1982

An empirical study of insertion and deletion in binary search trees

Jeffrey L. Eppinger

Carnegie Mellon University

Follow this and additional works at: http://repository.cmu.edu/compsci
NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.
An Empirical Study of Insertion and Deletion in Binary Search Trees

Jeffrey L. Eppinger

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

December 2, 1982

Abstract: This paper describes an experiment on the effect of insertions and deletions on the path length of unbalanced binary search trees. Given a random binary tree, repeatedly inserting and deleting nodes yields a tree that is no longer random. The expected internal path length differs when different deletion algorithms are used. Previous empirical studies indicated that expected internal path length tends to decrease after repeated insertions and asymmetric deletions. This study shows that performing a larger number of insertions and asymmetric deletions actually increases the expected internal path length, and that for sufficiently large trees, the expected internal path length becomes worse than that of a random tree. However, with a symmetric deletion algorithm, the results indicate that performing a large number of insertions and deletions decreases the expected internal path length, and that the expected internal path length remains better than that of a random tree.

This research was sponsored in part by the Office of Naval Research under contract N00014-76-C-0370.
1. Introduction

A binary tree created by inserting $n$ randomly chosen keys into an empty tree has an expected internal path length of $I_n \approx 1.386n \lg n$.† Randomly deleting $k$ nodes from such a tree yields a tree whose expected internal path length is $I_{n-k}$. Unfortunately, performing insertions after deletions does not produce binary trees whose internal path length is predicted by this function. A theoretical explanation of the effect of performing deletions and then insertions on binary trees is still lacking. [Knuth 73, Section 6.2.2]

This paper presents an empirical study on the effect of applying random insertions and deletions to random binary search trees and analyzes results of experiments comparing asymmetric and symmetric deletion algorithms. In a previous empirical study, Knott [Knott 75] suggests that the expected internal path length tends to decrease after repeated insertions and asymmetric deletions. In this study, the large number of insertions and asymmetric deletions performed suggests that the expected internal path length first decreases but eventually begins to increase. For sufficiently large trees, expected internal path length becomes worse than that of a random tree. However, experiments using the symmetric deletion algorithm show that performing a large number of insertions and symmetric deletions decreases the expected internal path length (making the trees better than random).

Section 2 describes the insertion and deletion algorithms used in this study and provides an overview of some of the previous work in this area. The statistics used in this study are defined in Section 3. Section 3 also mentions a few specifics about how the data was gathered. The observations in Section 4 give an interpretation of the data and the conclusions are summarized in Section 5.

2. Background

Insertion Algorithm: The structure of binary trees naturally leads to one insertion algorithm. To insert a node into a binary tree (known not to contain the node), compare the new and current keys and insert the node into the left or right subtree, whichever maintains the invariant of the data structure. The Pascal code for this algorithm is provided in Figure 1, below. For further

† Throughout this paper, $\lg x$ denotes $\log_2 x$. 
PROCEDURE Insert(VAR root : NodePtr; x : DataType);
BEGIN
  IF root = NIL
    THEN BEGIN
      NEW(root); root^.data := x;
      root^.lChild := NIL; root^.rChild := NIL
    END
  ELSE IF x < root^.data
    THEN Insert(root^.lChild, x)
  ELSE Insert(root^.rChild, x)
END;

Figure 1: The insertion procedure.

Unlike insertion, there are many reasonable deletion algorithms from which to choose. This paper describes experiments with Knuth's asymmetric deletion algorithm and a trivially modified version of this algorithm to make it symmetric.

Asymmetric Deletion Algorithm: A node's successor is defined to be the smallest node in the right subtree. Similarly a node's predecessor is defined to be the largest node in the left subtree. To delete a node from a binary tree, replace the node with its successor, i.e., the node that contains the next larger key. The Pascal code for this algorithm is given in Figure 2, below. Figure 4↑ shows examples of the insertion algorithm and this deletion algorithm applied to a particular binary tree; for further explanation see [Knuth 73, Section 6.2.2, Algorithm T].

Symmetric Deletion Algorithm: To delete a node from a binary tree, replace the node with its successor or predecessor. Alternately choose the successor and predecessor (so that half the time the RightDelete routine is called and half the time a suitably modified version of this routine, LeftDelete, is called).

Consider building a binary tree using n keys chosen randomly from a uniform distribution (i.e., all n! permutations of the keys are equally likely). There are \( \binom{2^n}{n} / (n + 1) \) possible shapes for this tree [Knuth 68, Section 2.3.4.4], each with some probability of occurring; call the distribution \( D_n \). By this definition, inserting a new node into this binary tree would yield a tree of size \( n + 1 \) whose shape occurs with a probability defined by \( D_{n+1} \). Binary trees whose distribution of shapes

↑ Figures 4-11 are at the end of the paper.
PROCEDURE RightDelete(VAR root : NodePtr; x : DataType);
VAR copy, successor, succPtr : NodePtr;
BEGIN
  IF x < root↑.data
    THEN RightDelete(root↑.lChild, x)
  ELSE IF x > root↑.data
    THEN RightDelete(root↑.rChild, x)
  ELSE BEGIN
    copy := root;
    IF root↑.rChild = NIL
      THEN root := root↑.lChild
    ELSE IF root↑.rChild↑.lChild = NIL
      THEN BEGIN
        root↑.rChild↑.lChild := root↑.lChild;
        root := root↑.rChild
      END;
      { Case I: There is no successor. }
    ELSE BEGIN
      succPtr := root↑.rChild;
      WHILE succPtr↑.lChild↑.lChild <> NIL DO
        succPtr := succPtr↑.lChild;
        successor := succPtr↑.lChild;
        succPtr↑.lChild := successor↑.rChild;
        successor↑.lChild := root↑.lChild;
        successor↑.rChild := root↑.rChild;
        root := successor
      END;
      { Case II: The successor is the right child. }
      DISPOSE(copy)
    END;
    { Case III: The successor is the leftmost child in the right subtree. }
    END
  END;
END;

Figure 2: The asymmetric deletion procedure.

is $D_n$ are called random binary trees.

Thomas Hibbard [Hibbard 62] proved that deleting a random node (i.e., where each node has an equal probability of being deleted) from a binary tree of size $n$, with distribution of shapes $D_n$, yields a tree with a distribution of shapes $D_{n-1}$.

Strangely, performing random insertion and deletion operations on a random tree does not preserve this distribution of shapes. Consider building a binary tree of size $n$, as described above. Since the keys are chosen from a uniform distribution, the probability of inserting a new node in any particular interkey gap is $\frac{1}{n+1}$. After one random deletion, the distribution of shapes will be
Knuth [Knuth 73, Section 6.2.2] describes this phenomenon as follows:

The shape of the tree is random after deletions, but the relative distribution of values in a given tree shape may change, and it turns out that the first random insertion after a deletion actually destroys the randomness property on shapes. This startling fact, first observed by Gary Knott in 1972, must be seen to be believed. Empirical evidence suggests strongly that the path length tends to decrease after repeated deletions and insertions, so the departure from randomness seems to be in the right direction; a theoretical explanation for this behavior is still lacking.

Knuth feels that binary trees tend to improve because "path length tends to decrease." One way to compare binary trees is to measure their internal path lengths. The internal path length of a tree is defined as the sum of the depths of the nodes in the tree,

\[ IPL = \sum_{i \in \text{nodes}} \text{distance(root, i)}. \]

For a random tree containing \( n \) nodes, the expected IPL is denoted as \( I_n \) and the expected number of comparisons in a successful search is denoted as \( C_n \). Knuth [Knuth 73, Section 6.2.2] gives the expected number of comparisons in a successful search, \( C_n \), as approximately equal to \( 1.386 \lg n \). Substituting into the relation \( I_n = n(C_n - 1) \), one obtains the approximation \( I_n \approx 1.386n \lg n \). A distribution of trees is said to be "better than random" when the expected IPL is less than \( I_n \) (since the expected number of comparisons is proportional to the IPL).

3. Methodology

If a random sequence of insertions and deletions were applied to a random tree of size \( n \), the resulting tree would probably not have the same number of nodes. The original tree's IPL would therefore not be directly comparable with the IPL of the new tree. In this study, sequences of insertion/deletion pairs (I/D pairs) are applied to random trees. Since the resulting tree always has the same size, it is easy to see whether any improvement has been made. (Knott's data was also obtained by using I/D pairs.) The first step of the simulation is therefore to insert \( n \) nodes into an empty tree, after which successive pairs of insertions followed by deletions are performed.

Let \( IPL_{n,i} \) denote the measured mean IPL of an \( n \)-node binary tree after applying \( i \) I/D pairs.
Figures 5 through 10 show $IPL_{n,i}/I_n$ plotted as a function of $i$. This ratio shows the improvement of the resulting tree's expected IPL as a fraction of the random tree's expected IPL.

The deletion algorithm given above generally replaces the node to be deleted with its successor, the "left-most node in the right subtree". The left and right subtrees are treated differently and, as observed below, this appears to have a profound affect on the behavior of binary trees. Such a deletion algorithm is called an asymmetric deletion algorithm. The symmetric deletion algorithm which is examined in this study is a trivially modified version of the asymmetric algorithm. This symmetric algorithm alternately replaces the node to be deleted with its successor or its predecessor. The algorithm requires a small amount of state information, but similar results have been obtained by randomly replacing the node to be deleted by its successor or predecessor.

To ensure that the results were not an artifact of the random number generator, simulations were performed on both DEC-20s and Perqs. In the DEC-20 simulations the random number generator used the linear congruential method to produce 36-bit pseudorandom numbers [Knuth 69, Section 3.2]. The random number generator for the Perqs is the feedback shift-register pseudorandom number generator as described in [Lewis 73]. The data presented in this paper was generated on the Perqs and took about one month of CPU time, but similar results were obtained for the smaller trees on the DEC-20s.

The outer loop of the simulation program is very simple. First, build a tree with tsize nodes, then gather data before and after each interval of isize input/output pairs.

```
FOR i := 1 TO tsize DO RndInsert;
... gather data ...
FOR i := 1 TO intervals DO BEGIN
    FOR j := 1 TO isize DO BEGIN RndInsert; RndDelete END;
    ... gather data ...
END;
FreeTree;
```

Figure 3: The inner loop of a simulation.

4. Observations

The graphs in Figures 5 and 6 show the expected internal path length of $n$-node binary trees plotted against the number of insertion and asymmetric deletion pairs. Initially, $IPL_{n,i}$
decreases, as Knott and Knuth observed. After some critical point, though, $\frac{IPL_{n,i}}{I_n}$ starts to increase, eventually levelling off after approximately $n^2$ I/D pairs. Figure 7 is a comparison chart in which $\frac{IPL_{n,i}}{I_n}$ is plotted as a function of $i/n^2$ for each of the values of $n$ tested. (The latter ratio normalizes the $x$-axis.)

Perhaps the most significant observation is that as $n$ increases so does the asymptotic value for $\frac{IPL_{n,i}}{I_n}$. Since binary trees can be modeled by Markov Chains, and any binary tree may be obtained by applying some combination of I/D pairs to any other binary tree, the $\lim_{i \to \infty} IPL_{n,i}$ exists [Ross 70, Theorem 4.9]. Figure 7 suggests that 

$$\lim_{i \to \infty} IPL_{n,i} > I_n$$

for sufficiently large values of $n$ (roughly greater than 128). Thus binary trees seem to become “worse than random” after many insertions and deletions.

The comparison chart in Figure 11 shows the asymptotic values of $\frac{IPL_{n,i}}{I_n}$ for both deletion algorithms plotted against $n$ (on a log scale). The data given in Table 1 was obtained by summing all the $IPL_{n,i}$ and $IP^2_{n,i}$, when $i > n^2$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>Samples</th>
<th>$\frac{IPL_{n,i}}{I_n} &gt; n^2$</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>6000</td>
<td>0.97</td>
<td>0.01652</td>
</tr>
<tr>
<td>128</td>
<td>6800</td>
<td>1.00</td>
<td>0.01340</td>
</tr>
<tr>
<td>256</td>
<td>2300</td>
<td>1.06</td>
<td>0.00985</td>
</tr>
<tr>
<td>512</td>
<td>1200</td>
<td>1.16</td>
<td>0.00970</td>
</tr>
<tr>
<td>1024</td>
<td>750</td>
<td>1.30</td>
<td>0.01013</td>
</tr>
<tr>
<td>2048</td>
<td>5340</td>
<td>1.49</td>
<td>0.00771</td>
</tr>
</tbody>
</table>

Table 1: Data for Asymmetric Deletions.

The asymmetric curve appears to be quadratic. A least-squares multiple regression weighted by the inverse of the variance yields the following approximation:

$$\lim_{i \to \infty} \frac{IPL_{n,i}}{I_n} \approx 0.0202 \log^2 n - 0.241 \log n + 1.69.$$ 

Substituting $I_n \approx 1.386 n \log n$ we obtain

$$\lim_{i \to \infty} IPL_{n,i} \approx 0.0280 n \log^3 n - 0.334 n \log^2 n + 2.34 n \log n.$$
The graphs in Figures 8 and 9 show the corresponding plots of the data in Table 2 for the expected internal path length for symmetric deletions.

<table>
<thead>
<tr>
<th>n</th>
<th>Samples</th>
<th>( I_{PL,n,i} &gt; n^2 )</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>6000</td>
<td>0.905</td>
<td>0.01654</td>
</tr>
<tr>
<td>128</td>
<td>6800</td>
<td>0.890</td>
<td>0.00916</td>
</tr>
<tr>
<td>256</td>
<td>2300</td>
<td>0.888</td>
<td>0.00615</td>
</tr>
<tr>
<td>512</td>
<td>1200</td>
<td>0.890</td>
<td>0.00347</td>
</tr>
<tr>
<td>1024</td>
<td>750</td>
<td>0.881</td>
<td>0.00235</td>
</tr>
<tr>
<td>2048</td>
<td>5340</td>
<td>0.883</td>
<td>0.00269</td>
</tr>
</tbody>
</table>

Table 2: Data for Symmetric Deletions.

The \( I_{PL,n,i} \) decreases initially, as in the case of asymmetric deletions, but the asymptotic value of the expected internal path length seems to remain lower than that of a random tree. The comparison charts in Figures 10 and 11 indicate that

\[
1 > \lim_{i \to \infty} \frac{I_{PL,n,i}}{I_n} \approx 0.88
\]

or that

\[
I_n > \lim_{i \to \infty} I_{PL,n,i} \approx 1.22n \log n.
\]

The comparison chart in Figure 11 shows the asymptotic value of \( I_{PL,n,i} \) slowly decreasing as \( n \) increases. Since a binary tree with \( n \) nodes cannot have an internal path length less than that of a perfect tree, we know that

\[
\lim_{i \to \infty} I_{PL,n,i} = \Omega(n \log n).
\]

5. Conclusions

The expected internal path length of a random binary tree is \( I_n = \Theta(n \log n) \). Empirical evidence suggests that performing many insertion and asymmetric deletions yields binary trees with an expected internal path length of \( I_{PL,n,i} = \Theta(n \log^2 n) \). Thus performing asymmetric deletions causes binary trees to become more unbalanced. Amazingly, the expected path length does not increase by a constant factor, but rather by a factor of \( \log^2 n \). However, experiments show
that the symmetric deletion algorithm improves the balance of binary trees leaving the expected internal path length $\Theta(n \log n)$, but with a smaller constant coefficient than the expected internal path length of a random binary tree.

Because this is an empirical study, the above conclusions can only be conjectures. No one has provided a theoretical explanation of the behavior of a binary tree's path length after applying deletions and then insertions. There is no proof that the asymptotic value of $IPL_{n,i}$ is less than $I_n$ when performing random insertions and symmetric deletions or that the asymptotic value of $IPL_{n,i}$ is greater than $I_n$ when applying insertions and asymmetric deletions.

In closing, it should be noted that the results of this study will have little impact on the use of binary trees in practice. It takes approximately 1.5 million random insertions and asymmetric deletions to make a 2048-node binary tree worse than a random tree, and 4 million before its expected internal path length reaches the asymptotic value (which is just 50% worse). When so many operations are required, other data structures are probably more appropriate.
6. Acknowledgements

I would like to thank Jon Bentley, James Gosling, Diane Lambert, and Jim Saxe for their help and guidance.

7. References

[Hibbard 62] Hibbard, Thomas N.
Some Combinatorial Properties of Certain Trees
with Applications to Searching and Sorting.

[Knott 75] Knott, Gary D.
*Deletion in Binary Storage Trees.*
STAN-CS-75-491.

[Knuth 68] Knuth, Donald E.
The Art of Computer Programming.
Addison-Wesley, 1968, Section 2.3.4.4.

[Knuth 69] Knuth, Donald E.
The Art of Computer Programming.
Volume II: Seminumerical Algorithms.
Addison-Wesley, 1969, Section 3.2.

[Knuth 73] Knuth, Donald E.
The Art of Computer Programming.
Addison-Wesley, 1973, Section 6.2.2.
Note: The Second Printing contains important changes in Section 6.2.2.

[Lewis 73] Lewis, T. G., and W. H. Payne
Generalized Feedback Shift Register Pseudorandom Number Generator

[Ross 70] Ross, Sheldon M.
*Applied Probability Models with Optimization Applications.*
Holden-Day, 1970, Section 4.3.
Figure 4: Examples of Insertion and Asymmetric Deletion.
Figure 5

Asymmetric Deletions, 64 Node Tree, 200 Runs

Asymmetric Deletions, 128 Node Tree, 200 Runs

Asymmetric Deletions, 256 Node Tree, 100 Runs
Figure 6

Asymmetric Deletions, 512 Node Tree, 50 Runs

Asymmetric Deletions, 1024 Node Tree, 25 Runs

Asymmetric Deletions, 2048 Node Tree, 20 Runs
Figure 7

Comparison Chart for Asymmetric Deletions

(Number of I/D Pairs) / n²
Figure 8

Symmetric (Alternating) Deletions, 64 Node Tree, 200 Runs

Symmetric (Alternating) Deletions, 128 Node Tree, 200 Runs

Symmetric (Alternating) Deletions, 256 Node Tree, 100 Runs
Figure 9

Symmetric (Alternating) Deletions, 512 Node Tree, 50 Runs

Symmetric (Alternating) Deletions, 1024 Node Tree, 25 Runs

Symmetric (Alternating) Deletions, 2048 Node Tree, 20 Runs
Comparison Chart of the Asymptotic Values of IPL(n,i)
<table>
<thead>
<tr>
<th>REPORT NUMBER</th>
<th>CMU-CS-82-146</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. TITLE</td>
<td>AN EMPIRICAL STUDY OF INSERTION AND DELETION IN BINARY SEARCH TREES</td>
</tr>
<tr>
<td>7. AUTHOR(s)</td>
<td>Jeffrey L. Eppinger</td>
</tr>
<tr>
<td>9. PERFORMING ORGANIZATION NAME AND ADDRESS</td>
<td>Carnegie-Mellon University Computer Science Department Pittsburgh, PA. 15213</td>
</tr>
<tr>
<td>11. CONTROLLING OFFICE NAME AND ADDRESS</td>
<td>Office of Naval Research Arlington, VA 22217</td>
</tr>
<tr>
<td>12. REPORT DATE</td>
<td>December 2, 1982</td>
</tr>
<tr>
<td>13. NUMBER OF PAGES</td>
<td>19</td>
</tr>
<tr>
<td>16. DISTRIBUTION STATEMENT (of this Report)</td>
<td>APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED</td>
</tr>
<tr>
<td>17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)</td>
<td>Approved for public release; distribution unlimited</td>
</tr>
<tr>
<td>20. ABSTRACT (Continue on reverse side if necessary and identify by block number)</td>
<td></td>
</tr>
</tbody>
</table>