Average Case Analysis of Marking Algorithms

D.S. Hirschberg[†] and *L.L. Larmore*[‡]

Technical Report #223

March, 1984

Abstract. The Lindstrom marking algorithm uses bounded workspace. Its time complexity is $O(n^2)$ in all cases, but it has been assumed that the average case time complexity is $O(n \lg n)$. It is proven that the average case time complexity is $H(n^2)$. Similarly, the average size of the Wegbreit bit stack is shown to be $H(n)$.

This research was supported in part by National Science Foundation Grant MCS-82-00362, and by a California State University PAID grant.

[†]Department of Information and Computer Science, University of California, Irvine, CA 92717.
[‡]Department of Computer Science, California State University, Dominguez Hills, CA 94707.

Average Case Analysis of Marking Algorithms

D.S. Hirschberg[†] *and L.L. Larmore*[‡]
University of California, Irvine
California State University, Dominguez Hills

1. Introduction

Consider a data store organized into nodes, each of which has two link fields, L and R. Each link field contains either the address of a node in the store, or an uninterpreted atomic value. One specific node will be designated as the root.

In garbage collection, we may wish to mark all nodes accessible from the root; assume then that each node has a mark bit. Several algorithms to do this are known. Baer and Fries [1] analyze some of these, including the Schorr-Waite tag bit algorithm [4], the Wegbreit bit-stack algorithm [5] and the Lindstrom algorithm [3].

Both the Schorr-Waite and Wegbreit algorithms have time complexity $O(n)$ where n is the number of nodes accessible from the root, and both use $O(n)$ additional workspace. The Schorr-Waite algorithm requires a tag bit in each node whereas the Wegbreit algorithm uses a bit stack, whose size is $O(n)$ in the worst case, for the same purpose. At first glance, it may appear that the maximum size of the Wegbreit bit stack is $O(\lg n)$ in the average case, but we show that it is $\Theta(n)$.

The Lindstrom algorithm uses bounded workspace; in particular, it uses no stack or tag bits. Its time complexity is $O(n^2)$ in all cases, but Lindstrom asserted [3] that the average case time complexity is $O(n \lg n)$. We show that this assertion is false. In

This research was supported in part by National Science Foundation Grant MCS-82-00362, and by a California State University PAID grant.

[†]Department of Information and Computer Science, University of California, Irvine, CA 92717.
[‡]Department of Computer Science, California State University, Dominguez Hills, CA 94707.

particular, we prove that the average case time complexity is $\Omega(n^2)$.

2. The Lindstrom Algorithm

We refer the reader to [3] for a description of the algorithm. It is important to note that λ, the nil address, is *not* an atom. Essentially, descent is always to the left, providing the left link leads to a previously unvisited node. Otherwise, descent is to the right. If no descent is possible because both links are either atomic or lead to previously visited nodes, ascent is initiated and continues until a node with an unvisited right child is found.

At all times, all fully-processed nodes (all of whose descendents have been visited) have a marked mark field (value 1), and all unvisited nodes have an unmarked mark field (value 0). All other nodes must be on the trace path, the path of nodes from the root to the current node. For a node on the trace path, the mark field is 0 if both links are non-atomic and descent from that node was to the right. Otherwise, it is 1.

The algorithm uses link reversal so it is necessary, during ascent, to determine whether that ascent is from the left or the right. If one link field is atomic, there is no problem. Otherwise, the mark bit contains the information.

During descent, if a node whose mark bit is 0 (unmarked) and both of whose link fields are non-atomic is encountered, there is no immediate way to determine whether the node is unvisited or is on the trace path. The algorithm must search for this node upward along the trace path. If the node is unvisited, the time required for this step is clearly $\Omega(k)$, where k is the length of that trace path. Lindstrom's error was his supposition that the length of the trace path is $O(\lg n)$ in the average case. Under that hypothesis, the upward searching portion of the algorithm could be done in $O(n \lg n)$

time, and other parts of the algorithm are clearly $O(n)$. It may actually be true that the depth of an average node is $O(\lg n)$, measured along the shortest possible rooted path. But the Lindstrom algorithm does not visit each node first by the shortest path, rather by the leftmost path which may be far longer. In fact, by Theorem 2 of this paper, the average case length of that path is $\Theta(n)$.

3. The Trace Tree

We use the term ***list structure*** to refer to the collection of all nodes accessible from the root, together with their pointers. The Schorr-Waite, Wegbreit, and Lindstrom marking algorithms visit the nodes of a list structure, X, in the same order (counting only first visits). We call this the ***visitation order***.

We denote the ***trace tree*** of list structure X by $\mathcal{T}(X)$, or by just $\mathcal{T}$ when X is understood. $\mathcal{T}$ is a binary tree which has exactly the same nodes as does X, but only a subset of the links. For all nodes in X, the father of node x in $\mathcal{T}$ is the node in X from which x was first visited in the visitation order. All other links of $\mathcal{T}$ are λ. As a result of this definition, the preorder of $\mathcal{T}$ will be the visitation order of X, and a path in $\mathcal{T}$ from the root to a given node P will be exactly the trace path when that node is visited for the first time by one of the marking algorithms.

The visitation order $P_1, P_2, \ldots P_n$ on X can be characterized by the following four rules, which also characterize the trace tree. L(P) and R(P) denote the left and right link fields of node P in X. Similarly, $L_{\mathcal{T}}(P)$ and $R_{\mathcal{T}}(P)$ denote link fields of P in $\mathcal{T}$.

(i) $P_1 = root$

(ii) For $1 \leq k < n$, if $L(P_k)$ is not an atom and is not in the set $\{P_1, \ldots P_k\}$ then $P_{k+1} = L(P_k) = L_{\mathcal{T}}(P_k)$.

(iii) For $1 \leq k < n$, if $L(P_k)$ is an atom or if $L(P_k) \in \{P_1, \ldots P_k\}$, let $1 \leq j \leq k$ be the largest value such that $R(P_j)$ is not atom and not in the set

$\{P_1, \ldots P_k\}$. Then $P_{k+1} = R(P_j) = R_{\mathcal{T}}(P_j)$.

(iv) For $1 \leq k \leq n$, $L_{\mathcal{T}}(P_k) = \lambda$ unless otherwise specified by rule (ii), and $R_{\mathcal{T}}(P_k) = \lambda$ unless otherwise specified by rule (iii).

4. Time Analysis

For any node P, let $depth_{\mathcal{T}}(P)$ be the depth of P in $\mathcal{T}$. The time required for the upward searching portion of the Lindstrom algorithm is clearly $\Theta(\Sigma_{P \in X} depth_{\mathcal{T}}(P))$.

Let *height* be the height of $\mathcal{T}$. We can put a lower bound on the time complexity of the Lindstrom algorithm as follows.

Theorem 1. The time complexity of the Lindstrom algorithm is $\Omega(n \log n + (height)^2)$.

Proof. Since the average depth of a node must be $\Omega(\log n)$, the number of upward searches must be $\Omega(n \log n)$. On the other hand, let $Q_0, Q_1, \ldots Q_{height}$ be a longest rooted path in $\mathcal{T}$, where Q_0 is *root*. Then

$$\Sigma_{P \in X} depth_{\mathcal{T}}(P) \geq \Sigma_{0 \leq i \leq height} depth_{\mathcal{T}}(Q_i) = \Sigma_{0 \leq i \leq height} i = height(height+1)/2$$

□

5. Average Case Analysis

In order to do an average case analysis, it is necessary to have some understanding of what an "average case" is. We shall assume that we are given some distribution on the class of *all* list structures of size *n*, and that when we say "average" we shall mean average weighted by this distribution. We will freely use words such as ***probability***, ***conditional probability*** and ***expected value*** in their usual meanings, where we assume that a list structure has been selected at random from all list structures, using the given distribution.

Let *probatom*(X) be the proportion of address fields in a list structure X which are atoms. If there are no atoms, we say that X is non-atomic.

Fix n, and fix $0 \leq p < 1/2$. What kind of properties would we want a "random" distribution of list structures whose size is n and whose *probatom* is approximately p to have? Intuitively, we would want to describe these properties in terms of what value we expect a "new" (i.e., previously unexamined) link to have, given whatever information we have about previously examined links. In an ideal sense, we would want each new link to be an atom with probability p, and, given that it is not an atom, we would want the probability that it points to any given node in the structure to be $1/n$. It turns out that such a strong randomness property is unnecessary. We will just assume that our distribution satisfies a weaker set of conditions, which we call Weak Randomness.

Weak Randomness Assumption: There exist constants $0 \leq \alpha < 1/2$ and $\beta > 0$ (not dependent on n) such that, if P is a new link and k is the number of nodes already visited:

(i) The probability P is an atom does not exceed α.

and (ii) The probability that P is an old node does not exceed $\beta k/n$.

Intuitively we mean that, although we may allow knowledge of previously examined links to influence the expectation of P, we place a definite bound on that effect. In particular, the probability that a given link is an atom is bounded below $1/2$, and the probability that it points to an old node is of the order of the proportion of old nodes in the structure.

Theorem 2. In the average case, ***height*** $= \Theta(n)$.

Proof. We present an algorithm for visiting all nodes of X in visitation order (i.e., preorder for $\mathcal{T}$). Let S be a stack.

```
Step 1:    $S \leftarrow \emptyset$
Step 2:    $P \leftarrow root$
Step 3:    While S is non-empty or P is a new node, do steps 4 and 5
Step 4:        If P is an atom or an old node,
                   $P \leftarrow$ top of stack S;  pop S
Step 5:        Otherwise (i.e., P is a new node),
                   push R(P) onto stack S;  $P \leftarrow L(P)$
Step 6:    Halt
```

Let us now examine what the situation is after t iterations of the loop in the above algorithm. Stack S will have some size, say $size(t)$, P will be some node, say $node(t)$, and P will be at depth $depth_{\mathcal{T}}(P)$ in the trace tree. But note that each element on stack S is the right child of some node in the trace path of P. It follows that $depth_{\mathcal{T}}(node(t)) \geq size(t)$.

We think of $size$ as a stochastic function. It is clear that the number of old nodes after t iterations of the loop of the above algorithm is $\leq t$, and that

$$size(t+1) = \begin{cases} size(t) + 1 \text{ if } node(t) \text{ is a new node} \\ size(t) - 1 \text{ otherwise} \end{cases}$$

By the Weak Randomness Assumption, the probability that $node(t)$ is a new node is at least $1 - \alpha - \beta t/n$, and the probability that $node(t)$ is not a new node is at most $\alpha + \beta t/n$. Therefore (see, for example, [2])

$$\begin{aligned} E(t) &= \text{the expected value of } size(t) \\ &= E(t-1) + \text{Prob}(node(t) \text{ is a new node}) - \text{Prob}(node(t) \text{ is not a new node}) \end{aligned}$$

$$\geq E(t-1) + (1 - \alpha - \beta t/n) - (\alpha + \beta t/n)$$

Since *size*(0) = 0, we therefore have

$$E(t) \geq \Sigma_{1 \leq i \leq t} (1 - 2\alpha - 2\beta i/n) = t(1 - 2\alpha) - \beta t(t+1)/n$$

Now choose $t = \lfloor n(1-2\alpha)/(2\beta) \rfloor$. It is seen that the expected value of *size*(*t*), and hence the expected value of $depth_T(node(t))$, must be at least $n(1-2\alpha)^2/4\beta - 1$. $\square$

Corollary. The size of the Wegbreit bit-stack is $\Theta(n)$ in the average case.

Theorem 3. The average case time complexity of the Lindstrom algorithm is $\Theta(n^2)$.

Proof. It is already known [3] that the time is $O(n^2)$, and it is $\Omega(n^2)$ by Theorems 1 and 2. $\square$

6. An Open Question

The Schorr-Waite algorithm has both time and space complexity $\Theta(n)$. The Wegbreit algorithm has time complexity $\Theta(n)$ and average case space complexity $\Theta(n)$. The Lindstrom algorithm has bounded space complexity and average case time complexity $\Theta(n^2)$.

A natural question to ask is whether there exists any marking algorithm, the product of whose average time and space complexities is less than quadratic.

References

[1] J.-L. Baer and M. Fries, *On the Efficiency of Some List Marking Algorithms*, Information Processing 77, IFIP, North-Holland Publishing Co., 1977, pp. 751-756.

[2] P. G. Hoel, S. C. Port, and C. J. Stone, *Introduction to Stochastic Processes*, Houghton Mifflin, 1972.

[3] G. Lindstrom, *Copying List Structures in Bounded Workspace*, CACM 17 (1974), pp. 198-202.

[4] H. Schorr and W. M. Waite, *An Efficient Machine Independent Procedure for Garbage Collection in Various List Structures*, CACM 10 (1967), pp. 501-506.

[5] B. A. Wegbreit, *A Space Efficient List Structure Tracing Algorithm*, IEEETC, C-21 (1972), pp. 1009-1010.