Technical Note
Computer Architecture and Systems
John P. Hayes*
Editor

Searching in a Dynamic Memory with Fast Sequential Access

Om Vikas
National Informatics Centre, New Delhi, India
V. Rajaraman
Indian Institute of Technology, Kanpur, India

This communication presents an algorithm for searching in the Aho–Ullman dynamic memory consisting of $(2^m - 1)$ cells. Mean search time of $1.5m$ steps to the first specified record is obtained with a subsequent sequential access capability. Thus, in such a dynamic memory, the mean access time for content addressing is the same as the mean access time for random addressing.

CR Categories and Subject Descriptors: H.3.3 **[Information Storage and Retrieval]**: Information Search and Retrieval—*search process*; B.3.2 **[Memory Structures]**: Design Styles—*sequential-access memory*
General Term: Algorithms
Additional Key Words and Phrases: dynamic memory, shuffle transformation, content addressing

1. Introduction

A dynamic memory consists of N cells interconnected by several memory transformations. At each clock time one of these transformations is selected and applied to the data stored in the memory. The conventional shift-register is a dynamic memory with a cyclic memory transformation. In this memory the mean access time to access a specified cell is $(N - 1)/2$ assuming unit time to transfer a datum from one cell to another.

Stone [3] proposed a pair of memory transformations, namely, *Shuffle* and *Exchange-Shuffle* in a dynamic memory of $N = 2^m$ cells. The mean random access time in such a memory is $(\log_2 N - 1) = (m - 1)$. Following Stone, we have counted *Exchange-Shuffle* as being a single transformation taking unit time even though, in practice, it takes two steps.

Aho and Ullman [1] considered a memory of $N = (2^m - 1)$ cells with two transformations, namely, modified *Shuffle* and *Cyclic* transformations. The cyclic transformation is also known as the *Minus-one* transformation. Aho and Ullman's scheme yields a mean random access time of $1.5m$[1] and a worst access time of $2m - 1$. Besides random access, this scheme has an additional advantage for sequential access, requiring one *Minus-one* transformation per access, after retrieving the first two data words of a block of data. Stone [4] indicated minor variations and improvements to the random access algorithm for the Aho–Ullman memory.

We develop an algorithm to retrieve a record with a given key in the Aho–Ullman memory. It is assumed that the records are presorted and stored in ascending order of the keys. The proposed method yields a mean access time of $1.5m$ to the first cell with the specified key and subsequent sequential access requiring one *Minus-one* transformation per access. The longest time in the worst case to retrieve a record with a specified key is $(2m - 1)$. Thus, access of contents takes the same average time as the access of an address in the Aho–Ullman memory. After the completion of the retrieval, the memory is returned to its original status with keys in ascending sequence to facilitate further retrieval. This is done using the Aho-Ullman algorithm and takes $1.5m$ units of time.

2. The Aho–Ullman Memory

The Aho–Ullman memory consists of $(2^m - 1)$ cells. The cells are numbered as $1, 2, \ldots, N$, where $N = 2^m - 1$. Cell 1 is the read/write port. Two memory transformations, namely, *Shuffle* and *Minus-one* are employed.

The *Shuffle* transformation transfers the datum in position α to position $S(\alpha)$ given by

$$S(\alpha) = \uparrow\alpha \qquad (1)$$

where $\uparrow$ is an operator which left circular shifts the m-bit binary representation of the address α by 1 bit.

The *Minus-one* transformation transfers the datum in position α to position $M(\alpha)$ given by

$$M(\alpha) = \alpha \oplus (-1), \qquad \alpha \neq 0 \qquad (2)$$

where $\oplus$ denotes one's complement addition of m-bit integers.

---

* Former editor of Computer Architecture and Systems, of which Duncan Lawrie is the current editor.

Authors' Present Addresses: Om Vikas, National Informatics Centre, Electronics Commission, E-Wing, Pushpa Bhavan, Chirag Delhi-Madangir Road, New Delhi 110 062, India; V. Rajaraman, Computer Centre, Indian Institute of Technology, Kanpur 208 016, India.

[1] We have assumed that *Minus-one* and *Shuffle* take unit time each.

Observe that the *Minus-one* transformation is a left circular shift of the contents of the memory and the *Shuffle* transformation a left circular shift of the address bits.

Data stored in the memory have distinct logical addresses, namely, 1, 2, ..., N. In order to access a datum with a specified logical address, a sequence of *Shuffle* and *Minus-one* transformations are applied and eventually the datum with the desired logical address reaches the read/write port, that is, cell 1. The sequence of transformations can be determined as follows: Let t be the logical address of the datum to be accessed and s be the logical address of the datum currently available at the read/write port. We define the distance d between t and s as $d = t \oplus (-s)$. Let $(d_{m-1}, d_{m-2}, \ldots, d_1, d_0)$ be the binary representation of d. Thus $d = d_{m-1}2^{m-1} + d_{m-2}2^{m-2} + \cdots + d_1 2^1 + d_0$.

The distance d determines the sequence of transformations required to bring the datum at logical address t to the read-write port and place the memory in a cyclic shift of the initial state which allows data with successive logical addresses to be accessed sequentially. The algorithm which achieves this is

Algorithm 1: Aho-Ullman random access algorithm
For $k = m - 1$ **step** -1 **until** 0 **do**
begin comment: (**Minus** d_k means **Minus-one** if $d_k = 1$ and **skip** if $d_k = 0$);
Shuffle;
Minus d_k;
end.

The above algorithm for random accessing yields a mean access time of $1.5m$.

3. Searching

In a number of applications it is desirable to retrieve a group of records from a memory with the first record in the group having a given key. This requires us to develop an algorithm to search in a memory for a record with a given key at the lowest logical address and to retrieve sequentially a specified number of records following this record.

We assume that the records are presorted by key and stored in a nondecreasing order in a memory of size $(2^m - 1)$. A binary search method [2] may now be employed to retrieve a record with a specified key.

The following Lemma establishes the address of the cell whose contents will be brought to cell 1 (i.e., the read/write port) of the memory. S is used as an abbreviation for *Shuffle* and M for *Minus-one*.

LEMMA 1. Let the sequence of transformations applied to the memory be

$$S\, t_1 t_2 \cdots t_p \qquad p \leq (m-1)$$

where each t_i is either S or MS. Then the contents of the cell

$$\underbrace{b_1 b_2 \cdots b_p 100 \cdots 0}_{m}$$

(where b_i is 1 if t_i is MS and b_i is 0 if t_i is S) will be brought to cell 1.

PROOF. The proof is by induction on the length of memory transformation sequence. From the definitions of S and M in Eqs. (1) and (2), it is easy to verify that

$$SS\,(0100 \cdots 0) = 00 \cdots 01$$

and

$$SMS\,(1100 \cdots 0) = 00 \cdots 01$$

Let the lemma be true for

$$S\, t_1 t_2 \cdots t_p, \qquad p < (m-1) \text{ (Induction hypothesis)}$$

Therefore

$$S\, t_1 t_2 \cdots t_p \underbrace{(b_1 b_2 \cdots b_p 10 \cdots 0)}_{m} = \underbrace{000 \cdots 1}_{m} \qquad (p < m-1)$$

Let us now consider a sequence of transformations

$$\sigma = S\, t_1 t_2 \cdots t_{p+1}$$

Let B be defined as:

$$B = \underbrace{b_1 b_2 \cdots b_p b_{p+1} 100 \cdots 0}_{m}$$

where $b_i = 0$ if t_i is S and 1 if t_i is MS. We are required to show that

$$\sigma(B) = \underbrace{00 \cdots 1}_{m}$$

Now

$$\sigma(B) = S\, t_1 t_2 \cdots t_{p+1}(b_1 b_2 \cdots b_p b_{p+1} 100 \cdots 0)$$
$$= t_1 t_2 \cdots t_{p+1}(b_2 \cdots b_p b_{p+1} 100 \cdots b_1)$$

We have two cases:

Case 1: $t_1 = S$

Then by definition of B, $b_1 = 0$, therefore

$$\sigma(B) = S\, t_2 \cdots t_{p+1}(b_2 \cdots b_p b_{p+1} 100 \cdots 0)$$
$$= 000 \cdots 1 \text{ (invoking the induction hypothesis)}$$

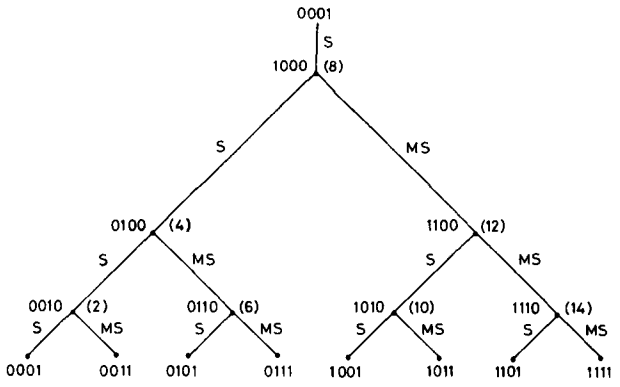
Case 2: $t_1 = MS$

Then by definition of B, $b_1 = 1$, therefore

$$\sigma(B) = MSt_2 \cdots t_{p+1} \cdot (b_2 \cdots b_p b_{p+1} 100 \cdots 1)$$
$$= St_2 \cdots t_{p+1} \cdot (b_2 \cdots b_p b_{p+1} 100 \cdots 0)$$
$$= 000 \cdots 01 \text{ (invoking the induction hypothesis)} \quad \square$$

Using this lemma we observe that a judicious application of a sequence of S and MS transformations leads to the binary search tree of Figure 1. We interpret the tree as follows: Take any node in the tree. The node is labeled with the address of the cell whose contents will

Fig. 1. Binary Search Tree Obtained Using Lemma 1.



reach cell 1 by applying the transformations indicated on the branches of the tree leading to this node from the root of the tree. For example, take the node labeled 0110. Then employing *SSMS* transformation to the memory will take the contents of location 0110 to cell 1. This can be verified by observing that $m = 4$, $b_1 = 0$, $b_2 = 1$, $p = 2$. Thus the sequence *SSMS* applied to 0110 gives, by Lemma 1

$$SSMS\ (0110) = 0001$$

4. Algorithm for Search

Let $R_1, R_2, \ldots, R_N$ be records whose keys $K_1, K_2, \ldots, K_N$ are in nondecreasing order, so that $K_1 \leq K_2 \cdots \leq K_N$. Let Z be the key of the record in cell 1 and K be the key of the record to be retrieved. Let S be the m-bit logical address of the record brought to cell 1 by a sequence of S and MS transformations. Let r be a flag which is set to 1 if the search for the key succeeds before applying mS transformations. The value of s is derived using Lemma 1. The symbol $\oplus$ indicates one's complement addition of m-bit numbers.

Algorithm 2: Algorithm for binary search in Aho-Ullman memory
begin
  If $Z = K$ **then Write** "No Search needed"
  Else If $Z > K$ **then Write** "Record not in memory"
  Else If $Z < K$ **then Call** *Search*
end
Search:
begin
  $r \leftarrow 0$
  Shuffle
  $s \leftarrow 2^{m-1}$
  For $i = 1$ **to** $(m-1)$ **step** 1 **do**
  begin
    If $Z < K$ **then begin Minus-one**
      Shuffle
      $s \leftarrow s \oplus 2^{m-i-1}$ **end**
    Else If $Z = K$ **then begin Shuffle**
      $r \leftarrow 1$
      $s \leftarrow s \oplus (-2^{m-i-1})$ **end**
    Else If $Z > K$ **then begin Shuffle**
      $s \leftarrow s \oplus (-2^{m-i-1})$ **end**
  end
If $r = 1$ **and** $Z \neq K$ **then begin Minus-one**
  $s \leftarrow s \oplus -1$ **end**
Else If $r = 0$ **and** $Z \neq K$ **then Write** "Search Fails"
end.

Proof of the algorithm

The algorithm enters the routine *Search* when the key $K > C(00 \cdots 1)$, where the notation $C(00 \cdots 1)$ is used to indicate the contents of cell $00 \cdots 1$. Thus, if K is in the memory, then

$$\underbrace{C(000 \cdots 01)}_{m} < K \leq \underbrace{C(11 \cdots 1)}_{m}$$

The routine *Search* first effects an S transformation and then it enters the **for** loop which iterates exactly $(m-1)$ times. In each iteration of the **for** loop either an S or an MS transformation is applied to the memory. Therefore after p iterations of the **for** loop the transformations applied on the memory is given by the sequence:

$$S\ t_1 t_2 \cdots t_p$$

where t_i is either S or MS depending on the transformation applied in the ith iteration. From Lemma 1 it follows that cell 1 will contain at the end of p iterations the contents of cell

$$\underbrace{b_1 b_2 \cdots b_p 100 \cdots 0}_{m}$$

where b_i is 1 if t_i is MS and b_i is 0 if t_i is S. Thus at the end of p iterations

$$Z = C(b_1 b_2 \cdots b_p \underbrace{100 \cdots 0}_{m-p})$$

LEMMA 2. Let the contents of Cell 1 at the end of $p (0 < p \leq m-1)$ iterations of the **for** loop be (from Lemma 1)

$$Z = C(b_1 b_2 \cdots b_p 100 \cdots 0)$$

If the record with key K is in the memory, then if $r = 0$ at the end of p iterations of the **for** loop

$$C(b_1 b_2 \cdots b_p \underbrace{00 \cdots 01}_{m-p-1}) \leq K \leq C(b_1 b_2 \cdots b_p \underbrace{11 \cdots 1}_{m-p})$$

If r becomes 1 at the end of the pth iteration, then

$$K = C(b_1 b_2 \cdots b_p \underbrace{11 \cdots 1}_{m-p} + 1)$$

If r becomes 1 before p iterations of the **for** loop, then the **first** occurrence of K will be in the range of addresses:

$$b_1 b_2 \cdots b_p \underbrace{0 \cdots 0}_{m-p-1} 1 \leq A(K) \leq b_1 b_2 \cdots b_p \underbrace{11 \cdots 1}_{m-p}$$

where the notation $A(K)$ indicates the address of K.

PROOF. We will prove this lemma using induction on p.

At the end of 0th iteration $r = 0$, and clearly if K is in the memory then

$$\underbrace{C(00 \cdots 0)}_{m-1} 1 \leq K \leq \underbrace{C(1 \cdots 1)}_{m}$$

This serves as the base for the case $r = 0$.

For the induction step of this case let us *assume* that at the end of $(p-1)$ iterations

$$C(b_1 \cdots b_{p-1} \underbrace{0 \cdots 01}_{m-p}) \leq K \leq C(b_1 \cdots b_{p-1} \underbrace{1 \cdots 1}_{m-p+1})$$

At the pth iteration the key K is compared with Z the contents of cell 1. The original contents of cell 1 are from Lemma 1:

$$Z = C(b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p})$$

If $K > Z$, then, as the list is sorted, K obeys the inequality

$$C(b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p-1} 1) \leq K \leq C(b_1 \cdots b_{p-1} 1 \underbrace{1 \cdots 1}_{m-p})$$

The pth iteration will effect an MS transformation on the memory (as $K > Z$), and thus

$$Z = C(b_1 \cdots b_{p-1} 1 1 \underbrace{00 \cdots 0}_{m-p-1})$$
$$= C(b_1 \cdots b_p 1 \underbrace{00 \cdots 0}_{m-p-1})$$

Thus $b_p = 1$ and we see that Lemma 2 is true for $r = 0$ at the end of the pth iteration.

Similarly, we can verify the lemma when $K < Z$. The base case for the remaining parts of Lemma 2 is that value of p when r becomes 1 from 0. Let this happen at the pth iteration.

At the beginning of the pth iteration using Lemma 1, we see that

$$K = Z = C(b_1 b_2 \cdots b_{p-1} 1 \underbrace{00 \cdots 0}_{m-p})$$

The pth iteration effects an S transformation on the memory as $Z = K$. Thus by Lemma 1 after the pth iteration

$$Z = C(b_1 \cdots b_{p-1} 0 1 \underbrace{00 \cdots 0}_{m-p-1})$$

Thus $b_p = 0$.
Therefore

$$b_1 \cdots b_p \underbrace{1 \cdots 1}_{m-p} + 1 = b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p}$$

and

$$K = C(b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p})$$

Till the end of the $(p-1)$th iteration (by assumption) $r = 0$. Invoking the first part of this lemma and knowing that

$$K = C(b_1 \cdots b_{p-1} 1 \underbrace{00 \cdots 0}_{m-p})$$

We can conclude that if K repeats before this value, then

$$C(b_1 \cdots b_{p-1} \underbrace{0 \cdots 0}_{m-p} 1) \leq K \leq C(b_1 \cdots b_{p-1} 0 \underbrace{1 \cdots 1}_{m-p})$$

From Lemma 1 we observe that $b_p = 0$. Thus, if more than one record has key K its first occurrence must be in the range of addresses

$$b_1 \cdots b_p \underbrace{0 \cdots 0}_{m-p-1} 1 \quad \text{to} \quad b_1 \cdots b_p \underbrace{1 \cdots 1}_{m-p}$$

For the induction part of the second part of Lemma 2, let the lemma be true at the end of $(p-1)$th iteration where $r = 1$. At the pth iteration also r will remain to be 1.

From the induction hypothesis

$$K = C(b_1 \cdots b_{p-1} \underbrace{1 \cdots 1}_{m-p+1} + 1)$$

and

$$C(b_1 \cdots b_{p-1} \underbrace{00 \cdots 0}_{m-p} 1) \leq K$$
$$\leq C(b_1 \cdots b_{p-1} \underbrace{1 \cdots 1}_{m-p+1})$$

We compare the key K with the contents of location $(b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p})$ at the pth iteration. K cannot be less than this value. If K is equal to this, then b_p becomes 0 to make Z at the end of pth iteration $= C(b_1 \cdots b_{p-1} 0 1 \underbrace{0 \cdots 0}_{m-p-1})$ But

$$b_1 \cdots b_p \underbrace{1 \cdots 1}_{m-p} + 1 = b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p}$$

and thus the lemma holds.

If K is found greater than the contents of $(b_1 \cdots b_{p-1} 1 \underbrace{0 \cdots 0}_{m-p})$ then the first occurrence of K must be after this value. Moreover, in this case when K is found greater b_p becomes 1. Thus the lemma holds again. □

The correctness of the algorithm is established from Lemma 2. First we observe that at the completion of the **for** loop exactly S transformations have been carried out on the memory, interspersed with one or more M transformations. From Aho–Ullman [1] it follows that the contents of the memory will be in the original order with a cyclic shift of addresses. The contents of $(b_1 b_2 \cdots b_{m-1} 1)$ will be at cell 1. If $r = 0$, then from Lemma 2, if K is in the memory then

$$C(b_1 \cdots b_{m-1} 1) \leq K \leq C(b_1 \cdots b_{m-1} 1)$$

In other words, if $K = Z = C(b_1 \cdots b_{m-1} 1)$, with normal exit from the **for** loop, then K is in the memory, otherwise it is not.

On the other hand if $r = 1$ at the normal exit from the **for** loop, then Lemma 2 asserts that

$$K = C(b_1 b_2 \cdots b_{m-1} 1 + 1)$$

and if K repeats, then its first occurrence must be at $(b_1 \cdots b_{m-1} 1)$. Thus, if $K = Z$ it is the first occurrence of K. Otherwise the first (and maybe only) occurrence of K is at $b_1 \cdots b_{m-1} 1 + 1$. Therefore if K is not equal to the contents of $b_1 \cdots b_{m-1} 1$ then by doing a M transformation we get the contents of $(b_1 \cdots b_{m-1} 1 + 1)$ at cell

Table I. Searching in a Dynamic Memory.

Physical address	Contents with logical address	$Z < K$ Apply S	Contents	$i = 1$ $Z = K$ Apply S	Contents	$i = 2$ $Z < K$ Apply MS	Contents	$Z < K$ Apply M	Contents
				Search for $K = 489$, $m = 3$					
1	150(1)	$r = 0$	489(4)	$r = 1$	200(2)	$r = 1$	305(3)	$r = 1$	489(4)
2	200(2)		150(1)		489(4)		489(4)		520(5)
3	305(3)		520(5)		520(6)		520(5)		520(6)
4	489(4)		200(2)		150(1)		520(6)		699(7)
5	520(5)		520(6)		305(3)		699(7)		150(1)
6	520(6)		305(3)		520(5)		150(1)		200(2)
7	699(7)		699(7)		699(7)		200(2)		305(3)
	↑ Logical address								
				Initialization after retrieval					
1	489(4)	Apply SM	489(4)	Apply S	200(2)	Apply S	150(1)		
2	520(5)		150(1)		489(4)		200(2)		
3	520(6)		520(5)		520(6)		305(3)		
4	699(7)		200(2)		150(1)		489(4)		
5	150(1)		520(6)		305(3)		520(5)		
6	200(2)		305(3)		520(5)		520(6)		
7	305(3)		699(7)		699(7)		699(7)		

(·) *denotes logical address of datum.*

1. This is because after mS transformations, the memory has reverted back to the original order within a cyclic permutation, and $C(b_1 \cdots b_{m-1}1 + 1)$ will be at the next higher location to cell $(b_1 \cdots b_{m-1}1)$.

Initialization of memory to original state

After retrieving the record(s) with the desired key the contents of the memory is to be placed back in the original sorted order with the record with the lowest key in cell 1. This is done by applying the Aho–Ullman algorithm of Sec. 2 as follows: At the termination of the search algorithm the variable s contains the initial address of the record which is now in cell 1. We define the distance d as $1 \oplus (-s)$. Let $(d_{m-1}d_{m-2} \cdots d_1d_0)$ be the binary representation of d. We use this with Algorithm 1 (Sec. 2) to initialize the memory to its original state.

Illustrative example

Table I traces Algorithm 2. The initial and subsequent contents of the memory are listed. We consider $m = 3$. Initially $r \leftarrow 0$. *Search* for $K = 489$ shows a successful match when $i = 1$; consequently, $r \leftarrow 1$ and $s \leftarrow 100$. In order to test for a multiplicity of records containing the specified key, we examine records at addresses less than s, and hence apply an S transformation. At $i = 2$, $Z < K$ and hence MS is applied. When we leave the loop, $Z \neq K$, and $r = 1$. A *Minus-one* transformation brings the most recently matched record (Key 489) to cell 1. Now the memory is ready for consecutive retrievals. The value of s is incremented by one for every subsequent retrieval. Initialization is carried out by knowing the binary representation of $d = 1 \oplus (-s)$. In our example, $d = 001 \oplus (-100) = 100$. Thus the sequence of transformations (SM S S) places the memory in its initial state.

5. Mean Access Time Estimation

Every search, successful or unsuccessful, ends after m *Shuffle* transformations are applied to the initial memory state. If a search is successful at the last state, no reordering is required. If it is successful at any state but the last, a search for a possible match over lower addresses is required. If this search fails, a *Minus-one* transformation is applied.

To obtain the mean access time, we assume that each transformation is performed in unit time and the keys are uniformly distributed. For a dynamic memory of size $(2^m - 1)$, the average number of *Shuffle* and *Minus-one* transformations is $1 + 1.5(m - 1)$ if the search succeeds at the $(m - 1)$th step. It is $1 + 1.5(m - 1) + 1$ if the search succeeds at an earlier stage. Hence, the average number of transformations for a successful search is $1.5m$ and for an unsuccessful search is $1 + 1.5(m - 1)$.

For initialization, if $d = (000 \cdots 0)$ is detected then no transformation is applied. The case $d = (11 \cdots 1)$ never occurs. The remaining values of d are equally likely. A 0 in the binary representation of d corresponds to the transformation S, and 1 to the transformation S followed by M. As 0 and 1 are equally probable, the average number of transformations for initialization is $1.5m$.

Thus, it is concluded that the mean search time is $1.5m$, the mean initialization time is $1.5m$, and the worst case search time is $1 + 2(m - 1) = 2m - 1$. It may be

assumed that the mean interrequest time is sufficiently large compared to the mean initialization time so that a queue of requests does not build up. The initialization overhead is needed to bring the memory to its initial state so as to facilitate another binary search in the memory.

6. Conclusions

It has been shown that if records in a dynamic memory of size $(2^m - 1)$ are stored in a nondecreasing order of their keys, then it is possible to retrieve a record with a given key by applying a pair of memory transformations called *Shuffle* and *Minus-one*. A binary search algorithm has been implemented by an appropriate sequence of applications of *Shuffle* and *Minus-one Shuffle* transformations. The mean time to search and retrieve a record with a specified key is shown to be $1.5m$. This time is identical to that needed to retrieve a record from a specified random address [4]. After the search is over, it is necessary to spend an average of $1.5m$ steps to return the dynamic memory to its initial state, namely, with the keys in nondecreasing order, to facilitate further search. This initialization may, however, be done between search requests.

Acknowledgment. The authors thank S. Biswas for critically reading the manuscript and for assistance in developing a proof of the algorithm.

Received 8/79; revised 5/81; accepted 9/81

References

1. Aho, A. and Ullman, J.D. Dynamic memories with rapid random and sequential access. *IEEE Trans. Comput., C.23,* (March 1974) 271–276.
2. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching.* Addison Wesley, Reading, MA, 1973.
3. Stone, H.S. Dynamic memories with enhanced data access. *IEEE Trans. Comput., C.21,* (April 1972) 359–366.
4. Stone, H.S. Dynamic memories with fast random and sequential access. *IEEE Trans. Comput., C.24,* (Dec. 1975) 1167–1174.

Technical Note
Systems Modeling and Performance Evaluation

Herbert D. Schwetman
Editor

Estimating Block Accesses and Number of Records in File Management

To-Yat Cheung
University of Ottawa

We consider the problems of estimating the number of secondary storage blocks and the number of distinct records accessed when a transaction consisting of possibly duplicate requested records is presented to a file management system. Our main results include (1) a new formula for block access estimation for the case where the requested records may have duplications and their ordering is immaterial and (2) a simple formula for estimating the number of distinct records in the transaction.

CR Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design; H.2.3 [**Database Management**]: Languages
General Terms: Design, Theory
Additional Key Words and Phrases: files, block access, number of records, transactions

1. Introduction

In file management, in order to estimate the sizes of the buffer, secondary storage space and core space for a transaction, or the cost of its transmission, two seemingly different problems frequently arise.

I. *The block access estimation problem*: Suppose that the n records of a file are stored randomly and uniformly in m blocks of secondary storage and that k records are

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant #8963.

Author's Present Address: To-Yat Cheung, Computer Science Department, University of Ottawa, 375 Nicholas, Ottawa, Ontario, Canada, K1N 9B4.