The Effect of Asymmetric Deletions on Binary Search Trees

by

Joseph Carl Culberson

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Research Report CS-86-15
May 1986

The Effect of Asymmetric Deletions on Binary Search Trees

by

Joseph Carl Culberson

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, 1986

Abstract

The Effect of Asymmetric Deletions on Binary Search Trees

This thesis deals with the study of binary search trees subjected to insertions and deletions using the standard Hibbard deletion scheme. It has been widely believed that deletions followed by insertions caused the internal path length of binary trees to be reduced on average. Recent empirical results obtained by Eppinger and Culberson cast doubt on this assumption. We present strong theoretical and empirical evidence that long sequences of deletion and insertion pairs result in trees with an expected internal path length of $\Theta(N^{3/2})$. We also show that this theory applies to a wide variety of asymmetric deletion algorithms, including an improved one given by Knuth.

We present the results of extensive simulations in an attempt to verify the estimates of the leading coefficient and lower order terms of the internal path length. A simulation of special features of trees provides results on trees as large as 100,000 nodes.

Finally, we consider the problem of designing deletion algorithms without using rebalancing techniques that yield good average case behavior.

Acknowledgements

I would like to thank my supervisor Professor Ian Munro for suggesting this topic and for his advice during the development of this thesis. I would also like to thank those on my committee for taking the time to read this thesis and providing much helpful criticism. Special thanks then to Professor Derick Wood, Professor G. V. Cormack, Professor J. Kalbfleisch, and Professor P. Purdom. In addition, a special thanks to Gregory Rawlins for many helpful conversations and in particular for his help in solving some of the recurrences.

In addition, I owe a debt of gratitude to my wife Deborah Culberson, my sons Luke and Adam, and daughter Shannon for their support while I spent many weekends and evenings in pursuit of my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Binary search trees are well known data structures, often used when fast search, insertion and deletion are required. They also support nearest neighbor and range queries. When the search trees are well balanced, any of these operations can be done in $O(\log N)$ time on trees containing N items. Although binary search trees have been intensively studied, there is still much to be learned, particularly in the dynamic use of these trees.

We are concerned with the effect of deletions and insertions on the expected cost of accessing elements in a binary search tree using various deletion and insertion schemes that do not explicitly rebalance the tree. Thus, we do not consider either height or weight balanced trees, and this will be the only reference to such trees in this thesis. In particular, we are concerned with the effect of the Hibbard [11] deletion scheme, which was first published in 1962. The long term effect of this scheme has been an open problem since its publication. In Chapter 2 we show that under the assumption that each deletion is paired with an insertion, in a tree with N nodes, then there is strong reason to believe that the expected cost becomes $\Theta(N^{1/2})$. In Chapter 3 the results of extensive simulations are used to strongly support this conclusion. A preliminary study of these conclusions appears in [6]. We now introduce the problem more formally and follow that with a history of previous work.

1.1. Asymptotic Notation

We use the following definitions for our asymptotic notation. This notation is fairly standard and follows Gonnet [10].

$$f(n) = O(g(n)) \rightarrow \exists\ k \text{ and } n_0 \ni \quad |f(n)| < kg(n)\ \forall\, n > n_0$$

$$f(n) = \Omega(g(n)) \rightarrow g(n) = O(f(n))$$

$$f(n) = \Theta(g(n)) \rightarrow f(n) = O(g(n)) \text{ and } f(n) = \Omega(g(n))$$

$$f(n) = o(g(n)) \rightarrow \lim_{n\rightarrow\infty} \frac{f(n)}{g(n)} = 0$$

$$f(n) = \omega(g(n)) \rightarrow g(n) = o(f(n))$$

1.2. Definitions

We follow the definitions of the area given by Knuth [16]. A *binary search tree* is a finite set of nodes which is either empty, or consists of a root (containing a key) and two disjoint binary trees called the left and right subtrees. If v is a non-empty node of a tree, then $l(v)$ designates the left subtree, $r(v)$ the right subtree, and $k(v)$ the key of v. The *father* $f(v)$ of v is defined by the relation $f(l(v))=f(r(v))=v$. The *search property* is defined by the rule that for each node $v \in T$, each key in the left subtree of v is less than $k(v)$, and each key in the right subtree of v is greater than $k(v)$. This implies that each key in a search tree is unique. Henceforth, binary search trees will be called simply trees.

The *successor* $s(v)$ of a node $v \in T$ is defined to be that node in the right subtree with the minimum key. The *predecessor* $p(v)$ is similarly defined to be the node with the maximum key in the left subtree. If either subtree is empty, then the corresponding function is undefined. The natural definition for $s(v)$ when the subtree is empty would be the ancestor of v which contains the next largest key; however, for purposes of the description of the algorithm and the subsequent analysis, the above definition is preferable.

We measure the cost of searching for a key in a tree by the number of unsuccessful comparisons required to find the key, starting from the root. The *Internal Path Length* (IPL) of a tree, is the sum over all the nodes in a tree of the depth of the node, where the root is at depth 0. The average length of a search path is the IPL of the tree divided by the number of nodes N. The average length of a search path represents the average cost of accessing a key.

There are two basic operations that change trees, *insertion* and *deletion* of keys. To insert a key, a leaf is created containing the key and is attached in the unique position that maintains the search property of the tree. The natural and usual method for doing this can be stated recursively as

```
Insertion
    Insert($x$,$v$)
        if $v$ is empty, add a node $n$ with $k(n)=x$.
        else if $k(v)<x$ Insert($x$,$r(v)$)
        else Insert($x$,$l(v)$).
```

It is not known who first discovered this algorithm, although the first published descriptions were by Windley [21], Booth and Colin [2] and Hibbard [11] who apparently arrived at the method independently.

In contrast to the insertion problem, there does not seem to be any one natural algorithm for deletions. Deletions seem to be intrinsically more difficult. The algorithm which is the most widely known and used is the following due to Hibbard [11].

```
Hibbard Algorithm
    To delete a key $d$ from a tree $T$, find $v \in T$ such that $k(v)= d$.
    if $r(v)=\varnothing$ then do
        { The right subtree is empty, so delete the node containing key}
        { and re-attach the left son as the appropriate left or right son }
        { of the parent }
        If $v$ is not the root $[l/r](f(v)) \leftarrow l(v)$;
        { If $v$ is the root then $l(v)$ becomes the new root }
        remove $v$ from $T$.
    else do
        { Replace the key $d$ with the key from the successor }
        $k(v) \leftarrow k(s(v))$;
        { Delete the successor node, re-attaching its right subtree. Note that
        the successor never has a left subtree }
        if $s(v)= r(v)$ then $r(v) \leftarrow r(r(v))$
        else $l(f(s(v))) \leftarrow r(s(v))$
        remove $s(v)$ from $T$.
```

The following examples applied to the tree in Figure 1.1 may clarify these algorithms. For convenience the keys are chosen from the integers.

To insert the value 5 in this tree we simply attach it at the point indicated by the dotted line. To Hibbard delete the value 6 from the tree in Figure 1.1, we replace it by its successor, which is 7. The node containing 8 becomes the left son of the node containing 9, while the node which contained 7 is deleted. To Hibbard delete 4, which has no right son, the node is deleted and 2 becomes the new left son of 6.

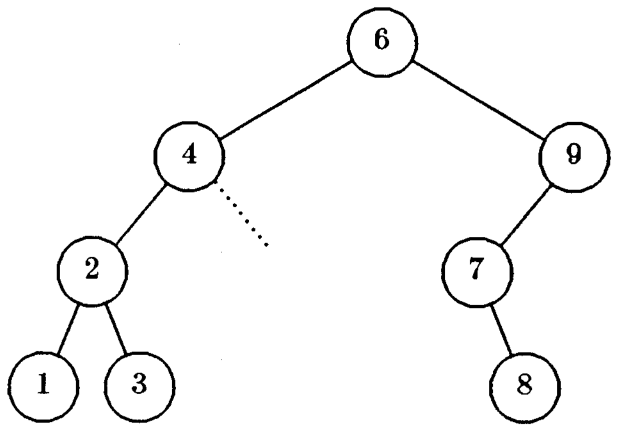


Figure 1.1 Example Binary Search Tree

By a *random tree* on N nodes we mean a binary tree generated by inserting a random sequence of N keys (all permutations equally likely), using the above algorithm. That is, we are referring to the probability distribution of the shapes of trees generated by random insertion sequences. It is well known (See for example, Knuth [16] Chapter 6.2.2) that the expected IPL of a random tree is approximately $1.386\, N \log_2 N$. We caution the reader that this is quite different from all trees being equally likely.

The general problem we will address is that of determining the expected IPL of a tree generated by a random sequence of random insertions and random deletions. The problem as stated is somewhat ill-defined. To be more precise (and to make the problem more tractable) we define an *update* on a tree to be the deletion of one key selected at random from the tree, followed by an insertion into the resulting tree of a new random key.

We begin with any tree T_0 on N nodes, for example one generated by inserting N random keys from the domain $D=(0,1)$. For times $j>0$, we define the tree T_j as the tree created by performing one update (consisting of a random deletion followed by a random insertion) on the tree T_{j-1}. The insertions are performed by the standard leaf insertion algorithm [16]. The deletions are performed by the Hibbard algorithm. It is assumed that the keys are drawn from a fixed dense domain and that the probability of deletion is the same for each key in the tree. For purposes of simplifying the analysis, we consider the keys to be drawn from a uniform(0,1) domain, although this does not affect the validity of the result for other domains. (See section 2.8). Under these assumptions, we will estimate, under certain reasonable assumptions, the asymptotic upper and lower bounds on the expected value of the Internal Path Length (IPL) for T_j as $j \to \infty$.

1.3. Historical Background

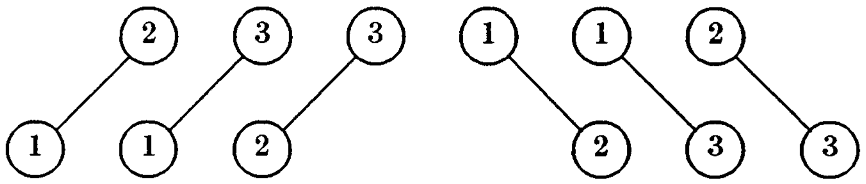
As has already been indicated, the history of binary search trees extends back into the late 1950's. However, the problem with which we are concerned, namely the effect of sequences of deletions and insertions on binary search trees, can be considered as having originated with Hibbard's [11] publication of a deletion algorithm. The algorithm is decidedly asymmetric, in that whenever the successor of the node containing the key scheduled for deletion exists, that key is used to replace the key in the node. Thus, the node effectively moves to the right in the domain from which the keys are chosen. In light of this observation, the following theorem, which Hibbard proved in his original paper [11], is quite surprising.

Theorem

If one key in a random tree of $N+1$ nodes is selected at random and deleted using Hibbard's algorithm, the result is a random tree on N nodes.

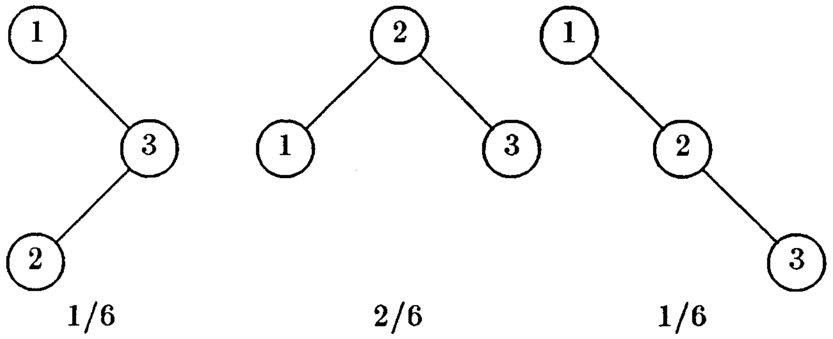
Recalling our earlier definition of a random tree, what this means is that the probability of obtaining any particular shape of tree by inserting $N+1$ elements in random order and then deleting one at random, is the same as the probability of obtaining that shape by inserting N elements in random order. The intuitive conclusion is that the Hibbard deletion algorithm leaves the distribution of trees unchanged, and thus sequences of insertions and deletions generate trees with the same shape distribution as those generated by pure insertions. This, however, is false. It was first observed by Knott [14] in 1972 that the first insertion after a deletion changes the distribution of shapes.

The reason for this is that, although the shape distribution is maintained for a deletion, the distribution of values within shapes is not maintained. To properly describe the distribution of trees of size N, we must include in the definition both shape and values. For example, in Figure 1.2.A we see the distribution of trees of size 2 generated by insertions drawn at random from the domain {1,2,3}. Also displayed is the probability of each of these trees



2 Insertions Only	1/6	1/6	1/6	1/6	1/6	1/6
3 Insertions 1 Deletion	1/6	2/9	1/9	1/6	1/9	2/9

A: Tree Probabilities



B: Delete 1 to Obtain Rightmost Above

Figure 1.2 Illustration of Probability Changes Caused by Deletion

being generated from a sequence of three random insertions followed by one random deletion. There are only two shapes for trees of size 2, and each of these shapes occurs with probability of one-half under either method of generation. However, the probabilities of the individual trees do change. For example, the probability of the rightmost tree in the figure increases from 1/6 to 2/9. In Figure 1.2.B, the only trees that can generate the rightmost tree of size two through a deletion (of key 1) are shown, together with the probability of their occurrence.

It is now readily seen how subsequent insertions change the distribution of shapes. By assuming that a new key is equally likely to fall into any of the four intervals (0,1), (1,2), (2,3), or (3,4), we can compute the probability of occurrence for each of the tree shapes of size three given the above probabilities for the trees of size two. For example, the insertion of a key in the interval (1,2) only creates a balanced tree of size three when inserted into the rightmost tree of Figure 1.2.A. Summing over all the cases, we see that the probability of the balanced tree is increased when a deletion is included.

In his thesis, Knott [14] reported the results of extensive simulations of the Hibbard deletion algorithm for trees of size up to 98 nodes. In these simulations a random tree was first generated through a sequence of random insertions, then the tree was subjected to a sequence of deletion/insertion pairs. In all cases, what appeared to be the asymptotic average IPL was less than the expected IPL of a random tree. This led him to conjecture that the Hibbard algorithm causes the search time to be improved when used with the standard insertion algorithm for random sequences of insertions and deletions.

In 1978, Jonassen and Knuth [12] published a lengthy and difficult analysis of the algorithm, proving that Knott's conjecture is correct for trees of size 3, thus strengthening the conjecture for trees of all sizes. This paper is an excellent exposition of the extreme difficulties involved in a complete and exact analysis of this problem. An analysis of trees with four nodes was done by Baeza-Yates [1] in 1985 with similar results.

In 1982, Eppinger [7] performed much more extensive simulations of the Hibbard algorithm, for trees as large as 2048 nodes and for more than N^2 updates. The surprising result was that for trees of size greater than about 128 nodes, the asymptotic IPL appeared to increase. Using regression techniques, Eppinger conjectured that the expected IPL was $O(N\log^3 N)$, which is considerably worse than the $O(N\log N)$ expected IPL of a random tree. It was further noted that to reach the plateau assumed to be the long term expected IPL, required on average N^2 updates.

Eppinger also simulated the effects of a symmetric version of Hibbard's algorithm, in which for each deletion it is randomly decided whether to use the predecessor or the successor as the replacement key. In all the tests performed, the average IPL was reduced after many updates. These results are considered further in Chapter 4.5.

In 1983, the author [5] ran further simulations on a variety of algorithms, generally confirming Eppinger's results. However, it was not clear that the $O(N\log^3 N)$ conjecture could be supported over other simple formulations that gave equally good fits to the limited data afforded by the simulations. It was conjectured that one significant effect of the asymmetry of the Hibbard algorithm is that the tree should become skewed to the left. This is a direct result of the root of any subtree always receiving a larger key when its current key is deleted and the right subtree is not empty. Various measures were devised to measure the imbalance of a binary tree and when applied, these measures showed strong empirical evidence of the conjectured skewing. One of these measures was the number of nodes in the left linear descendents of the root; this set of nodes was called the backbone of the tree.

In the following chapter, an approximate mathematical analysis of the length of the backbone and the size of its subtrees, subject to certain hypotheses, is presented. The results of this analysis are quite convincing and support the theory that the expected IPL of trees subjected to a large number of random updates is $\Theta(N^{3/2})$. In Chapter 3 the data from previous simulations and from further extensive simulations is used to strongly support this theory.

Chapter 2

Models and Analyses

In this chapter we develop models to explain the behavior of binary trees when modified by long sequences of random updates. We first present a descriptive analysis of the behavior of the tree, laying an intuitive basis for understanding the later refined models. Formal analysis of the simpler models then follows.

2.1. The Intuitive Basis for Our Models

As has been previously noted, when a node that has a non-empty right subtree has its key scheduled for deletion, the Hibbard algorithm brings up the next larger key from the right subtree to replace it, and further adjustments are then made in the right subtree. However, our approach ignores the shape of the right subtree. This process can be thought of as moving the node to the right in the domain.

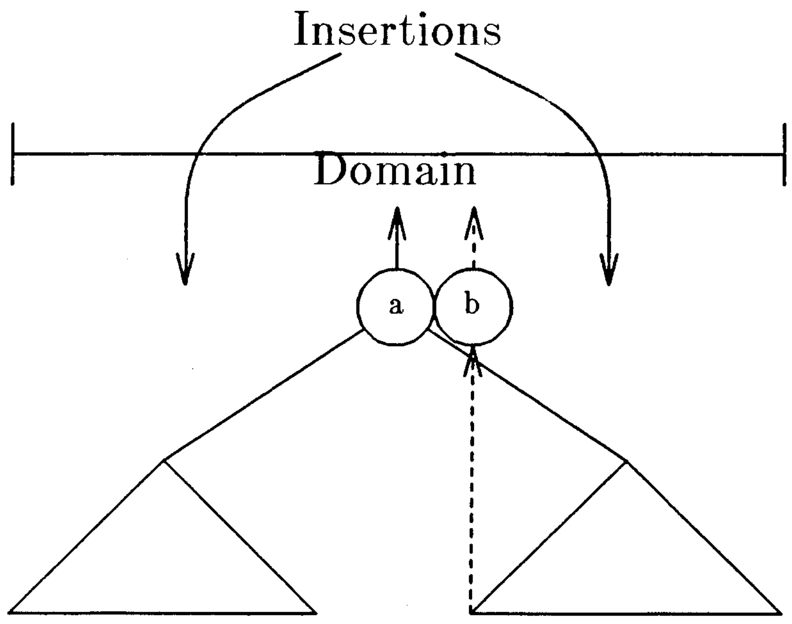


Figure 2.1 Domain Splitting by Root.

For example, if we consider Figure 2.1, we see that on deleting the key ‘a’ from the tree, the root gets the new key 'b'. The root acts as a divider for all future insertions, and so after the deletion the probability of falling into the left subtree is increased for each insertion. Intuitively, the left subtree should be increased on average by one node, and the right subtree should be decreased by one during the next few moves. The interval (0,*Root*) has been increased by the distance between the two adjacent keys. The interval between the root and its left son has also been increased, and so the right subtree of the left son of the root is expected to show the actual increase as future insertions are made. The size of the interval between any two nodes in the tree determines the expected number of nodes between them.

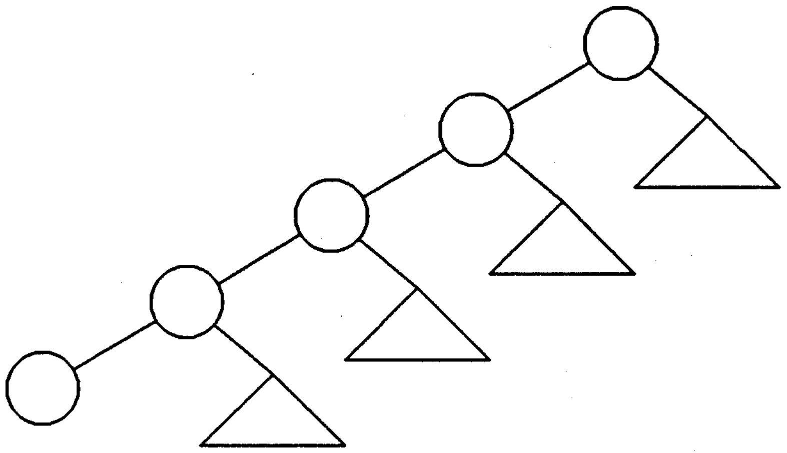


Figure 2.2 Tree After Many Updates.

However, the roots of each of the subtrees in the tree are also equally subject to deletion, and so we expect the tree to eventually come to look like that in Figure 2.2.

The nodes down the left side of Figure 2.2 are called the ***backbone*** of the tree. Note that the left-most node in the backbone must contain the smallest key in the tree. We will argue that after N^2 or more updates have occurred, there are on average $\Theta(N^{1/2})$ nodes in the backbone, and that each of the subtrees of the backbone are of average size $O(N^{1/2})$.

We wish to study the behavior of the backbone, and in particular the intervals between the backbone nodes. One approach to this is to study the node motions over the domain. When the right subtree of a backbone node is empty, the node is deleted from the backbone when its key is deleted. This complicates the picture somewhat.

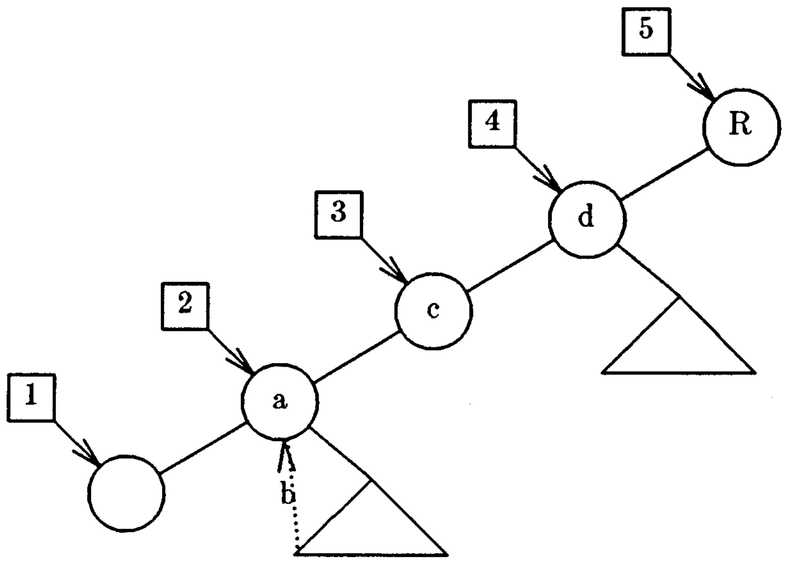


Figure 2.3 Tag Model.

To eliminate the problem of disappearing nodes, we track the backbone nodes by attaching *tags* in the manner discussed below and pictured in Figure 2.3. There is a tag on each of the backbone nodes of the initial tree. When a node is inserted into a backbone at time j, a tag with label j is attached to the node. This can occur only when the key to be inserted is smaller than any other key in the tree. When a node is deleted from the backbone, all tags attached to it are reattached to the next larger member of the backbone. Thus, tags may clump together on a node, and if so, will remain together thereafter. Note that when a key is deleted from the backbone, and the node containing the key has a right son, then the node is not deleted and so the tags attached to it stay on that node.

We require a device that allows the tags to move freely off the tree without introducing special boundary conditions. To do this we create a dummy root element (R), with key value $k(R)=1$. The dummy root is not subject to deletion or alteration. (Note that $1 \notin D$). Since 1 is larger than any key in the tree, the dummy root will have only a left subtree, which is the entire tree of N nodes. We emphasize that the root of the tree is still the real root, and that the dummy is just a position for the tags to move smoothly onto when they leave the tree.

The *value* of a tag is the key of the node with which it is associated. The tag *moves* at time j if its value changes.

Again, note that it is **not** necessary to reattach a tag to move it, since replacing the key in the node moves the tag. It is important to maintain the distinction between deleting keys and deleting nodes from the tree.

To make all this a bit clearer, consider the following examples based on Figure 2.3. If the key 'c' is deleted, then the node containing 'c' is deleted making the node with 'a' the left son of 'd'. The tag with label '3' will be reattached to 'd', giving that node two tags. (If more tags had been on 'c', they too would have been reattached.) If key 'a' were deleted, then 'b', the smallest key in the right subtree of the node, would be brought up to replace it. The subtree would also be altered in this case, but we will not be concerned with that. Note, however, that the tags in each case moved to the next larger key in the tree. Eventually, all these tags would be moved to the dummy root R. New tags will be created one at a time at the far left whenever a new smallest key is inserted.

The following observation is the key to our analysis. It is easily proved by analyzing the various cases that can occur during the deletion and insertion of keys in the backbone.

Critical Observation:

If a tag moves at time j, then it assumes the value of the next larger key in the tree at time $j-1$.

We can now partially understand the behavior of trees by studying the behavior of tags. We will show that the tags tend to become clumped as they move, and that the intervals between tags, and thus between backbone nodes, apparently become distributed to achieve a $\Theta(N^{1/2})$ average search path length.

Intuitively, the tags, starting near zero, perform a random walk to the right, taking a step of approximate average size $1/(N+1)$ with probability of $1/N$ per update. After an average of N steps (requiring an average N^2 updates) the tag will reach the dummy root; that is, it will have fallen off the tree. Knowing the nature of random walks, it seems reasonable that a tag would deviate $\Theta(N^{1/2})$ steps from its expected position before reaching the right side. Since the keys are uniformly distributed, this implies $O(N^{1/2})$ nodes between tags. Since the backbone nodes are precisely the ones with tags, achieving the square root deviation implies that there are $\Theta(N^{1/2})$ nodes in the backbone. It also means that each right subtree of the backbone is expected to contain $O(N^{1/2})$ nodes. Assuming that the simplified models of this tagging process given in the next section are asymptotically accurate, we can prove that after $\Omega(N^2)$ updates a tree will have an expected search path length of $\Theta(N^{1/2})$.

2.2. Analysis of A Special Case: EFD's

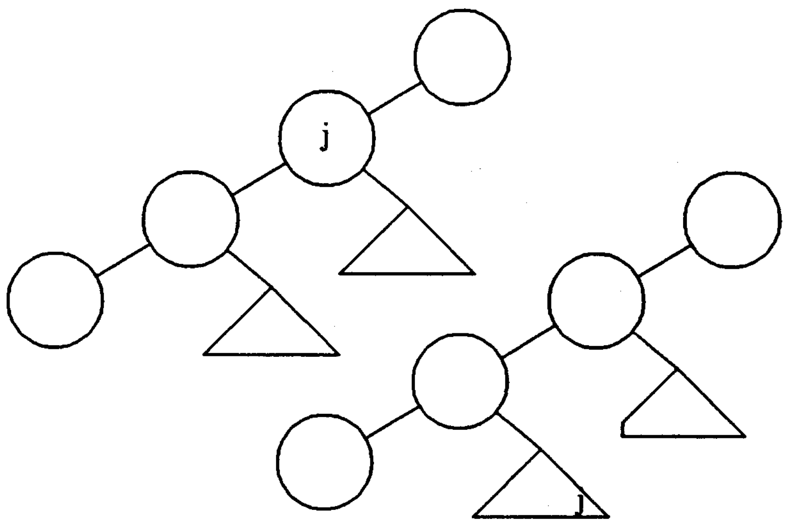
We begin our modeling with the study of a special case, namely that the universe of keys is severely restricted. We define an *Exact Fit Domain* (EFD) to mean that there were exactly N keys available in the domain, and that an update therefore consists of deleting a key and then re-inserting that same element into the tree, using the deletion and insertion algorithms of interest. The effect of various updating schemes on trees of small size in such domains was studied by the author in [5].

We now study the asymptotic behavior of EFD trees on N nodes, with the updating process using Hibbard deletions and leaf insertions. New tags are created whenever the smallest key is inserted. It is easily seen that any tag created after time zero will move exactly N times until it falls off an EFD tree. If a tag moves, then exactly one new node is inserted to its left on that update, and it falls between the tagged node and the previous immediate predecessor of the tagged node, unless the node is the smallest in the tree, in which case the inserted node is a new backbone node. Since the relative size of the keys is irrelevant to a tree on an EFD, we consider the keys to be equally distributed, and therefore we say that a tag moves one ***step*** when it moves.

In this section, we derive the expected number of nodes in the subtree containing the kth smallest node when the node is first inserted into the subtree. From this we are able to obtain bounds upon the expected size of the subtrees of the backbone, as well as the expected number of nodes in the backbone. These bounds can then be used to show that the EFD tree has an expected IPL of $\Theta(N^{3/2})$ in the long term.

We study the size of an interval between two adjacent tags for those cases in which the two tags remain apart. The number of changes to a non-collapsing interval between two initially adjacent tags during the lifetime of the upper tag must be less than or equal to $2N-2$, since the upper tag can move no more than $N-1$ times after the lower tag is created, and the lower tag must move no more times than the upper, or they will collide.

Note that only those updates which affect the size of the interval in question concern us here. Updates on keys other than the two at the extrema of the interval have no effect on the number of keys in the interval. Updates on the keys within the interval may change the shape of the subtree, but we do not analyze this shape. Since an update reinserts the deleted key, the new key must fall in the same interval, and so the number of keys in the interval does not change for such updates. Similar remarks apply to updates on keys which fall outside the interval. We refer to the number of keys in the interval as the size of the interval, or equivalently as the size of the subtree.



Update On j Moves It Into The Subtree

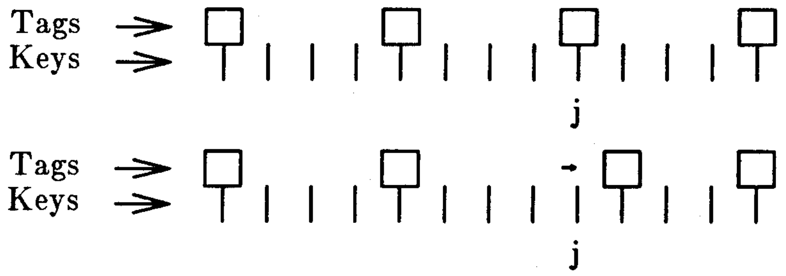


Figure 2.4 Moving The jth Node Into The Interval

We now compute the expected size of the interval when the upper tag moves past the jth key as in Figure 2.4. To understand the motivation behind this, notice that as long as j is the largest (i.e. rightmost) key in the interval, then the size of the interval can only be less than or equal to its size when j entered it. We see that the expected size of the right subtree of the root will be less than or equal to the value obtained by setting $j=N$ in the following lemma.

Lemma 2.1

In an EFD, the expected size of the interval containing the jth smallest key at the time it enters the interval is

$$E_j = \frac{2^{2j-2}}{\binom{2j-2}{j-1}} - 1$$

$$\approx \sqrt{\pi j}$$

Proof: On the first move of the upper tag, the second smallest key is added to the interval between the two tags. Similarly, the jth key is added to the interval on the $j-1$st move of the upper tag. To simplify the algebra, we let $k=j-1$, and solve for the expected number of nodes in the interval between two tags after the upper tag has moved k times under the condition that after any ith move of the upper tag, $1 \le i \le k < N$, the lower tag has moved fewer than i times.

We can model the behavior of the number of nodes in the interval by a simple coin flipping process. Here a head corresponds to a move of the right tag, or an increase of one in the number of nodes, and a tail to a move of the left tag, or a decrease of one in the number of nodes. The number of nodes in the interval corresponds to the number of heads minus the number of tails. We start with no heads or tails, and we wish to know what the expected difference is after some number of heads, given that the number of tails never exceeds the number of heads. This process is known in the literature as a simple random walk with an absorbing barrier at -1. See, for example, Feller [8].

To aid in understanding the analysis, we represent the possible walks for $k \le 3$ in Figure 2.5. Initially, we start at the origin, which is the top circle of the diagram. We use k to represent the number of steps to the right, and i

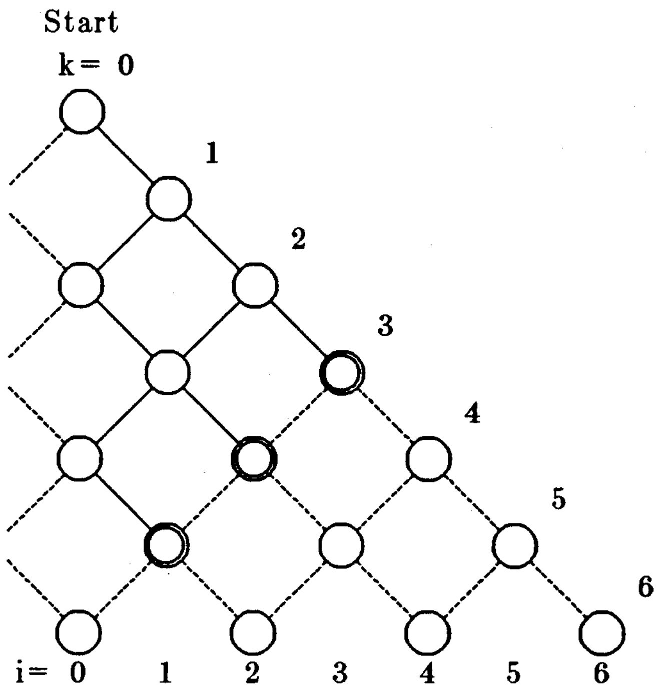


Figure 2.5 Exact Fit Domain Analysis

the distance from the origin. Thus, i under the coin model is the difference between the number of heads and the number of tails. In the figure all moves are downward, and either to the right, corresponding to a head (move of the upper tag), or to the left corresponding to a tail (move of the lower tag).

Using our oblique coordinate system, we let (i,k) be the position in the diagram representing a distance of i from the origin after k heads. We let $P_{i,k}$ represent the number of ways of reaching (i,k) with no point on the path having a negative i component, and also the last move being be a head. In the diagram, this corresponds to the number of paths by which we could reach the ith position from the origin along the upward diagonal ending at k. For example, the positions reachable with $k=3$ are indicated by the double circles in the diagram. Here, $P_{1,3} = 2$, $P_{2,3} = 2$ and $P_{3,3} = 1$. Note that $P_{0,3} = 0$, since we are looking at the state immediately after the kth right move. Thus, in the diagram we do not include the dashed edges along the third diagonal in computing $P_{i,3}$. Note the implication that there is always at least one node in the initial right subtree of the root (for $N>1$).

Since left and right moves are equally probable, each time we move down one level in the diagram we would double the number of possible paths available in an unbounded version of the random walk. The depth of a point (i,k) is $2k-i$, and so the probability of reaching (i,k) on a random walk by one of the $P_{i,k}$ paths is $(1/2)^{2k-i}P_{i,k}$. The expected value of i for a given k and under the condition that $i \geq 0$ at all times is

$$E_k = \frac{\sum_{i=1}^{k} i\left(\frac{1}{2}\right)^{2k-i} P_{i,k}}{\sum_{i=1}^{k} \left(\frac{1}{2}\right)^{2k-i} P_{i,k}}$$

We multiply the numerator and denominator by 2^{2k-1} to obtain

$$E_k = \frac{\sum_{i=1}^{k} i2^{i-1}P_{i,k}}{\sum_{i=1}^{k} 2^{i-1}P_{i,k}}$$

Note that $2^{i-1}P_{i,k}$ is the number of ways of reaching the $2k-1$st level by passing through (i,k) and not $(i+1,k)$. Notice that the $2k-1$st level is the level of the point $(1,k)$ and the point (k,k) is at level k. Thus, there is exactly one way to get from $(1,k)$ to the $2k-1$st level. From $(2,k)$ there are two ways to reach the $2k-1$st level; either move down and left or down and right. In general, from (i,k), level $2k-1$ is achieved by $i-1$ moves, each with two choices, giving us 2^{i-1} ways of reaching the $2k-1$st level. Thus, the denominator in the right hand side is equivalent to the number of ways of reaching the $2k-1$st level without passing to the left of the origin. We designate this value by

$$D_k = \sum_{i=1}^{k} 2^{i-1}P_{i,k}$$

We get the recurrence

$$D_{k+1} = 4D_k - C_k$$

with $D_1 = 1$ and C_k is the number of ways of reaching the first node on level $2k-1$, that is the position $(1,k)$. We note that the point $(1,k+1)$ is two levels deeper than $(1,k)$. From each node at level $2k-1$, there are 4 ways of reaching level $2k+1$, except that from the leftmost node one of these ways leads to the left of the origin and so must be discounted. Thus, we obtain the recurrence. It is well known that C_k is the kth Catalan number, (see for example Jow [13])

$C_k = \frac{1}{k+1}\binom{2k}{k}$. The reader may recognize this more easily as the number of binary trees on k nodes. It now easily follows by induction that

$$D_k = \frac{1}{2}\binom{2k}{k}$$

Similarly, we let

$$N_k = \sum_{i=1}^{k} i2^{i-1}P_{i,k}$$

which is the numerator for the expectation of i. The left and right moves from (i,k) to the $2k-1$st level are entirely symmetric. That is, for every path from any position (i,k) to a position $i-j$ on the $2k-1$st level, there is a mirror image path from (i,k) to a position $i+j$ on the $2k-1$st level. Thus, N_k is equal to the sum of the distances of the ith position times the number of ways of reaching the ith position on the $2k-1$st level without ever passing to the left of the origin. (Notice that only the odd positions are reachable on these odd levels). We get the recurrence

$$N_{k+1} = 4N_k + C_k$$

by noticing that, except for the first node, from position i on level $2k-1$, there is one way to reach position $i-1$ on level $2k+1$, two ways to reach position i and one way to reach position $i+1$. If the contribution to N_k of position i is $(2i-1)Z_i$, then the contribution to N_{k+1} is $((2i-3)+2(2i-1)+(2i+1))Z_i$ which is 4 times its contribution to N_k. The first node does not contribute the $-1C_k$ implied by this argument, since we disallow the path to the left of the origin, and thus it must be added in to complete the recurrence.

If we form the sum

$$N_{k+1} + D_{k+1} = 4(N_k + D_k)$$

and note that $N_1 = D_1 = 1$, we obtain

$$N_k = 2^{2k-1} - D_k$$

Thus,

$$E_k = \frac{N_k}{D_k}$$

$$= \frac{2^{2k-1} - D_k}{\frac{1}{2}\binom{2k}{k}}$$

$$= \frac{2^{2k}}{\binom{2k}{k}} - 1$$

$$= \frac{2^{2k}(k!)^2}{(2k)!} - 1$$

Using Stirling's approximation $k! \approx \sqrt{2\pi k}(k/e)^k$, we obtain

$$E_k \approx \sqrt{\pi k}$$

Finally, recall that the upper tag starts on the 2nd smallest key, and thus the kth move of the upper tag corresponds to adding the $k+1$st key to the subtree. Thus, by substituting $j-1$ for k we finish our proof.

□

Note that in the preceding lemma we did not compute $P_{i,k}$. This can be computed using the ballot theorem of Feller [8] Chapter III.1 and is

$$P_{i,k} = \frac{i}{2k-i}\binom{2k-i}{k}$$

In what follows, the jth subtree refers to the right subtree of the jth backbone node, where the root is the first backbone node. If there is no jth backbone node, because the backbone has fewer than j nodes, then we consider the jth subtree to be empty; that is, it has a size of zero.

Lemma 2.2

The expected size of the jth subtree on an EFD after sufficiently many updates is $O(N^{1/2})$, for all j.

Proof: This follows from the previous lemma on observing that the expected size of the subtree to the left of a tag on the kth node is $O(\sqrt{k}) = O(\sqrt{N})$. The topmost subtree is $O(\sqrt{N})$ when the Nth node is first introduced, and further updates can only reduce this size until the Nth node is eventually tagged again.

□

In the next lemma we compute a bound on the length of the backbone.

Lemma 2.3

The expected number of nodes in the backbone of the EFD tree is $O(\sqrt{N})$ after sufficiently many updates.

Proof: We define the *weight* w_k of the backbone with respect to the kth node as follows. Assume that the jth node is tagged, and that the following i

nodes are untagged. Then the weight of the kth key, $j \le k \le j+i$ is $1/(i+1)$. Thus, we see that the number B of nodes in the backbone of an N node tree is

$$B = \sum_{k=1}^{N} w_k$$

Intuitively, the density of the backbone nodes near the right of a tree is not increased by the fact that they are near the upper boundary. More formally, if we consider the first N nodes of an M node tree, where M may be arbitrarily large, we see that the number of nodes in the backbone over the first N nodes is identical to the number of backbone nodes in an N node tree in which the set of updates is the same as the set of updates over the first N nodes of the M node tree. This is true since an update on any key k does not affect the structure of the backbone to the left of the key. Thus, where we now consider the weights to be over the M node tree we see that the number of backbone nodes over the first N nodes is

$$B \le 1 + \sum_{k=1}^{N} w_k$$

where the 1 comes from the observation that the rightmost backbone node may now be distributed over some of the keys to the right of the Nth node. If we consider the average weight taken over time on the kth key, we see then that

$$\mathrm{E}[B] \le 1 + \sum_{k=1}^{N} \mathrm{E}[w_k]$$

We now compute an upper bound on $\mathrm{E}[w_k]$, for $k<M$. We proceed in a manner similar to that in the proof of lemma 2.1 First, we compute the expected weight R_k when the kth key first falls into the subtree. (Actually, we compute the ratio for the $K+1$st, but this won't make any difference in the asymptotic result that we derive).

$$R_k = \frac{\displaystyle\sum_{i=1}^{k} \frac{1}{i+1} \left(\frac{1}{2}\right)^{2k-i} P_{i,k}}{\displaystyle\sum_{i=1}^{k} \left(\frac{1}{2}\right)^{2k-i} P_{i,k}}$$

where the term $\frac{1}{i+1}$ is the assigned weight, given that there are i untagged ones for the one tagged one at the left of the interval. Again we multiply the numerator and denominator by 2^{2k-1} to obtain

$$R_k = \frac{\sum_{i=1}^{k} \frac{1}{i+1} 2^{i-1} P_{i,k}}{\sum_{i=1}^{k} 2^{i-1} P_{i,k}}$$

Using the result that $P_{i,k} = \frac{i}{2k-i}\binom{2k-i}{k}$, which comes from applying the Ballot theorem of Feller [8] we find that the expected ratio is

$$R_k = \frac{\sum_{i=1}^{k} 2^{i-1} \frac{i}{2k-i}\binom{2k-i}{k}\frac{1}{i+1}}{D_k}$$

where D_k is defined as in lemma 2.1. We now compute the value of the numerator by

$$\sum_{i=1}^{k} 2^{i-1}\frac{i}{2k-i}\binom{2k-i}{k}\frac{1}{i+1} < \sum_{i=1}^{k} 2^{i-1}\frac{1}{2k-i}\binom{2k-i}{k}$$

$$= \frac{1}{k}\sum_{i=1}^{k} 2^{i-1}\binom{2k-i-1}{k-1}$$

$$= \frac{1}{k}\sum_{j=0}^{k-1} 2^{k-j-1}\binom{k+j-1}{k-1}$$

$$= \frac{1}{k}2^{k-1}\sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^{j}\binom{k+j-1}{k-1}$$

and using the equivalence (see Feller [8] page 50)

$$\binom{x}{r-1} = \binom{x+1}{r} - \binom{x}{r}$$

we continue with

$$= \frac{2^{k-1}}{k}\sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^{j}\left[\binom{k+j}{k} - \binom{k+j-1}{k}\right]$$

$$= \frac{2^{k-1}}{k}\left[\sum_{j=0}^{k-1}\left[\left(\frac{1}{2}\right)^{j}\binom{k+j}{k} - \left(\frac{1}{2}\right)^{j-1}\binom{k+j-1}{k}\right] + \sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^{j}\binom{k+j-1}{k}\right]$$

and using the telescoping property we get

$$= \frac{2^{k-1}}{k}\left[\binom{2k-1}{k}\left(\frac{1}{2}\right)^{k-1} + \sum_{j=0}^{k-1}\left(\frac{1}{2}\right)^{j}\binom{k+j-1}{k}\right]$$

and similarly

$$= \frac{2^{k-1}}{k}\left[\binom{2k-1}{k}\left(\frac{1}{2}\right)^{k-1} + \binom{2k-1}{k+1}\left(\frac{1}{2}\right)^{k-1} + .. + \binom{2k-1}{2k-1}\left(\frac{1}{2}\right)^{k-1}\right]$$

$$= \frac{1}{k}\sum_{i=k}^{2k-1}\binom{2k-1}{i}$$

$$= \frac{2^{2k-1}}{2k}$$

Recalling that

$$D_k = \frac{1}{2}\binom{2k}{k}$$

from lemma 2.1, and using Stirling's approximation we find that the expected weight is

$$R_k < \frac{\sqrt{\pi}}{2\sqrt{k}}$$

where the inequality follows from the first inequality above. To see that this also acts as an approximate upper bound on the expected weight at all other times, we note that at any time other than when the key first enters a subtree one of two cases must hold. First some number of updates may have occurred since the key entered. However, when the next key enters the tree, we see that the number of untagged keys in the interval is expected to be larger, or more to the point, that the weight on k is reduced. Thus, our bound acts as an upper bound to this case. In the second case, the key may be tagged. In this case, the argument applies to the interval to the right, and we are counting the tag on the left of the interval as being part of the interval. Thus, our upper bound again applies.

To complete the proof,

$$E[B] \le 1 + \sum_{k=1}^{N} R_k$$

$$= \sum_{k=1}^{N} O(\frac{1}{\sqrt{k}})$$

$$= O(\sqrt{N})$$

where the last follows by comparing the summation of $1/\sqrt{k}$ with the integral to see that the result is $\approx 2\sqrt{N}$.

□

We note that in fact the upper bound is approximately $\sqrt{\pi N} \approx 1.77..\sqrt{N}$. Since this is based on the expected weight when the kth key first enters the interval, it is in a sense based on the minimal expected interval, that is on a maximal expected weight. On the other hand, if we look at the expected interval size, we can see that when the kth key is first tagged the size of the interval to its right is still $O(\sqrt{k})$. This suggests that the expected size of the interval is $\sqrt{\pi k} \pm o(\sqrt{k})$ over all time. We conjecture that a better approximation to the coefficient can be obtained by ignoring the small order term and taking the inverse of this expectation to be the expected proportion of the time that the kth key is tagged. Thus,

$$\begin{aligned} \mathrm{E}[B] &\approx \sum_{k=1}^{N} \frac{1}{\sqrt{\pi k}} \\ &\approx \frac{2}{\sqrt{\pi}}\sqrt{N} \\ &\approx 1.128..\sqrt{N} \end{aligned}$$

This conjecture has the virtue of being in excellent agreement with the results of the simulations performed upon EFD trees, as presented in Chapter Three.

We now have two pieces of information which together imply that the Internal Path Length of the EFD tree is $\Theta(N^{3/2})$ if more than N^2 updates have been performed. These are the upper bound on the subtree size, and the upper bound on the expected number of backbone nodes. We first prove the lower bound on the IPL.

Theorem 1

The asymptotic IPL of a tree is $\Omega(N^{3/2})$.

Proof: We let B be the number of backbone nodes in a tree, and N_i, the number in the ith interval from the right of the tree. Clearly,

$$\mathrm{IPL} \geq \sum_{i=1}^{B} iN_i$$

Now consider an ideal tree T, in which each interval between backbone nodes contains $K=\Theta(N^{1/2})$ keys, and the constant is larger than the constant in the bound of lemma 2.3. If we consider the K largest keys, they will contribute less to the above sum in the ideal tree than the K largest keys would on average contribute to the sum on an average tree, since the top subtree would on average be smaller than our ideal subtree, and so some of these nodes would on average be in the second subtree. Similarly, the jth largest subset of K

keys corresponding to any interval j of the ideal tree would contribute more to the average tree, since

$$\sum_{i=1}^{j} i\mathrm{E}[N_i] \geq \sum_{i=1}^{j} iNT_i$$

where NT_i is the number in each subtree of the ideal tree, and thus some of the keys in the ith subset would be in a deeper subtree on average. Summing over all N nodes in this way leads to the conclusion that the IPL of the average tree is greater than the IPL of the ideal tree. Noting that having $\Theta(N^{1/2})$ nodes in each subtree implies that the ideal tree has $\Omega(N^{1/2})$ such subtrees, we get

$$\mathrm{IPL} \geq \sum_{i=1}^{\Omega(N^{1/2})} \Theta(N^{1/2})i$$

$$= \Omega(N^{3/2})$$

□

Next we prove the upper bound.

Theorem 2

The asymptotic IPL of a tree is $O(N^{3/2})$.

Proof: Now we build a pessimal tree, in which we put $\Theta(N^{1/2})$ nodes in the backbone with a constant larger than that of the expected number of nodes. We then add nodes until each subtree contains $\Theta(N^{1/2})$ nodes, as in the previous theorem. Note that this tree will have more than N nodes. For our upper bound we assume that each subtree is linear, which is the worst case. We note that the IPL can be computed as the sum over each subtree of the IPL of that subtree, plus the sum of the distances to the roots of the subtrees times the number of nodes in the corresponding subtree. The IPL of each subtree is

$$IPL_s = \sum_{i=1}^{\Theta(N^{1/2})} i$$

$$= \Theta(N)$$

Since there are $\Theta(N^{1/2})$ such subtrees, this implies that the sum of all the IPL's of the subtrees is $\Theta(N^{3/2})$. The sum of the distances to the roots of the subtrees is

$$\sum_{i=1}^{\Theta(N^{1/2})} i\Theta(N^{1/2}) = \Theta(N^{3/2})$$

Thus, the total of these two is also $\Theta(N^{3/2})$. We see that for each subtree of the average tree, the contribution to the IPL is less than the contribution of the corresponding subtree in the pessimal tree, and there are more subtrees in the pessimal tree than in the average tree. Hence, the IPL of the average tree is less than or equal that of the pessimal tree, which completes the theorem.

□

Finally, we combine these two theorems to form

Corollary
The Internal Path Length of the EFD tree is $\Theta(N^{3/2})$.

2.3. An Alternate Model

We remind the reader that the only requirements for the above two theorems and corollary to hold are that the expected number of nodes in the backbone is $O(N^{1/2})$ and that the expected number of nodes in any subtree of the backbone is $O(N^{1/2})$. We now give an alternate proof of the first fact for the EFD model. Although this method gives much weaker bounds in the sense that any coefficients computed by this method are less precise than the above method, it does appear that the arguments given here should also apply to trees in which the keys for insertion are drawn at random from the domain (0,1), independently of the keys already in the tree. After completing this alternate proof for the EFD model, we will attempt to justify this intuition more fully.

Finally, we will present an intuitive argument that the subtrees of the root when first created should have an average size of $\sqrt{2\pi N}$, and that the average number of nodes in the backbone should be approximately $\sqrt{2N/\pi}$ for asymptotically large N. This argument will parallel very closely the above proof for the EFD model, but relies on the convergence of certain distributions which is not yet proven. The results of simulations presented in subsequent chapters tend to confirm these values.

We will be using the uniform(0,1) distribution throughout this discussion, and in particular we will frequently require the distribution of the kth gap $X_{(k)}-X_{(k-1)}$, where $X_{(k)}$ denotes the kth order statistic from N independent random variables with the uniform distribution. The gap distribution is the same for all k, including the values $X_{(1)}-0$ and $1-X_{(N)}$, and is given by

$$P\{\mathbf{U}_N \le u\} = 1-(1-u)^N, \quad 0 \le u \le 1 \tag{2.1}$$

See for example Feller [9] pages 22ff. The expected value is

$$E[\mathbf{U}_N] = \frac{1}{N+1} \tag{2.2}$$

and

$$\begin{aligned} E[\mathbf{U}_N{}^2] &= \int_0^1 u^2 N(1-u)^{N-1} du \\ &= \frac{2}{(N+1)(N+2)} \end{aligned} \tag{2.3}$$

from which we can derive the variance

$$\begin{aligned} \mathrm{Var}(\mathbf{U}_N) &= E[\mathbf{U}_N{}^2] - E[\mathbf{U}_N]^2 \\ &= \frac{N}{(N+1)^2(N+2)} \end{aligned} \tag{2.4}$$

2.4. The Model

Although analyzing the intervals between a pair of non-colliding tags works well for the EFD tree, when we allow the new keys that are to be inserted during an update to be drawn at random from the (0,1) domain, the problem becomes much more difficult. In the following, we study the behavior of a single, but typical tag from the time it is first created until it falls off the tree. The intuition behind this analysis is that if we can bound the expected deviation of a typical tag at each point in its lifespan, and we can say something about the average distance between the tags, then we can bound the expected size of the gaps that open up due to the deviations of the tags from their expected positions.

Also, since the distributions of the tag motions are different for the real tree (i.e. the tree wherein the keys are drawn at random) than for the EFD tree, we will try to build a model which is general enough to capture both the EFD and the real tree. That is, we specify a set of *conditions* which are sufficient to allow us to draw some conclusions about the behavior of any process which satisfies these conditions. It will then be shown that the tags on an EFD do satisfy these conditions while they remain on the tree, but that the those on the real tree do not. Thus, we develop an alternate proof, subject to some simplifying assumptions, for the fact that the EFD develops an IPL of $\Omega(N^{3/2})$.

Although the model is still not general enough to capture all the intricacies of the real tree, it is possible to carefully pinpoint the areas for which the model fails. In particular, it will be seen that condition 2.7, relating to the covariances between successive moves, has not been proven. However, it seems pretty clear from an intuitive viewpoint that this condition must hold. In addition, condition 2.4 is shown not to hold for the real tree, although in this case bounds can be placed on the distances moved which are sufficient to allow the proof to proceed subject to condition 2.7 and the assumptions made in this section. Here again, it seems intuitively obvious that for asymptotically large N the distribution should converge to that in condition 2.4.

We now build the model. Conditions 2.1 to 2.7 define a random walk which models the behavior of a single tag. We define random variables called *tag positions* where the tth tag position on the Kth update (time step) after its creation is designated by $\mathbf{P}_{t,K}$, and

Condition 2.1

$$\mathbf{P}_{t,K} = \sum_{i=0}^{K} \mathbf{M}_{t,i}$$

where $\mathbf{M}_{t,i}$ is the distance moved by the tag t on the ith update and will be further refined shortly. If two or more tags are being referred to, this definition admits of ambiguity. In this case, the time subscript will refer to the time since the creation of the most recently created tag.

The above definition produces an unbounded walk. On the EFD this would be equivalent to allowing the tags to continue walking after leaving the tree. Since we will require a bound on the expected deviation of a tag during its lifetime, we specify a bounded walk, $\mathbf{P}^{b}_{t,K}$, corresponding to the bounded walk of tags on the tree, with the following boundary condition.

Condition 2.2

$$\mathbf{P}^{b}_{t,K} = \min(\mathbf{P}_{t,K}, 1)$$

We note that the motion of a tag across the domain of a tree depends upon two random variables. First the tag must be moved by having the node to which it is attached selected for deletion, and then the distance it moves is the distance to the next closest key. We mimic this behavior with the following two conditions. First we define a selection variable $\mathbf{Z}_{t,i}$, which, for $i \geq 1$ has the distribution,

Condition 2.3

$$P\{\mathbf{Z}_{t,i}\} = \begin{cases} \frac{1}{N} & \mathbf{Z}_{t,i}=1 \\ \frac{N-1}{N} & \mathbf{Z}_{t,i}=0 \end{cases}$$

The second part of the random move is the distance moved. We now define a random variable $\mathbf{U}_{t,i}$ for the distance moved. For the EFD at least, these intervals are identical for each i, and have the distribution

Condition 2.4

$$P\{\mathbf{U}_{t,i}<u\} = P\{\mathbf{U}_N<u\} = 1-(1-u)^N \tag{2.4}$$

It is easy to see that as long as a tag remains on the tree, whether or not it is moved on update i is independent of whether or not it is moved on any update $j \neq i$. Also, whether or not a tag moves on update i is independent of the size of the interval following the tag. Thus,

Condition 2.5

We require that for a given tag t the $\mathbf{Z}_{t,i}$ are independent of each other and of the $\mathbf{U}_{t,i}$; that is, $\mathrm{Cov}(\mathbf{Z}_{t,i},\mathbf{U}_{t,i}) = 0$.

We call $\mathbf{M}_{t,0}$ the initial value, or the creation value (distance) of the tag t, and define $\mathbf{M}_{t,0} = \mathbf{U}_{t,0}$.

Condition 2.6

We define $\mathbf{M}_{t,i}$, for $i \geq 1$, by $\mathbf{M}_{t,i} = \mathbf{Z}_{t,i}\mathbf{U}_{t,i}$, and thus

$$\begin{aligned} P\{\mathbf{M}_{t,i} \leq u\} &= 1-P\{\mathbf{Z}_{t,i}=1\}P\{\mathbf{U}_N>u\} \\ &= 1-\frac{1}{N}(1-u)^N \end{aligned} \tag{2.6}$$

To obtain the desired bound on the expected deviation we also need the following bound, which will be proven for the EFD.

Condition 2.7

$$\mathrm{Cov}(\mathbf{U}_{t,i},\mathbf{U}_{t,j}) \leq 0, \quad i \neq j \tag{2.7}$$

The above are sufficient to obtain bounds upon the expected deviation of a tag during its lifetime on the bounded walk. We will develop that analysis in the next section, and then following that show some of the complexities that arise when we treat the tags on the real tree.

To show more precisely how this information can be used to attack the problem of the distances between backbone nodes, and thus bound the IPL, we need to delimit the interactions between tags. Since the interactions between tags on the real tree are apparently very complex, we attack this part of the problem in a high level manner. Namely, we try to specify a minimal amount of information which appears to be true on the two types of tree, but which should be sufficient to provide us with the results we desire. Note that we do not specify how the new tags are generated, nor how they might come together to form clumps. We will see that both the EFD and the real tree satisfy the upper bound implied by the following condition, and that the EFD satisfies the second half of the condition also. We say that two tags u,v are *adjacent* if they are created in sequence.

Condition 2.8

The expected distance between adjacent tags u and v, where u is created before v, is, for all updates i after the creation of v,

$$E[\mathbf{P}_{u,i}]-E[\mathbf{P}_{v,i}] = O\left(\frac{1}{N}\right)$$

and is identical for all i.

The following we call the sweeping rule, because it says that a tag sweeps all the tags to its right ahead of it. This clearly applies to both types of trees.

Condition 2.9

For all tags u,v such that u is created before v, $\mathbf{P}_{v,i} \le \mathbf{P}_{u,i}$ for all updates i after the creation of v.

The next condition we call the gluing rule, and it says that once two tags are clumped together they stick together. Note that we do specify how the tags come together.

Condition 2.10

If on any update i, $\mathbf{P}_{u,i}=\mathbf{P}_{v,i}$, then $\mathbf{P}_{u,j}=\mathbf{P}_{v,j}$ for all $j>i$.

Using our model, we can now prove the following lemmas.

Lemma 2.4

$$E[\mathbf{M}_{t,i}] = \frac{1}{N(N+1)}, \quad i \geq 1$$

Proof: Using conditions (2.4), (2.3) and (2.6) we have

$$\begin{aligned} E[\mathbf{M}_{t,i}] &= E[\mathbf{M}_{t,i} \text{ and } \mathbf{Z}_{t,i}=1]P\{\mathbf{Z}_{t,i}=1\} + E[\mathbf{M}_{t,i} \text{ and } \mathbf{Z}_{t,i}=0]P\{\mathbf{Z}_{t,i}=0\} \\ &= E[\mathbf{U}_N]\frac{1}{N} \\ &= \frac{1}{N(N+1)} \end{aligned}$$

□

Lemma 2.5

$$\mathrm{Var}(\mathbf{M}_{t,i}) = \frac{2N^2+N-2}{N^2(N+1)^2(N+2)}, \quad i \geq 1$$

Proof: Using the previous lemma, and conditions (2.4), (2.3) and (2.6) we have

$$\begin{aligned} \mathrm{Var}(\mathbf{M}_{t,i}) &= E[(\mathbf{M}_{t,i})^2] - E[\mathbf{M}_{t,i}]^2 \\ &= E[(\mathbf{M}_{t,i})^2 \mid \mathbf{Z}_{t,i}=1]P\{\mathbf{Z}_{t,i}=1\} + E[(\mathbf{M}_{t,i})^2 \mid \mathbf{Z}_{t,i}=0]P\{\mathbf{Z}_{t,i}=0\} - E[\mathbf{M}_{t,i}]^2 \\ &= E[\mathbf{U}_N{}^2]\frac{1}{N} - \left(\frac{1}{N(N+1)}\right)^2 \\ &= \frac{2N^2+N-2}{N^2(N+1)^2(N+2)} \end{aligned}$$

□

Applying conditions (2.5) and (2.7) leads to

Lemma 2.6

$\mathrm{Cov}(\mathbf{M}_{t,i},\mathbf{M}_{t,i+j}) \leq 0, \; i\geq 0, j>0.$

Proof:

We treat the case $i>0$. The case $i=0$ will follow. By definition we have

$$\mathrm{Cov}(\mathbf{M}_{t,i},\mathbf{M}_{t,i+j}) = E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j}] - E[\mathbf{M}_{t,i}]E[\mathbf{M}_{t,i+j}]$$

The distributions of $\mathbf{M}_{t,i}$ and $\mathbf{M}_{t,i+j}$ are given by condition (2.6), thus the product of expectations is

$$E[\mathbf{M}_{t,i}]E[\mathbf{M}_{t,i+j}] = \frac{1}{N^2(N+1)^2}$$

by lemma 2.4 and thus

$$= E[\mathbf{U}_{t,i}]E[\mathbf{U}_{t,i+j}]\frac{1}{N^2}$$

by applying equation (2.2) and condition (2.4).

Now consider the first term on the right hand side, that is, the expected product.

$$\begin{aligned} E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j}] &= E[E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j}|\mathbf{Z}_{t,i}]] \\ &= E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j} \mid \mathbf{Z}_{t,i}=1]P\{\mathbf{Z}_{t,i}=1\} \\ &\quad + E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j} \mid \mathbf{Z}_{t,i}=0]P\{\mathbf{Z}_{t,i}=0\} \\ &= E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j} \mid \mathbf{Z}_{t,i}=1]\frac{1}{N} \end{aligned}$$

where the last step follows since the probability of moving on any update is $1/N$, and the product is zero if the tag does not move. Similarly,

$$E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j}] = E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j} \mid \mathbf{Z}_{t,i}=1 \text{ and } \mathbf{Z}_{t,i+j}=1]\frac{1}{N^2}$$

This term be rewritten as

$$E[\mathbf{M}_{t,i}\mathbf{M}_{t,i+j}] = E[\mathbf{U}_{t,i}\mathbf{U}_{t,i+j}]\frac{1}{N^2}$$

Combining these results, we have

$$\begin{aligned} \mathrm{Cov}(\mathbf{M}_{t,i},\mathbf{M}_{t,i+j}) &= (E[\mathbf{U}_{t,i}\mathbf{U}_{t,i+j}]-E[\mathbf{U}_{t,i}]E[\mathbf{U}_{t,i+j}])\frac{1}{N^2} \\ &= \mathrm{Cov}(\mathbf{U}_{t,i},\mathbf{U}_{t,i+j})\frac{1}{N^2} \end{aligned}$$

and applying condition (2.7) leads to

$$\mathrm{Cov}(\mathbf{M}_{t,i},\mathbf{M}_{t,i+j}) \le 0$$

when $i \ge 1$. When $i=0$, the proof is similar, except that $1/N^2$ becomes $1/N$.

□

Lemma 2.7

$$\mathrm{Var}(\mathbf{P}_{t,K}) \le \frac{2N^2+N-2}{N^2(N+1)^2(N+2)}K + \frac{N}{(N+1)^2(N+2)}$$

Proof: From condition (2.1) we can obtain

$$\mathrm{Var}(\mathbf{P}_{t,K}) = \sum_{i=0}^{K}\mathrm{Var}(\mathbf{M}_{t,i}) + 2\sum_{i<j}\mathrm{Cov}(\mathbf{M}_{t,i},\mathbf{M}_{t,j})$$

$$\leq \sum_{i=0}^{K} \mathrm{Var}(\mathbf{M}_{t,i})$$

since the covariances are non-positive by lemma 2.6. Now using lemma 2.5 we continue

$$\leq \frac{2N^2+N-2}{N^2(N+1)^2(N+2)}K + \mathrm{Var}(\mathbf{M}_{t,0})$$

$$\leq \frac{2N^2+N-2}{N^2(N+1)^2(N+2)}K + \frac{N}{(N+1)^2(N+2)}$$

□

For ξ a constant greater than zero, we define

$$L = (1+\xi)N^2$$

Lemma 2.8

For $K \geq L$ updates

$$\mathrm{P}\{\mathbf{P}_{t,K} < 1\} \leq \frac{2(1+\xi)}{N\xi^2} + O\left(\frac{1}{N^2}\right)$$

Proof: Using condition (2.1) and lemma 2.4 we see that the expected value of the sum $\mathbf{P}_{t,L}$ is

$$\mathrm{E}[\mathbf{P}_{t,L}] = \sum_{i=1}^{L} \mathrm{E}[\mathbf{M}_{t,i}] + \mathrm{E}[\mathbf{M}_{t,0}] = \frac{(1+\xi)N^2}{N(N+1)} + \frac{1}{N+1}$$

$$= 1 + \xi\frac{N}{N+1}$$

Thus, for $\mathbf{P}_{t,L} < 1$ the tag must deviate by $\xi N/(N+1)$ from its expected position. The variance of $\mathbf{P}_{t,L}$ can be determined using the previous lemma to be

$$\mathrm{Var}(\mathbf{P}_{t,L}) \leq \frac{2N^2+N-2}{N^2(N+1)^2(N+2)}(1+\xi)N^2 + \frac{1}{(N+1)(N+2)}$$

$$= \frac{2(1+\xi)}{N} + O(\frac{1}{N^2})$$

Using Kolmogorov's inequality then,

$$\mathrm{P}\{\mathbf{P}_{t,L} < 1\} \leq \frac{\frac{2(1+\xi)}{N} + O\left(\frac{1}{N^2}\right)}{\left(\xi\frac{N}{N+1}\right)^2}$$

$$= \frac{2(1+\xi)}{N\xi^2} + O\left(\frac{1}{N^2}\right)$$

To complete the proof, we note that for any $\xi_0 \geq \xi$ the term on the right is reduced, and thus the bound holds for $K \geq L$.

□

Lemma 2.9

$$\mathrm{Var}(\mathbf{P}^b_{t,K}) \leq \mathrm{Var}(\mathbf{P}_{t,K})$$

Proof: We use throughout this proof the fact that

$$\mathrm{E}[(X-z)^2] \geq \mathrm{E}[(X-\mu_X)^2]$$

Case (1). $K \leq N^2$.
From the above fact we can conclude

$$\mathrm{E}[(\mathbf{P}^b_{t,K}-\mathrm{E}[\mathbf{P}^b_{t,K}])^2] \leq \mathrm{E}[(\mathbf{P}^b_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}])^2]$$

For $K \leq N^2$, $\mathrm{E}[\mathbf{P}_{t,K}] \leq 1$. For $\mathbf{P}_{t,K} \leq 1$, $\mathbf{P}^b_{t,K} = \mathbf{P}_{t,K}$ by condition (2.2) and for $\mathbf{P}_{t,K} > 1$, $\mathbf{P}^b_{t,K} < \mathbf{P}_{t,K}$ and thus

$$|\mathbf{P}^b_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}]| \leq |\mathbf{P}_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}]|$$

From this we can conclude

$$\mathrm{E}[(\mathbf{P}^b_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}])^2] \leq \mathrm{E}[(\mathbf{P}_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}])^2]$$
$$= \mathrm{Var}(\mathbf{P}_{t,K})$$

Case (2). $K > N^2$
Again using our fact

$$\mathrm{E}[(\mathbf{P}^b_{t,K}-\mathrm{E}[\mathbf{P}^b_{t,K}])^2] \leq \mathrm{E}[(\mathbf{P}^b_{t,K} - 1)^2]$$

Now for $\mathbf{P}_{t,K} \leq 1$, the relation $\mathbf{P}^b_{t,K} \leq 1 < \mathrm{E}[\mathbf{P}_{t,K}]$ follows by lemma (2.4). For $\mathbf{P}_{t,K} > 1$, $\mathbf{P}^b_{t,K} = 1$ and so

$$|\mathbf{P}^b_{t,K}-1| \leq |\mathbf{P}_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}]|$$

This leads to our conclusion that

$$\mathrm{E}[(\mathbf{P}^b_{t,K}-1)^2] \leq \mathrm{E}[(\mathbf{P}_{t,K}-\mathrm{E}[\mathbf{P}_{t,K}])^2]$$
$$= \mathrm{Var}(\mathbf{P}_{t,K})$$

Lemma 2.10

$$\mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]|] \leq \left(\frac{2(1+\xi)}{N}\right)^{1/2} + O\left(\frac{1}{N^{3/2}}\right), \quad K \leq L$$

Proof: Using the fact $\mathrm{E}[|X|^2]^{1/2} \geq \mathrm{E}[|X|]$ (Feller [9] Chapter V.8(c)) we see that the expected deviation is less than or equal to the standard deviation, which is the quantity in the right hand side of the lemma for L updates, which holds by the previous lemmas.

□

We now have an upper bound on the expected (absolute) deviation of a tag during $K \leq L$ updates. However, it may require more than L updates before the tag reaches the boundary.

Lemma 2.11

$$\mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]|] \leq \left(\frac{2(1+\xi)}{N}\right)^{1/2} + O\left(\frac{1}{N^{3/2}}\right), \quad K \leq \infty$$

Proof: We already have the proof for $K \leq L$. For $K > L$, we have

$$\begin{aligned}\mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]|] &= \mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]| \mid \mathbf{O}_{t,L}=0]\mathrm{P}\{\mathbf{O}_{t,L}=0\} \\ &+ \mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]| \mid \mathbf{O}_{t,L}=1]\mathrm{P}\{\mathbf{O}_{t,L}=1\}\end{aligned}$$

where

$$\mathbf{O}_{t,L} = \begin{cases} 1 & \text{if } \mathbf{P}_{t,L} < 1 \\ 0 & \text{if } \mathbf{P}_{t,L} \geq 1 \end{cases}$$

From lemma (2.8) we have that $\mathrm{P}\{\mathbf{O}_{t,L}=1\} = O\left(\frac{1}{N}\right)$. Thus,

$$\mathrm{E}[\mathbf{P}^{b}_{t,K}] = 1 - O\left(\frac{1}{N}\right), \quad K > L$$

since the smallest value it can take on is 0, and the probability of being less than 1 is $O(1/N)$. Thus,

$$\begin{aligned}\mathrm{E}[|\mathbf{P}^{b}_{t,K} - \mathrm{E}[\mathbf{P}^{b}_{t,K}]| \mid \mathbf{O}_{t,L}=0]\mathrm{P}\{\mathbf{O}_{t,L}=0] &= \left|1 - \left(1 - O\left(\frac{1}{N}\right)\right)\right|\left(1 - O\left(\frac{1}{N}\right)\right) \\ &= O\left(\frac{1}{N}\right)\end{aligned}$$

If $\mathbf{O}_{t,L}=1$, the tag may be no farther from its expected position than is zero, which is a distance of less than one. And so for $K>L$,

$$\mathrm{E}[|\mathbf{P}^b_{t,K} - \mathrm{E}[\mathbf{P}^b_{t,K}]| \mid \mathbf{O}_{t,L}=1]\mathrm{P}\{\mathbf{O}_{t,L}=1] = O\left(\frac{1}{N}\right)$$

Thus,

$$\mathrm{E}[|\mathbf{P}^b_{t,K} - \mathrm{E}[\mathbf{P}^b_{t,K}]|] = O\left(\frac{1}{N}\right)$$

Combining this result with the previous lemma completes our proof.

□

We have now achieved our first sub-goal of proving a bound upon the expected deviation of any tag from its expected position throughout its lifetime. To show how this piece of information can be useful, we now turn to the problem of the interaction between tags, and in particular the forming of intervals between clumps of tags. Because of the simplifications we have made, we will be forced to proceed somewhat informally, making some intuitive assumptions along the way.

We let I_j be the size of the jth interval from the right of the bounded walk, where the first interval is the interval between the boundary and the rightmost clump of tags to the left of the boundary. If j is greater than the number of intervals, then we define I_j to be zero.

Lemma 2.12

Subject to the assumption that the Law of Large Numbers applies, for all j, and any $\xi>0$, we have asymptotically that

$$\mathrm{E}[I_j] \leq C\left(\frac{2(1+\xi)}{N}\right)^{1/2} + O\left(\frac{1}{N^{3/2}}\right)$$

for some constant C.

Proof: We proceed informally. If at some time T, an interval is defined by two clumps, C_l and C_r, the left and right clumps respectively, then each tag whose expected position lies within that interval at that time must deviate at least as far as the nearer of the two clumps from its expected position. They may be even further away from their expected positions than this, since all the tags in the clumps may have expected positions outside of the interval. However, to prove our claim, we must maximize the expected size of an interval, and so we must minimize the total deviations of the tags. Thus, we may specify that the tags are no further from their expected positions than is the nearest clump. Thus, under this optimization scheme, all the tags which have expected positions between C_l and C_r are in one of those clumps.

We first analyze the rightmost interval, I_1. Assume at time T_1, $T_1 > N^2$, that this interval is defined by the clump C_1 and the boundary. Let S_1 be the set of tags whose expected positions (with respect to the unbounded walk) are in I_1 at T_1; that is, those tags t for which $E[\mathbf{P}_{t,i}]$ for $i = T_1$ is in the interval.

At some later time T_2, the clump C_1, along with any new tags which have joined it in the interim, will reach the boundary, and a new clump C_2 will define I_1. (We are implicitly using condition 2.10 here). We let S_2 be the set of tags with expected positions in I_1 at T_2. Note that under this scheme, S_1 and S_2 are not necessarily disjoint.

We continue this process indefinitely until M tags are involved, for some arbitrarily large M.

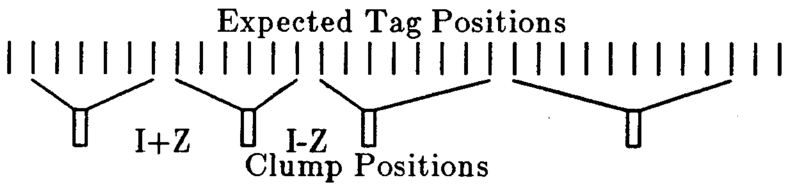


Figure 2.6 The Minimal Deviations Tags Must Make To Form Clumps.

In Figure 2.6, we represent this sequence of intervals laid out end to end, showing how the tags must move to reach the nearest clump. Note that this is not a snapshot taken at one particular time, but a sequence of shots, taken one per interval.

We now make the assumption that we can apply the law of Large Numbers to the absolute deviations. Note that the absolute deviations are not independent, since tags that are joined together in a clump are clearly correlated. Thus, the Law of Large Numbers can not be proven applicable unless we are willing to be more specific about the random distributions involved. It is difficult to see how there could be any covariance between the absolute deviation of tags which have reached the boundary and those not yet created. Assuming this is true, the Law of Large Numbers should apply. (See for example, Feller [8] Volume I, Chapter X, exercise 10).

Assuming this result, we have with probability one the result

$$\sum_{t=1}^{M} d_t \leq M\left(\left(\frac{2(1+\xi)}{N}\right)^{1/2} + O\left(\frac{1}{N^{3/2}}\right)\right) \quad \xi>0$$

where d_t is the deviation of the tag t from its expected position at the times when the measurements are taken. That is, different t may refer to the same tag at different times.

Condition 2.8 tells us that the expected distance between tags is less than the expected deviation of the tags, and thus the size of the intervals between clumps will be limited by the deviations. Let the expected interval size between adjacent tags be γ. We now show that having all of the distances between the expected positions equal to γ implies that the expected interval size can be maximized if all the intervals between clumps are made equal. Consider a pair of adjacent intervals such that one has $I+Z$ and the other has $I-Z$ expected tag positions between the clumps. We represent the total interval by I'. Then recalling that the sum is minimized if the tags go to the clump closest to their expected positions, the sum of the absolute deviations of the tags over these intervals is at least

$$\sum_{i\in I'} d_i \geq 2\sum_{i=1}^{\frac{I+Z}{2}} i\gamma + 2\sum_{i=1}^{\frac{I-Z}{2}} i\gamma$$

$$= \gamma\left(\frac{(I+Z)^2+2(I+Z)}{4} + \frac{(I-Z)^2+2(I-Z)}{4}\right)$$

$$= \gamma\left(\frac{I^2+2I+Z^2}{2}\right)$$

which is minimal for $Z=0$. If we treat all intervals as equal, then the average deviation per tag is equivalent to the average over any interval. Thus, dividing by the $2I$ tags in the two intervals we obtain

$$\mathrm{E}[d_t] \geq \gamma\left(\frac{I+2}{4}\right)$$

which implies

$$I\gamma \leq 4\mathrm{E}[d_t]$$

Recalling that $\mathrm{E}[d_t] = \left(\frac{2(1+\xi)}{N}\right)^{1/2} + O\left(\frac{1}{N^{3/2}}\right)$ for $\xi>0$, this gives us our bound for I_1 under the condition that we measure the interval just when a clump reaches the boundary. For any time between T_i and T_{i+1}, the times when clumps C_i and C_{i+1} reach the boundary respectively, the interval will be at most the size of the interval at time T_i. Since this is true for all i, it follows that the above is a bound on the expected size of I_1.

For I_j, $j>1$, we observe that the bound we obtained above was on the size of the interval to the left of a clump when it first arrived at the boundary, using the upper bound on the expected deviation of a tag. This bound clearly applies to the interval to the left of any clump on the bounded interval, and so applies for all j. In fact, for the intervals to the far left, the expected deviation of the tags should be far less, and so the intervals should be much smaller.

□

2.5. Fitting the Model

We now verify that the EFD tree satisfies all the conditions of our model. Condition (2.1) is trivially satisfied, since we are describing a random walk. The tag moves exactly N times, and the sum of all the intervals moved plus the initial interval is exactly one, so condition (2.2) is satisfied. We choose the keys for the EFD at random from the (0,1) domain, and the tags move over these fixed intervals, so condition (2.4) is satisfied. As long as a tag remains on the tree, it moves exactly when the key it is attached too is updated, which occurs with the probability given in condition (2.3). From these last two conditions, we see that condition (2.6) also holds.

Considering condition (2.5), it is easy to see that the intervals were chosen independently of any move sequence, and so the $\mathbf{Z}_{t,i}$ are independent of the $\mathbf{U}_{t,i}$. As to the mutual independence of the $\mathbf{Z}_{t,i}$, we have to be careful in that knowing a tag moved on an earlier update changes the probability that the tag is no longer on the tree, and thus conditions the probability that the tag moves on a subsequent update. However, removing this dependence was precisely the purpose of treating the walk as unbounded in the model. For the purpose of studying the behavior of a single tag, we can add an unlimited number of dummy keys to the right of the domain, and allow the tag to continue moving as long as we wish. This clearly does not have any effect on the key while it is on the domain. Thus, condition (2.5) applies to the domain of interest.

We now prove that the covariances are negative for the intervals over which the moves take place, and thus prove that condition (2.7) is satisfied. Since the intervals are fixed from the start, we drop the subscript t.

Lemma 2.13

$$\mathrm{Cov}(\mathbf{U}_i,\mathbf{U}_j) \le 0$$

Proof: A set of N keys divides the interval into $N+1$ subintervals, each with the distribution given by (2.1), each with variance given in (2.4). If we sum over all $N+1$ intervals, the sum is exactly 1, which means that the variance of the sum is 0. The variance of a sum of random variables is the sum of the variances plus twice the sum of covariances. (See for example, [18] Theorem 5.8) Thus,

$$0 = \sum_{i=1}^{N+1} \frac{N}{(N+1)^2(N+2)} + 2\sum_{i<j} \mathrm{Cov}(\mathbf{U}_i,\mathbf{U}_j)$$

Noting that the pairwise covariances are all equal, and that there are $\binom{N+1}{2}$ such pairs, we can solve for the general pairwise covariance to obtain

$$\mathrm{Cov}(\mathbf{U}_i,\mathbf{U}_j) = \frac{-1}{(N+1)^2(N+2)}$$

□

For condition (2.8), we notice that the expected number of moves of each tag after the creation of the second one on the unbounded version of our EFD is the same for all subsequent updates. Thus, the expected distance between the tags is the expected size of an interval, which is $1/(N+1)$, and this satisfies the condition. As was stated when they were defined, conditions (2.9) and (2.10) are trivially satisfied by the EFD.

This completes the list of conditions. We did not solve for the exact distribution of our model. However, the results we obtained depended only upon the conditions defining our model, and thus must also apply to the EFD since it meets each of these conditions.

Lemma 2.14

The expected number of nodes in any subtree of the backbone of the EFD is $O(N^{1/2})$ after sufficiently many updates.

Proof: Applying lemma 2.12 to the EFD, the expected size of any interval on the backbone is $O\left(\frac{1}{N^{1/2}}\right)$. We let N_j be the number of nodes in the jth subtree of the backbone, where the first subtree is the right subtree of the root. We let S_j be the size of the interval between the nodes defining the jth subtree. Since the keys are distributed uniformly over the interval (0,1), we have that the expected number of keys between backbone nodes is (keeping in mind that the backbone nodes themselves cannot be in the interval)

$$\begin{aligned} \mathrm{E}[N_j] &= \mathrm{E}[\mathrm{E}[N_j \mid I_j]] \\ &\le \mathrm{E}[NI_j] \\ &= NO\left(\frac{1}{N^{1/2}}\right) \\ &= O(N^{1/2}) \end{aligned}$$

□

This is enough to conclude Theorem 1 holds on the EFD, which we proved before.

2.6. What About the Real Tree?

We now consider how well the real tree satisfies the conditions of our model. Condition (2.1) is trivially satisfied, since we are describing a random walk. The sum of all the intervals moved plus the initial interval is exactly one, so condition (2.2) is satisfied. As long as a tag remains on the tree, it moves exactly when the key it is attached to is updated, which occurs with the probability given in condition (2.3).

Again as in the case of the EFD, it is easy to see that the interval following a tag is independent of whether or not the tag moves on any particular update, if the tag is on the tree. As to the mutual independence of the $\mathbf{Z}_{t,i}$, we again have to be careful in that knowing a tag moved on an earlier update changes the probability that the tag is no longer on the tree, and thus conditions the probability that the tag moves on a subsequent update. As in the case for the EFD, we again can add an unlimited number of dummy keys to the right of the domain, and allow the tag to continue moving as long as we wish. This clearly does not have any effect on the key while it is on the domain. Thus, condition (2.5) applies to the domain of interest.

Again conditions (2.9) and (2.10) are trivially satisfied by the real tree.

This leaves us with three problem conditions, namely (2.4), (2.7) and (2.8). This section concerns the problems associated with attempting to make the real tree fit these assumptions. We will find a bounding distribution to replace that in (2.4) which should be sufficient to implement the proof subject to the other two conditions. It will become obvious that the expected distance between adjacent tags is $O(1/N)$, but that the value is not identical for all subsequent updates. However, lemma (2.12) could be adjusted to accommodate this, provided that the expected positions of the tags do not somehow form clusters, which seems unlikely. Thus, it seems that the real tree is sufficiently close to satisfying condition (2.8) that the proof could be attained.

Proving that the covariances between the intervals over which a tag moves are non-positive has proven to be even more elusive. Attempts to analyze these covariances occupy several pages of this section. Although a full proof is not obtained, we present very convincing intuitive arguments that indeed the covariance between successive updates is sufficiently small that it does not change the $\Omega(N^{3/2})$ expected internal path length. We thus have

Theorem

Subject to the assumptions implicit in the preceding paragraphs, the expected Internal Path Length of a tree whose keys are drawn from the Uniform (0,1) domain and which is subjected to a sequence of random updates, will become $\Omega(N^{3/2})$.

Although we cannot prove this theorem without the conditions, we can turn to the evidence of very large scale simulations which we have performed. The results of these simulations, which are presented in the following chapter, strongly confirm this conjecture.

We turn now to the analysis of the behavior of a tag on the real tree. To get some feel for the complexity of the situation we begin by considering the backbones of the tree of three nodes.

A backbone *configuration* on N nodes consists of specifying whether the ith smallest node is a member of the backbone, for $1 \leq i \leq N$. The smallest node is always a member of the backbone, thus there are 2^{N-1} possible configurations. If we know the sizes of each of the subtrees, then we also know precisely the backbone configuration. It is well known that the number of binary trees on N nodes is $\frac{1}{N+1}\binom{2N}{N}$, the Nth Catalan number. Thus, reducing our problem to studying the backbone greatly reduces the complexity of the problem.

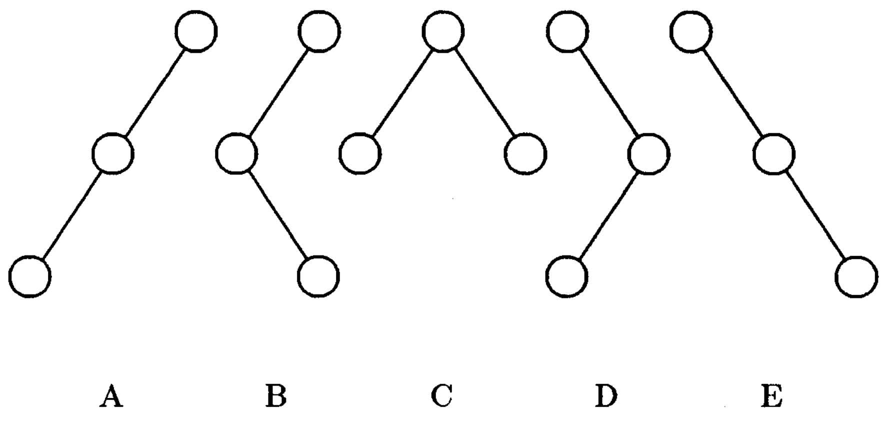


Figure 2.7 The Five Trees of Size Three

For example, if we consider the five trees of size three (see Figure 2.7) we see that there are four distinct backbone configurations, with only trees 'D' and 'E' not being uniquely identified by their backbone configuration. Since 'D' and 'E' have the same Internal Path Length, it follows that knowing the asymptotic probability distribution of the four backbone configurations would be sufficient to determine the asymptotic expected Internal Path Length.

Thus, we have reduced the number of variables we have to consider from five to four and can still determine the expected IPL exactly for three nodes. However, this is the bad news, not the good news. A closer inspection of the methods used by Jonassen and Knuth [12] shows that the inherent difficulties of their analysis must also apply to our simplification, since combining only the two rightmost trees still leaves us with essentially the same complexity. Furthermore, for larger trees, the number of backbone configurations is still exponential in the number of nodes in the tree. Thus we are unlikely to be able to obtain the exact asymptotic distribution of the backbone configurations as N becomes arbitrarily large.

In considering the application of our model to trees in which the insertions are made using new keys selected at random from the interval (0,1), there is a problem in that the distribution of the moves after the creation of the tag are not identical. To indicate the difficulties involved, we derive the distribution for the distance moved on the first move of a tag.

To simplify notation, we define $\mathbf{X}_0 = \mathbf{M}_{t,0}$, which is the distance of the tag from the origin at the time of its creation. We define $\mathbf{X}_1$ to be the distance moved on the tag's first move, which must occur on some update $i \geq 1$. The distribution of $\mathbf{X}_0$ is just the distribution $\mathbf{U}_N$ defined previously. To see this, note that the tag is attached to the smallest of N randomly distributed variables. At first glance, knowing that the smallest variable is the last one chosen may lead to the conclusion that the intervals surrounding it are smaller, because it divides one of the N random intervals defined by the $N-1$ previous variables. However, the fact that it fell in the first interval also conditions the size of the interval, and there is no paradox. See Feller [9] page 23 for a full discussion. Thus,

$$P\{\mathbf{X}_0 < \mathbf{x}_0\} = 1-(1-\mathbf{x}_0)^N$$

We call the set of $N-1$ keys, other than the smallest, which are in the tree at the time of the creation of the tag, the set of *old* keys. Some random number of updates will occur before the key supporting the tag is deleted, and thus some of the old keys will be replaced by *new* keys. Some of the new keys may also be deleted and replaced by other new keys before the tag moves, but we are only concerned with the set of remaining old keys and the final set of new keys. We let $\tilde{\mathbf{K}}_f = \langle \mathbf{K}_{f,0}, \mathbf{K}_{f,1} \rangle$ where $0 \leq \mathbf{K}_{f,0} \leq N-1$ is the number of old keys which existed at time of the creation of the tag and still remain when the tag makes its first move, and $\mathbf{K}_{f,1}$ is the number of new keys inserted after the creation of the tag but prior to its first move. Thus, $\mathbf{K}_{f,0} + \mathbf{K}_{f,1} = N-1$. In Figure 2.8, we have a representation of this situation. The old keys are distributed over the subinterval to the right of the tag, while the new keys are distributed over the entire interval. Thus, assuming $\mathbf{X}_0 = \mathbf{x}_0$, $P\{\mathbf{X}_1 > \mathbf{x}_1 \mid \mathbf{X}_0\}$, is the probability that all $\mathbf{K}_{f,0}$ old keys fall in the interval $(\mathbf{x}_0+\mathbf{x}_1, 1)$, which has size $1-\mathbf{x}_0-\mathbf{x}_1$, given that they fall in the interval $(\mathbf{x}_0, 1)$, which has size $1-\mathbf{x}_0$, and that all the new keys fall in the interval $(0,\mathbf{x}_0) \cup (\mathbf{x}_0+\mathbf{x}_1, 1)$, which has size $1-\mathbf{x}_1$. Thus, for $\tilde{\mathbf{K}}_f = \langle k, N-k-1 \rangle$ we get the probability conditional on k

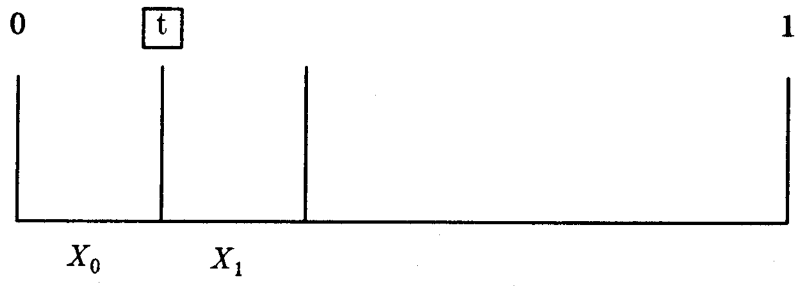


Figure 2.8 Situation for the First Two Moves of a Tag

$$P_k\{X_1>x_1 \mid X_0=x_0\} = \left(1-\frac{x_1}{1-x_0}\right)^k \left(1-x_1\right)^{N-k-1}$$

Since X_0 is independent of $\hat{K}_1^f$ and has the probability density function $N(1-x_0)^{N-1}$, it follows that

$$P_k\{X_1>x_1\} = \int_0^{1-x_1} P_k\{X_1>x_1\}P'\{X_0\}dx$$

$$= \int_0^{1-x_1} \left(1-\frac{x_1}{1-x_0}\right)^k \left(1-x_1\right)^{N-1-k} N(1-x_0)^{N-1}dx_0$$

$$= \int_0^{1-x_1} N(1-x_0-x_1)^k((1-x_1)(1-x_0))^{N-1-k}dx_0$$

The old keys including the tagged key are equally likely to be deleted in any order, and so

$$P\{K_{f,0}=k\} = \frac{1}{N}, \quad 0\le k\le N-1$$

Thus,

$$P\{X_1>x_1\} = \frac{1}{N}\sum_{k=0}^{N-1}\int_0^{1-x_1} N(1-x_0-x_1)^k((1-x_1)(1-x_0))^{N-1-k}dx_0$$

This can be simplified to

$$P\{X_1>x_1\} = (1-x_1)^N\sum_{j=0}^{N-1}\frac{x_1{}^j}{j+1}$$

but we will study it in the preceding form. For $K_{f,0} = N-1$, we have

$$\mathbf{P}_{N-1}\{\mathbf{X}_1 > \mathbf{x}_1\} = \int_{\mathbf{x}_0=0}^{1-\mathbf{x}_1} \left(1 - \frac{\mathbf{x}_1}{1-\mathbf{x}_0}\right)^{N-1} N\left(1-\mathbf{x}_0\right)^{N-1} dx_0$$

$$= \int_{\mathbf{x}_0=0}^{1-\mathbf{x}_1} N(1-\mathbf{x}_0-\mathbf{x}_1)^{N-1} dx_0$$

$$= (1-\mathbf{x}_1)^N$$

which is of course the probability for a random interval as we would expect, since in this case no key is deleted before the one which moves the tag. For $\mathbf{K}f_{,0}=0$, we get

$$P_0\{\mathbf{X}_1 > \mathbf{x}_1\} = \int_{\mathbf{x}_0=0}^{1-\mathbf{x}_1} (1-\mathbf{x}_1)^{N-1} N(1-\mathbf{x}_0)^{N-1} dx_0$$

$$= (1-\mathbf{x}_1)^{N-1}(1-\mathbf{x}_1{}^N)$$

Alternatively, we can compute this in the following way.

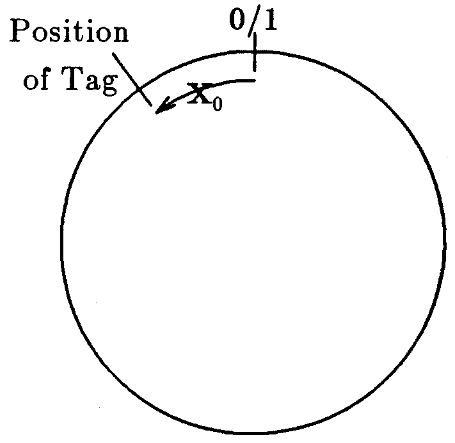


Figure 2.9 Interval as a Closed Loop

Assume the endpoints of the (0,1) interval are joined into a circle as in Figure 2.9. The new keys are distributed over the entire interval. However, the tagged key has the distribution $1-(1-\mathbf{x}_0)^N$ with respect to the endpoint, and thus $P\{\mathbf{X}_1 > \mathbf{x}_1\}$ is the probability that the nearest of the other $N-1$ keys is further than $\mathbf{x}_1$ and that $1-\mathbf{x}_0$ is greater than $\mathbf{x}_1$. Since the new keys are independently chosen, we have

$$\begin{aligned} P\{\mathbf{X}_1 > \mathbf{x}_1\} &= (1-\mathbf{x}_1)^{N-1} P\{1-\mathbf{X}_0 > \mathbf{x}_1\} \\ &= (1-\mathbf{x}_1)^{N-1} P\{\mathbf{X}_0 < 1-\mathbf{x}_1\} \\ &= (1-\mathbf{x}_1)^{N-1}(1-\mathbf{x}_1{}^N) \end{aligned}$$

We note that, for $0<\mathbf{x}_1<1$, $P_k\{\mathbf{X}_1>\mathbf{x}_1\}$ is monotonic in k, and $(1-\mathbf{x}_1)^{N-1} > (1-\mathbf{x}_1)^{N-1}(1-\mathbf{x}_1{}^N) \geq P_k\{\mathbf{X}_1>\mathbf{x}_1\} \geq (1-\mathbf{x}_1)^N$. Noting that the first term is the probability distribution for intervals between $N-1$ keys, we can thus prove

Lemma 2.15

$$\frac{1}{N+1} \leq E[\mathbf{X}_1] \leq \frac{1}{N}$$

Proof: The distribution function for $\mathbf{X}_1$ is $F(\mathbf{x}_1) = 1-P\{\mathbf{X}_1>\mathbf{x}_1\}$. The expected value is

$$\begin{aligned} E[\mathbf{X}_1] &= \int_0^1 P\{\mathbf{X}_1>\mathbf{x}_1\}\, dx_1 \\ &\leq \int_0^1 (1-\mathbf{x}_1)^{N-1} dx_1 \\ &= \frac{1}{N} \end{aligned}$$

Alternatively,

$$\begin{aligned} E[\mathbf{X}_1] &\geq \int_0^1 (1-\mathbf{x}_1)^N dx_1 \\ &= \frac{1}{N+1} \end{aligned}$$

□

Lemma 2.16

$$\mathrm{Var}(\mathbf{X}_1) \leq \frac{2}{N(N+1)} = O\left(\frac{1}{N^2}\right)$$

Proof: Similar to the previous lemma, we have

$$\mathrm{Var}(\mathbf{X}_1) \leq E[\mathbf{X}_1{}^2]$$

$$= \int_0^1 x_1^2 F'(x_1)\,dx_1$$

$$= 1 - \int_0^1 2x_1 F(x_1)\,dx_1$$

$$= \int_0^1 2x_1 P\{X_1 > x_1\}\,dx_1$$

$$\le \int_0^1 2x_1 (1-x_1)^{N-1}\,dx_1$$

$$= \frac{2}{N}\int_0^1 x_1 N(1-x_1)^{N-1}\,dx_1$$

$$= \frac{2}{N(N+1)} = O\left(\frac{1}{N^2}\right)$$

□

We thus have the expected step size and the variance bounded for the distance of the creation and the first move of a tag. We now proceed to show that the upper bounds also hold for the distance moved on the second and subsequent moves of the tag.

First, we extend our notation. Let $\vec{X}_m = <X_0, X_1, X_2, \cdots, X_m>$ be the vector of values on the zeroth, first, second and up to the mth move. We call $\vec{K}^f_m$ the *final partition* of the keys prior to the mth move, where $\vec{K}^f_m = <K^f_{m,0}, K^f_{m,1}, \cdots, K^f_{m,m}>$. This vector represents the number of keys in common with those present at the zeroth, first and up to the $m-1$th move, with $K^f_{m,m}$ the number of keys which are new, at the time the tag makes its mth move. We will say a key is in common with the ith move on the mth move, if it was inserted after the $i-1$th move but prior to the ith move, and was not deleted prior to the mth move. We exclude the tagged key from our count, and thus, $\sum_{0 \le j \le m} K^f_{m,j} = N-1$.

Let $\vec{K}^i_m = <K^i_{m,0}, K^i_{m,1}, \cdots, K^i_{m,m}>$ be the number of keys in common with the zeroth, first and up to the $m-1$th move, immediately after the $m-1$th move. This is called the *initial partition* prior to the mth move. It is easily seen that $K^i_{m,m} = 1$ for $2 \le m$, since one key is inserted as the final part of the update that moves the tag. Also, there is one value of $j, 0 \le j \le m-1$, such that $K^i_{m,j} = K^f_{m-1,j} - 1$, since the tag moves onto one of the keys and it is no longer counted.

For $m \geq 2$, we obtain conditional probabilities only, and obtain our bounds from these. First, we restate what we have so far.

$$P\{\mathbf{X}_0 > \mathbf{x}_0\} = (1-\mathbf{x}_0)^N$$

$$P\{\mathbf{X}_1 > \mathbf{x}_1 \mid \mathfrak{X}_0, \mathfrak{K}\ell_1\} = \left(1 - \frac{\mathbf{x}_1}{1-\mathbf{x}_0}\right)^{k\ell_{1,0}} \left(1-\mathbf{x}_0\right)^{k\ell_{1,1}}$$

which we write in the form

$$= \prod_{j=0}^{1} \left(1 - \frac{\mathbf{x}_1}{1 - \sum_{i=j}^{0} \mathbf{x}_i}\right)^{k\ell_{1,j}}$$

For $m=2$, we see that the full probability distribution is complex, since the distributions of $\mathbf{K}\ell_{2,0}$ and $\mathbf{K}\ell_{2,1}$ will depend not only on the number of keys deleted and the order of deletion between the first and second moves, but also on the values of $\mathbf{K}\ell_{1,0}$ and $\mathbf{K}\ell_{1,1}$. Furthermore, the size of the first move depends upon whether the tag moves onto an old or new key, and $\mathbf{K}\ell_{2,0}$ and $\mathbf{K}\ell_{2,1}$ also depend on this.

However, by using the following conditional probability, we circumvent this complexity.

$$P\{\mathbf{X}_2 > \mathbf{x}_2 \mid \mathfrak{X}_1, \mathfrak{K}\ell_2\} = \left(1 - \frac{\mathbf{x}_2}{1-\mathbf{x}_0-\mathbf{x}_1}\right)^{k\ell_{2,0}} \left(1 - \frac{\mathbf{x}_2}{1-\mathbf{x}_1}\right)^{k\ell_{2,1}} \left(1-\mathbf{x}_2\right)^{k\ell_{2,2}}$$

$$= \prod_{j=0}^{2} \left(1 - \frac{\mathbf{x}_2}{1 - \sum_{i=j}^{1} \mathbf{x}_i}\right)^{k\ell_{2,j}}$$

One interpretation of this is as follows. For each combination of values $\mathbf{x}_0$ and $\mathbf{x}_1$, and for each partition of the keys at the time the tag is moved, the probability that the distance moved is greater than $\mathbf{x}_2$ is the probability that

1 Each of the keys remaining from the initial set fall outside the interval of size $\mathbf{x}_0+\mathbf{x}_1+\mathbf{x}_2$ given that they must fall in the interval of size $1-\mathbf{x}_0-\mathbf{x}_1$.

2 Each of the keys remaining from those inserted subsequent to the creation of the tag but prior to the first move fall outside the interval of size $\mathbf{x}_1+\mathbf{x}_2$ given that they must fall in the interval of size $1-\mathbf{x}_1$.

3 Each of the keys inserted after the first move fall outside the interval of size $\mathbf{x}_2$.

The above distribution is the consequence of the facts that all insertions are independent, and that the conditional distribution of a uniform random variable given that it falls in a given subinterval is uniform over the subinterval.

Noting that each of the terms in the above product is less than or equal $1-\mathbf{x}_2$, and that the sum of the $k^f_{2,j}$ is $N-1$, we obtain

$$P\{\mathbf{X}_2>\mathbf{x}_2 \mid \tilde{\mathbf{X}}_1, \tilde{\mathbf{K}}^f_2\} \leq (1-\mathbf{x}_2)^{N-1}$$

which immediately leads to

$$P\{\mathbf{X}_2>\mathbf{x}_2\} \leq (1-\mathbf{x}_2)^{N-1}$$

Generalizing from this discussion we obtain

Lemma 2.17

$$P\{\mathbf{X}_m>\mathbf{x}_m\} \leq (1-\mathbf{x}_m)^{N-1}$$

Proof: Analogous to the above discussion we obtain

$$P\{\mathbf{X}_m>\mathbf{x}_m \mid \tilde{\mathbf{X}}_m, \tilde{\mathbf{K}}^f_m\} = \prod_{j=0}^{m}\left(1-\frac{\mathbf{x}_m}{1-\sum_{i=j}^{m-1}\mathbf{x}_i}\right)^{k^f_{m,j}}$$

$$\leq (1-\mathbf{x}_m)^{N-1}$$

since again each of the terms in the product is less than or equal to $1-\mathbf{x}_m$, and the number of keys other than the tagged key is always $N-1$.

□

Lemma 2.18

$$E[\mathbf{X}_m] \leq \frac{1}{N}$$

Proof: As in Lemma 2.15.

□

Lemma 2.19

$$\mathrm{Var}(\mathbf{X}_m) \le \frac{2}{N(N+1)} = O\left(\frac{1}{N^2}\right)$$

Proof: As in Lemma 2.16.

□

Recalling that the probability of a move, given that a tag is on the tree, is $1/N$ on any update, we easily find that the expected distance moved per update is $O(1/N^2)$ and that the variance is $O(1/N^3)$.

Lemma 2.20

$$\mathrm{E}[\mathbf{M}_{t,i}] = O\left(\frac{1}{N^2}\right)$$

Proof: From Lemma 2.18 we have an upper bound on the conditional expectation given that the tag moves. If the tag is not on the tree, then it cannot move. If the tag is on the tree, and it does not move then the distance moved is also zero, and so

$$\begin{aligned}\mathrm{E}[\mathbf{M}_{t,i}] &= \mathrm{E}[\mathrm{E}[\mathbf{M}_{t,i} \mid \mathbf{Z}_{t,i}]] \\ &\le \mathrm{E}[\mathbf{M}_{t,i} \mid \mathbf{Z}_{t,i}=1]\mathrm{P}\{\mathbf{Z}_{t,i}=1\} \\ &= O\left(\frac{1}{N^2}\right)\end{aligned}$$

□

Lemma 2.21

$$\mathrm{Var}(\mathbf{M}_{t,i}) = O\left(\frac{1}{N^3}\right)$$

Proof: We have

$$\mathrm{Var}(\mathbf{M}_{t,i}) \le \mathrm{E}[\mathbf{M}_{t,i}^2]$$

and as in the previous lemma this

$$\begin{aligned}&= \mathrm{E}[\mathrm{E}[\mathbf{M}_{t,i}^2 \mid \mathbf{Z}_{t,i}]] \\ &= O\left(\frac{1}{N^2}\right)\frac{1}{N} \\ &= O\left(\frac{1}{N^3}\right)\end{aligned}$$

We turn now to proving a lower bound on the expected distance moved on the *m*th move, provided that the tag moves at least *m* times. We have already shown in Lemma 2.15 that $E[\mathbf{X}_1] \geq 1/(N+1)$, and of course $E[\mathbf{X}_0] = 1/(N+1)$. Recall that

$$P\{\mathbf{X}_2 > \mathbf{x}_2 \mid \vec{\mathbf{X}}_1, \vec{\mathbf{K}}_1'\} = \left(1 - \frac{\mathbf{x}_2}{1-\mathbf{x}_0-\mathbf{x}_1}\right)^{k'_{2,0}} \left(1 - \frac{\mathbf{x}_2}{1-\mathbf{x}_1}\right)^{k'_{2,1}} \left(1-\mathbf{x}_2\right)^{k'_{2,2}}$$

$$\geq \left(1 - \frac{\mathbf{x}_2}{1-\mathbf{x}_0-\mathbf{x}_1}\right)^{N-2} (1-\mathbf{x}_2)$$

where the second value is the probability distribution only if the first and second moves occur without any intervening updates (with the exception that the key inserted after the first update may be replaced by another random key). In this case, the first two moves were just over the first two intervals out of $N+1$, and noting that the joint density of two intervals specified by N uniform random variables is $N(N-1)(1-\mathbf{x}_0-\mathbf{x}_1)^{N-2}$, we have

$$P\{\mathbf{X}_2 > \mathbf{x}_2\} \geq$$

$$\int_{\mathbf{x}_1=0}^{1-\mathbf{x}_2} \int_{\mathbf{x}_0=0}^{1-\mathbf{x}_1-\mathbf{x}_2} \left(\frac{1-\mathbf{x}_0-\mathbf{x}_1-\mathbf{x}_2}{1-\mathbf{x}_0-\mathbf{x}_1}\right)^{N-2} \left(1-\mathbf{x}_2\right) N(N-1)(1-\mathbf{x}_0-\mathbf{x}_1)^{N-2} dx_0\, dx_1$$

$$= \int_{\mathbf{x}_1=0}^{1-\mathbf{x}_2} \int_{\mathbf{x}_0=0}^{1-\mathbf{x}_1-\mathbf{x}_2} \left(1-\mathbf{x}_2\right) N(N-1)(1-\mathbf{x}_0-\mathbf{x}_1-\mathbf{x}_2)^{N-2} dx_0\, dx_1$$

$$= \int_{\mathbf{x}_1=0}^{1-\mathbf{x}_2} \left(1-\mathbf{x}_2\right) N(1-\mathbf{x}_1-\mathbf{x}_2)^{N-1} dx_1$$

$$= (1-\mathbf{x}_2)^{N+1}$$

This is the density of an interval specified by $N+1$ keys.

There is an easy and intuitive way to obtain this result. If for the first move, assuming that no deletions have yet occurred, we move the tag to the next largest key without deleting the key under the tag and then insert a new key at random, we see that we have $N+1$ random keys. The presence of the undeleted key (let us call it a ***shadow key)*** will in no way affect the next move of the tag, since it is to the left of the tags current position. If we now immediately move the tag again, then the distribution of the distance moved is just that of the distance between $N+1$ keys, since the new key is equally likely to fall in any interval. The fact that the tag is on the second largest key before the new key is inserted does not change this. In fact, we could place the tag on any key before inserting the final key, and the distribution would obviously remain the same.

We now extend this observation as follows. We define a *shadow tag* which moves whenever the tag moves, but the keys over which the shadow move are never deleted. The shadow and its tag begin at the same time and position. New keys are inserted into the shadows domain only when the shadow moves, that is only when the tag moves. We then consider the race between the tag and its shadow to reach the end point of the domain, the point 1. We designate the outcomes of this race by W_t meaning the tag wins, and W_s meaning the shadow wins.

Lemma 2.22

$$P\{W_t\} \geq P\{W_s\}$$

Proof: Any intermediate deletion of a key which occurs in the tag process removes the key from a class $k^i_{m,j}$ and inserts a key into the class $k^f_{m,m}$, where $j \leq m$. This means that the replacement key is distributed uniformly over a possibly larger interval, and thus the probability that the new key lies between the tag and the end point is reduced, increasing the probability that the end point is reached in any subsequent move. Note that the same argument applies to any point z, $x_0 < z \leq 1$, not just the end point.

□

We now attempt to justify the claim that the covariance between any pair of moves is non-positive. Consider two moves $\mathbf{X}_k$ and $\mathbf{X}_m$, where without loss of generality we assume $m > k$. Then

$$\begin{aligned}
\mathrm{Cov}(\mathbf{X}_k,\mathbf{X}_m) &= E[(\mathbf{X}_k - \mu_{x_k})(\mathbf{X}_m - \mu_{x_m})] \\
&= E[E[(\mathbf{X}_k - \mu_{x_k})(\mathbf{X}_m - \mu_{x_m}) \mid \mathbf{X}_k]] \\
&= \int_0^1 E[(\mathbf{x}_k - \mu_{x_k})(\mathbf{X}_m - \mu_{x_m}) \mid \mathbf{X}_k = \mathbf{x}_k] f'(\mathbf{x}_k)\,dx_k \\
&= \int_0^{\mu_{x_k}^-} (\mathbf{x}_k - \mu_{x_k}) E[(\mathbf{X}_m - \mu_{x_m}) \mid \mathbf{X}_k = \mathbf{x}_k] f'(\mathbf{x}_k)\,dx_k
\end{aligned}$$

$$+ \int_{\mu_{x_k}^+}^{1} (\mathbf{x}_k - \mu_{x_k}) \mathrm{E}[(\mathbf{X}_m - \mu_{x_m}) \mid \mathbf{X}_k = \mathbf{x}_k] f'(\mathbf{x}_k) dx_k$$

Now to show that the covariances are non-positive, we need only show that $\mathrm{E}[(\mathbf{X}_m - \mu_{x_m}) \mid \mathbf{X}_k]$ and $(\mathbf{x}_k - \mu_{x_k})$ are of opposite sign in each term of the above sum. Unfortunately, we currently do not have a formal proof of this, but the following intuitive argument seems convincing.

Consider the case $\mathbf{X}_k = \mathbf{x}_k > \mu_{x_k}$. In this case, after the *k*th move the remaining $N-2$ keys not affected by that move must be uniformly distributed over the interval $1-\mathbf{x}_k$, which is less than the expected size. In addition, as the move is big, it seems reasonable to expect that the tag is closer to the end point than in the unconditional case. Thus the expected distance between the tag and the next smaller key or the end point whichever is closer is reduced under this supposition. Subsequent updates delete some of these abnormally dense keys and replace them with new keys independently distributed over the entire domain. After some number of updates, the tag moves for the *m*th time. If the tag moves from one of the keys in common with the *k*th move to another, or to the end point, then the expected distance moved would be less when $\mathbf{x}_k > \mu_{x_k}$ than otherwise. Thus, the expected distance would be reduced and the sign inverted.

The problem with this argument is that the probability distribution for the various $\mathbf{K}^f_{m,j}$ are not known to be independent of $\mathbf{X}_k$. Changing these probability distributions could negate the above argument, for example by increasing the probability of $k^f_{m,j}$ for the larger j at the expense of the smaller j.

If we change our process model slightly to allow deletions and insertions to be performed at random instead of being coupled, then it is easier to argue for positive correlations between distances moved. In this case, $\mathbf{x}_k > \mu_{x_k}$ could mean that there is a high probability that the number of keys in the tree has been reduced because the number of deletions has exceeded the number of insertions. But assuming that the number of keys is reduced would imply that the expected distance moved on the next move is increased. This shows that it is possible that one move being larger than expected may indicate an increase in the expected size of the next move, even though the total remaining distance is reduced. For our model, however, this argument does not apply, but we cannot prove that a similar argument does not apply. For example, knowing that $\mathbf{x}_k > \mu_{x_k}$ might lead us to believe that $\mathbf{K}^{\prime}_{i,k}$ would be larger than expected since that condition produces larger than expected distances. But if this is the case, then the next interval at the time of the move might also be larger than expected.

But any such effect would appear to be insignificant. The distribution of the distance between keys is always $(1-x)^N$. The fact that the distance moved by the tag differs slightly from this distribution occurs only because the tag has a tendency to be attached to a key preceding a slightly non-random interval. Whatever the distribution of this interval may be (and it is probably different for different updates), knowing that the interval is larger than expected means that the rest of the keys are more densely distributed than expected. This is because the remaining keys must be distributed over a smaller interval than expected. This would seem to indicate the claimed negative correlation. Successive updates would delete some of these negatively correlated keys and replace them with independent keys, but this could only increase the covariance to at most zero for totally independent keys. Since the walk is bounded by the end point of the domain there must always be some slight negative correlation, even when all keys are replaced between moves.

In what follows, we make the assumption that the covariances are non-positive between pairs of moves for a given tag. We also assume for similar reasons that the correlation between moves of tags on distinct nodes is negative.

For the shadow walk, we can do better. We indicate the distance moved by the shadow on the ith move by $\mathbf{Y}_i$.

Lemma 2.23

$$\mathrm{Cov}(\mathbf{Y}_i, \mathbf{Y}_j) \leq 0$$

Proof: On the ith move of a shadow the density of the interval is given by $(1-\mathbf{y}_i)^{N+i-1}$, and this is the density for every interval, since it is derived by the rule that there are $N+i-1$ keys. For this process, knowing that a key is tagged does not change the distribution of the following interval. If we are given $\mathbf{y}_i$, then the remaining keys have the uniform distribution over the remaining interval. Thus, if $\mathbf{y}_i > \mu_{y_i}$ then the expected distance between other keys is reduced. Inserting new independent keys does not change that result for these keys are never deleted. Thus, the covariance is negative.

□

We now show that the shadow tag leaves the tree within $O(N)$ moves with high probability.

Lemma 2.24

The probability that a shadow tag remains on the tree for more than $(1+\xi)(e-1)N$ moves is $O(1/N)$.

Proof: We compute a lower bound for the expected value of the sum of $\mathbf{Y}_i$ ignoring the creation or zeroth term, assuming an unbounded walk. A simple model for this unbounded walk is to consider the keys to be on a circle. Then in the unbounded walk the shadow simply continues around the circle indefinitely. Our bound on the probability of not exceeding one after m moves is then a bound on the probability of not traversing the circle at least once. Noting that each move increases the number of keys by one, and that no key is deleted we see that

$$
\begin{aligned}
\mathrm{E}[\sum_{i=1}^{m}\mathbf{Y}_i] &= \sum_{i=1}^{m}\frac{1}{N+i} \\
&= \sum_{i=1}^{m+N}\frac{1}{i} - \sum_{i=1}^{N}\frac{1}{i} \\
&\approx \log(m+N) - \log(N) = \log\left(\frac{m+N}{N}\right)
\end{aligned}
$$

and for $m=(1+\xi)(e-1)N$

$$
\begin{aligned}
&= \log\left(\frac{(1+\xi)(e-1)N+N}{N}\right) \\
&= \log(e+\xi(e-1)) \\
&= 1+\xi'
\end{aligned}
$$

for some $\xi'>0$ where $\xi>0$. Thus, for the sum to be less than one would require a deviation of at least ξ'. By Chebyshev's inequality we have

$$
\mathrm{P}\{|\sum_{i=1}^{m}\mathbf{Y}_i - \mathrm{E}[\sum_{i=1}^{m}\mathbf{Y}_i]|>\xi'\} \le \frac{\mathrm{Var}(\sum_{i=1}^{m}\mathbf{Y}_i)}{\xi'^2}
$$

Clearly Lemma 2.19 applies to the shadow tags as well as tags and thus $\mathrm{Var}(\mathbf{Y}_i) = O(1/N^2)$. Since by Lemma 2.23 the covariances are less than zero, then

$$\begin{aligned}\mathrm{Var}(\sum_{i=1}^{m}\mathbf{Y}_i) &\le \sum_{i=1}^{m}\mathrm{Var}(\mathbf{Y}_i)\\ &= O\left(\frac{m}{N^2}\right)\end{aligned}$$

and for $m=(1+\xi)(e-1)N$

$$= O\left(\frac{1}{N}\right)$$

Making the substitution into the probability inequality above completes the proof.

□

Lemma 2.25

Assuming an unbounded walk,

$$P\{\sum_{i=1}^{(1+\xi)mN}\mathbf{Z}_{t,i}<m\} = O\left(\frac{1}{m}\right)$$

Proof: The number of moves is given by the binomial distribution, with a probability of each success of $1/N$. The expected value is $(1+\xi)m$ and variance is $(1+\xi)m\frac{N-1}{N}$. (See for example Feller [8] chapter IX) Using Chebyshev's inequality leads to the result

$$\begin{aligned}P\{|\sum_{i=1}^{(1+\xi)mN}\mathbf{Z}_{t,i} - (1+\xi)m| \ge \xi m\} &\le \frac{(1+\xi)m}{(\xi m)^2}\\ &= O\left(\frac{1}{m}\right)\end{aligned}$$

□

We now conclude that $O(N^2)$ updates are sufficient to move the tag off the tree with high probability.

Lemma 2.26

$$P\{\sum_{i=1}^{(1+\xi)^2(e-1)N^2}\mathbf{M}_{t,i}<1\} = O\left(\frac{1}{N}\right)$$

Proof: With $m=(1+\xi)(e-1)N$, we have from Lemma 2.25 that the probability that more than m moves are made is $1-O(1/N)$. Given m or more moves, the probability that the tag moved off the tree is $1-O(1/N)$, by Lemmas 2.24 and 2.22. Thus, the probability that the tag moved off the tree is at least $(1-O(1/N))^2$, which means the probability that it remains on the tree is $O(1/N)$.

□

We have now completed proofs for the tree on the real domain showing, similar to lemma 2.8, that the probability of remaining on the tree for more than $O(N^2)$ updates is $O(1/N)$. To compute bounds on the variance, and expected deviation as in lemmas 2.9 through 2.11 requires us to assume that the covariances between the moves of the tag are non-positive. Having the covariances negative for the shadow walk is not sufficient for this purpose, since the shadow walk only gives us a bound in one direction.

Lemma 2.27

Assuming the covariances are non-positive, the expected deviation of a tag from its expected position is $O(1/\sqrt{N})$.

Proof: We have from lemma 2.21 that the variance per update is $O(1/N^3)$, and assuming the covariances are non-positive, this leads to the variance on $O(N^2)$ updates of $O(1/N)$. Using this and the result of the previous lemma, we derive the result using the analogous proofs of lemmas 2.9 through 2.11.

□

The next problem we encounter is that the expected distance between tags can no longer be considered to remain constant throughout their life times. The distribution of the moves is not the same for every update, and so the expected distance between tags may vary. Arguing informally, we see that bounding the expected deviation should bound the expected interval size as in lemma 2.12. Consider those tags which are $\Omega(1/\sqrt{N})$ apart. Then if more than a constant number of these merge, the average deviation of these (and the intervening tags as well) will be $\omega(1/\sqrt{N})$, violating the expected deviation computed above. This argument assumes that the expected positions of the tags are not somehow clustered around the expected position of the clumps.

We now have an incomplete argument that the intervals between backbone nodes of the tree on the real domain are $O(1/\sqrt{N})$. This would then lead us to conclude that the expected IPL of such a tree is $\Omega(N^{3/2})$ as claimed, subject to the hypotheses described above.

In the next section an approximation is presented representing what we feel is the correct asymptotic coefficient for the size of the top most subtree when it first becomes the right subtree of the root. This result is in excellent agreement with the results of the extensive simulations we have performed.

In what follows, we compute a slightly different lower bound for the Internal Path Length using a different assumption. However, this should be taken only as another indication that the trees do indeed get bad. We firmly hold that the previously computed lower bound is correct. The best bound on the covariances that have yet been obtained is $O(1/N^4)$, as follows.

Lemma 2.28

$$\mathrm{Cov}(\mathbf{M}_{t,i}\mathbf{M}_{t,j}) = O\left(\frac{1}{N^4}\right)$$

Proof:

$$\mathrm{Cov}(\mathbf{M}_{t,i}\mathbf{M}_{t,j}) = \mathrm{E}[\mathbf{M}_{t,i}\mathbf{M}_{t,j}] - \mathrm{E}[\mathbf{M}_{t,i}]\mathrm{E}[\mathbf{M}_{t,j}]$$
$$\leq \mathrm{E}[\mathbf{M}_{t,i}\mathbf{M}_{t,j}]$$

since the expectations are positive.

$$= \mathrm{E}[\mathrm{E}[\mathbf{M}_{t,i}\mathbf{M}_{t,j} \mid \mathbf{Z}_{t,i}]]$$
$$= \mathrm{E}[\mathbf{M}_{t,i}\mathbf{M}_{t,j} \mid \mathbf{Z}_{t,i}=1]\mathrm{P}\{\mathbf{Z}_{t,i}=1\}$$

since the distance moved when the tag doesn't move is zero

$$= \mathrm{E}[\mathbf{X}_i\mathbf{M}_{t,j}]\frac{1}{N}$$
$$= \frac{1}{N^2}\mathrm{E}[\mathbf{X}_i\mathbf{X}_j]$$

using the same argument for the second update,

$$= \frac{1}{N^2}\mathrm{E}[\mathbf{X}_i\mathrm{E}[\mathbf{X}_j \mid \mathbf{X}_i]]$$
$$= \frac{1}{N^2}\mathrm{E}[\mathbf{X}_i O\left(\frac{1}{N}\right)]$$

since we already have shown in the proof of Lemma 2.18 that the expectation given any previous set of moves is $O(1/N)$

$$= O\left(\frac{1}{N^3}\right)\mathrm{E}[\mathbf{X}_i]$$

$$= O\left(\frac{1}{N^4}\right)$$

□

If we consider the sum of the covariances over $\Theta(N^2)$ updates, the sum of the covariances between all pairs is $O(1)$, which is insufficient as a bound for our purposes. To get a total of $O(1/N)$ would require a bound of $O(1/N^5)$.

If we assume that the covariance between moves in which there are no common keys is negligible then we can obtain a different bound. Note that for two sets of keys with none in common, the intervals between the keys of one set are independent of the other. This implies that the covariance between any interval of one set and one in the other is zero. However, knowing that we are to the right of some interval which is larger than expected in the first set indicates that we are closer to the end point than expected after the update that moved over that interval, and so the expected distance to the end point may be reduced. We assume this is the case, and thus that the covariance is either zero or negative, and can be safely neglected in our upper bound computation.

We now consider updates which are at least $2N\ln N$ updates apart.

Lemma 2.29

The probability that a key remains on the tree throughout $2N\ln N$ updates is $O(1/N^2)$.

Proof: The probability that the key remains is

$$\left(1-\frac{1}{N}\right)^{2N\ln N} = N^{2N\ln\left(1-\frac{1}{N}\right)}$$

$$= N^{2N(-\frac{1}{N}-\frac{1}{2N^2}-\cdots)}$$

$$= \left(\frac{1}{N}\right)^2\left(\frac{1}{N}\right)^{(\frac{1}{2N}+\cdots)}$$

$$= O\left(\frac{1}{N^2}\right)$$

Using this lemma we see immediately that the expected number remaining after $2N\ln N$ updates is $NO(1/N^2) = O(1/N)$. Thus, the probability that there are 1 or more keys in common is $O(1/N)$.

Lemma 2.30

For any update, the total covariance of that update with all updates more than $2N\ln N$ updates away, but within $O(N^2)$ updates, is $O(1/N^3)$, under the assumption of negligible covariances for independent key sets.

Proof: Under the assumption, we only need compute the sum for those which contain at least one key in common. From the preceding comments, the probability of this is $O(1/N)$, and from Lemma 2.28 the covariance in that case is $O(1/N^4)$. We have $O(N^2) - 2N\ln N$ updates to consider, and so the sum is $O(N^2)O(1/N^4)O(1/N)$ which is $O(1/N^3)$.

□

Lemma 2.31

The sum of covariances for any update with those within $2N\ln N$ updates is $O(\ln N/N^3)$.

Proof: The covariances are $O(1/N^4)$ by Lemma 2.28, so for $O(N\ln N)$, the total is $O(\ln N/N^3)$.

□

Lemma 2.32

$$\sum_{O(N^2)} \mathrm{Cov}(\mathbf{M}_{t,i}\mathbf{M}_{t,j}) = O\left(\frac{\ln N}{N}\right)$$

Proof: From the preceding two lemmas, the total covariance of any update of a tag with all other updates is $O(\ln N/N^3)$, so for $O(N^2)$ the result follows by multiplication.

□

With this bound on the sum of the covariances, we get the same bound for the variance of the sum, since the sum of variances is $O(1/N)$. This means that the expected deviation is $O(\sqrt{\ln N/N})$. Arguing in a manner similar to that used to obtain the previous bounds this gives us a bound on the expected size of a subtree of the backbone of $O(\sqrt{N\ln N})$. This means there must be at least $\Omega(\sqrt{N/\ln N})$ such subtrees. And so we obtain

Lemma 2.33

Under the hypothesis that the covariances are negligible when moves are over independent sets of keys the expected Internal Path Length is $\Omega(N^{3/2}/\sqrt{\ln N})$.
Proof: As in Theorem 1, we form the sum

$$\sum_{i=1}^{\Omega(\sqrt{N/\ln N})} i\sqrt{N\ln N} = \Omega\left(\frac{N^{\frac{3}{2}}}{\sqrt{\ln N}}\right)$$

□

We emphasize that this bound is considered to be too weak.

2.7. What We Really Expect to Happen

In this section, an informal argument will be given to suggest that the asymptotic expected size of the subtree of the root when it first becomes the right son of the root is approximately $\sqrt{2\pi N} \approx 2.506\sqrt{N}$. This model will also be used to argue that the number of backbone nodes is $\sqrt{2/\pi N} \approx 0.80\sqrt{N}$.

We have developed a series of models of the behavior of trees, from the EFD model to the more general but incompletely analyzed models where keys are drawn from the uniform domain at random. In the EFD model we computed the expected interval size of the topmost interval at the time the upper tag reached the boundary by computing the expected interval between the tags under the condition that they did not bump into one another. Thus we obtained the result that the lower tag would be an expected $\sqrt{\pi N}$ steps from the boundary at this time.

The next model was that of a random walk with random step sizes. The distribution for each step was that of the interval between two adjacent keys out of N distributed over the interval (0,1). Let us alter this model so that the domain is $(0,N+1)$, thus making the absolute expected step size 1. That is, it will now have the same expected step size as the EFD model. The probability distribution of an interval is now

$$P\{X>x\} = \left(1-\frac{x}{N+1}\right)^N$$

We can now describe the interval between an adjacent pair of tags as a random walk in a manner analogous to the analysis of the EFD. That is, we assume that it is equally probable that the lower or upper tag is moved each time the interval is changed, and that the distribution of the size of the move is given in each case by the above distribution. We let **C** be the random variable indicating the size of the change and thus we can describe the distribution $f(c)$ in terms of the random variable **X** by

$$f(c) = \frac{1}{2} f_x(|c|)$$

It is easily seen that, due to symmetry, all odd moments of this distribution are zero. Also, applying symmetry again, we find the second moment of **C** to be

$$\begin{aligned} \mathrm{E}[\mathbf{C}^2] &= \int_0^{N+1} x^2 \frac{N}{N+1}\left(1-\frac{x}{N+1}\right)^{N-1} dx \\ &= 2\frac{N+1}{N+2} \\ &\approx 2, \quad N \to \infty \end{aligned}$$

This is also the variance of **C**, since the expected value is zero.

In comparison we note that the variance per change for the EFD model was 1. Intuitively, the primary difference between the EFD model and the system using the uniform domain with independent insertions is that in the latter the variance of the steps is increased. The number of steps may now be greater than or less than N, but asymptotically at least it seems reasonable to assume that taking steps of size approximately 1 would mean that we require N steps on average for the tags to fall off the tree.

We repeat that this model assumes that the absolute sizes of the changes in the interval are asymptotically independent random variables with the distribution of the intervals between N uniformly distributed variables. This is the reasonable, if unproven, extrapolation of the results in the preceding section. We now want to consider the conditional random walk based on this distribution wherein we want the expected size of the interval given that the interval has never been less than or equal to zero.

We approximate the preceding random walk with the following discrete random walk. We define a random walk by

$$S_N = \sum_{i=1}^{N} X_i$$

where the distribution of the random variables X_i are given by

$$r(x) = \begin{cases} \frac{1}{2} & x = -\sqrt{2} \\ \frac{1}{2} & x = \sqrt{2} \end{cases}$$

We have then a simple random walk, as defined in Feller [8] for example, except that the step sizes are $\sqrt{2}$. We choose this step size so that the second moment will be equivalent to that of the preceding random walk. This walk can be readily modeled by coin flipping, where a head corresponds to a positive step and a tail to a negative one.

It can be readily seen that this distribution has all odd moments equal zero, and that the second moment (and the variance) is equal to 2. We expect S_N to behave then somewhat like the previous random walk. It is easy to see that asymptotically these two unbounded walks should converge in distribution. Whether convergence also holds when we apply the condition that the walks never go below the origin is open, but it seems reasonable that first and second moments would be asymptotically equivalent and thus that the expected interval would be equivalent.

Recalling that the expected deviation for the EFD is $\sqrt{\pi N}$, and that the step size is one, we see that the deviation for this walk is $\sqrt{2\pi N}$, the proof being as in Lemma 2.1. We note that this does not take into account the slight negative correlation that we believe may occur. In particular, if the variance in the intervals is partly responsible for the final deviation, then having a positive deviation means the upper tag has moved slightly farther than expected per move, and thus the number of moves of the upper tag may be slightly less than N. We could guess that the upper tag moves $N - O(\sqrt{N})$ times when it and the lower tag are known to diverge. Thus, the estimate of $\sqrt{2\pi N}$ may be slightly high for smaller N, but we expect it to be asymptotically correct. This argument applies less strongly to intervals generated by $i < N - O(\sqrt{N})$ moves, since the effect is entirely due to the fact that the upper tag cannot move beyond the endpoint.

Using the approximation $\sqrt{2\pi i}$ for the interval following a tag at position i, we find that the probability of the ith node being tagged is approximately $1/\sqrt{2\pi i}$. And thus the expected number of backbone nodes is

$$\mathrm{E}[B] = \sum_{i=1}^{N} \frac{1}{\sqrt{2\pi i}} \approx \frac{2\sqrt{N}}{\sqrt{2\pi}} \approx 0.798..\sqrt{N}$$

However, this assigns a probability of $1/\sqrt{2\pi} = 0.3989$ to the first node being tagged when in fact we know that it is always tagged. Similarly, this formula slightly underestimates the other low order nodes as well. Thus, for any N, the number of nodes predicted by this formula should be a bit less than actually measured.

Assuming that the convergences outlined above are correct, we have again that the expected IPL of the tree after sufficiently many updates is $\Theta(N^{3/2})$. The simulations presented in the next chapter lend strong support to these conjectures, the values for large N being almost exactly those predicted.

We can get a crude estimate of the coefficient of the leading term for the expected IPL by pushing our approximation a bit further, although we now depart somewhat from reality. In the idealized situation in which the ith backbone node is at its expected position, we assume that the size of the ith subtree should be linear in i. Thus, the ith subtree from the left should contain

$$\sqrt{2\pi N}\frac{i}{\sqrt{2N/\pi}} - c \approx \pi i$$

nodes, where c is a constant and will now be ignored. Since there are $\mathbf{B}_K$ backbone nodes on average, the root of the ith subtree from the left of the tree is at a depth of approximately $\sqrt{2N/\pi} - i$ from the root. Ignoring any contribution the IPL's of the right subtrees might make to the leading coefficient, that is assuming the subtrees are roughly balanced, we can compute an approximation to the leading term of the IPL by

$$\int_{i=1}^{\sqrt{2N/\pi}} \pi i(\sqrt{2N/\pi} - i)di = \int_{i=1}^{\sqrt{2N/\pi}} \sqrt{2\pi N} i di - \int_{i=1}^{\sqrt{2N/\pi}} \pi i^2 di$$

$$\approx \frac{\sqrt{2\pi N}\frac{2N}{\pi}}{2} - \frac{\pi\left(\frac{2N}{\pi}\right)^{3/2}}{3}$$

$$= \frac{1}{3}\left(\frac{2}{\pi}\right)^{1/2} N^{3/2}$$

$$\approx 0.266 N^{3/2}$$

Despite the lack of justification for this model (or perhaps because of it) this value nevertheless agrees quite well with the simulation results presented in Chapter 3.

2.8. Further Comments and Observations

In a previous section, it was claimed that a tree on an EFD would be more skewed because the variance of the tag motions would be reduced by the absence of insertions, resulting in a reduction of expected subtree sizes. In view of this, it seems somewhat paradoxical to claim that the analysis for general trees holds true for insertions from any distribution, provided the insertions are identically and independently distributed random variables with finite mean and variance. To verify the claim, we consider the usual method of generating random variables of some desired distribution $F(x) = P\{X \le x\}$ from the uniform distribution U by setting $X = F^{-1}(U)$; that is, generating the inverse function value of a uniform variate. (See Knuth [15] chapter 3.4.1). We note that such a transformation does not alter the relative rank of the variates when applied uniformly to each, and that the shape of the tree depends only on the relative ranks of the keys. Thus, applying such a transformation does not affect our result.

The resolution of the paradox lies in the observation that the reduction in variance on EFD trees comes from a perfect correlation between a deletion and the subsequent insertion. Thus, on EFD trees the number of nodes in any right subtree of a backbone node changes if and only if the backbone node or its parent moves. We assume on general trees that no such correlation exists between deletions and insertions.

We conjecture that the subtrees remain somewhat balanced, so that they do not contribute significantly to the $N^{3/2}$ term. The only theoretical justification for this is that each subtree is moving as a whole to the right. Thus, on average every node is moving at the same rate, and there should therefore be no overall skewing effect. Of course, we know this is not strictly true, since some trees must be growing as they move, while others shrink away and disappear. In fact, as will be evidenced in the next chapter, results from simulations suggest that the subtrees are somewhat right skewed, but that the IPL is less than the IPL of a random tree of the same size. Whether these effects persist for arbitrarily large trees is open and would make an interesting simulation project. The large variance due to random rates of motion may be offset somewhat by the rebalancing effect that occurs when the right subtree of a node scheduled for deletion is empty.

We can view the dynamics of a subtree as a complex flow of nodes from right to left over the subtree as it moves to the right. The two backbone nodes defining the interval containing the subtree will move gradually to the right, as will the root of the subtree. Insertions will fall at random over the entire interval, adding to both the left and right portions of the subtree. Deletions of the key at the left end of the interval cause the leftmost node of the subtree to be removed, thus reducing the left portion of the subtree. Deletions of the key at the right end of the interval open the gap to the right, thus causing the right side of the subtree to grow on average. However, deletions of the root of the subtree will cause right to left rebalancing within the subtree even as they do within the entire tree. Deletions of the subroots of the subtree will have similar effect on those smaller subtrees. Thus, there will be a tendency for the subtree to grow to the right and be reduced at the left, offset by the motion of the root of the subtree which causes the subtree to rebalance from the right to the left.

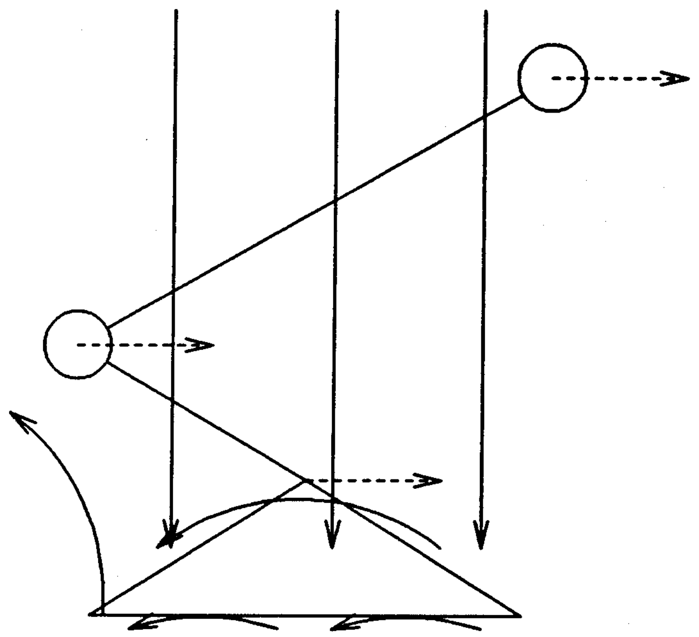


Figure 2.10 Flows In Subtrees

In Figure 2.10 the dashed arrows indicate the motion of the backbone nodes defining the interval containing the subtree. The vertical straight lines indicate the addition of nodes to the subtree through insertions. The arcs indicate the rebalancing of the subtree induced by deletions and the deletion of material from the tree as the lower backbone node moves to the right. The entire process appears to be very complex.

Generally speaking, it is unlikely that every deletion will be exactly paired with a subsequent insertion in most real world uses of binary trees. This problem is also mentioned by Knott [14] and others. We present no theory to generalize our results, however it seems intuitive that the result of our scheme is worse than any scheme in which the deletions and insertions occur in random fashion. If a large batch of deletions occur at random and then an equal number of insertions follow, the conjecture is that the resulting tree will on average be better balanced than a similar tree in which the deletions and insertions are paired. The idea is that in the batch version the insertions will be split up over a reduced number of nodes forming larger subtrees that are randomly balanced, but are fewer in number than in the paired version. More specificly, the number of backbone nodes will be reduced after the batch of deletions on average, and the average interval between the backbone nodes should be larger, than would be the case for a tree of N nodes after the same number of updates. Since the next batch of insertions will be uniformly distributed over each of these intervals, the tree will be less skewed on average than the paired model suggests. As an extreme example, if all N nodes are deleted, then N new keys inserted, the resulting tree will have an expected IPL of $O(N\lg N)$.

In the analysis of the Hibbard scheme, the expected IPL is $\Theta(N^{3/2})$ independent of the shape of the subtree. Thus, the analysis applies to any deletion scheme that replaces the deleted element with its successor when it exists, and with the left son otherwise, but makes any changes it likes to the rest of the right subtree of the deleted node; the only limitation is that no other changes outside the right subtree can be made.

The lower bound of $\Omega(N^{3/2})$ applies to an even larger class of algorithms. For example, algorithm 1 of [5] uses the rule choose the successor or else choose the predecessor as the replacement node. A quick analysis shows that the expected interval sizes between backbone nodes will be much reduced since tags can only become co-joined if two successive subtrees are empty when the tagged key is deleted. In Chapter 6 of [5] a deletion algorithm called RR for recursive replacement is given. This algorithm has the rule recursively delete the successor or else the predecessor and replace the deleted node with the resulting key. Again, it is easily seen that in this scheme tags are never co-joined, since if a backbone node cannot move right when its key is deleted, it moves left, pushing as many nodes as necessary to the left as well. It is conjectured that this algorithm results in trees in which the expected IPL is $\Omega(N^2)$.

Knuth [16] presents an improved algorithm that differs from Hibbard's only in checking for an empty left subtree before choosing the successor to replace the key. If the left subtree is empty, then the right son replaces the node containing the deletion key. He shows that for one application, his rule always results in a tree that is at least as well balanced as the one produced by the Hibbard algorithm when applied to the same tree, and is often better. However, we have the following corollary to our previous claims.

Corollary

Assuming that the model using identical distributions for the moves is sufficiently accurate, updates using the Knuth algorithm for deletions results in a tree with $\Theta(N^{1/2})$ expected search cost.

Proof: In the previous analysis only the backbone deletions are considered, and thus the only deletions that could change the results of the analysis are those involving the leftmost node in the tree. Even then, the result differs only if the right subtree of that node has at least two nodes in its backbone. In that case, the leftmost tag moves to the right as before, but one or more additional tags must now be inserted between the two smallest tags. For example, if we delete 'a' from the left tree in Figure 2.11 using the Knuth algorithm, then both 'b' and 'c' are added to the backbone, forcing us to introduce the tag labeled '0'. This can only reduce the size of the resulting subtrees, and increase the length of the backbone, and so the average IPL is still $\Omega(N^{3/2})$.

We turn now to the upper bound. We observe that for any sequence of insertions and deletions, the tags created by the Hibbard scheme will also be created by the Knuth algorithm, and that for any update these tags behave in identical fashion under either algorithm. Furthermore, the Knuth algorithm can only create additional tags (i.e. backbone nodes) from nodes which are in the right subtree of the leftmost backbone node. Finally, if after any update using the Knuth algorithm, we switch to the Hibbard algorithm, the tags which already exist at the time of the switch will behave exactly the

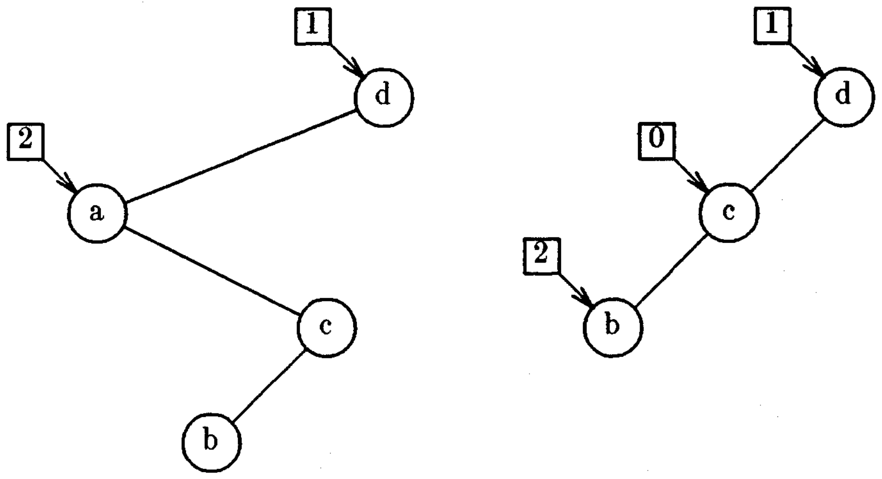


Figure 2.11 Adding New Tags by Knuth Deletion

same as if we continued with the Knuth algorithm. Let $f(N) = o(\sqrt{N})$, but $\omega(1)$. We can use Kolmogorov's inequality (See e.g. Feller [[9] Chapter V.8 (e)) in the following way to see that the probability of the $f(N)$th node being in the leftmost subtree is zero. We pick a random time t, and note that for the $f(N)$th node to be in the leftmost subtree, there can be no tagged node between it and the smallest node. We now consider the tag which was on the smallest node at time $t-f(N)N/2$. Its expected position is currently $f(N)/2N$. However, it must now either be on the current smallest node or to the right of the $f(N)$th node. In either case, it must either have deviated by $\Omega(f(N)/N)$ from its expected position, or the number of nodes in an interval of size $O(f(N)/N)$ must deviate by $\Omega(f(N))$ from the expected number, or both. The variance in the position of a tag over $f(N)N/2$ updates is $O(f(N)/N)$, and the variance of the number of nodes in an interval of size $f(N)/2N$, is the variance of the binomial with the corresponding probability, which is

$$N\frac{f(N)}{2N}\left(1-\frac{f(N)}{2N}\right) < \frac{f(N)}{2}$$

Applying Kolmogorov's inequality to either of these situations leads to the probability being $O(1/f(N))$ which is asymptotically zero. We have already observed that the Knuth algorithm increases the density of nodes on the backbone, and so the probability of the $f(N)$th node being in the first subtree is even less under the Knuth algorithm. Thus, the special effect of the Knuth algorithm cannot operate to the right of the $f(N)$th node, meaning the tags to the right of that node behave exactly as they would under the Hibbard algorithm. We conclude by noting that even if the Knuth algorithm were able to linearize the leftmost $f(N)$ nodes of the tree, the number of nodes on the backbone would still be $O(\sqrt{N})$, since the tags would still have $N-f(N)$ nodes over which the deviations could occur. Since the subtrees also still contain $O(\sqrt{N})$ nodes each, the asymptotic result is unchanged.

□

We should mention that the Knuth algorithm is free to increase the rebalancing effects on the right subtrees, and, guessing that the leftmost subtree is probably bounded by a small constant, we expect almost no change in the behavior of the backbone. Simulation results suggest that the Knuth algorithm produces a slightly smaller average cost than the Hibbard.

Chapter 3

Extended Hibbard Simulations

3.1. Introduction

In Chapter 2, it was argued that we expect the IPL to be $\Theta(N^{3/2})$ for both the Hibbard and Knuth schemes after more than N^2 updates have been made. The leading coefficient was estimated, but there is reason to believe that this estimate is not very precise, and, even if it is, the resulting estimate of the expected IPL is not precise. In large part this is because of the unknown structure of the subtrees of the nodes of the backbone. Furthermore, neither the expected length of the backbone nor the expected size of the right subtree of the root has been precisely determined. Finally, the distribution of the sizes of the subtrees of the backbone nodes is also unknown.

In this chapter we improve our understanding of the updating process by studying various simulations that we and others have performed. In this way we are able to estimate some of the relevant parameters of the process, including the coefficient of the leading term of the IPL. First, we review the data obtained from the simulations in Eppinger [7] and Culberson [5] using the evidenced lower bound as a guide to further analysis, and thus obtain our first estimate of the coefficient of $N^{3/2}$. During the process of developing the theoretical analysis of Chapter 2, we performed many further simulations to test various parameters and to guide the analysis. In this chapter we present the results of these simulations which lead to further insights into the updating process.

Although we have packaged the simulations and the theory into separate chapters, we wish to emphasize the interaction of the two during the development of our understanding of the problem. In particular it should be noted that the simulations performed by Eppinger were the first published indication that the Internal Path Length increased under the updating process.

3.2. Previous Simulations

In this section we re-evaluate the data from the extensive simulations of Eppinger [7] and Culberson [5] using our estimated lower bound as a guide.

Previously, they conjectured that the IPL was $\Omega(N\log^3 N)$ when the Hibbard or Knuth algorithms were used for many deletions.

Table 3.1

Empirical Mean Internal Path Lengths
from Simulations by Culberson and Eppinger

Number of	Hibbard Algorithm			Knuth Algorithm	
Nodes	Mean IPL	# Iterations	# Trees	Mean IPL	# Iterations
32	-	-	-	132.4	7500
64	349.9	10000	200	345.2	50000
128	889.8	50000	200	880.0	80000
256	2251.3	120000	100	2232.2	150000
512	5737.1	500000	50	5643.1	500000
1024	14687.2	2500000	25	14348.3	2000000
2048	37876.0	9000000	20	-	-

Table 3.1 presents average IPL's from simulations of the Hibbard algorithm as measured by Eppinger [7] and for the Knuth algorithm from Culberson [5]. These values were obtained from the published ratios by multiplying by the expected IPL's of random trees using the formula for random trees given in Knuth [16] pp 427. In these simulations, a random tree was first generated, then the prescribed number of iterations were performed. During this process the IPL was computed at frequent intervals. The process was then repeated for the indicated number of trees. The average IPL's shown in Table 3.1 are obtained only from data collected after the first N^2 updates, since only then did the trees appear to be stable. The analysis of Chapter 2 shows that this conjecture was justified. The extent of the simulations is shown by the large number of iterations and the number of trees simulated in Table 3.1. The Knuth algorithm was run on 50 trees in each case.

A regression analysis was used to fit the data from Table 3.1 to functions involving $N^{3/2}$, $N^{5/4}$, $N\lg(N)$ and N, where $\lg(N) = \log_2(N)$. The term $N\lg(N)$ arises naturally in an estimate of the IPL if we assume the subtrees are balanced. The term $N^{5/4}$ is included as an arbitrary term asymptotically between $N^{3/2}$ and $N\lg(N)$. Observe, however, that for N less than 65536, $N^{5/4}$ is less than $N\lg(N)$. A tabulation of the coefficients obtained are presented in

Table 3.2

Regression Coefficients for Data from Eppinger and Culberson

	$N^{3/2}$	$N^{5/4}$	$N\lg(N)$	N	Constant	Mean Square
Hibbard	0.405	-	-	-	634.73	304850.0
	0.270	-	0.570	-	-14.38	79.1
	0.259	-	0.726	-1.228	18.44	5.5
	0.201	1.369	0.018	-	-7.42	3.9
	0.199	1.412	-	-	-7.19	2.7
	0.199	1.405	-	0.021	-7.79	4.0
Knuth	0.434	-	-	-	272.04	60566
	0.256	-	.582	-	-9.51	5.9
	0.248	-	0.659	-0.527	-1.42	2.0
	0.190	1.210	0.112	-	-13.21	4.7
	0.174	1.497	-	-	-14.08	3.3
	0.172	1.512	-	-0.040	-13.51	4.9

Table 3.2. Also included is the residual mean square to show how well the various formulations fit the data. All the regressions are unweighted.

When we compare the leading coefficient of these regressions with the value 0.266 computed in the approximation model of section 2.4, we see that the experimental results strongly support the conjecture that the subtrees are balanced well enough that they do not contribute to the $N^{3/2}$ term. On the other hand, excellent fits are obtained by replacing the $N\lg(N)$ term with $N^{5/4}$. When both of these terms appear in a regression for the Hibbard algorithm, more significance is attached to the $N^{5/4}$ term than to $N\lg(N)$.

3.3. Measuring the Right Subtree

The simulations in this section were designed to measure the size and characteristics of the right subtree of the root after N^2 updates have been made. These simulations were progressively extended to include extra data gathering as our understanding increased and we required more detailed information. Originally only the length of the backbone and the size of the right subtree were measured. Later, we extended the lengths of the runs measuring the IPL in addition to the backbone and right subtree size. The purpose was to allow comparison with the IPL's obtained by Eppinger. Since we gathered data at specific times as described below, we wished to test that this did not significantly affect our measures. Finally, the lengths of the runs were extended even further, this time gathering information on the structure of the top five right subtrees of the backbone in addition to the previous data. In addition, three long runs were performed for trees of size 8192 gathering all types of data throughout. We were now particularly interested in determining whether the subtrees themselves exhibited any left or right imbalance. In all cases, the data was gathered only when a new right subtree of the root was created.

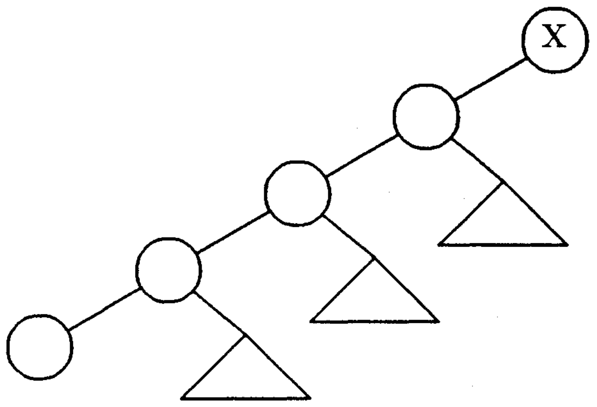


Figure 3.1 Collect Data When X Deleted.

Figure 3.1 illustrates the times at which the data was gathered. A new right subtree for the root is created when the key in the current root is deleted, and the right subtree of the root is empty. Note that at these times, the interval containing the new subtree has had the maximal time for growth, since any further change to this interval can only decrease the interval size by moving the root towards the right of the domain. This, of course, is not the same as saying that the right subtree is at its maximum size, or that it is the maximal subtree in the tree. However, it should give a good approximation to the subtree sizes as estimated in Chapter 2.

Unlike previous simulations, this one performed updates for a long time on a single tree. This makes the simulation more efficient since the subtrees in the initial tree must all be discarded from the analysis, to ensure that all the measured subtrees are between tags created during the updating process. The data analysis routines checked the number of nodes in the backbone of the initial tree and discarded that many subtrees from the calculations. This means that approximately N^2 updates were wasted on each initial tree. If there are B nodes in the backbone of a tree at any point in the simulation, then after B more subtrees have been removed, the tree is made up entirely of new keys, and the shape of this new tree will be independent of the shape of the previous one.

Table 3.3

Right Subtree For Hibbard with Leaf Insertion

Number of	Millions of	Number of	Subtree Size		Backbone	
Nodes	Updates	Subtrees	Avg	Var	Avg	Var
512	7	236	51.9	801.8	18.4	4.2
1024	22	237	79.6	1765.0	25.4	4.4
2500	70	209	118.4	3652.2	40.8	6.4
4096	140	188	158.3	6529.5	51.5	8.4
8192	600	270	233.1	15466.2	71.1	14.7
8192	400	169	229.6	15250.6	72.3	13.7
8192	400	181	219.3	13532.8	73.2	10.0

Table 3.3 summarizes the results of these simulations. These are the cumulative results from all three stages of the simulations. The column titled "Number of Subtrees" shows how many subtrees were used in computing the averages. The standard deviation of the subtree sizes is high, being approximately equal to one half of the mean in each case. The variance on the length of the backbone, on the other hand, is small, indicating a negative correlation between the subtrees. The number of updates is the number performed during the simulation. In general, the last new subtree occurred slightly prior to this time, and thus the number of updates required to generate the indicated subtrees was slightly less. Again recall that the first B subtrees generated are not counted, where B is the number of nodes in the backbone of the initial tree.

In Table 3.4, the averages for the sizes of the subtrees and the lengths of the backbone are expressed as multiples of $N^{1/2}$. These values are close to, but generally less than, the value of $\sqrt{2\pi} \approx 2.5$ computed in the approximation model found in Chapter 2. The data in Table 3.4 tend to confirm this conjecture. If we compute the average of the three runs for 8192, we find the coefficient for the subtree size is 2.512 and for the backbone it is 0.798 which agree fairly well with the conjectured 2.506 and 0.798 respectively.

Table 3.4

Average Subtree and Backbone Values
Expressed as Ratios to $\sqrt{N}$

Number of Nodes In the Tree (N)	Right Subtree	Standard Deviation	Backbone Length	Standard Deviation
512	2.29	1.25	0.813	0.091
1024	2.49	1.31	0.793	0.066
2500	2.37	1.21	0.816	0.051
4096	2.47	1.26	0.805	0.045
8192	2.58	1.37	0.786	0.042
8192	2.54	1.36	0.799	0.041
8192	2.42	1.29	0.809	0.035

Table 3.5

Average IPL From Extended Hibbard Simulations

Tree Size	Number of Subtrees	Average IPL	Variance
512	107	5,505	112,004
1024	137	13,999	645,567
2500	112	48,604	4,230,873
4096	118	94,987	21,450,140
8192	270	250,201	214,230,784
8192	169	254,515	201,874,000
8192	181	259,200	77,005,696

Table 3.5 shows the average IPL computed during the two extensions to the simulations. We repeat that these values are computed using data collected at the special times when a new right subtree has been created at the root. Deleting the root when it has no right subtree moves every other node in the tree up one level. Thus, we would expect the IPL to be reduced by $O(N)$ from the IPL previous to that update. However, in comparing the values obtained for trees of sizes 512 and 1024 in Table 3.5 with the values in Table 3.1, we see that the results are in reasonable agreement nonetheless. The column indicating the number of subtrees indicates the number of data

points used in computing the means and variances.

Table 3.6

Regression Coefficients for Data from
New Simulations of Hibbard's Algorithm.

$N^{3/2}$	$N^{5/4}$	$N\lg(N)$	N	Constant	Mean Square
0.339	-	-	-	3975.60	11644000
0.277	-	0.461	-	310.01	10326000
0.368	-	-2.216	26.907	-2514.70	13641000
0.656	-9.645	4.895	-	-1361.40	13615000
0.238	1.000	-	-	494.64	10351000
0.460	-3.041	-	18.689	-2178.60	13632000

In Table 3.6 we have the results of various regressions of the data in Table 3.5. Notice that the coefficient of $N^{3/2}$ seems to be fairly stable, but that the coefficient of $N^{5/4}$ varies greatly, even changing sign. Invariably, the error associated with $N^{3/2}$ was much better than that of $N^{5/4}$.

Table 3.7

Regression Coefficients for Combined Data
from Simulations of the Hibbard Algorithm

$N^{3/2}$	$N^{3/2}/\sqrt{\lg N}$	$N^{5/4}$	$N\lg(N)$	N	Constant
0.341	-	-	-	-	2707.50
0.269	-	-	0.517	-	191.09
0.317	-	-	-0.732	12.003	-567.80
0.528	-	-6.373	3.380	-	-326.23
0.223	-	1.145	-	-	295.98
0.359	-	-1.213	-	10.201	-551.13
-0.216	2.017	-	-	-	369.87
1.885	-6.705	-	2.220	-	-327.23
0.747	-1.828	-	-	9.4249	-544.30
7.869	-32.499	-	14.806	-57.989	1345.00
143.670	-715.320	719.730	-149.470	80.270	-1760.90
-	1.233	-	-	-	1271.70
-	1.150	-	-	2.200	121.36
-	1.115	-	0.234	-	274.45
-	1.361	-	-1.370	14.815	-640.95
-	1.021	0.569	-	-	326.36

If we combine this data with that from Eppinger [7] we increase the number of data points to 9, at the same time obtaining duplicate values for 512 and 1024 and of course 8192. Regressions were performed for various combinations of $N^{3/2}$, $N\log_2 N$, $N^{3/2}/\sqrt{\lg N}$, N, $N^{5/4}$ and a constant term. The results are tabulated in Table 3.7. The $N^{3/2}/\sqrt{\lg N}$ term was included since it is an alternative lower bound computed in the previous chapter. No clear preference was indicated throughout the regressions for this term over the $N^{3/2}$ term, and this is reflected in the rather erractic behavior of the coefficients when these two terms are both included. However, the length of the backbone and the size of the subtrees reported in Table 3.4 strongly support the $N^{3/2}$ model over the $N^{3/2}/\sqrt{\lg N}$ model. In particular, in the latter model the length of the backbone should differ by a factor of about 1.2 for trees with 8192 nodes from those with 512, as should the size of the right subtree. No such difference is evident. These results are in good agreement with those of Table 3.2. There seems to be good evidence for the approximation model.

We now turn to the results of the third stage of the simulation. Following [5] we define the *Left Normal Measure* (LNM) of a tree to be the sum over all nodes of the tree of the number of nodes in the left subtree of the node. The *Right Normal Measure* (RNM) is defined analogously to be the sum over the right subtrees. If the tree is balanced, we expect LNM $\approx$ RNM.

In these simulations, the measure was taken for each of the topmost five right subtrees of the backbone each time data was gathered. We define the *Left Measure* (L) of a subtree to be the total over all the samples of the LNM of the subtree divided by the total over all samples of the number of nodes in the subtree. That is, it is an average measure of the LNM, although the term average is a bit fuzzy here since the size of the subtree also varies from sample to sample. The Right Measure R is similarly defined.

The Left and Right measures for each of the top five subtrees is presented in Table 3.8. Notice that the subtrees exhibit a tendency to be skewed to the right. This would seem to contradict the assumption that the subtrees are balanced. Also presented is an estimate of the average search path length. This is just the sum of the Left and Right measures. (In [5] it is shown that LNM + RNM = IPL). For comparison, the expected search cost of the random tree of the nearest integral size (See Knuth [[16] Chapter 6.2.2), is also listed. In every case, the random cost exceeds our average cost. We thus seem to confirm the conjecture that the subtrees are reasonably well balanced. We conjecture that the difference $R-L$ is bounded by a constant, or at least that $\frac{R-L}{R+L}$ converges to zero. That is, we guess that the right

Table 3.8

	Tree size = 512		Number of Subtrees Used = 35	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
51.171	2.280	2.683	4.963	5.215
57.771	2.254	2.852	5.105	5.453
49.800	2.180	2.687	4.867	5.178
43.486	2.096	2.756	4.852	4.902
46.400	2.088	2.723	4.811	5.025

	Tree size = 1024		Number of Subtrees Used = 73	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
75.918	2.602	3.052	5.654	5.958
77.753	2.492	3.051	5.543	6.007
72.301	2.446	3.096	5.543	5.857
72.096	2.472	2.996	5.467	5.857
71.301	2.473	2.978	5.451	5.830

	Tree size = 2500		Number of Subtrees used = 47	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
117.872	2.782	3.385	6.168	6.795
119.745	2.792	3.447	6.239	6.827
103.553	2.653	3.365	6.018	6.553
116.234	2.791	3.398	6.189	6.762
106.468	2.662	3.390	6.052	6.590

	Tree size = 4096		Number of Subtrees Used = 27	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
173.444	3.165	3.722	6.887	7.533
179.111	3.139	3.754	6.893	7.599
177.333	3.112	3.752	6.863	7..577
163.815	3.176	3.598	6.774	7.430
162.630	3.087	3.679	6.766	7.418

subtrees remain well balanced for large trees.

Finally, we include a plot of the 270 subtrees from the first simulation of 8192 in Figure 3.2. This gives at a glance an idea of the range and scatter of the subtree sizes when they first are made right sons of the root.

	Tree size = 8192		Number of Subtrees Used = 270	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
233.081	3.352	3.988	7.340	8.113
231.537	3.335	4.021	7.355	8.096
222.900	3.337	3.931	7.268	8.027
214.963	3.318	3.930	7.248	7.956
220.885	3.315	4.003	7.318	8.009

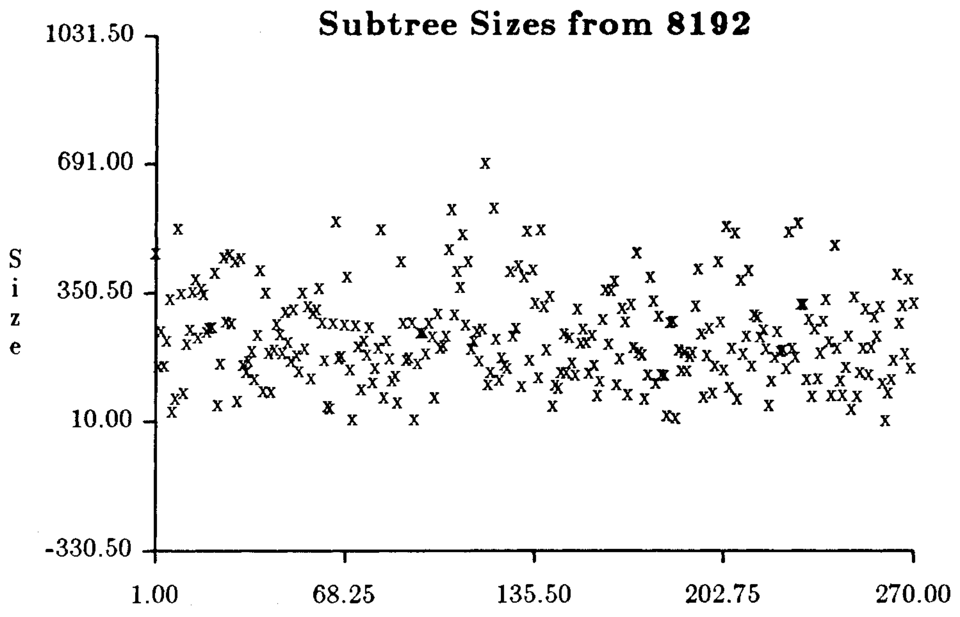
	Tree size = 8192		Number of Subtrees Used = 169	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
229.621	3.387	3.970	7.356	8.087
218.663	3.341	3.956	7.297	7.983
226.746	3.334	3.993	7.327	8.062
217.970	3.291	3.970	7.261	7.983
218.089	3.267	3.945	7.213	7.983

	Tree size = 8192		Number of Subtrees Used = 181	
Average Subtree Size	Average Left Measure	Average Right Measure	Average Path Length	Expected Path of A Random Tree
219.271	3.352	3.947	7.299	7.992
209.569	3.354	3.848	7.201	7.910
205.840	3.320	3.856	7.175	7.872
208.033	3.255	3.904	7.159	7.891
209.370	3.322	3.845	7.167	7.901

3.4. Further Subtree Simulations

The simulations described in this section have two purposes. First, they extend the results on the size of the subtrees previously obtained to significantly larger trees. Second, they provide confirmation of those previous results since, as the following description will make clear, the method of the simulation is different.

For the first point, we first notice that to achieve useful results from the above simulations, we require $\Omega(N^2)$ updates. Since the expected cost of an update becomes $\Omega(N^{1/2})$, this implies that the cost of the simulations increases at the rate $\Omega(N^{5/2})$. After describing the following simulation, we will show that this cost grows as $\Omega(N^{3/2}\lg N)$. This gives us a speedup factor of $\Omega(N/\lg N)$.



From 600 Million Updates

Figure 3.2 The 270 Subtrees of 8192

Note that in the previous simulation for the tree of size 8192, 600 million updates were performed to obtain the desired number of subtrees. This requires the generation of 1.2 billion random numbers. The random number generator used was algorithm M of Knuth [15], which has been extensively tested. The agreement of the results of the following simulation with those of the previous further increases our confidence in this generator.

In this simulation, only the backbone nodes of the tree are stored. This not only reduces the simulation time, but also reduces the memory required for large trees.

The updates are classified according to their effect on the backbone. There are four possibilities:

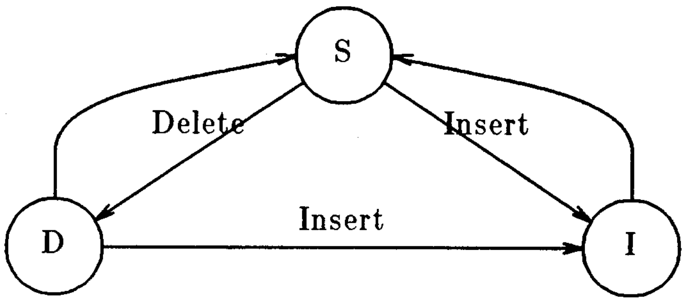
1 delete from the backbone, and insert in a subtree,

2 delete from the backbone, and insert into the backbone,

3 delete from a subtree, and insert into the backbone,

4 delete from a subtree, and insert into a subtree.

The last of these has no effect on the backbone, and yet it accounts for all but $O(1/\sqrt{N})$ of the updates, under the assumption that the backbone contains $O(1/\sqrt{N})$ nodes. Elimination of these updates is the basis for the reduction in the cost of these simulations.



S - Start. D - Delete. I - Insert.

Figure 3.3 Backbone Simulation Model

In Figure 3.3, node S indicates the state of the process before starting an update. Node D corresponds to the deletion of a key from the backbone, and I to the insertion of a new key into the backbone. An update of type 1 corresponds to the path $S-D-S$, an update of type 2 to the path $S-D-I-S$ and one of type 3 to $S-I-S$. All updates of type 4 are ignored. In our model, a deletion occurs in the backbone with probability $P(D_B) = B/N$, where B is the number of nodes in the backbone at the time of the update, and N is the total number of nodes in the tree. An insertion is made into the backbone if the new key is smaller than the smallest key in the backbone, so the probability of this event per insertion is $P(I_B) = K_s$ where K_s is the value of the smallest key, and the keys are Uniform(0,1). On each update, the process must first simulate either event D, corresponding to the deletion in an update of type 1 or 2, or event I corresponding to the insertion in an update of type 3.

Referring to Figure 3.3, the process should reach node D with probability

$$P(D) = \frac{P(D_B)}{P(D_B) + (1-P(D_B))^* K_s}$$

and node I with probability $P(I) = 1-P(D)$ from state S. If D is chosen, then an insertion may be performed after the deletion, with a probability determined by the smallest key after the deletion.

When the deletion of a key must be simulated, it is first necessary to select one backbone node v at random. Then the number of nodes N_{vr} in its right subtree must be determined. We make the assumption that for large N a sufficient approximation to the behavior of the trees is achieved by assuming that the nodes which are not in the backbone are randomly distributed over the tree. Thus, we assume that the probability of a node falling in the interval following a backbone node is proportional to the size of the interval. We assume that the number of nodes in the interval is independent of all other previous updates. We generate a Binomial random variable, Binomial($K_{v+1}-K_v$, $N-B$), where the first parameter is the size of the interval to the right of the node, and the second is the number of nodes not in the backbone. We use $N-B$ since the B nodes of backbone cannot be in any right subtree. If v is the root of the tree, then the next larger key is the dummy root of the tree with value 1. An algorithm for generating Binomial variables using the Beta algorithm is given by Knuth [15] and runs in log(N) time. The Beta algorithm used was algorithm BB of Cheng [3].

If the number of nodes in the right subtree of the selected node comes up zero, then the node is deleted from the backbone. Otherwise, we generate a random value having the distribution of the smallest of N_{vr} keys uniformly distributed over the interval, and use that as the new key for the selected backbone node. Again, we assume the uniform distribution and that the interval is independent of previous updates. Recalling that the distribution of the smallest of t keys over (0,1) is $1-(1-u)^t$, and noting that $1-u$ is uniformly distributed if u is, we see that variates having the distribution of the smallest key can be generated by $1-U^{1/t}$, where U is a random variable with the uniform distribution, and scaling over the given interval.

To insert a key, we generate a Uniform($0,K_s$) random variable, where K_s is the smallest key in the backbone.

To allow the efficient insertion and deletion of keys in the backbone, we make use of the data structure in Figure 3.4.

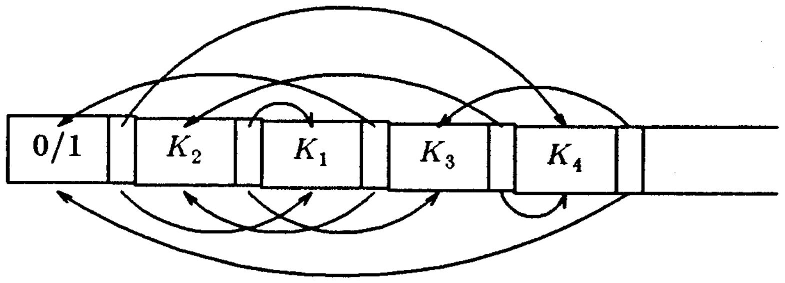


Figure 3.4 Data Structure for the Backbone

The operations that must be accommodated are:

1 Delete a random node.

2 Insert a smallest new node.

These operations can be easily accommodated in $O(1)$ time by keeping the nodes in the first B elements of an array. The elements are also members of a doubly linked circular list, where the pointers point to the next larger and next smaller members of the backbone, as in Figure 3.4. There is a dummy zeroth element assumed to contain 0 or 1, depending upon which pointer is used to reach it. To select a random node from the backbone, we select a random index from 1 to B. To delete a node, we simply move the last element of the array into the position occupied by the node, and adjust the pointers appropriately. To insert a node, we simply add it to the end of the array, and link it as the next larger node after the zeroth element, since to be inserted it must contain the new smallest key.

From the theory developed in Chapter 2 we know that the number of nodes in the backbone is $\Theta(N^{1/2})$, and that the probability of inserting a new node into the backbone is $O(1/N)$. Thus, $P(D)$ $P(I)$, and so most of the updates that we do will require a deletion. Since we must generate a Binomially distributed random variable for each deletion, we require $\log(N)$ time to do an update. Each simulated update is worth $N^{1/2}$ 'real' updates, and so the total time required to simulate $\Omega(N^2)$ updates is $\Omega(N^{3/2}\log N)$. Comparing this with the $\Omega(N^{5/2})$ for the simulation of the actual trees computed above, we have a speed up factor of $\Omega(N/\log N)$.

These simulations were run for each size of tree used in the above simulations, and in addition for trees of size 25,600 and 100,000. Table 3.9 shows

Table 3.9

Backbone Simulation Results

Number of	Millions of	Number of	Subtree Size		Backbone	
Nodes	Updates	Subtrees	Avg	Var	Avg	Var
512	4	139	50.27	755.6	19.55	4.1
1024	10	106	78.38	1747.9	27.12	4.9
2500	35	93	115.54	3734.3	42.27	7.7
4096	80	94	153.41	7440.8	54.01	8.8
8192	220	84	206.18	10267.0	74.56	11.4
25600	1700	95	391.00	32750.8	133.02	14.5
100000	20000	122	776.15	170902.3	260.31	28.4

the results of these simulations. Note that the number of updates is an approximation of the simulated updates that would have been performed on a real tree to obtain the same number of subtrees. As a note of interest, the simulation of the 100,000 node tree required 179 hours of CPU time on a vax 11/780. The simulation of 8192 using actual trees for 600 million updates required more than 115 hours. It is estimated that more than 13,000 hours would be required to do a real simulation of 20 billion updates on a tree of size 100,000.

The following table compares the results of these simulations.

Table 3.10

Subtrees and Backbones from Simulations
Values Expressed as Ratios to $\sqrt{N}$

	Full Tree Simulations		Backbone Model	
Number of	Right		Right	
Nodes	Subtree	Backbone	Subtree	Backbone
512	2.29	0.81	2.22	0.86
1024	2.49	0.79	2.45	0.85
2500	2.37	0.82	2.31	0.85
4096	2.47	0.81	2.40	0.84
8192	2.51	0.80	2.28	0.82
25600	-	-	2.44	0.83
100000	-	-	2.45	0.82

As can be seen, the results are reasonably close to the earlier simulations and theory, but the subtrees are a bit smaller and the backbones are a bit longer in the backbone simulation. For 100,000, the difference in the length of the backbone and the predicted length assuming the $\sqrt{2/\pi}$ coefficient is about 8 nodes. The difference in the subtree size and the predicted value is about 16 nodes. The values are generally smaller for smaller trees. These differences seem small compared to the measured size of the subtrees, but the consistency of the difference would seem to indicate that the backbone simulation is not as accurate as we would like.

There are several possibilities for the difference. Perhaps there is some loss of accuracy due to truncation in the computation of the Beta variates and the resulting Binomial variates. However, double precision was used throughout, and furthermore it seems unlikely that there would be any discrepancy in the smaller trees such as 512.

A more realistic hypothesis is the following. If we consider for example two tags when they are first created, then there is no key between them, but the interval between them has an expected size of approximately $1/N$. We expect about one half of the keys to be deleted and replaced before the lower tag moves in the real tree. Assuming the lower tag moves before the upper, we then have that the expected number of keys in the interval is the expectation generated by $N/2$ random keys falling at random. But the backbone simulation assumes that all the non-backbone keys are falling at random, and thus over estimates the probability of there being a key in the interval. Similar arguments would seem to apply for all inter-tag intervals, the effect being greatest when the tags are close together. This means that the tags are slightly less likely to clump together and thus the backbone would be longer on the backbone model. Whether or not the two models converge is not clear.

3.5. Exact Fit Domain Simulations

In the discussion on trees on Exact Fit Domains in Chapter 2, it is shown that the expected interval size of the right subtree at the time it first becomes the right son of the root is approximately $\sqrt{\pi N}$. Simulations were run for EFD's for various values of N. The following tabulation shows the accuracy of our simulation methods.

Table 3.11

Average Right Subtrees on EFD's

Number of Nodes	Right Subtree	Variance	Backbone	Variance	Number of subtrees	Ratios to $\sqrt{N}$ RST	Ratios to $\sqrt{N}$ Backbone
256	28.07	205.4	17.49	3.6	256	1.755	1.093
512	40.07	375.3	24.86	4.7	269	1.771	1.099
1024	54.13	787.8	36.18	6.7	224	1.692	1.131
2500	88.32	1995.9	56.20	12.0	192	1.766	1.124
4096	115.02	4358.5	71.27	16.1	215	1.797	1.114

We see that the subtree sizes agree with $1.772\sqrt{N}$ as predicted and that the length of the backbone also agrees with the predicted $1.128\sqrt{N}$.

Table 3.12

Average Internal Path Lengths of EFD Trees

Tree Size	Number of Subtrees	Mean Internal Path Length	Variance
256	135	2408.2	44388
512	92	6186.4	181531
1024	108	17282.4	1210317
2500	88	60395.6	9617220
4096	115	124916.9	33556912

As for general trees, these simulations were also extended and data gathered on the Internal Path Length, again at the special times when a new right subtree of the root was created. In Table 3.12 the average IPL's are displayed. Using regression analysis, the coefficient of $N^{3/2}$ ranges from 0.422 to 0.478. The results found were $0.474N^{3/2} - 0.565N\log_2 N + 6.919N$, $0.422N^{3/2} + 0.285N\log_2 N$, and $0.478N^{3/2}$.

Chapter 4

In Search of Improved Algorithms

4.1. Introduction

In this chapter we seek deletion algorithms that result in trees with good average internal path length, without resorting to any explicit rebalancing schemes. Empirically Eppinger [7] and Culberson [5] found that alternating between left and right versions of the Hibbard or Knuth algorithms resulted in $O(N\log N)$ internal path length. The intuition behind this is discussed in section 4.5, but the result has not yet been proven.

One of the most intriguing aspects of the Hibbard algorithm is that despite its obvious asymmetry, if we do a sequence of random insertions and then do one or more random deletions, the distribution of tree *shapes* is the same as that generated by the appropriate number of random insertions alone. However, deletions do not preserve the distribution of trees generated by random insertions, and as a result insertions following the deletions destroy this randomness property of the shapes. Knott [14] was the first to observe this subtle change in the distribution. The distinction hinges on the idea that the random tree shapes and the random values remaining after a deletion are not independent.

In the following sections we consider the problem of finding a deletion algorithm that maintains the distribution of trees generated by random insertions. The probability distribution of the IPL for randomly built trees using leaf insertion is known (See Knuth [16] section 6.2.2), and in particular, the average IPL is $O(N\log N)$. Thus, knowing that a deletion algorithm leaves us with random trees implies good average case behavior. We also explore the possibility of using the root insertion algorithm of Stephenson [19] with some of our deletion algorithms.

We continue to assume that insertions are random and that deletions are equally probable over all keys in a tree. A deletion *scheme* consists of a pair of algorithms; one for insertion and one for deletion. A deletion scheme is *deletion insensitive* (DI) if the expected cost of accessing an item after a random sequence of deletions and insertions is the same as in a tree of the same size built by a random sequence of insertions. Otherwise the scheme is *deletion sensitive.* Knott's result shows that coupling Hibbard deletions with leaf insertions yields a deletion sensitive scheme. Knuth [17] discussed general properties of data structures that preserve randomness.

We wish to study Deletion Insensitive schemes, and in particular we are interested in those schemes that use the leaf insertion algorithm. In the following sections we will consider schemes in which the time requirements are restricted to $O(\log N)$ for any access on average.

4.2. Exact History Algorithms

One way of obtaining a deletion insensitive scheme is to design our deletion algorithm so that deleting an element leaves us with the same tree that would have been generated if the deleted element had never appeared in the insertion sequence. We now explore some of the problems and possibilities of such an algorithm.

To aid in the arguments that follow, some notation will be necessary. We designate by k_i the ith element of an insertion sequence, and by d_i the deletion of k_i from the tree. Thus, a sequence of 4 insertions followed by the deletion of the 2nd key inserted followed by a fifth insertion could be designated by $<k_1,k_2,k_3,k_4,d_2,k_5>$.

Suppose we build a tree T from an insertion sequence $\sigma = <k_1, \cdots k_n>$. We say that σ *generates* T. We then use an algorithm A to delete an element k_i from T. If A always generates the tree that would be generated by the sequence $\sigma - <k_i>$, we say that A is an *Exact History Deletion* (EHD) algorithm. Trivially any EHD algorithm is DI when coupled with the leaf insertion algorithm. Unfortunately, there can be no EHD algorithm unless we store information about the order of the insertions. This is because many trees have several possible generating sequences. For example, the sequences <2,1,3> and <2,3,1> each generate the same tree. Thus, when we store a sequence in a tree we may lose information about the order of the elements in the sequence.

If we store both the key and the index of the key in the sequence (the *sequence number),* then it is possible to design an efficient algorithm that is EHD. At each node of the tree we will have a tuple, (k,s), which consists of a key k and a sequence number s. It is trivial to prove that leaf insertion

acting in the usual way on the keys preserves the heap property on the sequence numbers; i.e. the sequence number at any node is less than any of its descendants.
If the nodes of a tree are tuples and the tree is so ordered that

(1) the search property is maintained on the first elements of the tuples and

(2) the heap property is maintained on the second elements of the tuples

we call the tree a *history tree.*
These are equivalent to the Cartesian Trees described in Vuillemin [20]. However, we wish to view deletions from the point of view of rotations and then extend these notions to the probabilistic algorithms of the following sections.

First, we prove the following lemma.

Lemma 4.1

A set of tuples with unique keys and sequence numbers determines a unique history tree.
Proof: By induction, on noting that the root of a tree is uniquely determined by the smallest sequence number. The basis for trees of size one is obvious. For the induction step, assume that it is true for trees of size $1, \cdots, n-1$. Then for trees of size n, uniquely determining the root determines the set of tuples in the left and right subtrees, but these are of size $<n$.

□

To show that a deletion algorithm is EHD, we need only show that given a history tree as input, performing the deletion produces a new history tree. We note that by the heap property on sequence numbers, the replacement for the doomed node must be either the left or right son. By doing the proper single rotation (see for example [10] pp 87), we can move the node with the smaller sequence number up, and the doomed node toward a leaf position. We continue to rotate the doomed node in this fashion, until one of the sons of the doomed node is empty, then simply replace the node with the remaining son. It is easily seen that this algorithm visits each of the nodes in the paths to the successor and the predecessor nodes at most once. Note that it merges those nodes into a single path.

Figure 4.1 is an example of an exact history tree. We wish to delete the node (7,1) using our EHD algorithm. We first rotate the root left since the

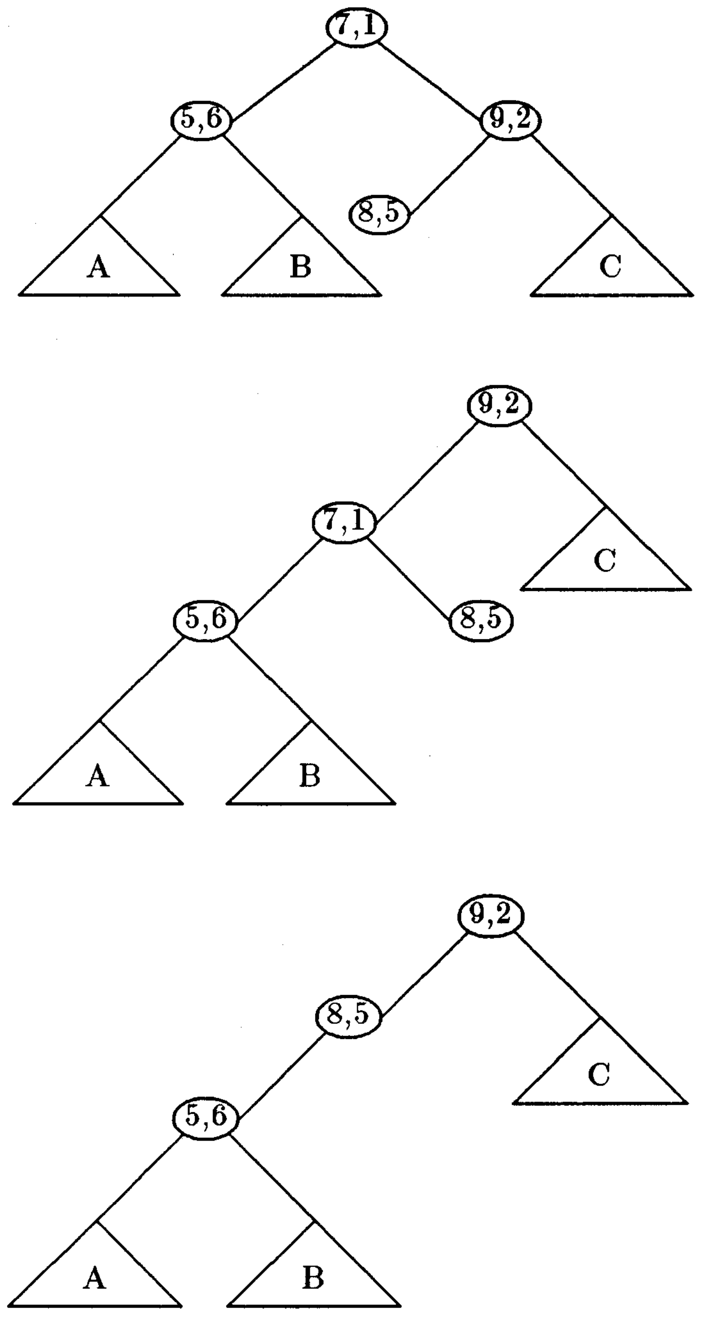


Figure 4.1 Exact History Deletion Using Rotations

sequence number '2' of the right son of the root is less than the '6' in the left son. To go from the second tree to the third, we do another rotate left, since now the '5' is less than '6'. However, the right son now becomes empty, so we can delete the node and use the left son as the replacement.

Stephenson [19] gives a method for inserting values into a binary search tree by inserting the new element at the root. The algorithm is best explained by an example.

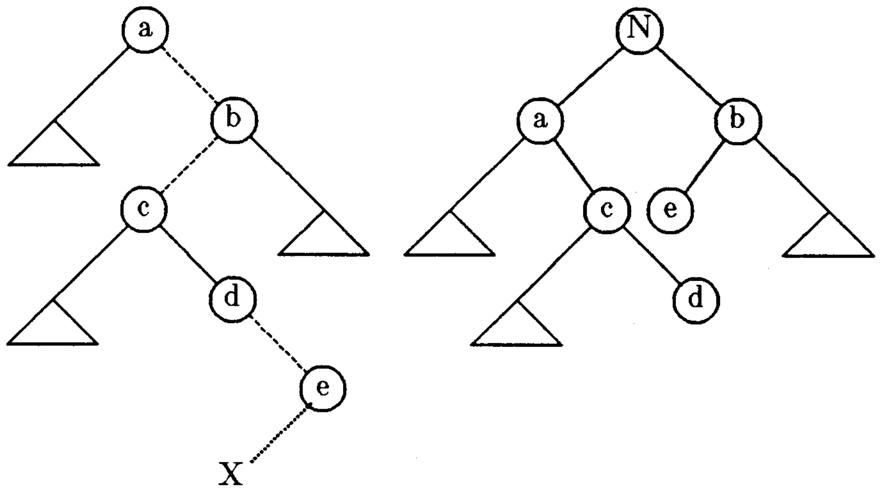


Figure 4.2 An Example of Root Insertion

We wish to insert a new value 'N' into the tree on the left of Figure 4.2, where the position of the new value would be at the external node 'X' if we used leaf insertion. The dashed lines indicate portions of the tree that are changed when we use Stephenson's root insertion algorithm. The tree on the right shows the result after the new node is inserted. The insertion at the root has the effect of splitting the tree on the dashed edges.

If we were inserting tuples using the root insertion, then it is easy to see that the heap property is again maintained, except now we have a max-heap; i.e. the maximum, or newest, sequence number is at the root. If we now apply the EHD algorithm described above (suitably modified to rotate up the maximum instead of the minimum) to delete 'N', it has the effect of 'zipping up' the paths to 'd' and 'e', the predecessor and successor of 'N', into the single path of the original tree. Stephenson [19] shows that the root insertion algorithm operating on any input sequence σ, generates the same tree as the leaf insertion algorithm operating on the sequence $\overline{\sigma}$, the reverse of σ. Thus, our EHD algorithm can be coupled with either insertion algorithm to give us a DI algorithm pair, with expected IPL of $O(N\log N)$. However, it does require the storage of sequence numbers.

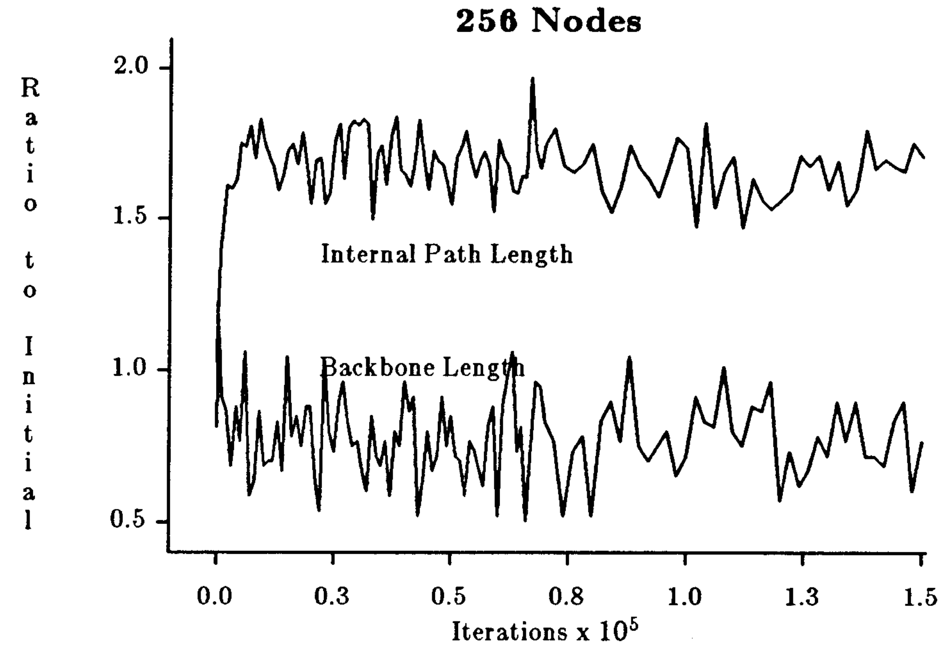
4.3. Possible History Deletion Algorithms

In this section we consider algorithms that do not require the storage of historical information. Note that the heap property of history trees implies that some information about the input sequence is implicit in the tree structure. In particular, if an insertion sequence $\sigma = <k_1, \cdots, k_n>$ would generate a given tree T, we then say that σ is a *possible history sequence* for T. In general, the set $\Psi(T)$ of possible history sequences for a given tree T will be a proper subset of the set of $n!$ possible orderings of the elements of T, whenever $n > 2$.

If a deletion algorithm deleting a key k_i from a tree T always produces a tree T', such that there is a sequence $\sigma' \in \Psi(T')$ with $\sigma' = \sigma - <k_i>$ and $\sigma \in \Psi(T)$, we say that the algorithm is a *Possible History Deletion* (PHD) algorithm. The idea behind this definition is that if we could design an algorithm that generates the left and right rotations with the proper probabilities, then we should have a DI algorithm. It is easy to see that the set of possible left and right rotation sequences for moving a node to a leaf position (or equivalently until one son is empty) and then deleting the node, results in the set of trees generated by the set $\Psi(T)$ with the corresponding element deleted. If there are a nodes in the path to the predecessor and b in the path to the successor, then there are $\binom{a+b}{a}$ possible rotation sequences.

Again, unfortunately, we suffer from the information loss due to many sequences mapping onto one tree. The proper left-right probability for any rotation depends on the relative sizes of the two subtrees. Unless we are willing to store this information, and keep it updated, or to count the nodes before each rotation, it does not appear that such proper probabilistic rotations are possible.

One obvious PHD algorithm is to choose between left and right rotations at random as they arise with equal probability. Figure 4.3 shows an example of the type of behavior of this algorithm. The IPL increases quickly, then levels off near 2.35 times its initial expected value for trees of size 1024. The ratio of final to initial path lengths appears to increase with tree size. At the same time, the length of the backbone decreases from its initial expected value. The trees remain left-right balanced, so it seems that somehow the lengths of the paths near the center portion of the tree must increase. We have no analysis of this algorithm.



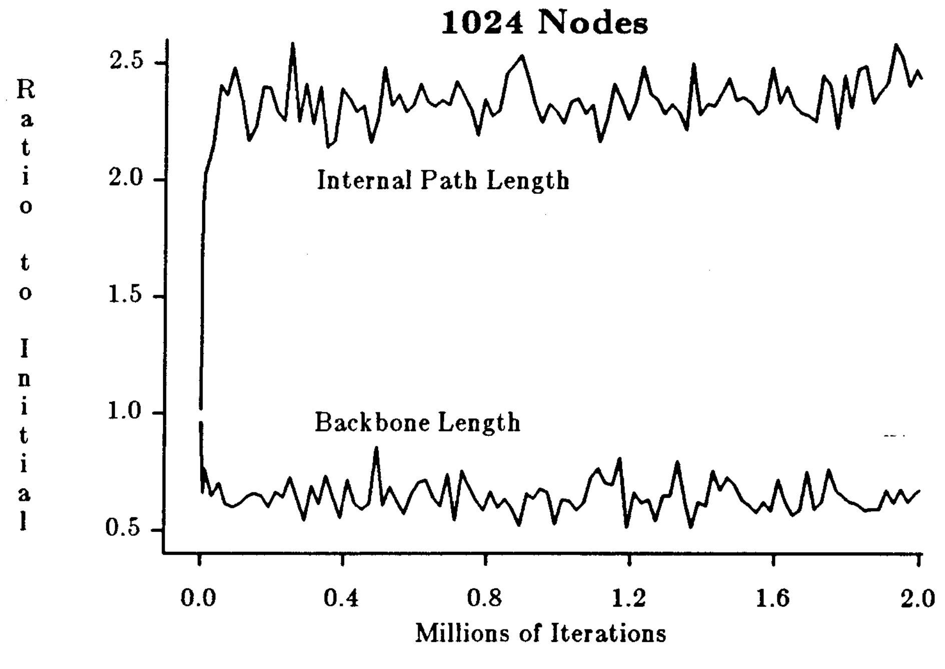


Figure 4.3 PHD Simulation Results

In Table 4.1, the ratio of the average final Internal Path Length to the expected initial value is given for various tree sizes. A good fit for these values is $0.237\log n - 0.14$. This implies that the IPL is $\Omega(n\log^2 n)$. The expected length of the backbone is the nth harmonic number. From the table it

Table 4.1

Simulations of the Possible History Deletion Algorithm Using Equally Probable Rotations.

Number of Nodes	Average Ratio of Final IPL/Initial	Average Measured Backbone
32	1.139	3.842
64	1.269	4.317
128	1.443	4.382
256	1.681	4.754
512	1.946	4.774
1024	2.345	4.831

appears that the backbone under the PHD scheme is growing much more slowly than for random trees, being almost constant over the range 256 to 1024.

The above approaches to finding a deletion insensitive algorithm are based on changing the tree during deletions to approximate the effect of deleting the element from the input sequence that generated the tree. It is interesting to note that the Hibbard algorithm preserves the shape distribution of trees under pure deletions, yet it is not a PHD algorithm. Whenever the successor exists and is not the right son of the deleted node, the Hibbard algorithm fails to generate a historically consistent tree.

4.4. Coupling Root Insertion with Hibbard Deletion

Another approach to deletion insensitivity is to couple one of our deletion algorithms with the root insertion algorithm mentioned above. By considering trees of size three, it is possible to show that coupling root insertion with Hibbard deletion does not preserve the random distribution precisely. Nevertheless, root insertion does have the property that insertions tend to reorganize the existing tree in a random fashion. For example, the distribution of the key of the root is random, depending only on the last insertion. Simulations have been run for trees of size 512 and 1024 nodes using this combination. No significant shift in either left-right balance, length of backbone or internal path length was detected. Thus, from an experimental viewpoint, this appears to be a deletion insensitive algorithm pair.

4.5. Symmetric Algorithms

The main result of this thesis is the claim that the Hibbard algorithm, and many other asymmetric algorithms similar to it, cause harmful deterioration in binary search trees when used for a long sequence of updates. Although this result is of benefit in that it warns against the use of asymmetric algorithms, it would be useful to have an algorithm that has guaranteed good average case behavior.

In the preceding sections, we searched for algorithms that were deletion insensitive, thus guaranteeing good average case behavior. Unfortunately, it seems that any such algorithm requires either extra memory or extra processing time or both. However, one positive indication from this search was that the use of a root insertion scheme did, at least empirically, preserve the expected IPL of random trees, even when coupled with the asymmetric Hibbard algorithm. An interesting open problem is to prove that this is indeed the case.

Eppinger [7] also simulated a symmetric version of the Hibbard algorithm, in which each deletion decides randomly whether to use the successor or the predecessor as a replacement, using the opposite son if the chosen replacement does not exist. In all the simulations that he ran, the IPL was reduced from that of the random tree, and did not show any signs of increasing again even after several times N^2 updates. Symmetric versions of various other deletion algorithms, including Knuth's [16] were tested empirically in [5] and in each case the IPL was reduced over that of random trees. However, to date there is no theory that proves that the trees do not eventually degenerate. Note that the PHD algorithm of section 4.3 was symmetric, randomly deciding between left and right rotations at each step. Yet, empirical evidence suggests that the IPL under that scheme increased by a factor of at least $O(\log N)$. Thus, symmetry is not a guarantee of good behavior.

The following is an intuitive argument that the symmetric Hibbard and Knuth algorithms do indeed yield good average case behavior. We will base the argument on the Hibbard algorithm, but the line of thinking clearly applies even more strongly to the symmetric Knuth algorithm. If we consider the movement of the root over the domain, then we see that it performs a symmetric random walk. Furthermore, the ends of the domain act as a type of reflecting barrier for this walk. For example, if the right subtree is empty and the algorithm decides to choose the successor as replacement, it finds that the successor does not exist, so the left son becomes the new root. Since the left son may have (probably will have) a large right subtree, the root of the tree has effectively jumped back to a random point on the domain.

If every step of the random walk performed by the root had the same probability distribution, then we should expect that it would appear in any subinterval $x,x+\Delta x$ with equal probability. For example, with a simple discrete random walk, with simple reflecting barriers, the probability of being at each point is equivalent if the probability distribution is symmetric. (See Cox and Miller [4] Chapter 2.2(v)) These distributions correspond to the uniform distribution of the root of a random tree. If the reflecting barriers are changed so that on reflection the particle jumps to some random point on the interval instead of to an adjacent point (in the discrete case) then a simple application of the theory of Markov processes shows that the probability of being near the end points of the domain is reduced, increasing the probability of being near the center. (The discrete case applies directly to EFD trees and thus proves that the root of an EFD tree is at least as likely to be centered after some number of updates as is the root of a random EFD tree). In general then, we expect that the root is more likely to divide the tree in nearly equal portions after a large number of updates have been made, improving the balance of the tree at the root.

It seems reasonable that the argument should apply to the roots of each of the subtrees; however, in these cases one or more of the 'barriers' are ancestors of the root. Thus, these barriers are themselves moving at random. The interdependencies now become very complex. However, one further observation can be made. If indeed the root of a subtree is more likely to be centered on the sub-interval, even when the parent of the root is at the end of its interval, then deletion of the parent moves that root nearer to the center of its interval. Thus, if the conjecture that the subtrees are more likely to be balanced is true, there should be a positive feedback upwards to the root of the tree increasing even further our conjectures on the amount of improvement at the root.

Admittedly, the above arguments are imprecise and incomplete, perhaps to the point of wishful thinking. The point in presenting them is that they might suggest to some reader a method of analyzing these symmetric algorithms. An analysis showing that the average IPL of a tree is reduced when subjected to updates using a symmetric algorithm would be of great theoretical and even practical interest, in large part because it would be a positive result.

References

1. Baeza-Yates, Ricardo A., *Analysis of Algorithms in Search Trees*, Master's Thesis, Universidad de Chile, Santiago, Chile, Jan. 1985.
2. Booth, A. D. and Colin, A. J. T., On the Efficiency of a New Method of Dictionary Construction, *Information and Control 3*, (1960), 327-334.
3. Cheng, R. C. A., Generating Beta Variates with Non-integral Shape Parameters, *Comm. ACM 21(4)*, (1978), 317-322.
4. Cox, D. R. and Miller, H. D., *The Theory of Stochastic Processes*, Chapman and Hall, London, 1978.
5. Culberson, Joseph C., Updating Binary Trees, Technical Report CS-84-08, University of Waterloo, Waterloo, Canada, March 1984.
6. Culberson, Joseph, The Effect of Updates in Binary Search Trees, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, Providence R.I., 1985.
7. Eppinger, Jeffery L., An Empirical Study of Insertion and Deletion in Binary Trees, *Comm. ACM 26*, (Sept. 1983), .
8. Feller, William, *An Introduction to Probability Theory and Its Applications Volume I*, John Wiley & Sons, New York, 1968.
9. Feller, William, *An Introduction to Probability Theory and Its Applications Volume II*, John Wiley & Sons, New York, 1971.
10. Gonnet, G. H., Handbook of Algorithms and Data Structures, *International Computer Science Series*, 1984.
11. Hibbard, Thomas N., Some Combinatorial Properties of Certain Trees with Applications to Searching and Sorting, *Journal of Association of Computing Machinery 9(1)*, (Jan. 1962), 13-28.
12. Jonassen, Arne T. and Knuth, Donald E., A Trivial Algorithm Whose Analysis Isn't, *Journal of Computer and System Sciences 16*, (1978), 301-322.
13. Jow, Richard Leon, *A Study of Lattice-Generated Integer Sequences*, Master's Thesis, California State University, San Jose, 1972.
14. Knott, Gary D., Deletion in Binary Storage Trees, *Ph.D thesis STAN-CS-75-491*, (May 1975), , Stanford University.

15. Knuth, Donald E., Seminumerical Algorithms, *The Art of Computer Programming Vol II*, 1969.

16. Knuth, Donald E., Searching and Sorting, *The Art of Computer Programming Vol III*, 1973.

17. Knuth, Donald E, Deletions That Preserve Randomness, *IEEE Transactions on Software Engineering SE-3*, (Sept. 1977), 351-359.

18. Mendenhall, William and Scheaffer, Richard L., *Mathematical Statistics with Applications*, Duxbury Press, North Scituate, Massachusetts, 1973.

19. Stephenson, C. J., A Method for Constructing Binary Search Trees by Making Insertions at the Root, *International Journal of Computer and Information Sciences 9*, (1980), 15-29.

20. Vuillemin, Jean, A Unifying Look at Data Structures, *Communications of the ACM 23, No. 4*, (April 1980), 229-239.

21. Windley, P. F., Trees, Forests and Rearranging, *The Computer Journal 3*, (July,1960), 84-88.