MIT/LCS/TM-247

PROBABILISTIC SEARCHING IN SORTED LINKED LISTS

Tom Leighton

Margaret Lepley

November 1983

Probabilistic Searching in Sorted Linked Lists

Tom Leighton & Margaret Lepley

Department of Mathematics & Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract: Janko [2] and Bentley, Stanat, and Steele [1] have described probabilistic procedures for data manipulation in sorted linked lists. Their procedures are based on an algorithm which performs a Member search operation using $2N^{1/2} + O(1)$ expected steps where N is the number of elements in the list. In addition, Bentley, Stanat and Steele have shown that every Member search algorithm requires $(2N)^{1/2} + \Omega(1)$ expected steps. In this paper, we improve the lower bound result in order to prove that the known algorithm for Member search is optimal.

Keywords: data structure, probabilistic algorithm, searching, sorted linked list

---

Tom Leighton was supported in part by a Bantrell Fellowship and Margaret Lepley was supported in part by an MIT Applied Math Fellowship. A preliminary version of this paper was presented at the 20th Annual Allerton Conference on Communication, Control and Computing.

1. Introduction

In a sorted linked list, the operations Predecessor, Successor, Insert, Delete, and Last element can all be performed using an operator which searches for a specific element in the list. We call this operation *Member search*. Since traversal of an N-element linked list requires N steps, it is not difficult to see that any *deterministic* algorithm for Member search requires N steps in the worst case. By considering *probabilistic* procedures, however, Janko [2] showed that the *expected* number of steps necessary to perform a Member search operation is substantially less than that required in the worst case.

In [1], Bentley, Stanat and Steele formalize Janko's arguments and describe a probabilistic algorithm for Member search which requires only $2N^{1/2} + O(1)$ expected steps. As the existence of a better procedure seemed unlikely, Bentley, Stanat and Steele conjectured that their algorithm for Member search was optimal. In support of this conjecture, they showed that *every* probabilistic Member search algorithm requires $(2N)^{1/2} + \Omega(1)$ expected steps. In this paper, we improve the lower bound in order to prove that the Bentley-Stanat-Steele algorithm for Member search is optimal.

The remainder of the paper is divided into three sections. In Section 2, we describe the data structure and probabilistic algorithms in greater detail. In Section 3, we prove the optimality of the Bentley-Stanat-Steele algorithm. We conclude with some remarks and related results in Section 4.

2. Preliminaries

2a) Data structure

The data structure used to represent an N-element sorted linked list consists of two arrays, Link[$0...N$] and Value[$1...N$], where Link[0] points to the first element of the list, and Link[J] points to the element which follows Value[J]. The end of the list is denoted by an element whose Link field contains -1. Since the array is required to be dense, every cell in Value[$1...N$] must contain an element of the list. Since the list is sorted, we have

$$\text{Value}[J] \leq \text{Value}[\text{Link}[J]]$$

when Link[J] is not -1. The following figure illustrates the array representation of the sorted linked list $\langle 2.1, 3.5, 6.2, 7.4, 7.9, 9.6\rangle$.

J	0	1	2	3	4	5	6
Value[J]		3.5	6.2	9.6	2.1	7.4	7.9
Link[J]	4	2	5	−1	1	6	3

2b) Probabilistic algorithms

An element of the list is accessed via an index key which can be chosen in several ways. The index may be given as the link from the predecessor, a predetermined integer, or a randomly selected integer. Access via a predetermined integer is of use only when access to the beginning of the list is needed. By combining probabilistic access with access by

predecessor link, however, a wide range of algorithms can be considered. For example, the following pseudo-Pascal program searches for the element E, using the order of operations specified by the array Step. When Step[J] is zero, a random sample occurs. Otherwise the program follows the next link in the list. Note that when a random sample is chosen, the position in the list is updated only if the random position is closer to (but not at or beyond) the location of E. This strategy insures that the updated positions in the list never worsen and that when E is eventually found, its predecessor will also have been found (since E will have been reached via a link). (This particular code assumes that Value[0] is $-\infty$ and Value[-1] is ∞.)

```
P: = 0
J: = 0
do until exit
    J: = J + 1
    if Step[J] = 0 then do
                        R: = Random(1,N)
                        if Value[R]<E and Value[R]>Value[P] then P: = R
                    else do
                        if Value[Link[P]] = E then exit (*E is at Link[P]*)
                        if Value[Link[P]]>E then exit (*E is not in the list*)
                        if Value[Link[P]]<E then P: = Link[P]
```

It is easy to modify the preceding algorithm for Member search to perform Predecessor, Successor and Last element. In addition, the algorithm can be modified to perform Insert and Delete. For example, if E does not appear in the list, then the search will end when the position it ought to occupy in the list has been found. The element can then be inserted by placing it at index $N+1$ and changing the links. A Delete is more complex. First the element to be deleted is found and removed from the list. There is now an index in the array which contains no element of the list. To remedy this situation the element at index N is moved into the open spot. This of course requires another search operation to find the Predecessor of the element at index N. Therefore a Delete requires two Member searches if the arrays are to remain dense.

2c) Mathematical model

The lower bound for finding the largest element is also a general lower bound since a random element might be maximum in the list. So in studying the random algorithm presented above, we will assume that we are searching for the largest element in the list. The program is then a black box which must be told the order of random and deterministic operations. This behavior can be modelled by the following probabilistic game due to Bentley, Stanat, and Steele [1].

Definition: The *probabilistic game G-D* involves two integers, i and N. The value of N remains fixed throughout the game, and the value of i is originally N. The goal of the player is to reduce the value of i to zero in the minimum expected number of steps. A step consists of performing one of the following two operations –

*D(for Decrement): If $i>0$, then replace i by $i-1$.

*G(for Guess): Choose j to be an integer uniform from $1...N$ and replace i by j if $j<i$. The value of i is unknown to the player, except at the beginning of the game when $i=N$, and at the end when he is notified that i has reached zero. (In this game the value i represents the distance from the current element in the list – denoted by P in the code above – to the end of the list.)

A strategy or sequence of operations will be denoted by a character string σ, composed of G's and D's, to be performed in order from left to right. A sequence is said to be *complete* if it contains at least N D's. Note that operations written after the first N D's are superfluous and need not appear. For convenience, however, we will often end complete sequences with D^N.

A complete sequence will always reach $i=0$ and terminate the game after some number of steps t. The expected termination point for a complete sequence σ is denoted by

$$E(\sigma) = \sum_{j=1}^{\infty} j \Pr[t=j] = \sum_{j=0}^{\infty} Q_j(\sigma)$$

where $Q_j(\sigma) = \Pr[t>j]$. The object is to minimize $E(\sigma)$ over all complete sequences σ. We denote the minimum by $S(N)$.

2d) Previous results

Janko first investigated the effects of random sampling in a sorted linked list. He restricted his analysis to the class of strategies of the form G^kD^N, for some k. Within this restricted class he discovered the optimal value for k.

Theorem 1: *For any integer* $k\geq 0$, $E(G^rD^N) \leq E(G^kD^N)$ *where* $r = N^{1/2} - 1 - 1/(24N^{1/2}) + O(1/N)$.

Proof: From the definition,

$$E(G^kD^N) = \sum_{j=0}^{k+N} Q_j(G^kD^N) .$$

Since the first k operations are Guesses the game cannot end there, so $Q_j(G^kD^N) = 1$ for $j = 0,...,k$. The probability of not terminating during the first d Decrements is $(N-d)^kN^{-k}$, since all the Guesses must be larger than d. Therefore

$$\begin{aligned} E(G^kD^N) &= k + 1 + \sum_{d=1}^{N-1}(N-d)^kN^{-k} \\ &= k+1 + N^{-k}\sum_{d=1}^{N-1} d^k \\ &= k+1 + N/(k+1) - 1/2 + k/(12N) + O(k^2/N^2) \end{aligned}$$

The minimum occurs when

$$1 - N/(k+1)^2 + 1/(12N) + O(k/N^2) = 0$$

and thus when

$$k = N^{1/2} - 1 - 1/(24N^{1/2}) + O(1/N). \quad \square$$

Theorem 1 provides an upper bound of $S(N) \leq 2N^{1/2} - 1/2 + 1/(12N^{1/2}) + O(1/N)$

expected operations for the G-D game. Recently, Bentley, Stanat, and Steele found a lower bound for the game. In particular, they showed that if the player was allowed to know the value of i at all times, then the optimal strategy requires $\sim(2N)^{1/2}$ expected steps. Hence, $S(N) = \Theta(N^{1/2})$ where the constant is between $2^{1/2}$ and 2. We will now show that the lower bound for the general G-D game can be improved to $\sim 2N^{1/2}$, which is tight.

3. General Lower Bound

When a sequence ω is not complete, the value of i after the operations in ω have been performed may remain undetermined. Instead of knowing the exact value of i we define a probability vector $P(\omega)$ so that

$$P_j(\omega) = \Pr[i>j \text{ after executing the sequence } \omega] .$$

We can see from the definition that $P_j(\omega) \geq P_{j+1}(\omega)$. Moreover the vector can be computed for any sequence ω.

Lemma 1: *For any sequence* $\omega = G^{a_1}D^{b_1}\ldots G^{a_s}D^{b_s}$, *with* $b = b_1 + \ldots + b_s$ *and* $a = a_1 + \ldots + a_s$

$$P_j(\omega) = \begin{cases} (N-j-b)^{a_1}(N-j-b+b_1)^{a_2}\ldots(N-j-b_s)^{a_s}N^{-a} & \text{for } j < N-b \\ 0 & \text{for } j \geq N-b \end{cases}$$

Proof: During each block of Guesses, G^{a_m}, the value i must remain above j plus the number of D's which are still to be performed, $b_m + \ldots + b_s$. The probability that all the Guesses in the block are between $j + b_m + \ldots + b_s + 1$ and N is $(N - j - b_m - \ldots - b_s)N^{-a_m}$. □

These probabilities are important in determining the optimal strategy. The following lemma states one of the most useful properties of this vector.

Lemma 2: *For any sequence* ω, $P_j(\omega D)/P_0(\omega D) \leq P_j(\omega)/P_0(\omega)$ *for* $0 \leq j \leq N$.

Proof: Each ratio is a product of terms of the form

$$[(N-j-b_i-\ldots-b_s)/(N-b_i-\ldots-b_s)]^{a_i}.$$

The sequence ωD has one more D in the last block than ω, so b_s increases by one in ωD. When $P_0(\omega D) > 0$ a comparison of these terms, letting $K = N - b_i - \ldots - b_s$, reveals that

$$[(K-j-1)/(K-1)]^{a_i} < [(K-j)/K]^{a_i}$$

and thus that $P_j(\omega D)/P_0(\omega D) \leq P_j(\omega)/P_0(\omega)$. If $P_0(\omega D) = 0$, we define the ratio to be zero and the inequality still holds. □

Our goal in this section is to prove that the optimal strategy has the form G^rD^N. First we analyze the effect of minor variations in strategy on the expected number of operations. Then we will show that if a small variation improves the strategy then a larger change could mean even more improvement. The two sequences which we will compare first are $\sigma = \omega D G^k D^N$ and $\sigma^* = \omega G^k D^N$. The only difference between σ and σ^* is the position of the block G^k. The following lemma gives a method for comparing these two strings.

Lemma 3: $E(\omega DG^kD^N) \leq E(\omega G^kD^N)$ *if and only if* $V_k(\omega) \leq 1$, *where*

$$V_k(\omega) = \begin{cases} P_1(\omega)/P_0(\omega) + \sum_{d=1}^{N-1} P_d(\omega)/P_0(\omega)\left[(N-d+1)^k - (N-d)^k\right]k^{-1}N^{-k} & \text{if } P_0(\omega)>0 \\ 0 & \text{if } P_0(\omega)=0 \end{cases}$$

Proof: Let σ and σ^* be defined as above. We would like to know when $E(\sigma) - E(\sigma^*) \leq 0$. The following values for $Q_m(\sigma)$ and $Q_m(\sigma^*)$ can be easily verified.

$$Q_m(\sigma) = \begin{cases} Q_m(\sigma^*) & \text{if } m \leq |\omega| \\ P_1(\omega) & \text{if } |\omega|+1 \leq m \leq |\omega|+k+1 \\ P_{m-|w|-k}(\omega)(N-m+|\omega|+k+1)^kN^{-k} & \text{if } m > |\omega|+k+1 \end{cases}$$

$$Q_m(\sigma^*) = \begin{cases} Q_m(\sigma) & \text{if } m \leq |\omega| \\ P_0(\omega) & \text{if } |\omega|+1 \leq m \leq |\omega|+k \\ P_{m-|w|-k}(\omega)(N-m+|\omega|+k)^kN^{-k} & \text{if } m > |\omega|+k \end{cases}$$

Thus $E(\sigma) - E(\sigma^*) = \sum_{m=0}^{\infty}[Q_m(\sigma) - Q_m(\sigma^*)] \leq 0$ is equivalent to

$$(k+1)P_1(\omega) - kP_0(\omega) + \sum_{d=1}^{N-1} P_d(\omega)[(N-d+1)^k - (N-d)^k]N^{-k} - P_1(\omega) \leq 0.$$

Rearranging terms slightly gives

$$P_1(\omega) + \sum_{d=1}^{N-1} P_d(\omega)[(N-d+1)^k - (N-d)^k]N^{-k}k^{-1} \leq P_0(\omega). \quad \square$$

Combining Lemmas 2 and 3 enables us to extend a minor variation of the string into a more radical change. Specifically, if the last block of G's is more efficient when it is moved to the right one place, then it is best to remove the block of Guesses altogether.

Lemma 4: *For all* ω, *if* $E(\omega DG^kD^N) \leq E(\omega G^kD^N)$, *then* $E(\omega D^N) \leq E(\omega DG^kD^N)$.

Proof: If $E(\omega D^jG^kD^N) \leq E(\omega D^{j-1}G^kD^N)$ for some $j \geq 1$, then $V_k(\omega D^{j-1}) \leq 1$ by Lemma 3. By Lemma 2 and the definition of $V_k(\omega)$, we know that $V_k(\omega D^j) \leq V_k(\omega D^{j-1})$, and thus that $V_k(\omega D^j) \leq 1$. Thus, we can conclude by Lemma 3 that $E(\omega D^{j+1}G^kD^N) \leq E(\omega D^jG^kD^N)$. The proof of the lemma is completed by applying this process inductively. $\square$

It is now a simple matter to prove our main result.

Theorem 2: *For every starting sequence* ω, *there exists an integer* $r \geq 0$ *such that* $E(\omega G^rD^N) \leq E(\omega\sigma)$ *for every completed sequence* $\omega\sigma$.

Proof: Let σ denote the shortest (in length) sequence for which $\omega\sigma$ is complete and $E(\omega\sigma) = \min_\gamma E(\omega\gamma)$. If σ is of the form G^rD^N, then we are done. Otherwise $\sigma = \omega^*DG^kD^j$, where G^k is the last block of Guesses and $j>0$. By the optimality of σ, we know that $E(\omega\omega^*DG^kD^j) \leq E(\omega\omega^*G^kD^{j+1})$. Thus by Lemma 4, we know that $E(\omega\omega^*D^{j+1}) \leq E(\omega\omega^*DG^kD^j)$ which contradicts the minimality of σ. $\square$

Thus the best way to finish any initial sequence is by a block of Guesses followed by Decrements. By letting ω be the empty string, we find that the optimal strategy for the game is G^rD^N. Recalling Theorem 1, we find that the optimal value of r is $N^{1/2} - 1 - 1/(24N^{1/2}) + O(1/N)$ and thus

$$S(N) = 2N^{1/2} - 1/2 + 1/(12N^{1/2}) + O(1/N).$$

4. Extensions

At first glance, it might appear that our proof technique depends on not being able to Guess *0*. This is not the case. In fact, when Guesses of *0* are allowed, the optimal strategy differs from the above strategy by only $O(N^{-1/2})$ Guesses.

In the G-D game the value i in the box can be interpreted as the number of links between our present position in the list and the position of the element we are searching for. When searching for the last element we start at $i = N$. In Member search we are searching for an element whose position is unknown and randomly distributed. Therefore we should start at an unknown, random i, or equivalently start at $i = N$ and do one Guess to randomize i before counting operations. By Theorem 2 the optimal strategy is then $G^{r-1}D^N$ as opposed to G^rD^N when searching for the last element.

Sometimes the *jth* element in the list is needed. When j is small it is easy to follow links and when $j = N$ we can use the G-D strategy. Between these two extremes the strategies used thus far are not necessarily optimal since the value of the *jth* element is unknown. But if we know the value as well as the position of the element for which we are searching, then we can apply the above techniques to find an optimal G-D search time. Searching for an element at position j means starting the G-D game at $i = j$. This is equivalent to starting at $i = N$ and doing $N-j$ Decrements. By Theorem 2 the best way to continue from this point is G^kD^N for some k. Thus it is only necessary to compute the optimal number of Guesses, $k = r(j)$. By modifying Theorem 1 it can be shown that $r(j) = c(j)N^{1/2}$ where

$$c(j) = \begin{cases} 0 & \text{if } j \le (2N)^{1/2} \\ 1 - O(N^{-1/2}) & \text{if } j \ge \Omega(N^{1/2}). \end{cases}$$

The value of $c(j)$ can be determined numerically when j is between $(2N)^{1/2}$ and $\Omega(N^{1/2})$, but it cannot be written in closed form. Figure 1 shows the behavior of $c(j)$ in this region.

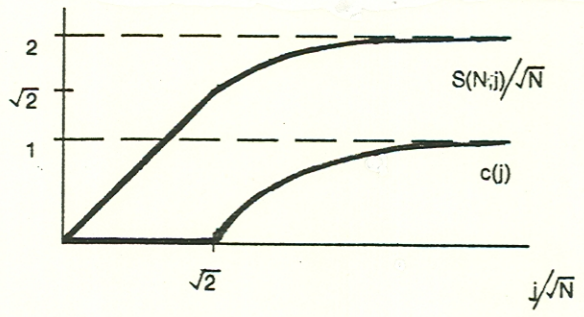

Figure 1: *Graph of $c(j)$ and $S(N,j)/N^{1/2}$ where $S(N,j)$ = optimal expected number of operations when searching for the j-th element.*

The G-D game can be modified in other ways. One particularly interesting modification allows the player to use the information about when a random sample is successful (i.e., closer to the target). For example, suppose the black box containing i is connected to a light which turns on every time i is decreased. At first glance, it appears as though such information could be quite useful in planning when to stop Guessing and start Decrementing. (For instance, if the light flashed on for a series of early Guesses, then the player might be led to believe that the early Guesses were very good and thus that i had become very small. Hence, the player might think it wise to start Decrementing early.) This is *not* the case, however, since if all the Guesses are required to be distinct, then it can be shown that the light does not *add* any useful information at all. It is worth remarking that the likelihood of two Guesses being identical is small and thus the constraint that all the Guesses be different has a negligible effect on the final result.

Such a counter-intuitive result requires some justification. First notice that when all the Guesses are distinct, the sequence of guesses is just a permutation of a subset R of $\{1,...,N\}$. Every sequence of guesses produces a unique light sequence, β, but one light sequence can be produced by many different guess sequences. In particular,

Lemma 5: *For every light sequence β of length k there exists an integer m, such that for any set of guesses $R \subseteq \{1,...,N\}$ of size k there are exactly m permutations of R which have light sequence β.*

Proof: Let $K = \{1,...,k\}$ and set m to be the number of permutations of K which produce the light sequence β. Now consider any other subset R of length k. There is a *1-1* order-preserving mapping between K and R, so there is also a *1-1* mapping between the permutations of the two sets which preserves light sequences. This means that there are also m permutations of R which fit β. □

We can now prove

Theorem 3: *When all the Guesses in the G-D game are distinct, then the light sequence adds no extra information about the value of i after a sequence of guesses.*

Proof: It is sufficient to show that the probability $\Pr[i = j \mid \beta]$, that $i = j$ after k guesses given a light sequence β, is the same as the probability $\Pr[i = j]$ that $i = j$ after k guesses (with no knowledge of the light sequence). From the definitions and Lemma 5,

$\Pr[i = j \mid \beta]$ = (# of sequences of guesses that fit β for which $i = j$)/(# of sequences of guesses that fit β)
= (# of R for which $i = j)m$/(# of $R)m$
= (# of R for which $i = j)k!$/(# of $R)k!$
= (# of sequences of guesses for which $i = j$)/(# of sequences of guesses)
= $\Pr[i = j]$. □

The G-D game might also be played with two different operations O_1 and O_2. It would be interesting to know what properties of O_1 and O_2 give an optimal sequence of the form $O_1^k O_2^m$, for some k and m. In addition to operators which act directly upon i, we might also consider comparison operations which compare i to a given input, n, and answer the

question, "$i \leq n$?" If the compare operation $i \leq (2N)^{1/2}$ is added to G-D, then the optimal strategy is $O(N^{1/4})$ Guesses followed by a Compare, repeated until $i \leq (2N)^{1/2}$, ending with Decrements. This strategy uses $(2N)^{1/2} + O(N^{1/4})$ expected steps.

Acknowledgements

We would like to thank Gary Miller, Ron Rivest, Jim Saxe, and Mike Sipser for helpful discussions. A special thanks to Jon Bentley for bringing this problem to our attention.

References

[1] J. L. Bentley, D. F. Stanat, J. M. Steele, "Analysis of a randomized data structure for representing ordered sets," *Proceedings Nineteenth Annual Allerton Conference on Communication, Control, and Computing*, pp. 364-372.

[2] W. Janko, "An insertion sort for keys with arbitrary key distribution," *ACM Transactions on Mathematical Software*, Vol.2, No.2, June 1976, pp. 143-153.