \qed
Theorem 4: A modified binary search tree with $n$ nodes may have at most $(n+1)/3$ degenerate nodes.

Proof: Let $t_n$ be an $n$-node modified binary search tree. By Lemma 2 $d = d(t_n) \leq \ell(t_n)$. Let $t_{n-d}$ be the binary tree obtained from $t_n$ by removing the leaves whose parents are degenerate. By the construction

$$\ell(t_{n-d}) = \ell(t_n).$$

Counting the number of nodes in $t_{n-d}$ yields:

$$n-d = 2\ell(t_{n-d}) - 1 = 2\ell(t_n) - 1.$$ 

Substituting for $d$ yields:

$$\ell(t_n) \geq (n+1)/3$$

and $d(t_n) \leq (n+1) - 2\ell(t_n) \leq (n+1)/3$. Figure 5 shows that this bound is exact. \qed

![Figure 5](image-url)
Theorem 5: The average number of degenerate nodes in a modified binary search tree is
\[ \bar{D}(M_n) = \frac{n+1}{7}, \quad (n > 5). \]

Proof: By induction.

Basis: For \( n = 6 \), inspecting Figure 4 shows that \( \bar{D}(M_6) = 1 \).

Induction Step: Let \( M_n \) be a modified tree obtained by random insertion. The subtree rooted at a degenerate node \( v \) has one of the following shapes:

\[ \text{\begin{tikzpicture}[scale=0.5]
        \node (v) at (0,0) {$v$};
        \node (left) at (-1,-1) {$\circ$};
        \node (right) at (1,-1) {$\circ$};
        \draw (v) -- (left);
        \draw (v) -- (right);
    \end{tikzpicture}} \]

By Lemma 2, the child of \( v \) is a leaf. Thus, in both cases there exist three null pointers. An insertion into such a subtree yields the full 3-node binary tree with no degenerate nodes. Thus, the number of degenerate nodes is decreased by one. This happens with probability \( \frac{3d(m_n)}{n+1} \). In all other cases the number of degenerate nodes increases by one. We get the following recursive formula:

\[ \bar{d}(m_{n+1}) = \frac{3d(m_n)}{n+1}(d(m_n) - 1) + \left(1 - \frac{3d(m_n)}{n+1}\right)(d(m_n) + 1) = 1 + \frac{n-5}{n+1}d(m_n). \]

Averaging over all \( n \)-node modified binary search trees yields:

\[ \bar{D}(M_{n+1}) = 1 + \frac{n-5}{n+1} \bar{D}(M_n). \]

By the induction hypothesis we get:

\[ \bar{D}(M_{n+1}) = 1 + \frac{n-5}{n+1} \frac{n+1}{7} = \frac{n+2}{7}. \]

Corollary: \( \bar{L}(M_n) = 3(n+1)/7 \). \( \square \)
Lemma 3: The average external path length of a modified binary search tree satisfies:

$$\overline{\text{EXT}(M_{n+1})} = 2 + \left(1 + \frac{1}{n+1}\right) \overline{\text{EXT}(M_n)} - \frac{2\overline{D(M_n)}}{n+1}.$$ 

Proof: Let $m_n$ be an $n$-node modified binary search tree and $v$ be a degenerate node whose child is $u$. A modification at $v$ occurs only when a node is inserted as $u$'s child. Thus, the probability of a modification is $2d(m_n)/(n+1)$.

A modification causes the external path length to decrease by one.

Thus, as in Lemma 1, the new external path length is

$$2 + \left(1 + \frac{1}{n+1}\right) \text{ext}(m_n)$$

before the modification. Bringing the modification into account and averaging over all $n$-node modified binary trees yields the desired formula.

\[\square\]

Theorem 6: The average number of nodes visited in an unsuccessful search is

$$\overline{\text{EXT}(M_n)}/(n+1) = c_3 \log_2(n+1) - c_4 + O(1/n)$$

where $c_3 \approx 1.188$ and $c_4 \approx 0.54109$.

Proof: First note that $\overline{\text{EXT}(M_6)} = 21 \times 2/5 + 20 \times 3/5 = 20.4$.

Next, substitute the value of $\overline{D(M_n)}$ from Theorem 5 in Lemma 3 to obtain:

$$\overline{\text{EXT}(M_n)} = 2 + (1 + 1/n)\overline{\text{EXT}(M_{n-1})} - (2/n) \cdot (n/7).$$
Thus,

\[ \frac{\text{EXT}(M_n)}{n+1} = \frac{12}{7} \cdot \frac{1}{n+1} + \frac{\text{EXT}(M_{n-1})}{n} \]

\[ = \frac{12}{7} \left( \frac{1}{n+1} + \frac{1}{n} + \ldots + \frac{1}{8} \right) + \frac{\text{EXT}(M_6)}{7} \]

\[ = \frac{12}{7} (H_{n+1} - H_7) + \frac{102}{35} \]

\[ = \frac{12}{7} H_{n+1} - \frac{75}{49} \]

\[ = \frac{12}{7} \ln(n+1) + \frac{84}{49} - \frac{75}{49} + O(1/n) \]

\[ = 1.188 \log_2 (n+1) - 0.54109 + O(1/n). \]

\[ \square \]

A comparison of modified binary search trees and regular binary search trees is given in Table 1.

<table>
<thead>
<tr>
<th></th>
<th>modified binary search tree</th>
<th>regular binary search tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum number of degenerate nodes</td>
<td>( \frac{(n+1)}{3} )</td>
<td>( n-1 )</td>
</tr>
<tr>
<td>minimum number of leaves</td>
<td>( \frac{(n+1)}{3} )</td>
<td>1</td>
</tr>
<tr>
<td>average number of degenerate nodes</td>
<td>( \frac{(n+1)}{7} )</td>
<td>( \frac{(n+1)}{3} )</td>
</tr>
<tr>
<td>average number of leaves</td>
<td>( 3\frac{(n+1)}{7} )</td>
<td>( \frac{(n+1)}{3} )</td>
</tr>
<tr>
<td>average number of complete nodes</td>
<td>( 3\frac{(n-4)}{7} )</td>
<td>( \frac{(n-2)}{3} )</td>
</tr>
<tr>
<td>average number of nodes visited in an unsuccessful search</td>
<td>( 1.188 \log_2(n+1) - 0.54109 + O(1/n) )</td>
<td>( 1.386 \log_2(n+1) - 0.8455 + O(1/n) )</td>
</tr>
<tr>
<td>average number of nodes visited in a successful search</td>
<td>( 1.188 \log_2(n+1) - 2.54109 + O(1/n) )</td>
<td>( 1.386 \log_2(n+1) - 2.8455 + O(1/n) )</td>
</tr>
</tbody>
</table>

Table 1
The last line of Table 1 is derived from the following relation between internal and external path lengths (the internal path length, denoted by \( \text{int} \), is the sum of the distance from the internal nodes to the root):

**Lemma 4**: [K1, p.400]. For every \( n \)-node binary tree \( t_n \)

\[
\text{ext}(t_n) = \text{int}(t_n) + 2n.
\]

6. CONCLUSIONS

Modified binary search trees compare favorably with regular binary trees, *search* is faster due to the smaller average internal path length. As to insertions the situation is unclear: On one hand, the average internal path length is smaller, but on the other hand modifications must be carried out (constant time) and to this end we must keep track of the parent of the degenerate node (time proportional to the distance of the node from the root).

An important advantage of modified binary search trees is the space savings. The wasted space is cut by 4/7 on the average.

Additional space may be saved by considering higher order modifications (such as the modification shown in Figure 6). Obviously, such modifications are more time consuming than the one considered in this paper.

Another extension is to conduct a probabilistic analysis for trees obtained from deletions as well as insertions, however, this case is considerably more difficult [K3].
REFERENCES


