RC 8550 (#37269) 11/7/80
Computer Science 50 pages

Research Report

A Sparse Table Implementation Of Priority Queues

Alon Itai[1], Alan G. Konheim[2] and Michael Rodeh[3]

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

IBM Research Division
San Jose · Yorktown · Zurich

RC 8550 (#37269) 11/7/80
Computer Science 50 pages

A Sparse Table Implementation Of Priority Queues

Alon Itai[1], Alan G. Konheim[2] and Michael Rodeh[3]

IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract : A data structure is suggested for efficient implementation of operations such as Search, Insert, Delete, Min, Next, Scan. The idea (proposed independently by Franklin and by Melville and Gries) is to maintain an ordered list of the elements (keys) in which an element may appear more than once. Insertion, as a typical operation, is done by (binary) search followed by some move operations. We show that the expected number of move operations is bounded (in the size of the table), provided that the density of *dummy keys* in the table is bounded away from 1. The probabilistic model assumes that all permutations of the input are equiprobable. As a result of the analysis, a flaw in both Franklin's and Melville and Gries' analyses was found. To improve the worst case behavior, a variant of the sparse table scheme is suggested. In that version, a sequence of n insertions into a table of size n may not take more than $O(n (\log n)^2)$.

[1] Computer Science Department, Technion – Israel Institute of Technology, Haifa, Israel

[2] Mathematical Sciences Department, IBM Thomas J. Watson Research Center

[3] IBM Israel Scientific Center, Technion City, Haifa, Israel

1. Introduction

Priority queues have been defined in several ways. In this paper a *priority queue* is a data structure on which the following operations can be executed:

Search(x)	Determine if x is in the queue.
Insert(x)	Enter x into the queue.
Delete(x)	Remove x from the queue.
Min	Find the smallest item in the queue.
Next(x)	Find the item following x in the queue.
Scan	Scan all members of the queue.

There are numerous implementations of priority queues, in which each of the first five operations requires no more than $O(\log_2 n)$ time (n is the number of keys currently in the queue). All of these implementations use trees: 2-3 trees [AHU], AVL-trees, weight balanced trees [RND], binomial trees [V] and require large overhead, both in time and in space. In many cases the algorithms devote much of their running time and storage manipulating the priority queue, often rendering a theoretically efficient algorithm to be infeasible or inefficient for all practical purposes.

An implementation of priority queues by means of *sparse tables* is presented in this paper. Data is stored in a linear array and requires only a single pointer. Insertion requires three operations:

- *searching* – to locate the table address at which a record will be inserted,
- *moving* – shifting of records in the table to free space for the record to be inserted, and
- *reconfiguring* – increasing the size of the table and distributing the keys evenly in the larger table.

Searching a table of size m requires $O(\log_2 m)$ time using binary search and $O(\log_2 \log_2 m)$ time using interpolation search [PIA]. While moving may in the worst case take O(m) time, the expected number of moves is bounded (in m) provided the density of records in the table is bounded from one. The main contribution of this paper is an exact analysis of the complexity of the move operation (see Sections 3-6).

Melville and Gries [MG] and Franklin [FR] have independently proposed a scheme which is similar to this sparse table. To carry out the analysis of their structure, both relate the insertion of keys in a sparse table to the length of a probe in hashing with linear probing [KN2, KW, BK]. While such a relationship exists, the probabilistic models underlying these two processes differ so that the analyses offered in [FR] and [MG] are incorrect. We discuss this in Section 7.

To improve the worst case behavior, a more complicated sparse table scheme is introduced in Section 8 which requires no more than $O(m(\log_2 m)^2)$ time.

Deletion may be implemented by maintaining a bit vector which marks the records that have been deleted. This idea was also used by Bentley et. al. [BDGS] and by Guibas et. al. [GMPR] and is discussed in Section 9.

2. A Sparse Table Scheme

Associate a *key* K = K(R) with each record R such that the correspondence between records and keys R $\leftrightarrow$ K(R) is biunique. When records are stored in a table with their keys in sorted order, *Search*(K) is efficient. If records are stored contiguously, *Insert*(K) is carried out by moving records to free space for the record to be inserted. The efficiency of insertion will be improved by introducing gaps in the table thereby storing n records in a table of capacity m for some $m \geq n$. A key is assigned to each address in the table by introducing fictitious or *dummy* keys. We describe the state of the table by the vector

$$\underline{y} = (y_0, y_1, \ldots, y_{m-1}) \qquad y_0 \leq y_1 \leq \ldots \leq y_{m-1}$$

indicating with this notation that the (genuine or dummy) key y_i is stored at address i for $0 \leq i < m$. If the genuine keys in the table are

$$y_{i_0} < y_{i_1} < \ldots < y_{i_{n-1}} \qquad 0 \leq i_0 < i_1 < \ldots < i_{n-1} = m-1$$

then dummy keys, each with the value y_{i_t}, are stored at each address in the contiguous block of $i_t - i_{t-1} - 1$ addresses

$$i_{t-1}+1, i_{t-1}+2, \ldots, i_t-1 \qquad (0 \le t < n;\ i_{-1} = -1)$$

so that

$$y_0 = \ldots = y_{i_0} < y_{i_0+1} = \ldots = y_{i_1} < \ldots < y_{i_{n-1}+1} = \ldots = y_{i_n-1}$$

For example, genuine keys are located at addresses 2, 5, 6 and 8 in the table

$$\underline{y} = (2, 2, 2, 3, 3, 3, 4, 5, 5) \qquad n = 4, m = 9$$

It will be useful to consider the table as a circular array; address 0 follows address $m-1$ and address calculations are made modulo m. A *head of the table pointer*[4] PB (initially 0) translates between the (logical) address in memory and the (virtual) address in the table

logical address i = (PB + *virtual address* i) (modulo m)

The building of a table by means of the insertion of keys is determined by two sequences of integers

$$\{n_k : 0 \le k < \infty\} \qquad \{m_k : 1 \le k < \infty\}$$

$$1 = n_0 < n_1 < n_2 < \ldots < n_k < \ldots \qquad n_k \le n_{k-1}m_k \quad 1 \le k < \infty$$

which have the following interpretation:

(i) The size of the table m can only take the values $n_{k-1}m_k$ for $k = 1, 2, \ldots$.

(ii) The size of the table is $m = n_{k-1}m_k$ when the number of distinct keys in the table n satisfies $n_{k-1} < n \le n_k$.

---

[4] The *state of memory* $\underline{z} = (z_0, z_1, \ldots, z_{m-1})$ and the state of the table $\underline{y} = (y_0, y_1, \ldots, y_{m-1})$ are related by $y_t = z_{t+PB \text{ (modulo m)}}$ $(0 \le t < m)$.

(iii) To insert the key x in a table of size $m = n_{k-1}m_k$ containing $n_{k-1} < n < n_k$ genuine keys, the address s satisfying

$$y_{s-1} < x < y_s \qquad 0 \le s < m$$

is determined by a binary search. $s = 0$ means that x is either smaller or larger than any key presently in the table. If y_s is a dummy key, it is replaced by x yielding the table

$$(y_0, \ldots, y_{s-1}, x, y_{s+1}, \ldots, y_{m-1})$$

If y_s is a genuine key, the block of t (say) consecutive genuine keys

$$y_s \neq y_{s+1} \neq \ldots \neq y_{s+t-1} \neq y_{s+t} = y_{s+t+1}$$

is *moved* circularly to the right one position, the pointer PB is adjusted if necessary and x is inserted at address s yielding the table

$$(y_0, \ldots, y_{s-1}, x, y_s, \ldots, y_{s+t-1}, y_{s+t+1}, \ldots, y_{m-1})$$

(iv) A *reconfiguration* of the table takes place when the key x is to be inserted into a table of size $m = n_{k-1}m_k$ containing n_k genuine keys. The size of the table is first increased to $n_k m_{k+1}$; the n_k genuine keys

$$y_{i_0} < y_{i_1} < \ldots < y_{i_{n_k-1}}$$

are uniformly distributed to obtain the table

$$((y_{i_0})^{m_{k+1}}, \ldots, ((y_{i_{n_k-1}})^{m_{k+1}})$$

where $(k)^s$ denotes s copies of k. A binary search for x in this expanded table is carried out and the key x is inserted as in step *(iii)*.

The numbers $\{m_k\}$ are the (multiplicative) expansion factors; the ratio $\rho = n/n_{k-1}m_k$ with $n_{k-1} < n \le n_k$ is the *density* of genuine keys in the table. Note that $1/m_k < \rho \le 1$. The expansion factor m_k adds approximately $\log_2 m_k$ steps to the binary search but reduces the numbers of keys which must be *moved* to insert a new key. The cost of reconfiguration is $O(n_k m_{k+1})$ so that if the number of keys inserted since the last reconfiguration, $n_k - n_{k-1}$, is proportional to the size of the

expanded table, the cost of reconfiguration per key is constant. As for the worst case, $O(n-n_{k-1})$ keys may be moved to insert the n^{th}-key. We will show in Sections 3-6 that the expected number of moves required remains bounded as $n_k \to \infty$ provided ρ is bounded away from one.

Example 2.1: $n_k = 2^k$ $(0 \le k < \infty)$ and $m_k = 3$ $(1 \le k < \infty)$.

Suppose the keys 0, 1, 2, ..., 7 are inserted in the order 7, 3, 1, 0, 5, 6, 4, 2. The state of the table and the number of moves to insert each key is shown below.

Key Inserted	(Virtual) Address	0	1	2	3	4	5	6	7	8	9	10	11	Number of Moves
7		7	7	7										0
3		3	7	7										0
1		1	3	3	7	7	7							0
0		0	1	3	7	7	7							1
5		0	0	0	1	1	1	3	3	3	5	7	7	0
6		0	0	0	1	1	1	3	3	3	5	6	7	0
4		0	0	1	1	1	3	3	3	4	5	6	7	3
2		0	0	1	1	1	2	3	3	4	5	6	7	0

In this example, the cost of reconfiguration is

$$m_k \big/ (1-n_{k-1}/n_k) = 6 \qquad (= O(1) \text{ as } k \to \infty).$$

per insertion.

3. An Occupancy Discipline

We consider an *occupancy discipline* for the placement of j (indistinguishable) balls $(0 \le j \le ic)$ into a system of i urns

$$URN_0, URN_1, ..., URN_{i-1}$$

each of capacity c (balls). The urns are arranged cyclically so that URN_{t+1} is "to the right" of URN_t for $0 \le t < i$ and $URN_i = URN_0$.

The process of placing balls in the urns is sequential; it starts from an *assignment sequence* $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ specifying the urn into which each ball is *assigned*. The constraint on an urn's capacity does not allow the *placement* of the t^{th}-ball into (its assigned urn) URN_{α_t} if this urn is now *full* – meaning that URN_{α_t} contains c balls after the placement of balls 0, 1, ..., t−1. In this case the t^{th}-ball is placed into the first urn "to the right" of URN_{α_t} which is not presently full. The *placement value* of the t^{th}-ball β_t is the index of the urn into which the t^{th}-ball is *placed*.

We have thus defined a mapping from a sequence of assignment values $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ to a sequence of placement values $\beta_0, \beta_1, \ldots, \beta_{j-1}$

$$P_c(\alpha_0, \alpha_1, \ldots, \alpha_{j-1}) = \beta_0, \beta_1, \ldots, \beta_{j-1}$$

We associate with the sequences of assignment and placement values an *assignment vector* $\underline{a}$ and a *placement vector* $\underline{b}$

$$\alpha_0, \alpha_1, \ldots, \alpha_{j-1} \rightarrow \underline{a} = (a_0, a_1, \ldots, a_{i-1})$$

$$\beta_0, \beta_1, \ldots, \beta_{j-1} \rightarrow \underline{b} = (b_0, b_1, \ldots, b_{i-1})$$

in which

a_t : number of terms in $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ equal to t

b_t : number of terms in $\beta_0, \beta_1, \ldots, \beta_{j-1}$ equal to t

The vectors $\underline{a}$ and $\underline{b}$ satisfy the constraints

(3.1a) $a_0 + a_1 + \ldots + a_{i-1} = j$

(3.1b) $a_t \geq 0 \qquad 0 \leq t < i$

(3.2a) $b_0 + b_1 + \ldots + b_{i-1} = j$

(3.2b) $0 \leq b_t \leq c \qquad 0 \leq t < i$

URN_t is full (after the placement of the j balls) if $b_t = c$. The set of (assignment) vectors $\underline{a}$ satisfying equations (3.1a-b) corresponds to the $C(i+j-1,i-1)$[5] *ordered partitions of j into i parts* and will be denoted by $\mathbf{A}(i,j)$. The set of (placement) vectors $\underline{b}$ satisfying equations (3.2a-b) corresponds to the *ordered partitions of j into i parts each no larger than c* and will be denoted by $\mathbf{B}_c(i,j)$.

While the placement function P_c has been defined as a mapping from assignment sequences $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ to placement sequences $\beta_0, \beta_1, \ldots, \beta_{j-1}$, it will be useful to consider P_c (with a slight abuse of notation) to be a mapping from assignment vectors in $\mathbf{A}(i,j)$ (with $0 \leq j \leq ic$) to placement vectors in $\mathbf{B}_c(i,j)$

$$P_c : \mathbf{A}(i,j) \rightarrow \mathbf{B}_c(i,j)$$

That P_c can be so defined follows from two observations; first, a placement sequence $\beta_0, \beta_1, \ldots, \beta_{j-1}$ determines in an obvious way a placement vector $(b_0, b_1, \ldots, b_{i-1})$ so that the mapping P_c induces an obvious mapping (also denoted by P_c) of assignment sequences $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}$ to placement vectors $(b_0, b_1, \ldots, b_{i-1})$

$$P_c(\alpha_0, \alpha_1, \ldots, \alpha_{j-1}) = (b_0, b_1, \ldots, b_{i-1})$$

Second, the mapping P_c (now with range $\mathbf{B}_c(i,j)$) is "constant" on assignment sequences which are related by a coordinate permutation. More precisely,

Lemma 3.1: Let π be an arbitrary permutation of $(0, 1, \ldots, j-1)$. The two assignment sequences

$$\alpha_0, \alpha_1, \ldots, \alpha_{j-1} \qquad \alpha_{\pi(0)}, \alpha_{\pi(1)}, \ldots, \alpha_{\pi(j-1)}$$

yield the same placement vectors.

Proof: The proof is by induction on j; for $j = 1$ the assertion is trivially true. By induction it suffices to prove that both assignment sequences

---

[5] $C(s,t)$ denotes the binomial coefficient $s!/t!(s-t)!$

$$\alpha_0, \alpha_1, \ldots, \alpha_{j-1}, \alpha_j \qquad \alpha_0, \alpha_1, \ldots, \alpha_j, \alpha_{j-1}$$

yield the same placement vector. Suppose $\alpha_j = i$ and that

$$P_c(\alpha_0, \alpha_1, \ldots, \alpha_{j-1}) = (b_0, b_1, \ldots, b_{i-1}).$$

There are three cases to examine:

Case 1: $b_r < c$

Case 2: $b_r = b_{r+1} = \ldots = b_{r+s-1} = c > b_{r+s}$ and $\alpha_{j-1} \notin \{r, r+1, \ldots, r+s-1\}$

In both *Cases 1* and *2*, it is easy to see that the insertion of balls with assignment values α_{j-1} and α_j do not mutually interfere so that

$$P_c(\alpha_0, \ldots, \alpha_{j-2}, \alpha_{j-1}, \alpha_j) = P_c(\alpha_0, \ldots, \alpha_{j-2}, \alpha_j, \alpha_{j-1})$$

Case 3: $b_r = b_{r+1} = \ldots = b_{r+s-1} = c > b_{r+s}$ and $\alpha_{j-1} \in \{r, r+1, \ldots, r+s-1\}$

For the assignment sequence $\alpha_0, \alpha_1, \ldots, \alpha_{j-1}, \alpha_j$ the $(j-1)^{st}$-ball (with assignment value α_{j-1}) has placement value $\beta_{j-1} = t$ with $r \le t < r+s$ while the j^{th}-ball (with assignment value α_j) has placement value $\beta_j = r+s$ (modulo i). For the assignment sequence $\alpha_0, \alpha_1, \ldots, \alpha_j, \alpha_{j-1}$ the $(j-1)^{st}$-ball (with assignment value α_j) has placement value $\beta_j = t$ while the j^{th}-ball (with assignment value α_{j-1}) has placement value $\beta_{j-1} = r+s$ (modulo i). ■

Thus, we may consider P_c to be a mapping from assignment vectors $\underline{a} \in \mathbf{A}(i,j)$ to placement vectors $\underline{b} \in \mathbf{B}_c(i,j)$.

We consider next the calculation of the number $N(\underline{b})$ of assignment vectors $\underline{a}$ which satisfy the equation

(3.3) $P_c(\underline{a}) = \underline{b} \qquad \underline{b} \in \mathbf{B}_c(i,j)$

for $0 \le j \le ic$. When $j = ic$, $\mathbf{B}_c(i,j)$ contains the single vector (c, c, ..., c) and each element of $\mathbf{A}(i,ic)$ is a solution of equation (3.3); henceforth we assume $0 \le j < ic$. In this case, there exists (at least one) index r with $0 \le r < i$ such that URN_r is not full. The cyclic arrangement of the urns and Lemma 3.1 imply that the cyclic shift operator

$$\sigma(a_0, a_1, \ldots, a_{i-1}) = (a_1, a_2, \ldots, a_{i-1}, a_0)$$

commutes with P_c

$$P_c \sigma = \sigma P_c$$

so that if $\underline{a}$ satisfies equation (3.3) then

$$P_c(a_{r+1}, \ldots, a_{i-1}, a_0, \ldots, a_r) = (b_{r+1}, \ldots, b_{i-1}, b_0, \ldots, b_r).$$

Thus when counting the solutions of $P_c(\underline{a}) = \underline{b}$ we may assume that the "rightmost" urn is not filled ($b_{i-1} < c$). In this case we have the explicit formula for P_c

$$\begin{aligned}
b_0 &= \min\{c, a_0\} \\
b_1 &= \min\{c, a_1 + (a_0 - c)^+\} \\
&\ldots \\
&\ldots \\
&\ldots \\
b_{i-1} &= \min\{c, (\ldots ((a_0 - c)^+ + a_1 - c)^+ + \ldots)^+ + a_{i-1}\}
\end{aligned}$$

where $z^+ = \max\{z, 0\}$.

If $(b_0, b_1, \ldots, b_{i-1})$ is a placement vector with $b_{i-1} < c$ let $r_0, r_1, \ldots, r_{s-1}$

$$0 \le r_0 < r_1 < \ldots < r_{s-1} = i-1$$

denote the urns which are not full

$$b_{r_t} < c \quad 0 \le t < s$$

$$b_i = c \quad i \notin \{r_0, r_1, \ldots, r_{s-1}\}$$

The occupancy discipline implies that $(b_{r_{t-1}+1}, b_{r_{t-1}+2}, \ldots, b_{r_t})$ is determined only by the values of $(a_{r_{t-1}+1}, a_{r_{t-1}+2}, \ldots, a_{r_t})$

$$P_c(a_{r_{t-1}+1}, a_{r_{t-1}+2}, \ldots, a_{r_t}) = (b_{r_{t-1}+1}, b_{r_{t-1}+2}, \ldots, b_{r_t})$$

so that

$$N(\underline{b}) = \prod_{0 \le t < s} N(b_{r_{t-1}+1}, b_{r_{t-1}+2}, \ldots, b_{r_t})$$

Thus the problem of calculating $N(\underline{b})$ reduces further to counting assignment vectors $\underline{a}$ which fill all but the "rightmost" urn in a system of urns.

Define the counting functions $\nu_c(i,j)$ and $\gamma_c(i,j)$ as follows:

Definition 3.1: For $1 \le i < \infty$ let $\nu_c(i,j)$ denote the number of i-tuples $(a_0, a_1, \ldots, a_{i-1})$ of non-negative integers which satisfy

$$a_0 + \ldots + a_{t-1} \ge tc \qquad 1 \le t < i$$

$$a_0 + \ldots + a_{i-1} = j$$

When $(i-1)c \le j < ic$, $\nu_c(i,j)$ is *the number of assignment vectors for j balls into i urns each of capacity c for which all but the right most urn are full.*

Definition 3.2: For $1 \le i < \infty$ and $0 \le j < ic$, let $\gamma_c(i,j)$ denote the number of i-tuples $(a_0, a_1, \ldots, a_{i-1})$ of non-negative integers which satisfy

$$a_0 + \ldots + a_{i-1} = j$$

$$(\ldots((a_0-c)^+ + a_1-c)^+ + \ldots)^+ + a_{i-1} < c$$

We define $\gamma_c(i,j)$ for other values of j by

$$\gamma_c(i,j) = \begin{cases} 0 & \text{for } j \ge ic \text{ and } i \ne 0 \\ 1 & \text{if } i = j = 0. \end{cases}$$

$\gamma_c(i,j)$ is *the number of assigment vectors for j balls into i urns each of capacity c in which urn $i-1$ is not full.*

In the next group of lemmata we obtain formulae for and relationships between the counting functions $\nu_c(i,j)$ and $\gamma_c(i,j)$.

Lemma 3.2:

(3.4) $\nu_c(i,j) = C(j+i,j+1) - (c+1)C(j+i-1,j+1)$

$= C(j+i-1,i-1) - cC(j+i-1,j+1)$[6]

Proof: The proof is by induction on i; for i = 1 we have $\nu_c(1,j) = 1$ which is consistent with equation (3.4). For i > 1, $\nu_c(i,j)$ obviously satisfies the recurrence relation

$$\nu_c(i,j) = \sum_{(i-1)c \le t \le j} \nu_c(i-1,t)$$

which is obtained by considering the possible values for

$$t = a_0 + a_1 + \ldots + a_{i-2}.$$

By the induction hypothesis

$$\nu_c(i,j) = \sum_{(i-1)c \le t \le j} C(i-1+t,t+1) - (c+1)C(i-2+t,t+1)$$

which yields equation (3.4) using the formula $C(i+j+1,j) = \sum_{0 \le t \le j} C(i+t,t)$ ■

Lemma 3.3: For i > 0, $\gamma_c(i,j)$ satisfies

(3.5) $\gamma_c(i,j) = \sum_{1 \le t \le i} \sum_{(t-1)c \le s < tc} \nu_c(t,s)\gamma_c(i-t,j-s)$

Proof: Let $\mathbf{B}_{c,0}(i,j)$ denote the subset of $\mathbf{B}_c(i,j)$ consisting of placement vectors which do not fill the rightmost urn ($b_{i-1} < c$). The inverse image $P^{-1}(\mathbf{B}_{c,0}(i,j))$ is of cardinality $\gamma_c(i,j) = |P^{-1}(\mathbf{B}_{c,0}(i,j))|$. Partition $\mathbf{B}_{c,0}(i,j)$ as follows; for fixed t and s, $\mathbf{B}_c(i,j \mid t,s)$ consists of all vectors $\underline{b}$ in $\mathbf{B}_{c,0}(i,j)$ for which

---

[6] Note that equation (3.4) expresses $\nu_c(i,j)$ as the difference of the number of *partitions of j into i parts* and c times the number of *partitions of j+1 into i−1 parts*. We have not found a direct combinatorial proof of this formula. Equation (3.4) also implies that $\nu_c(i,j)$ satisfies the recurrence $\nu_c(i,j) = \nu_c(i,j-1) + \nu_c(i-1,j)$.

$b_{i-t} = \ldots = b_{i-2} = c$ — *URN_{i-t}, URN_{i-t+1}, ..., URN_{i-2} are full*

$s = b_{i-t} + \ldots + b_{i-1}$ — *the rightmost t urns contain s balls*

$b_{i-t-1} < c$ — *URN_{i-t-1} is not full*

$j-s = b_0 + \ldots + b_{i-t-1}$ — *the leftmost $i-t$ urns contain $j-s$ balls*

$\mathbf{B}_c(i,j \mid t,s)$ may be identified with the Cartesian product of $\mathbf{B}_c(i\text{-}t,j\text{-}s)$ and $\mathbf{B}_c(t,s \mid t,s)$ in the sense that each $\underline{b} \in \mathbf{B}_c(i,j \mid t,s)$ is the concatention of the two vectors

$$(b_0, b_1, \ldots, b_{i-t-1}) \in \mathbf{B}_s(i\text{-}t,j\text{-}s) \qquad (b_{i-t}, b_{i-t+1}, \ldots, b_{i-1}) \in \mathbf{B}_c(t,s \mid t,s).$$

Since $N(\underline{b}) = N(b_0,b_1,\ldots,b_{i-t-1})N(b_{i-t},b_{i-t+1},\ldots b_{i-1})$ it follows that

$$| P^{-1}(\mathbf{B}_c(i,j \mid t,s)) | = | P^{-1}(\mathbf{B}_c(i\text{-}t,j\text{-}s)) | \times | P^{-1}(\mathbf{B}_c(t,s \mid t,s)) | = \gamma_c(i\text{-}t,j\text{-}s)\nu_c(t,s)$$

from which equation (3.5) follows. ■

Equation (3.5) is the (two-dimensional) convolution of the sequences

$$\{\gamma_c(i,j) : 0 \le j < ic, 0 \le i < \infty\}$$

$$\{\nu_c(i,j) : (i-1)c \le j < ic, 1 \le i < \infty\}.$$

Convolution of sequences corresponds to the multiplication of their respective generating functions. Thus, we introduce the generating functions

$$H_{i,c}(z) = \sum_{(i-1)c \le j < ic} \nu_c(i,j) z^j \qquad H_c(z,w) = \sum_{1 \le i < \infty} w^i H_{i,c}(z)$$

$$G_{i,c}(z) = \sum_{0 \le j < ic} \gamma_c(i,j) z^j \qquad G_c(z,w) = \sum_{0 \le i < \infty} w^i G_{i,c}(z)$$

Since both $\nu_c(i,j)$ and $\gamma_c(i,j)$ are bounded above by $C(i+j-1,i-1)$, the generating functions $H_c(z,w)$ and $G_c(z,w)$ are analytic in a neighborhood of $(z,w) = (0,0)$.

The term $\sum_{(t-1)c \le s < tc} \nu_c(t,s)\gamma_c(i-t,j-s)$ is the coefficient of z^j in $H_{t,c}(z)G_{i-t,c}(z)$ and the right-hand-side of equation (3.5) is the coefficient of $w^i z^j$ in $H_c(z,w)G_c(z,w)$ for $1 \le i < \infty$. We thus have proved

Lemma 3.4: $G_c(z,w) = \dfrac{1}{1 - H_c(z,w)}$

It remains to determine $H_c(z,w)$; for this purpose we use the following

Lemma 3.5: If $|\zeta| < c^c/(c+1)^{c+1}$, the equation

$$\zeta = (x-1)/x^{c+1}$$

has a root $X(\zeta)$ which is analytic in ζ in the open disk about $\zeta = 0$ of radius $c^c/(c+1)^{c+1}$ and takes the value 1 at $\zeta = 0$. Moreover, if $f(z)$ is analytic in the open disk $\{z : |z-1| < 1/c\}$ of radius $1/c$ about $z = 1$, then

$$(3.6)\quad f(X(\zeta)) = f(1) + \sum_{1 \le i < \infty} (\zeta^i/i!)\,[(d^{i-1}/du^{i-1})\, f'(u)u^{i(c+1)}]_{|u=1}$$

where $'$ denotes differentiation.

Proof: For each ζ, there are $c+1$ roots of the equation $\zeta x^{c+1} = x - 1$. When $|\zeta| < c^c/(c+1)^{c+1}$, the inequality

$$|\zeta x^{c+1}| < |x - 1|$$

is satisfied for all x on the boundary of the disk (in the complex plane) of radius $1/c$ about $x = 1$

$$\{x : |x-1| < 1/c\}.$$

Thus by Rouche's Theorem [AHL], the equation $\zeta x^{c+1} = x - 1$ has a unique root $X(\zeta)$ in the interior of this disk. At $\zeta = c^c/(c+1)^{c+1}$, the equation $\zeta x^{c+1} = x - 1$ has the root $(c+1)/c$ of multiplicity two. Equation (3.6) is an immediate consequence of Lagrange's Theorem [WW]. ∎

To evaluate $H_c(z,w)$ we use the special case of equation (3.6) with $f(z) = z^s$:

$$(3.7)\quad (X(\zeta))^s = \sum_{0 \le i < \infty} \zeta^i \,\frac{s}{i(c+1)+s}\, C(i(c+1)+s,i)$$

By Lemma 3.2

$$\nu_c(i,j) = C(i+j,i-1)[1 - (c+1)\frac{i-1}{i+j}]$$

so that

$$H_c(z,w) = w\sum_{0\le s<c} z^s \sum_{0\le i<\infty} (wz^c)^i \frac{s+1}{i(c+1)+s+1} C(i(c+1)+s+1,i)$$

$$= (w/z)\sum_{0\le s<c} (zX(wz^c))^{s+1}$$

$$= \frac{wX(wz^c)-X(wz^c)+1}{1-zX(wz^c)}.$$

Lemma 3.4 now yields

$$G_c(z,w) = \frac{1-zX(wz^c)}{(1-w-z)X(wz^c)}$$

Finally, we define the sequence of numbers $\{\lambda_c(i,j,k) : 1 \le i < \infty, 0 \le j < ic, 0 \le k\}$ by

$$(3.8)\quad \lambda_c(i,j,k) = \sum_{\{t,s:1\le t\le i,(t-1)c\le s<tc,s+t>k\}} \nu_c(t,s)\gamma_c(i-t,j-s)$$

$\lambda_c(i,j,k)$ counts the number of assignment vectors $(a_0, a_1, \ldots, a_{i-1})$ for which for some s,t with $s + t > k$

$b_{i-1} < c$	URN_{i-1} *is not full*
$b_{i-t} = b_{i-t+1} = \ldots = b_{i-2} = c$	$URN_{i-t}, URN_{i-t+1}, \ldots, URN_{i-2}$ *are full*
$b_{i-t} + b_{i-t+1} + \ldots + b_{i-1} = s$	$URN_{i-t}, URN_{i-t+1}, \ldots, URN_{i-1}$ *contain s balls*
$b_{i-t-1} < c$	URN_{i-t-1} *is not full*
$b_0 + b_1 + \ldots + b_{i-t-1} = j-s$	$URN_0, URN_1, \ldots, URN_{i-t-1}$ *contain j−s balls*

We shall show in Section 5 that, apart from the normalization constant $C(i+j,i)$, $\lambda_c(i,j,k)$ is the probability that an insertion will require the movement of k records in a table with $i = n_{k-1}$, $j = n-n_{k-1}$ and $c = m_k-1$. To calculate the expected number of moves required for an insertion we need to compute

$$\sum_k k\lambda_c(i,j,k)$$

For this calculation, we introduce the generating functions

$$\Lambda_{i,j,c}(u) = \sum_{k\geq 0} \lambda_c(i,j,k)u^k$$

$$\Lambda_c(z,w,u) = \sum_{1\leq i<\infty} \sum_{0\leq j<ic} \Lambda_{i,j,c}(u)w^i z^j$$

From equation (3.6) we obtain

Lemma 3.6: $\Lambda_{i,j,c}(u) = \sum_{\{s,t:1\leq t\leq i,(t-1)c\leq s<tc\}} \nu_c(t,s)\gamma_c(i-t,j-s)\ (1-u^{s+t})/(1-u)$

$$(3.9)\qquad \Lambda_c(z,w,u) = G_c(z,w)\ \frac{H_c(z,w) - H_c(uz,uw)}{1-u}$$

Proof: We have

$$H_c(uz,uw) = \sum_k u^k \sum_{\{i,j:1\leq i<\infty,(i-1)c\leq j<ic,\ j+i=k\}} \nu_c(i,j)w^i z^j$$

so that

$$[H_c(z,w)-H_c(uz,uw)]/(1-u) = \sum_k u^k \sum_{\{i,j:1\leq i<\infty,(i-1)c\leq j<ic,\ j+i>k\}} \nu_c(i,j)w^i z^j$$

from which equation (3.9) easily follows. ■

Equation (3.9) shows that to calculate the expected number of moves to insert a record, we need to calculate the coefficient of $w^i z^j$ in

$$0.5G_c(z,w)\ [(\partial^2/\partial v^2)\ H_c(uz,uw)]_{|\,u=1}$$

The calculation is straightfoward albeit tedious and is given in an appendix. The result is

Theorem 3.7: When $j = tc + T$ with

$$0 \le T < c \quad 0 \le t < i$$

$$(3.10) \quad \sum_k k\lambda_c(i,j,k) = -C(i+j,i) + \sum_{0\le k\le t}\sum_{kc\le s\le j} C(k+s,k)C(i-k+j-s,j-s)$$

4. The Relationship Between Key Insertion And The Occupancy Discipline

In this section we provide a correspondence between *the insertion of keys in a sparse table* (as defined in Section 2) and *the placement of balls in a system of urns* (as defined in Section 3). These processes differ in two respects:

- balls are indistinguishable, while keys are labelled by their values and are hence distinguishable.
- if the t^{th}-ball is assigned to URN_{α_t} which presently contains $c = m_k - 1$ balls, the t^{th} ball is placed in the first urn "to the right" of URN_{α_t} which contains less than c balls; if the t^{th}-key x_t is to be inserted at (virtual) address s which presently contains a genuine key, y_s rather than x_t is moved "to the right" and x_t is inserted at (virtual) address s.

Definition 4.1: Let the sparse table with state vector $\underline{y} = (y_0, y_1, \ldots, y_{m-1})$ with $m = n_{k-1}m_k$ result from the insertion of the n keys $x_0, x_1, \ldots, x_{n-1}$ $(n_{k-1} < n \le n_k)$, PB the value of the head of table pointer and $\underline{z} = (z_0, z_1, \ldots, z_{m-1})$ the corresponding state of memory.

The *t^{th}-block* (of logical addresses) in $\underline{z}$ is $tm_k, tm_k+1, \ldots, (t+1)m_k-1$.

Associate with the *insertion sequence* $x_0, x_1, \ldots, x_{n-1}$

(i) a sequence of *assignment values*

$$\alpha(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) \qquad 0 \le s < n-n_{k-1}$$

(ii) an *assignment vector* $\underline{a} = (a_0, a_1, \ldots, a_{n_{k-1}-1})$,

(iii) a sequence of *placement values*

$$\beta(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) \qquad 0 \le s < n-n_{k-1}$$

(iv) a *placement vector* $\underline{b} = (b_0, b_1, \ldots, b_{n_{k-1}-1})$

defined as follows: let the permutation $\eta = (\eta(0), \eta(1), \ldots, \eta(n_{k-1}-1))$ of the integers $0, 1, \ldots, n_{k-1}-1$ sort the key values $x_0, x_1, \ldots, x_{n_{k-1}-1}$

$$x_{\eta(0)} < x_{\eta(1)} < \ldots < x_{\eta(n_{k-1}-1)}$$

and define

$$\alpha(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) = \begin{cases} 0 & \text{if } x_{n_{k-1}+s} < x_{\eta(0)} \text{ or } x_{n_{k-1}+s} > x_{\eta(n_{k-1}-1)} \\ t & \text{if } x_{\eta(t-1)} < x_{n_{k-1}+s} < x_{\eta(t)} \text{ with } 1 \le t < n_{k-1} \end{cases}$$

a_t : number of indices s, $0 \le s < n-n_{k-1}$ for which $\alpha(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) = t$

$\beta(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1})$: the index of the block in $\underline{z}$ in which the key $x_{n_{k-1}+s}$ is inserted

b_t : number of placement values equal to t.

It is clear from these definitions that

(i) $\alpha(x_{n_{k-1}} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}), \alpha(x_{n_{k-1}+1} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}), \ldots, \alpha(x_{n-1} \mid x_0, x_1, \ldots, x_{n_{k-1}-1})$

constitutes a sequence of assignment values in the sense of Section 3

$$0 \le \alpha(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) < n_{k-1} \qquad (0 \le s < n-n_{k-1})$$

(ii) $\beta(x_{n_{k-1}} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}), \beta(x_{n_{k-1}+1} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}), \ldots, \beta(x_{n-1} \mid x_0, x_1, \ldots, x_{n_{k-1}-1})$

constitutes a sequence of placement values in the sense of Section 3

$$0 \le \beta(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) < n_{k-1} \qquad (0 \le s < n-n_{k-1})$$

(iii) $\underline{a} \in \mathbf{A}(n_{k-1}, n-n_{k-1})$ is the assigment vector determined by the assignment sequence

$$\{\alpha(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) : 0 \le s < n-n_{k-1}\}$$

in the sense of Section 3, and

(iv) $\underline{b} \in \mathbf{B}_{m_{k-1}}(n_{k-1}, n-n_{k-1})$ is the placement vector determined by the placement sequence

$$\{\beta(x_{n_{k-1}+s} \mid x_0, x_1, \ldots, x_{n_{k-1}-1}) : 0 \le s < n-n_{k-1}\}$$

Let I_{n_{k-1},m_k} denote the mapping from an insertion sequence $x_0, x_1, \ldots, x_{n-1}$ to its assignment vector $\underline{a} = (a_0, a_1, \ldots, a_{n_{k-1}-1})$

$$I_{n_{k-1},m_k}(x_0, x_1, \ldots, x_{n-1}) = (a_0, a_1, \ldots, a_{n_{k-1}-1})$$

We will prove that $\underline{a}$ and $\underline{b}$ defined above are related by $P_{m_{k-1}}(\underline{a}) = \underline{b}$. Thus the insertion of the n keys $x_0, x_1, \ldots, x_{n-1}$ into the sparse table corresponds to the placement of $n-n_{k-1}$ balls into a system of n_{k-1} urns each of capacity $c = m_k - 1$.

Example 4.1: Referring to ***Example 2.1*** where:

$$n_{k-1} = 4 \quad m_k = 3 \quad n = 8 \quad (x_0, x_1, \ldots, x_7) = (7, 3, 1, 0, 5, 6, 4, 2).$$

- $(\eta(0), \eta(1), \eta(2), \eta(3)) = (3, 2, 1, 0)$
- $\underline{y} = (0, 0, 1, 1, 1, 2, 3, 3, 4, 5, 6, 7)$
- $\underline{z} = (7, 0, 0, 1, 1, 1, 2, 3, 3, 4, 5, 6)$ (PB = 1)
- $\underline{a} = (0, 0, 1, 3)$
- $\underline{b} = (1, 0, 1, 2)$

Lemma 4.1: With $\underline{a}$ and $\underline{b}$ as in Definition 4.1, $P_{m_{k-1}}(\underline{a}) = \underline{b}$.

Proof: The proof makes use of the equivalence of the following three statements; if binary search determines the (virtual) address s for which $z_{(s-1+PB)(\text{modulo } m)} = y_{s-1} < x_n < y_s = z_{(s+PB)\ (\text{modulo } m)}$, then

1. insertion of the key x_n causes some genuine key to be placed in the first location at or "to the right" of (virtual) address s in $\underline{y}$ which presently contains a dummy key.

2. insertion of the key x_n causes some genuine key to be placed in the first location at or "to the right" of (logical) address (s+PB) (modulo m) in $\underline{z}$ which presently contains a dummy key.

3. insertion of the key x_n causes a genuine key to be placed in the first block at or "to the right" of the t^{th}-block in $\underline{z}$ which presently contains less than $c = m_k - 1$ genuine keys where $t = \lfloor (s+PB)\ (\text{modulo } m) / m_k \rfloor$.

where $\lfloor\ \rfloor$ denotes the *integer part of*.

Suppose $x_{\eta(r-1)} < x_n < x_{\eta(r)}$ with $1 \leq r < n_{k-1}$.[7] so that the assignment value of x_n is r. If the placement value of x_n is $t = \lfloor (s+PB)\ (\text{modulo } m) / m_k \rfloor$, we must prove that $b_r = \ldots = b_{t-1} = m_k - 1 \geq b_t$. Since $z_{(s-1+PB)(\text{modulo } m)} < x_n < z_{(s+PB)\ (\text{modulo } m)}$ we must have

$$(4.1)\quad x_{\eta(r-1)} \leq z_{(s-1+PB)(\text{modulo } m)} < x_n < z_{(s+PB)(\text{modulo } m)} \leq x_{\eta(r)}$$

But the key $x_{\eta(r)}$ was stored at logical address $(r+1)m_k - 1$ at the last reconfiguration, so that equation (4.1) implies

- if $y_s = z_{(s+PB)\ (\text{modulo } m)}$ is a dummy key, then the genuine key $x_{\eta(t)}$ remains stored at logical address $(r+1)m_k - 1$ implying

$$t = \lfloor (s+PB)\ (\text{modulo } m) / m_k \rfloor = r$$

---

[7] We leave the reader to make the minor modifications required in the proof when $r = 0$.

- if y_s is a genuine key, then all of the keys stored at the logical addresses $rm_k - 1$, rm_k, ..., (s+PB) (modulo m) are genuine. In this case we either have

$$z_{(s+PB)\ (\text{modulo } m)} < z_{(s+PB+1)\ (\text{modulo } m)} < \cdots < z_{(s+PB+u-1)\ (\text{modulo } m)} < z_{(s+PB+u)\ (\text{modulo } m)}$$

$$z_{(s+PB+u)\ (\text{modulo } m)} = z_{(s+PB+u+1)\ (\text{modulo } m)} = x_{\eta(r)}$$

implying $t = r$, or

$$z_{(s+PB)\ (\text{modulo } m)} < z_{(s+PB+1)\ (\text{modulo } m)} < \cdots$$

$$< z_{(s+PB+u-1)\ (\text{modulo } m)} < z_{(s+PB+u)\ (\text{modulo } m)} = x_{\eta(r)}$$

implying $t > r$ and $b_r = \ldots = b_{t-1} = m_k - 1 \geq b_t$.

In either case *the first block at or "to the right" of the assigned block of key* x_n *(block r) which is not full* is the same as *the first block at or "to the right" of the* t^{th}*-block which is not full* which proves the lemma. ■

Suppose the insertion sequence $x_0, x_1, \ldots, x_{n-1}$ is some permutation of the integers 0, 1, ..., n−1. The mapping I_{n_{k-1},m_k} from insertion sequences to assignment vectors

$$I_{n_{k-1},m_k}(x_0, x_1, \ldots, x_{n-1}) = \underline{a} \in A(n_{k-1}, n-n_{k-1})$$

is clearly many-to-one; indeed, any insertion sequence formed by permuting the values in the two subsets

$$\{x_0, x_1, \ldots, x_{n_{k-1}-1}\} \qquad \{x_{n_{k-1}}, x_{n_{k-1}+1}, \ldots, x_{n-1}\}$$

yields the same assignment vector $\underline{a}$.

Example 4.2: For assignment vector $\underline{a} = (0, 0, 1, 3)$ of ***Example 4.1*** with $n_{k-1} = 4$, $n = 8$ and $m_k = 3$

- $(x_{\eta(0)}, x_{\eta(1)}, x_{\eta(2)}, x_{\eta(3)}) = (0, 1, 3, 7)$
- $\{x_0, x_1, x_2, x_3\} = \{0, 1, 3, 7\}$
- $\{x_4, x_5, x_6, x_7\} = \{2, 4, 5, 6\}$
- there are $4! \times 4!$ insertion squences which produce the assignment vector $\underline{a} = (0, 0, 1, 3)$.

The cardinality of the inverse image of $\underline{a}$ under I_{n_{k-1},m_k} – the number cf insertion sequences (of the key values 0, 1, ..., n−1) which produce the same assignment vector $\underline{a}$ is given by the next lemma.

Lemma 4.2: For each $\underline{a} \in A(n_{k-1}, n-n_{k-1})$ there are $(1+a_0)n_{k-1}!(n-n_{k-1})!$ insertion sequences $x_0, x_1, ..., x_{n-1}$ – permutations of the key values 0, 1, ..., n−1 which yield the assignment vector $\underline{a}$.

Proof: Let $\underline{a} \in A(n_{k-1}, n-n_{k-1})$ and suppose

$$I_{n_{k-1},m_k}(x_0,x_1,...,x_{n-1}) = \underline{a}$$

If the permutation η of $\{0, 1, ..., n_{k-1}-1\}$ sorts $x_0, x_1, ..., x_{n_{k-1}-1}$

$$x_{\eta(0)} < x_{\eta(1)} < ... < x_{\eta(n_{k-1}-1)}$$

then

- the integers $\{0,1,...,x_{\eta(0)}-1\} \cup \{x_{\eta(n_{k-1}-1)}+1, ...,n-1\}$ have assignment value 0,
- the integers $\{x_{\eta(t-1)}+1, ..., x_{\eta(t)}-1\}$ have assignment value t for $1 \leq t < n_{k-1}$.

Thus

(4.2a) $a_0 = x_{\eta(0)} + n - 1 - x_{\eta(n_{k-1}-1)}$

(4.2b) $a_t = x_{\eta(t)} - x_{\eta(t-1)} - 1 \qquad 1 \le t < n_{k-1}$.

Equation (4.2a) shows that $0 \le x_{\eta(0)} \le a_0$ so that if $x_{\eta(0)} = a_0 - j$ for some j with $0 \le j \le a_0$, then

(4.3) $x_{\eta(t)} = a_0 + a_1 + \ldots + a_t + t - j \quad 1 \le t < n_{k-1}$

by equation (4.2b). Conversely, by setting $x_{\eta(0)} = a_0 - j$ for some j, $0 \le j \le a_0$, and defining a_t by means of equation (4.3), we obtain a set of possible values for $x_{\eta(0)}, x_{\eta(1)}, \ldots, x_{\eta(n_{k-1}-1)}$. ■

Remark 4.1: Note in passing the summation formula

(4.4) $$\sum_{\underline{a} \in A(n_{k-1}, n-n_{k-1})} |I_{n_{k-1}, n-n_{k-1}}^{-1}(\underline{a})| = \sum_{\underline{a} \in A(n_{k-1}, n-n_{k-1})} (1+a_0) n_{k-1}!(n-n_{k-1})!$$

$$= n!$$

which counts the n! insertion sequences formed from the key values $\{0, 1, \ldots, n-1\}$.

Remark 4.2: Lemma 4.2 characterizes the set $\Gamma_{\underline{a},t}$ of *possible key values* that may appear in the insertion subsequence $x_0, x_1, \ldots, x_{n_{k-1}-1}$. If a value in $\Gamma_{\underline{a},t}$ is specified for any $x_{\eta(t)}$, then the values of $x_{\eta(s)}$ for $0 \le s < n_{k-1}$, $s \ne t$ and $\{x_{n_{k-1}}, x_{n_{k-1}+1}, \ldots, x_{n-1}\}$ are then determined. This yields:

Corollary 4.3: For each assignment vector $\underline{a}$, index t ($0 \le t < n_{k-1}$) and (possible) key value $q \in \Gamma_{\underline{a},t}$, there are $n_{k-1}!(n-n_{k-1})!$ insertion sequences $x_0, x_1, \ldots, x_{n-1}$ such that

- $I_{n_{k-1}, n-n_{k-1}}(x_0, x_1, \ldots, x_{n-1}) = \underline{a}$
- $q = x_{\eta(t)}$, where η is the permutation that sorts $x_0, x_1, \ldots, x_{n_{k-1}-1}$.

We will need a slight extension of Lemma 4.2 in Section 5.

Example 4.3: For the assignment vector $\underline{a} = (0, 0, 1, 3)$ with $n_{k-1} = 4$, $n = 8$, $m_k = 3$ consider the insertion sequences which correspond to the cyclic shifts of $\underline{a}$:

$$I_{4,4}(x_{0,0}, x_{0,1}, \ldots, x_{0,7}) = \sigma^0\underline{a} = (0, 0, 1, 3)$$

$$I_{4,4}(x_{1,0}, x_{1,1}, \ldots, x_{1,7}) = \sigma^1\underline{a} = (0, 1, 3, 0)$$

$$I_{4,4}(x_{2,0}, x_{2,1}, \ldots, x_{2,7}) = \sigma^2\underline{a} = (1, 3, 0, 0)$$

$$I_{4,4}(x_{3,0}, x_{3,1}, \ldots, x_{3,7}) = \sigma^3\underline{a} = (3, 0, 0, 1)$$

Suppose the permutations $\eta_u = (\eta_u(0), \eta_u(1), \eta_u(2), \eta_u(3))$ sorts the values $x_{u,0}, x_{u,1}, x_{u,2}, x_{u,3}$ for $0 \le u < 8$

$$x_{u,\eta_u(0)} < x_{u,\eta_u(1)} < x_{u,\eta(2)} < x_{u,\eta_u(3)} \qquad 0 \le u < 4$$

The possible values of $x_{u,\eta_u(t)}$ are shown in the following table:

u	$x_{u,\eta_u(0)}$	$x_{u,\eta_u(1)}$	$x_{u,\eta_u(2)}$	$x_{u,\eta_u(3)}$
0	{0}	{1}	{3}	{7}
1	{0}	{2}	{6}	{7}
2	{0,1}	{4,5}	{5,6}	{6,7}
3	{0,1,2,3}	{1,2,3,4}	{2,3,4,5}	{4,5,6,7}

If we rearrange the table – tabulating $x_{u,\eta_u((i-u) \text{ modulo } 4}$ in the i^{th}-column

u	$x_{u,\eta_u(0-u)}$	$x_{u,\eta_u(1-u)}$	$x_{u,\eta_u(2-u)}$	$x_{u,\eta_u(3-u)}$
0	{0}	{1}	{3}	{7}
1	{7}	{0}	{2}	{6}
2	{5,6}	{6,7}	{0,1}	{4,5}
3	{1,2,3,4}	{2,3,4,5}	{4,5,6,7}	{0,1,2,3}

we note that in each column each key value in {0, 1, ..., 7} appears exactly once. Thus specifying a key value determines the cyclic shift. The general statement is given next in

Lemma 4.4: Let $\underline{a} \in A(n_{k-1}, n-n_{k-1})$ and suppose

$$I_{n_{k-1},m_k}(\underline{x}_u) = \sigma^u \underline{a} \qquad 0 \le u < n_{k-1}$$

where

- σ is the shift operator

$$\sigma^u(a_0, a_1, \ldots, a_{n_{k-1}-1}) = (a_u, a_{u+1}, \ldots, a_{n_{k-1}-1}, a_0, a_1, \ldots, a_{u-1})$$

- $\underline{x}_u = (x_{u,0}, x_{u,1}, \ldots, x_{u,n-1})$ is an insertion sequence of the key values $\{0, 1, \ldots, n-1\}$

Let $\eta_u = (\eta_u(0), \eta_u(1), \ldots, \eta_u(n_{k-1}-1))$ be the permutation of $0, 1, \ldots, n_{k-1}-1$ sorting $x_{u,\eta_u(0)} < x_{u,\eta_u(1)} < \ldots < x_{u,\eta_u(n_{k-1}-1)}$. If $\Gamma_{\underline{a},t,u}$ is the set of *possible values* for $x_{\eta_u(t-u)}$, then $\{\Gamma_{\underline{a},t,u} : 0 \le u < n_{k-1}\}$ is a partition of the integers $\{0, 1, \ldots, n-1\}$; each key value in $\{0, 1, \ldots, n-1\}$ is a possible value for $x_{u,\eta_u(t-u)}$ for some u, $0 \le u < n_{k-1}$.

Proof: We argue as in the proof of Lemma 4.2 that the possible values for $x_{u,\eta_u(t-u)}$ are

$$\Gamma_{\underline{a},t,u} = \begin{cases} \{a_{u+1}+a_{u+2}+\ldots+a_{t-1}+j-u-1+s : 0 \le s \le a_u\} & \text{if } 0 \le u < t \\ \{a_0+a_1+\ldots+a_{t-1}+a_{u+1}+a_{u+2}+\ldots+a_{n_{k-1}-1}+n_{k-1}-(u-t+1)+s : 0 \le s \le a_u\} & \text{if } t \le u < n_{k-1} \end{cases}$$

It is easily seen that the sets $\{\Gamma_{\underline{a},t,u} : 0 \le u < n_{k-1}\}$ form a partition of $\{0, 1, \ldots, n-1\}$. ■

Remark 4.3: The hypothesis that the key values are the integers $0, 1, \ldots, n-1$ was made to simplify the statement of the results; given any set of key values

$$k_0 < k_1 < \ldots < k_{n-1}$$

and an assignment vector $\underline{a}$, there is precisely one cyclic shift $\sigma^u \underline{a}$ of $\underline{a}$ for which k_t appears in $(x_0, x_1, \ldots, x_{n_{k-1}-1}) \in (I_{n_{k-1},n-n_{k-1}})^{-1}(\underline{a})$.

5. The Distribution Of The Number Of Moves

Assume now that

- $n_{k-1} < n < n_k$
- the table $(Y_0, Y_1, ..., Y_{m-1})$[8] $(m = n_{k-1}m_k)$ has been constructed by the insertion of n keys $X_0, X_1, ..., X_{n-1}$ using the sparse table scheme
- an $(n+1)^{st}$-key X_n is to be inserted into $(Y_0, Y_1, ..., Y_{m-1})$

Let $\pi = (\pi(0), \pi(1), ..., \pi(n))$ be the permutation of $\{0, 1, 2, ..., n\}$ which sorts the keys $X_0, X_1, ..., X_n$, namely,

$$X_{\pi(0)} < X_{\pi(1)} < ... < X_{\pi(n)}$$

Our probabilistic model describing the insertion of keys assumes each of the (n+1)! permutations are of equal probability. There is no loss of generality in further assuming that the key values $\{X_0, X_1, ..., X_n\}$ are the integers $\{0, 1, ..., n\}$.

Let $M_{m_k}(n_{k-1},n-n_{k-1})$ denote the number of moves needed to insert X_n into the table. If $X_n = i$ and $M_{m_k}(n_{k-1},n-n_{k-1}) = e$, binary search determines an address s for which $Y_{s-1} < i < Y_s$ if $0 < i < n$ and $s = 0$ if $i = 0$ or n. Since e moves are required to insert $X_n = i$, it follows that the (virtual) addresses s, s+1, ..., s+e−1 (modulo m) contain genuine keys

$$Y_s = i+1, Y_{s+1} = i+2, ..., Y_{s+e-1} = i+e$$

where the addition of the key values i+1, i+2, ..., i+e−1 is modulo n+1 and Y_{s+e} is a dummy key with value i+e+1 (modulo n+1). Define t_R, $1 \le t_R < m_k$ by

---

[8] We generally adhere to the convention of using capital letters to denote random variables. Thus $(Y_0, Y_1, ..., Y_{m-1})$ is the "random" table that results when the "random" sequence of keys $X_0, X_1, ..., X_{n-1}$ are inserted.

$$Y_{s+e} = Y_{s+e+1} = \ldots = Y_{s+e+t_R} = i+e+1 \text{ (modulo } n+1)$$

$$Y_{s+e+1+t_R} = i+e+2 \text{ (modulo } n+1)$$

so that $s+e+t_R$ (modulo m) is the *(virtual) address to the right of s* containing the genuine key with value $i+e+1$ (modulo $n+1$). Also define t_L by

$$Y_{s-t_L} = i-t_L, \ldots, Y_{s-1} = i-1$$

$$Y_{s-t_L-1} = Y_{s-t_L}$$

so that $s-t_L-1$ (modulo m) is the *first (virtual) address to the left of s* containing a dummy key. It is clear that $t_L \geq 1$.

Since Y_{s-t_L} and Y_{s+e+t_R} are genuine keys which have a dummy key with the same value "to their left", we may conclude that Y_{s-t_L} and Y_{s+e+t_R} were inserted in the table before the last reconfiguration and from this deduce that

- $t_L + e + t_R$ must be a multiple of m_k

$$t_L + e + t_R = Nm_k$$

- exactly $N-1$ of the keys $\{i-t_L+1, \ldots, i-1, i+1, \ldots, i+e\}$ (modulo $n+1$) were inserted before the last reconfiguration, and

- the remaining $t_L + e - N$ of these keys were inserted after the last reconfiguration.

Thus, $n_{k-1} - N \leq n - (e+t_L)$.

We have thus decomposed the event $\{M_{m_k}(n_{k-1}, n-n_{k-1}) = e, X_n=i\}$ into disjoint events $E(i,e,t_R,t_L,N)$. The probability of this latter event is the number of insertion sequences which produce a table state with the parameters i, e, t_R, t_L, N as described above divided by the normalizing factor $(n+1)!$. To make this count we use the correspondence in Section 4; let $\underline{b} = (b_0, b_1, \ldots, b_{n_{k-1}-1})$ be any placement vector satisfying

(i) $b_{n_{k-1}-N} = b_{n_{k-1}-N+1} = \ldots = b_{n_{k-1}-2} = m_k - 1$

(ii) $b_{n_{k-1}-1} = m_k - 1 - t_R < m_k - 1$

(iii) $b_{n_{k-1}-N} + b_{n_{k-1}-N+1} + \ldots + b_{n_{k-1}} = N(m_k-1) - t_R = t_L + e - N$

(iv) $b_{n_{k-1}-N-1} < m_k - 1$

(v) $\sum_{0 \le j < n_{k-1}-N} b_j = n - n_{k-1} - (t_L + e - N)$

The placement vector for a table described by the event $E(i,e,t_L,t_R,N)$ is clearly some cyclic shift of a placement vector $\underline{b}$ satisfying *(i)-(v)* above.

The number of assignment vectors $\underline{a}$ for which $P_{m_{k-1}}(\underline{a}) = \underline{b}$ satisfies *(i)-(v)* is

$$\nu_{m_{k-1}}(N, t_L + e - N)\, \gamma_{m_{k-1}}(n_{k-1} - N, n - n_{k-1} - (t_L + e - N)).$$

We now use Lemma 4.4; if $\underline{a}$ is any assignment vector for which $P_{m_{k-1}}(\underline{a}) = \underline{b}$ satisfies *(i)-(v)*, then

- there exists a unique cyclic shift $\sigma^u \underline{a}$ of $\underline{a}$ (*u depending on* $\underline{a}$) such that the key $i+e+1$ is a possible value in $\{X_0, X_1, \ldots, X_{n_{k-1}-1}\}$
- there are $n_{k-1}!(n-n_{k-1})!$ insertion sequences in $E(i,e,t_L,t_R,N)$ corresponding to each $\sigma^j \underline{a}$.

We have thus proved that $\Pr\{M_{m_k}(n_{k-1}, n-n_{k-1}) = e,\ X_n = i\}$ is the summation

$$(5.1) \quad \frac{n_{k-1}!(n-n_{k-1})!}{(n+1)!} \sum_{t_L, N} \nu_{m_{k-1}}(N, t_L + e - N)\, \gamma_{m_{k-1}}(n_{k-1} - N, n - n_{k-1} - (t_L + e - N))$$

over the admissible values of t_L and N. Finally, the summation in equation (5.1) is independent of the value of the key X_n so that we obtain

Theorem 5.1: When $n_{k-1} \le n < n_k$

$$(5.2) \quad \Pr\{M_{m_k}(n_{k-1}, n-n_{k-1}) = e\} =$$

$$\frac{(n-n_{k-1})!\, n_{k-1}!}{n!} \sum_{\{t,s:\, 1 \le t \le n_{k-1},\, (t-1)(m_k-1) \le s < t(m_k-1),\, s+t > e\}} \nu_{m_{k-1}}(t,s)\, \gamma_{m_{k-1}}(n_{k-1} - t, n - n_{k-1} - s)$$

Example 5.1: $m_k = 3,\ n_{k-1} = 5,\ n = 11,\ X_{11} = 5, e = 3,\ t_L = 1,\ t_R = 1,\ N = 2$

The vector $\underline{b} = (b_0, b_1, b_2, b_3, b_4)$ satisfies the conditions:

(i) $b_3 = 2$

(ii) $b_4 < 2$

(iii) $b_3 + b_4 = 3$

(iv) $b_2 < 2$

(v) $b_0 + b_1 + b_2 = 3$

We list below all solutions $\underline{b}$ of *(i)-(v)*, the corresponding assignment vectors $\underline{a}$, the cyclic shift $\sigma^u\underline{a}$ which permits $X_n+e+1 = 9$ to have been inserted before the last reconfiguration and the table state $\underline{y}$. (The keys in bold-face are those inserted before the reconfiguration.)

$\underline{b}$	$\underline{a}$	shift u	$\underline{y}$
(2,1,0,2,1)	(3,0,0,3,0)	0	(0,**1**,2,**2**,3,3,3,4,6,7,**8**,9,**9**,10,11)
	(3,0,0,2,1)	0	(0,**1**,2,**2**,3,3,3,4,6,7,8,9,**9**,10,11)
	(2,1,0,3,0)	0	(**0**,1,2,**2**,3,3,3,4,6,7,**8**,9,**9**,10,11)
	(2,1,0,2,1)	0	(**0**,1,2,**2**,3,3,3,4,6,7,8,9,**9**,10,11)
(2,0,1,2,1)	(2,0,1,3,0)	0	(**0**,1,1,**1**,2,3,3,3,4,6,7,**8**,9,**9**,10,11)
	(2,0,1,2,1)	0	(**0**,1,1,**1**,2,3,3,4,6,7,8,9,**9**,10,11)
(0,2,1,2,1)	(0,3,0,2,1)	1	(0,1,**2**,3,3,4,6,7,8,9,**9**,10,10,**10**,11)
	(0,2,1,2,1)	1	(0,**1**,2,3,3,4,6,7,8,9,**9**,10,10,**10**,11)
	(0,3,0,3,0)	1	(0,1,**2**,3,3,4,6,7,**8**,9,**9**,10,10,**10**,11)
	(0,2,1,3,0)	1	(0,**1**,2,3,3,4,6,7,**8**,9,**9**,10,10,**10**,11)
(1,2,0,2,1)	(1,2,0,3,0)	1	(0,1,**2**,3,3,3,4,6,7,**8**,9,**9**,10,11,**11**)
	(1,2,0,2,1)	1	(0,1,2,3,3,3,4,6,7,8,9,**9**,10,11,**11**)
(1,1,1,2,1)	(1,1,1,3,0)	1	(0,1,1,2,3,**3**,4,6,7,**8**,9,**9**,10,11,**11**)
	(1,1,1,2,1)	1	(0,1,**1**,2,3,3,4,6,7,8,9,**9**,10,11,**11**)

Equation (5.2) shows that

$$C(n,n_{k-1})\Pr\{M_{m_k}(n_{k-1},n-n_{k-1}) = e\} = \lambda_{m_k-1}(n_{k-1},n-n_{k-1},e)$$

so that

$$\Lambda_{m_{k-1}}(z,w,u) = \sum_{1\le i<\infty} w^i \sum_{0\le j<\infty} z^j \sum_{k\ge 0} u^k\, C(i+j,i)\, \Pr\{M_{m_k}(i,j)=k\}$$

Theorem 3.7 now yields

Theorem 5.2: When $t = \lfloor (n-n_{k-1})/(m_k-1) \rfloor$

(5.3) $E\{M_{m_k}(n_{k-1},n-n_{k-1})\} =$

$$-1 + (C(n,n_{k-1}))^{-1} \sum_{0\le i\le t} \sum_{i(m_k-1)\le j\le n-n_{k-1}} C(i+j,i)C(n-i-j,n-n_{k-1}-j)$$

In Figure 1 we plot $E\{M_4(i,j-1)\}$ for $1 \le j \le 3i$ and $i = 2, 3, \ldots, 20$.

Remark 5.1 $E\{M_{m_k}(n_{k-1},n-n_{k-1})\}$ is strictly increasing in n for $n_{k-1} \le n < m_k n_{k-1}$.

Proof: Let

$$T(n,n_{k-1},i,j) = C(n-i-j,n-n_{k-1}-j) \Big/ C(n,n_{k-1})$$

Then

$$T(n+1,n_{k-1},i,j) \Big/ T(n,n_{k-1},i,j) = (n+1-i-j)(n-n_{k-1}+1) \Big/ (n-n_{k-1}+1-j)(n+1) \ge 1 \ \blacksquare$$

Remark 5.2 $E\{M_{m_k}(n_{k-1},n_{k-1}(m_k-1)-1)\} = (n_{k-1}m_k-2)/2$.

Proof: The $n_{k-1}m_k-1$ possible states of the table after the insertion of the keys X_0, $X_1, \ldots, X_{n_{k-1}m_k-2}$ (a permutation of the integers $0, 1, \ldots, n_{k-1}m_k-2$) are

$$\underline{y}^{(r)} = (0, 1, \ldots, r-1, r, r, r+1, r+2, \ldots, n_{k-1}m_k-2)$$

with $0 \le r < n_{k-1}m_k-1$. A symmetry argument shows that each of these states is equally likely (and hence of probability $1/(n_{k-1}m_k-1)$). Inserting $X_{n_{k-1}m_{k-1}-1}$ into the table $\underline{y}^{(r)}$ requires an expected number of moves equal to

$$\Big(r + 0.5(n_{k-1}m_k-1)\ (n_{k-1}m_{k-1}-2)\Big) \Big/ n_{k-1}m_{k-1}$$

and hence averaging over the possible states $\{\underline{y}^{(r)} : 0 \le r < n_{k-1}m_k-1\}$ yields the result. ■

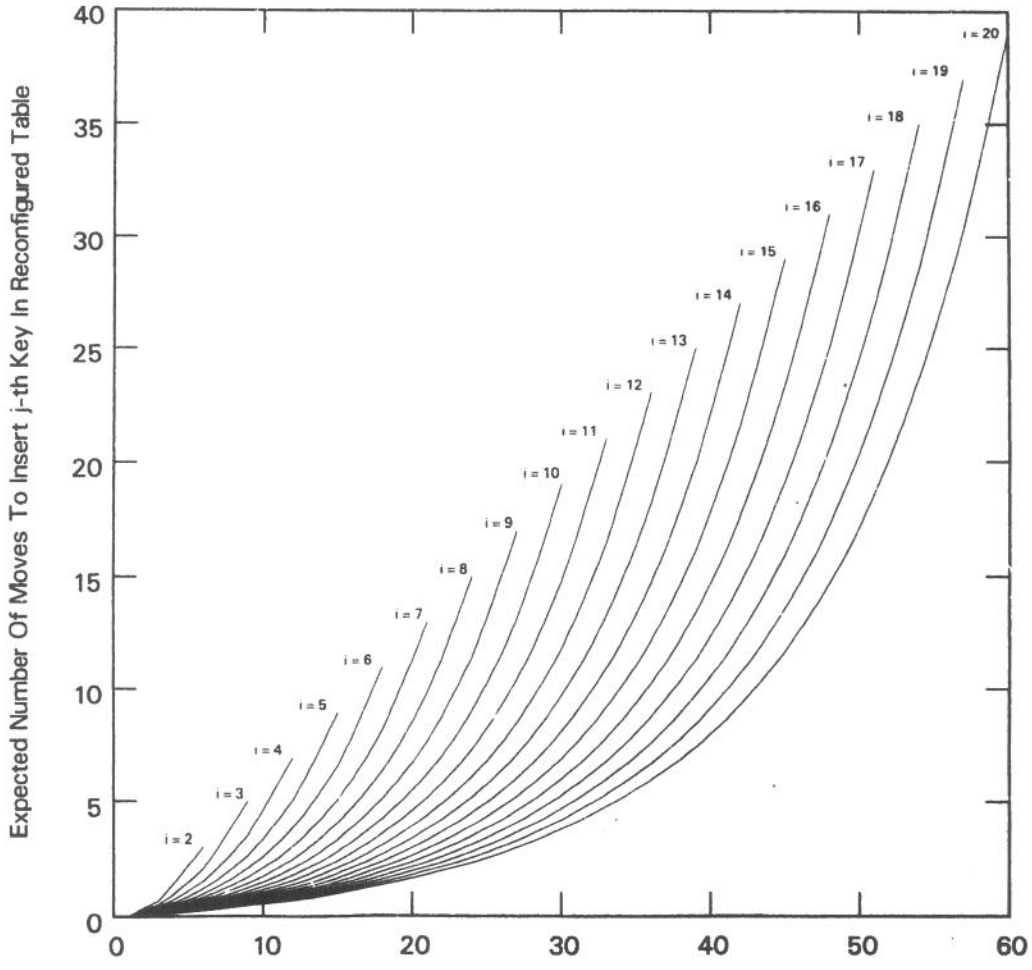

Figure 1

$j = 1, 2, \ldots, 3i \qquad i = 2, 3, \ldots, 20$

6. Asymptotic Analysis

We determine the asymptotic behavior of $E\{M_d(i,j)\}$ as $i,j \to \infty$ with a *fixed* density $\rho = (i+j)/id$. Let $\mu_d(\rho)$ denote this limit and assume that $(1/d) < \rho < 1$. Using Stirling's formula [FE]

$$(2\pi)^{1/2} n^{n+1/2} e^{-n+1/(12n+1)} < n! < (2\pi)^{1/2} n^{n+1/2} e^{-n+1/(12n)}$$

we find

$$C(i+j-k-s,j-s)/C(i+j,i) = (1/\rho d)^k (1 - 1/\rho d)^s \, R_1(i,k,s) R_2(i,k,s)$$

where

$$R_1(i,k,s) = [1 - (k+s)/i\rho d]^{i\rho d-k-s+0.5} \, [1 - k/i]^{-(i-k)-0.5} \, [1 - s/i(\rho d-1)]^{-i(\rho d-1)+s-0.5}$$

and

$$\begin{array}{l} 1/12(i\rho d-k-s)+1 \\ \quad - 1/12(i(\rho d-1)-s) \\ \quad - 1/12(i-k) \\ \quad - 1/12i\rho d \\ \quad + 1/12i(\rho d-1)+1 \\ \quad + 1/12i+1 \end{array} \quad \leq \log R_2(i,k,s) \leq \quad \begin{array}{l} 1/12(i\rho d-k-s) \\ \quad - 1/12(i(\rho d-1)-s)+1 \\ \quad - 1/12(i-k)+1 \\ \quad - 1/12i\rho d+1 \\ \quad + 1/12i(\rho d-1) \\ \quad + 1/12i \end{array}$$

If $k,s = O(\sqrt{i})$, $\lim_{i\to\infty} R_1(i,k,s) = \lim_{i\to\infty} R_2(i,k,s) = 1$ and

$$\lim_{i\to\infty} C(i+j-k-s,j-s)/C(i+j,i) = (1/\rho d)^k (1 - 1/\rho d)^s$$

leading to the formula

Theorem 6.1: If $i,j \to \infty$ with $\rho = (i+j)/id$ $(c = d-1)$, then[9]

$$(6.1)\quad \mu_d(\rho) = -1 + \sum_{0\le k<\infty} (1/\rho d)^k \sum_{kc\le s<\infty} C(k+s,k)(1-1/\rho d)^s$$

To simplify the right-hand-side of equation (6.1) we write

$$C(k+s,k)\,(1-1/\rho d)^s = (1/k!)\,[(d^k/du^k)\,u^{k+s}]_{|\,u=1-1/\rho d}$$

so that

$$(6.2)\quad \mu_d(\rho) = -1 + \sum_{0\le k<\infty} (1/\rho d)^k/k! \sum_{kc\le s<\infty} [(d^k/du^k)\,u^{k+s}]_{|\,u=1-1/\rho d}$$

$$= -1 + \sum_{0\le k<\infty} (1/\rho d)^k/k!\,[(d^k/du^k)\,u^{kd}/(1-u)]_{|\,u=1-1/\rho d}$$

By Cauchy's theorem

$$(1/k!)\,[(d^k/du^k)\,u^{kd}/(1-u)]_{|\,u=1-1/\rho d} = 1/2\pi i \oint_{\Psi} \frac{\psi^{kd}}{(1-\psi)(\psi-(1-1/\rho d))^{k+1}} d\psi$$

where Ψ is a circle centered at $\psi = 1-1/\rho d$, excluding $\psi = 1$ and on which the inequality $|\,(1/\rho d)\psi^d\,| < |\,\psi - (1-(1/\rho d))\,|$ is satisfied[10]. Substituting the

---

[9] The convergence of the series (equation (6.1)) when $\rho d > 1$ together with the monotonicity of $C(i+j-k-s,j-s)/C(i+j,i)$ with k and s implies that the "tail" $\sum_{\{k,s:\max(k,s)\ge O(\sqrt{i})\}} C(k+s,k)C(i+j-k-s,j-s)/C(i+j,i)$ is negligible as $i,j \to \infty$.

[10] To show the existence of a circle Ψ with the requisite properties, we need to prove that the inequality $(1/\rho d)\,(1 - 1/\rho d + \varepsilon)^d < \varepsilon$ is satisfied for some ε, $0 < \varepsilon < 1/\rho d$. If $f_1(\varepsilon) = \varepsilon$, $f_2(\varepsilon) = (1/\rho d)(1 - 1/\rho d + \varepsilon)^d$, the properties:

(i) $0 = f_1(0) < f_2(0)$ *(ii)* $f_1(1/\rho d) = f_2(1/\rho d)$

(iii) $(d/d\varepsilon)\,f_1(\varepsilon)_{|\,\varepsilon=1/\rho d} = 1 < 1/\rho = (d/d\varepsilon)\,f_2(\varepsilon)_{|\,\varepsilon=1/\rho d}$

imply the existence of ε in the interval $(0, 1/\rho d)$ such that $f_2(\varepsilon) < f_1(\varepsilon)$.

integral representation into equation (6.2), and interchanging the order of integration and summation, we obtain

$$\mu_d(\rho) = -1 + 1/2\pi i \oint_\Psi \frac{d\psi}{(1-\psi)(\psi-(1-1/\rho d)-(1/\rho d)\psi^d)}$$

We claim that the polynomial $P_d(z) = -(1/\rho d)z^d + z - (1-1/\rho d)$ has a single real simple zero $\omega_d(\rho)$ in the unit interval if $1/d < \rho < 1$; indeed since

$$P_d((1-1/\rho d)z) = (1-1/\rho d)[z - 1 - (1/\rho d)(1-1/\rho d)^{d-1}z^d]$$

the zeros of $P_d(z)$ are those of the polynomial considered in Lemma 3.5 with $d = c+1$ and $\zeta = (1/\rho d)(1-1/\rho d)^{d-1}$. Observe that $(1/\rho d)(1-1/\rho d)^{d-1}$ monotonically decreases to 0 from $(d-1)^{d-1}/d^d$ as ρ increases from $1/d$ to 1. Thus by the residue theorem

Theorem 6.2:

$$(6.5)\quad \mu_d(\rho) = -1 + \frac{\omega_d(\rho)}{(1-\omega_d(\rho))[d(1-1/\rho d) - \omega_d(\rho)(d-1)]}$$

Remark 6.1 If $i,j\to\infty$ such that $\rho(i,j) = (i+j)/id \to \rho < 1$, then

$$\text{limit}_{i,j\to\infty} E\{M_d(i,j)\} = \mu_d(\rho)$$

Remark 6.2 If $i,j(i)\to\infty$ such that $E\{M_d(i,j(i))\}$ is unbounded, then $\text{limit}_{i\to\infty} j(i)/ic = 1$.

Proof: Suppose on the contrary that $j(i)/ic \le B < 1$. Then the utilization

$$\rho(i) = (i+j(i))/id$$

is bounded away from 1, which yields a contradiction using *Remarks 5.1* and *6.1*.

Remark 6.3: $\omega_2(\rho) = 2\rho-1$ and $\mu_2 = -1 + \rho/2(1-\rho)^2$.

Remark 6.4. The root $\omega_d(\rho)$ is not analytic in ρ in a neighborhood of $\rho = 1$. Nevertheless the limits

$$\text{limit}_{\rho \uparrow 1} \, (d^k/d\rho^k) \, \omega_d(\rho) \qquad k = 0, 1, \ldots$$

exist. To show this, differentiate the defining relationship

$$(6.6) \quad 1 - (\omega_d(\rho))^d = \rho d(1 - \omega_d(\rho))$$

and obtain

$$(6.7) \quad [\rho - (\omega_d(\rho))^{d-1}] \, \omega_d'(\rho) = 1 - \omega_d(\rho)$$

$$[\rho - (\omega_d(\rho))^{d-1}] \, \omega_d''(\rho) = [-2 + (d-1)(\omega_d(\rho))^{d-2}\omega_d'(\rho)] \, \omega_d'(\rho)$$

where

$$\omega_d'(\rho) = (d/d\rho) \, \omega_d(\rho) \qquad \omega_d''(\rho) = (d^2/d\rho^2) \, \omega_d(\rho)$$

We claim that $\omega_d'(\rho) \geq 0$; using equation (6.7), it suffices to show that

$$(\omega_d(\rho))^{d-1} < \rho$$

Suppose on the contrary that $(\omega_d(\rho))^{d-1} \geq \rho$; then using equation (6.6) and the fact that $\omega_d(\rho) < 1$, we have

$$\rho d = 1 + \omega_d(\rho) + \ldots + (\omega_d(\rho))^{d-1} < \rho d$$

which is a contradiction. Thus $\omega_d'(\rho) \geq 0$, so that $\omega_d(1) = \text{limit}_{\rho \uparrow 1} \, \omega_d(\rho)$ exists and

$$(1 - (\omega_d(1))^d) = d(1 - \omega_d(1))$$

which implies that $\omega_d(1) = 1$. A similar argument proves that the derivatives from the left $\omega_d'(1)$ and $\omega_d''(1)$ are finite and

$$\omega_d'(1) = 2/(d-1)$$

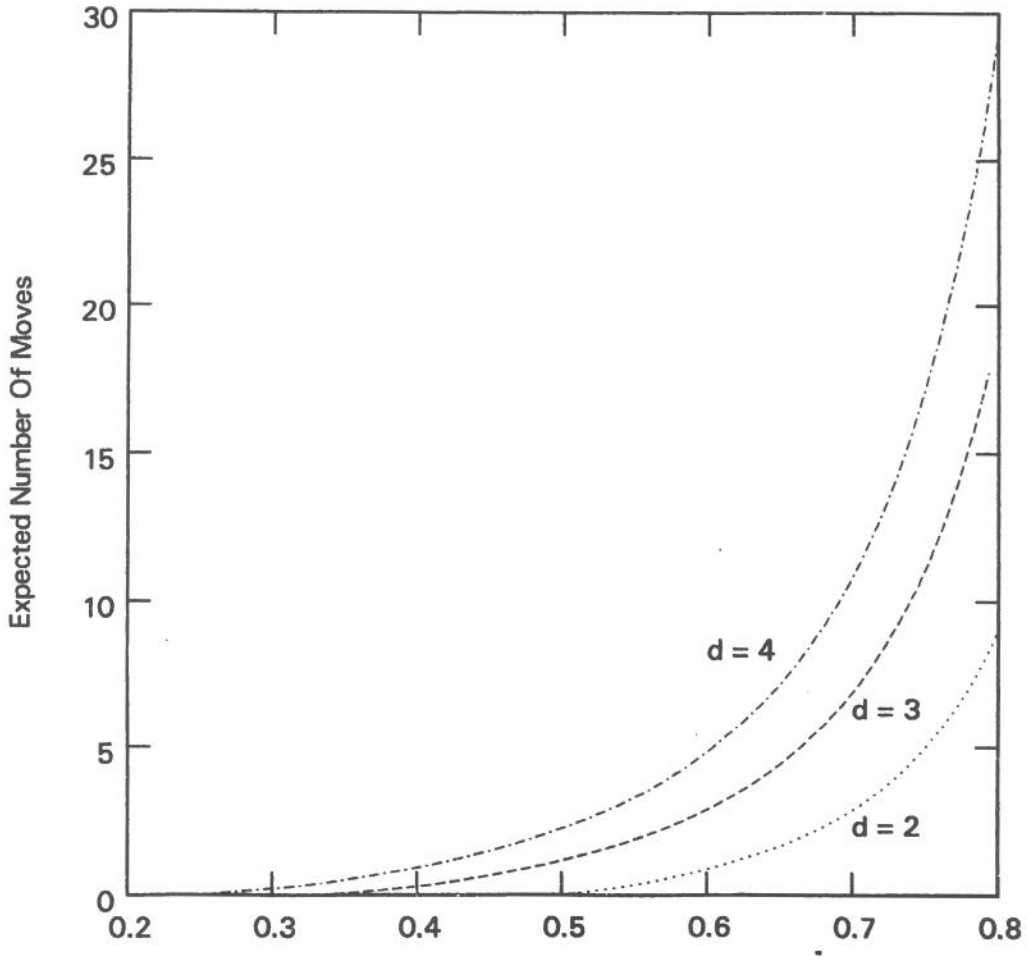

Figure 2

Expected Number Of Moves As A Function Of Utilization ρ

The mean value theorem now yields

$$(6.8)\quad \omega_d(\rho) = 1 + 2(\rho-1)/(d-1) + o((1-\rho) \qquad \rho \uparrow 1$$

Equation (6.8) then yields

$$\mu_d(\rho) \sim \frac{d-1}{2} \frac{1}{(1-\rho)^2} \qquad \rho \uparrow 1$$

for which the special case of $d = 2$ was noted in Remark 6.3.

In Figure 2 we plot $\mu_d(\rho)$ for $1/d \leq \rho \leq 0.8$ and $d = 2, 3, 4$.

7. The Relationship Of Key Insertion To Hashing With Linear Probing

A *hashing function* is a mapping from a set of possible key values to an address space $\{0, 1, ..., i-1\}$. The term *hashing with linear probing* is used to describe the following process whereby a set of keys $k_0, k_1, ..., k_{j-1}$ are assigned distinct addresses;

(i) the hashing function h determines the assignment sequence

$$(h(k_0), h(k_1), ..., h(k_{j-1}))$$

(ii) having stored keys $k_0, k_1, ..., k_{t-1}$ in addresses $\beta_0, \beta_1, ..., \beta_{t-1}$, the key k_t is stored in address $\beta_t = (h(k_t) + s)$ (modulo i) where s is the smallest non-negative integer such that $(h(k_t)+s)$ (modulo i) does not appear in the subsequence $\beta_0, \beta_1, ..., \beta_{t-1}$. The *length of the probe* is s; to retrieve a key k whose address has been determined by this hashing procedure requires a computation of h(k) and s+1 comparisons.

Thus the occupancy discipline defined in Section 3 with $c = 1$ corresponds to assignment of address by hashing with linear probing and the number of moves is equal to the length of a probe. An analysis of of the length of a probe with $c = 1$ is found in [KN2, KW] and for $c > 1$ in [BK].

Just as in Section 5, we may argue that a linear probe of length e corresponds to a configuration of the (hash) table with parameters t_R and t_L The number of assignment sequences

$$h(X_{n_{k-1}}), h(X_{n_{k-1}+1}), ..., h(X_{n-1})$$

which result in a probe of length e is

$$\nu_{m_{k-1}}(B,t_L+e-B)\ \gamma_{m_{k-1}}(n_{k-1}-B,n-n_{k-1}-(t_L+e-B)).$$

In the case of sparse tables, we assign each of these sequences the probability $(C(n,n_{k-1}))^{-1}$, while the linear probe model assigns the different probability $(n_{k-1})^{-(n-n_{k-1})}$. If $n_{k-1} = i$, $n-n_{k-1} = j$ with $\rho = (i+j)/2i$ and $i\to\infty$, then the asymptotic probabilities are

$$i^{-i(2\rho-1)} \quad \textit{in the linear probing model}$$

$$(2\pi i)^{-1/2}\,(1 - 1/2\rho)^{i(2\rho-1)+1/2}\,(1/2\rho)^{-i} \quad \textit{in the key insertion model}$$

Thus one cannot directly use the results in [KN2, KW] as Franklin [FR] and Melville and Gries [MG] both do. It should be mentioned that Melville and Gries are aware that their analysis is suspect since the positions of values are changed during an insertion.

8. Improving The Worst Case Of Insertion

As we remarked in Section 2, the worst case of insertion may be quite bad. For example, if $m_k = 2$, $n_{k-1} = 100$ and $n = 150$, inserting a new key may cause 100 keys to move while the expected value is 5 from Figure 2. To improve the worst case performance an additional structure is imposed on the sparse table, yielding a more complicated scheme which we refer to as the *hierarchical sparse table*. The basic idea is to *redistribute* the keys locally when the local density becomes high. The insertion algorithm to be described below is somewhat different than the one described in Section 2.

Let m be the size of the table, $h = \lfloor \log_2 m - \log_2\log_2 m \rfloor$ and $b = m/2^h$. Note that $\log_2 m \le b < 2\log_2 m$. Divide the table into $m/b = 2^h$ *blocks* $B_0, B_1, ..., B_{2^h-1}$; the

first $2^h\lceil b\rceil - m$ blocks are of size $\lfloor b\rfloor$ and the others are of size $\lceil b\rceil$ where $\lfloor x\rfloor$ and $\lceil x\rceil$ denote the integer part of x (*floor of x*) and the smallest integer $\geq x$ (*ceiling of x*), respectively.

Now consider a full binary tree of height h with *leaves* $L_0, L_1, \ldots, L_{2^h-1}$ (scanned from left to right) and associate with each of its nodes v a segment s(v) of the table as follows:

(i) to the leaf L_i associate the block B_i $(0 \leq i < 2^h)$

(ii) for an internal node with children u and w, $s(v) = s(u) \cup s(w)$

For every node v, let m(v) be the size of s(v). Thus if r is the root, then $s(r) = m$, the size of the table. The *density* $\rho(v)$ of s(v) is the number of genuine keys in s(v) divided by m(v).

The nodes of the tree are divided into levels; the root r is at level 0, and the level of any other node is greater by one than that of its parent. The level of the leaves is obviously h. A distinct maximum density is associated with each of the levels. Let $0 \leq \tau_L < \tau_U \leq 1$ and define the sequence $\tau_0, \tau_1, \ldots, \tau_h$ of *threshold densities* of nodes in levels 0, 1, ..., h by:

$$\tau_q = \tau_L + q(\tau_U - \tau_L)/h \qquad 0 \leq q \leq h$$

Thus $\tau_L = \tau_0 < \tau_1 < \ldots < \tau_h = \tau_U$ and $\tau_{q+1} - \tau_q = (\tau_U - \tau_L)/h$.

During the process of insertion into a hierarchical sparse table, the density $\rho(L_i)$ of each leaf L_i satisfies $\rho(L_i) \leq \tau_h = \tau_U$. However, it may happen that for an interval node v of level q that $\rho(v) > \tau_q$. An insertion is performed as follows:

(i) conduct a binary search and insert the new key as in Section 2

(ii) assume that the block into which the new key has been added is B_i. If the density of B_i is less than or equal to τ_h, then insertion process has been completed. Otherwise, consider the ancestors $r = v_0, v_1, ..., v_{h-1}$ of L_i and find the maximal value of q for which $\rho(v_q) < \tau_q$. If such a q is found, then the genuine keys of $s(v_q)$ are redistributed locally; the size of $s(v_q)$ is not changed, only its genuine keys are evenly distributed. However, if no such q exists, then the density of the entire table is greater than or equal to $\tau_0 = \tau_L$ and the table size is increased.

One way to increase the size of the table is by expanding to a table of size nm_{k+1} where n is the number of genuine keys currently in the table. Note that n may be different that n_k so that the sequence $\{n_k: 0 \leq k < \infty\}$ no longer plays its former role. Other possibility is to reconfigure when $n = n_k$ even if there is no need to do so according to the local densities criterion. In this case, one can use $\tau_L = n_k/n_{k-1}m_k$ which conforms with the (regular) sparse table scheme in the sense that in both schemata table expansion occurs for the same table state.

The advantage of hierarchical sparse tables is the improvement of their "worst case" performance over the original scheme.

Theorem 8.1: Performing $n-n_{k-1}$ insertions into a hierarchical sparse table of size $m = n_{k-1}m_k$ requires at most $O((n-n_{k-1})(\log_2 m)^2/(\tau_U-\tau_L))$ operations.

Proof: The density of a block is bounded by $\tau_h = \tau_U < 1$. Therefore, each block contains some dummy keys and the length of a move is less than the size of two blocks $2\lceil b \rceil \leq 2 + 4\log_2 m$.

Some insertions trigger a redistribution of the entire table which costs m operations. However others are not immediately followed by redistribution. We wish to bound the entire time spent on redistribution while inserting $n-n_{k-1}$ keys. To this end we first estimate the number of insertions into s(v) between two successive redistributions.

After redistributing s(v), the density of s(v) and therefore the densities of both of its children is at most τ_q where q is the level of v. At the next redistribution of s(v), v must have at least one child u with density τ_{q+1} or higher. Thus, the density of s(u) has increased by at least $\tau_{k+1}-\tau_k = (\tau_L-\tau_U)/h$. Hence at least $(\tau_U-\tau_L)m(u)/h$ dummy keys have been replaced by genuine keys in s(v) between two redistributions of s(v). The cost of a single redistribution of s(v) is m(v). Therefore, the cost per insertion is at most

$$m(v)\Big/(\tau_U-\tau_L)m(u)/h = (m(v)/m(u))h\Big/(\tau_U-\tau_L)$$

However,

$$m(v)/m(u) \leq 2 + 1/b$$

and thus the cost per insertion for s(v) is at most

$$(2+1/b)h\Big/(\tau_U-\tau_L)$$

Each block has one ancestor at each level, and therefore each insertion contributes to at most h redistributions. Thus the cost of inserting $n-n_{k-1}$ keys is at most

$$(n-n_{k-1})(2+1/b)h^2\Big/(\tau_U-\tau_L)$$

Since b and h are both of the order $\log_2 m$, the theorem is proved. ■

As for the implementation of this tree, several possibilities exist:

(i) *Explicit Representation:* The tree is stored by using nodes to contain the current density ($\rho(v)$), and pointers to the two children. The pointers can be eliminated if we use an array where locations 2i and 2i+1 are the children of location i. The number of leaves is $2^h = m/b$ and the number of nodes is less than twice as much. Thus the storage requirements are o(m). On each insertion the densities of the ancestors must be updated. This can be done within $O(h) = O(\log_2 m)$ time.

(ii) *Implicit Representation:* No tree structure is maintained. On insertion, we first calculate the boundaries of the block which has received an additional key. Then the entire block is scanned to calculate its density. If the density is found to be too high then the sibling block is scanned to calculate the density of the common parent. This process is continued until arriving at the node v to be redistributed. The scan time is O(m(v)) which is proportional to the redistribution time, and therefore the worst case time bound does not change. However, a scan of length b must be conducted even if no redistribution takes place, thus increasing the cost of insertion somewhat.

(iii) *Compromise Representation:* The insertion cost using the implicit representation can be reduced if a vector containing the densities of the blocks is maintained. If no redistribution is required then we must only update the density of a single block (in O(1) time). In case of redistribution of s(v) the densities of all of the offsprings of v must be updated but the time required for the update is negligible compared to the redistribution time. As for storage, the extra space is equal to the number of blocks $m/b = o(m)$.

Note that these schemata are equivalent in the sense that redistribution occurs for the same table states and affects the same blocks.

(iv) *Alternative Scheme* As in the implicit representation, no tree structure is maintained. The boundaries of the blocks and their densities are calculated when needed. Redistribution occurs whenever a scan of length 2b or more is conducted. This implies that at least one block is full and requires redistribution. Note that in this scheme, redistribution does not occur at the same time and does not occur for the same table state and does not have the same scope as in the other three schemata. An analysis similar to that used in proving Theorem 8.1 may be carried out.

9. Deletions

Deletions, though easy to implement, are difficult to analyze statistically. We propose two deletion schemes:

(i) *physical removal* – to delete the key k from the table $\underline{y} = (y_0, y_1, \ldots, y_{n_{k-1}m_{k-1}})$ conduct a search to find s such that

$$y_{s-1} < k = y_s.$$

Suppose L satisfies

$$y_s = y_{s+1} = \ldots = y_{s+L-1} \neq y_{s+L}$$

where the subscripts are taken modulo $n_{k-1}m_k$. Then replace the block $(y_s, y_{s+1}, \ldots, y_{s+L-1})$ by $(y_{s+L}, y_{s+L}, \ldots, y_{s+L})$ obtaining the table

$$\underline{y}' = (y_0, y_1, \ldots, y_{s-1}, y_{s+L}, y_{S+L}, \ldots, y_{s+L}, y_{s+L+1}, \ldots, y_{n_{k-1}m_{k-1}})$$

In addition to the reconfiguration which occurs whenever we attempt to insert a key into a table $\underline{y} = (y_0, y_1, \ldots, y_{n_{k-1}m_{k-1}})$ presently containing n_k keys, a reconfiguration will also occur whenever deletion reduces the number of genuine keys to some threshold. There are a variety of ways to specify these contraction thresholds; the simplest is to reconfigure (after deletion) when the number of genuine keys remaining is n_{k-2}.

We are not able to provide an analysis of sparse tables under a sequence of insertions/deletions. To begin with the set of possible table states attainable by a sequence of insertions/deletions is larger than the set of possible table states attainable by only insertions. (For example, delete the key 4 from the table (0, 0, 0, 1, 2, 2, 4, 4, 4, 6, 6, 6, 8, 8, 8).) The analysis of the pure insertion process is simplified by the existence of *renewal points* – the epochs of reconfiguration. The insertion/deletion process might be compared with a birth and and death process and the analysis given in Sections 3-5 has determined a probability distribution on the state space of the pure birth (= insertion) process.

(ii) *tagged deletions* – Like indexed sequential files (ISAM) this scheme requires an additional Boolean vector of length m to distiguish between genuine and dummy keys. A key is deleted by setting the appropriate entry to *false*. The physical removal of keys is postponed until reconfiguration time; until then, at least one copy of each key must remain. The time for deletion consists principally of the search time $O(\log_2 m)$. Additional $O(m_k)$ time is required to set the bits corresponding to all entries of the key to be deleted. By marking only the rightmost copy of a key as deleted, the additional operation requires only O(1) time.

10. Fingers

Guibas et. al. introduced the idea of *fingers* (see also Brown and Tarjan [BT]): Assume that many search operations accumulate near some prespecified keys, called fingers. Given a key k which is close to some finger f, it is required to design an algorithm which searches for k in time $O(\log_2 d)$ where d is the distance between the location of f and the location of k. This feature can be incorporated into the sparse table scheme by keeping the fingers in a special sorted list and their locations in the sparse table by means of an additional list of pointers. Searching for a key k is done by first finding the appropriate finger, using its corresponding pointer to access the table (updating the pointer if necessary), and then using the unbounded search technique of Bentley and Yao [BY].

11. Indexed Sequential Files

An indexed sequential file consists of a sorted disk file which resides on several cylinders. The value of the key uniquely determines the cylinder on which the record resides. The identity of this cylinder is found by means of an index. To enable insertions, each cylinder has several *overflow tracks*, into which all additions to the cylinder are placed. The advantage of this system is the single motion of the read arm required to locate a record. If many insertions occur, then the overflow tracks might become full after which additional records are placed in a general overflow area. To locate a record in the overflow area, two arm motions are required. To avoid excessive arm motion, it is advisable to reconfigure the entire file.

Given the characteristics of the file, it is interesting to estimate the average number of insertions until one of the cylinders overflows. Each cyclinder corresponds to an urn, whose capacity is equal to the size of the cylinder overflow area. Indexed sequential files resemble sparse tables also in the fact that the maximum key in each urn depends on the sequence of prior insertions, and the probabilistic model assigns equal probability to each sequence.

12. Linear Sparse Tables

Replace the circular table by a linear one, with additional space on the "right end". This extra space is used for storing keys which would otherwise shift the origin (PB) of the table. The additional amount of storage depends on the density. It is conjectured that for density ρ bounded away from unity, o(m) extra space is sufficient.

13. Conclusions

The sparse table scheme is an extremely simple data structure. As indicated by Melville and Gries [MG], it can be used for sorting. Another application is to B-trees, where all nodes have the same prespecified size m, and the number of keys may be as low as m/2. Implementing each node as a sparse table trades a reduced search time within a node (from O(m) to $O(\log_2 m)$) for an increased storage allocation. Even though many memory management systems (such as *Buddy systems* [KN1]) allocate space in predefined quantities, not many data structures take advantage of this. (The exceptions are hash tables, sparse tables and some list processing system with garbage collection.)

For constant m_k, average behavior of sparse tables is optimal (up to a constant). However the worse case behavior is O(n). To effectively control the worst case, a hierarchical scheme has been introduced, and an upper bound of $O((\log_2 n)^2)$ has been proved. This bound is not tight and the its true value is an open question. A second open question is the average number of moves in a hierarchical scheme. We conjecture the bound is O(1) for constant m_k.

14. References

[AHL] L. V. Ahlfors, "Complex Analysis", McGraw-Hill, 1953.

[AHO] A. V. Aho, J. E. Hopcroft and J. D. Ullman, "The Design And Analysis Of Computer Algorithms", Addison-Wesley, 1974.

[BDGS] J. L. Bentley, D. Detig, L. Guibas and J. Saxe, "An Optimal Data Structure For Minimal-Storage Dynamic Searching", Computer Science Department, Carnegie-Mellon University, 1978.

[BK] I. F. Blake and A. G. Konheim, "Big Buckets Are (Are Not) Better", *JACM*, ***24***, 4, October 1977, pp. 591-606.

[BT] M. R. Brown and R. E. Tarjan, "Design And Analysis Of A Data Structure For Representing Sorted Lists", *SIAM Journal of Computing*, ***9***, No. 9, pp. 594-614, August 1980.

[BY] J. L. Bently and A. C. Yao, "An Almost Optimal Algorithm For Searching", *Information Processing Letters*, ***5***, 3, 1976, pp. 82-87.

[FE] W. Feller, "An Introduction To Probability Theory And Its Applications", Volume 1, John Wiley, 1950.

[FR] W. R. Franklin, "Padded Lists: Set Operations In Expected O(log log N) Time", *Information Processing Letters*, ***9***, 4, November 1979, pp. 161-166.

[GMPR] L. J. Guibas, E. M. McCreight, M. F. Plass and J. R. Roberts, "A New Representation For Linear Lists", *9th Annual Symposium Theory Of Complexity*, pp. 49-60, 1977.

[KN1] D. E. Knuth, "The Art Of Computer Programming : Fundmanetal Algorithms", Addison-Wesley, 1969.

[KN2] _____, "The Art Of Computer Programming : Searching And Sorting", Addison-Wesley, 1973.

[KN3] _____, "Deletions Which Preserve Randomness", *IEEE Transactions On Software Engineering*, ***SE-3***, pp. 351-359, 1977.

[KW] A. G. Konheim and B. Weiss, "An Occupancy Discipline And Applications", *SIAM Journal Of Applied Mathematics*, ***14***, 6, November 1966, pp. 1266-1274.

[MG] R. Melville and D. Gries, "Sorting And Searching Using Controlled Density Arrays", ***TR 78-362***, Cornell University, Ithaca, New York.

[PIA] Y. Perl, A. Itai and H. Avni, "Interpolation Search a LogLog N Search", ***CACM***, ***21***, 1978, pp. 550-553.

[RND] E. M. Reingold, J. Nievergelt and N. Deo, "Combinatorial Algorithms: Theory And Practice", Prentice Hall, 1977.

[V] V. Vuillemin, "A Data Structure For Manipulating Priority Queues", *CACM*, ***21***, pp. 309-315, 1978.

[WW] E. T. Whittaker and G. N. Watson, "A Course Of Modern Analysis", Cambridge University Press, 1952.

[Y] A. C. Yao, "On Random 2-3 Trees", *Acta Informatica*, ***9***, pp. 159-170, 1978.

Appendix

Proof of Theorem 3.7: To compute the coefficient of $w^i z^j$ with $0 \le j < ic$ in

$$[(\partial/\partial u)\ \Lambda_c(z,w,u)]_{|u=1} = 0.5G_c(z,w)\ [(\partial^2/\partial u^2)\ H_c(uz,uw)]_{|u=1}$$

we expand the numerator and denominator of

$$H_c(uz,uw) = \frac{(uw-1)X(wu^{c+1}z^c)+1}{1-uzX(wu^{c+1}z^c)}$$

in Taylor series about $u = 1$

$$(uw-1)X(wu^{c+1}z^c) + 1$$

$$= [(w-1)X(wz^c) + 1] + (u-1)[wX(wz^c) + (w-1)D_1X(wz^c)]$$

$$+ 0.5(u-1)^2[(w-1)D_2X(wz^c) + 2wD_1X(wz^c)] + O((u-1)^3)$$

$$1-uzX(wu^{c+1}z^c) = 1 - zX(wz^c) - (u-1)[zX(wz^c) + zD_1X(wz^c)]$$

$$- 0.5(u-1)^2[2zD_1X(wz^c) + zD_2X(wz^c)] + O((u-1)^3)$$

where

$$D_iX(wz^c) = [(\partial^i/\partial u^i)\ X(wu^{c+1}z^c)]_{|u=1} \qquad i = 1,2$$

Then

$$[(\partial/\partial u)\ \Lambda_c(z,w,u)]_{|u=1} = T_1 + T_2 + T_3 + T_4 + T_5$$

where

$$T_1 = \frac{z/2}{X(wz^c)}\ \frac{1+(w-1)X(wz^c)}{1-zX(wz^c)}\ \frac{D_2X(wz^c)+2D_1X(wz^c)}{1-w-z}$$

$$T_2 = \frac{z^2}{X(wz^c)}\ \frac{1+(w-1)X(wz^c)}{1-w-z}\ \left(\frac{X(wz^c)+D_1X(wz^c)}{1-zX(wz^c)}\right)^2$$

$$T_3 = \frac{z}{X(wz^c)} \frac{wX(wz^c)+(w-1)D_1X(wz^c)}{1-zX(wz^c)} \frac{X(wz^c)+D_1X(wz^c)}{1-w-z}$$

$$T_4 = 1/2 \frac{w-1}{X(wz^c)} \frac{D_2X(wz^c)+2D_1X(wz^c)}{1-w-z}$$

$$T_5 = \frac{1}{1-w-z} \frac{D_1X(wz^c)}{X(wz^c)}$$

Expressions for the derivatives $D_iX(wz^c)$ (i = 1,2) may be found by differentiating the relationship

$$wz^cu^{c+1}\Big(X(wu^{c+1}z^c)\Big)^{c+1} = (X(wu^{c+1}z^c)-1)$$

yielding

(A.1) $D_1X(wz^c)[1 - c(X(wz^c-1)] = (c+1)X(wz^c)(X(wz^c)-1)$

(A.2) $D_2X(wz^c)[1 - c(X(wz^c-1)] = - D_1X(wz^c)[1 - c(X(wz^c-1)]$

$$+ (c+1)(2X(wz^c)-1)D_1X(wz^c) + c[D_1X(wz^c)]^2$$

Combining the terms we obtain

(A.3) $$T_1 + T_4 = \frac{-1}{2X(wz^c)} \frac{D_2X(wz^c)+2D_1X(wz^c)}{1-zX(wz^c)}$$

(A.4) $$T_2 + T_3 = - \frac{z}{X(wz^c)} \left(\frac{X(wz^c)+D_1X(wz^c)}{1-zX(wz^c)}\right)^2$$

$$+ \frac{z}{1-w-z} \frac{X(wz^c)+D_1X(wz^c)}{1-zX(wz^c)}$$

Let $\mathcal{H}_{i,j}$ denote the operator on generating functions F(z,w) defined by

$$\mathcal{H}_{i,j}F(z,w) = f_{i,j}$$

$$F(z,w) = \sum_{0\le i<\infty} w^i \sum_{0\le j<\infty} f_{i,j}z^j$$

Equation (3.6) shows that

$$\mathcal{H}_{i,j}X(wz^c) = 0 \quad 0 \le i < \infty,\ 0 \le j < ic \quad (i,j) \neq (0,0)$$

and that more generally

$$\mathcal{H}_{i,j}f(X(wz^c)) = 0 \quad 0 \le i < \infty,\ 0 \le j < ic \quad (i,j) \neq (0,0)$$

whenever f is analytic in a neighborhood of 1. From equations (A.1-2) it follows that

$$\mathcal{H}_{i,j}D_kX(wz^c) = 0 \qquad 0 \le j < ic \quad (i,j) \neq (0,0) \quad k = 1,2$$

and therefore from equations (A.3-4) that

$$\mathcal{H}_{i,j}(T_1+T_4) = 0 \quad 0 \le i < \infty \quad 0 \le i < jc \quad (i,j) \neq (0,0)$$

and

$$\mathcal{H}_{i,j}(T_2+T_3) = \mathcal{H}_{i,j}\ \frac{z}{1-w-z}\ \frac{X(wz^c)+D_1X(wz^c)}{1-zX(wz^c)}$$

$$0 \le i < \infty \quad 0 \le j < ic \quad (i,j) \neq (0,0)$$

so that it remains to identify the coefficient of w^iz^j in

$$\frac{-1}{1-w-z} + \frac{1}{1-w-z}\ \frac{X(wz^c)}{1-zX(wz^c)}\ \frac{1}{1-c(X(wz^c)-1)}$$

with $0 \le j < ic$. Writing

$$\frac{X(wz^c)}{1-zX(wz^c)}\ \frac{1}{1-c(X(wz^c)-1)} = \frac{X(wz^c)}{1-zX(wz^c)}\ \frac{1}{1-cwz^c(X(wz^c))^{c+1}}$$

$$= \sum_{0\le u<\infty} z^u \sum_{0\le v<\infty} (wz^c)^v\ (X(wz^c))^{u+1+(c+1)v}$$

and using equation (3.6) we find

$$\mathcal{H}_{k,ck+s}\,\frac{X(wz^c)}{1-zX(wz^c)}\qquad\frac{1}{1-c(X(wz^c)-1)}$$

$$=\sum_{0\le u\le k} c^u\,\frac{u(c+1)+s+1}{(c+1)k+s+1}\;C((c+1)k+s+1,s-u)$$

$$=\sum_{0\le u\le k}[c^uC((c+1)k+s+1,s-u)-(c+1)c^uC((c+1)k+s,s-u-1)]$$

$$=C((c+1)k+s,k)$$

Thus, when

$$j=tc+T\quad 0\le T<c\quad 0\le t<i$$

we have

$$\mathcal{H}_{i,j}\,[(\partial/\partial u)\,\Lambda_c(z,w,u)]_{|\,u=1}=-\,C(i+j,i)$$

$$+\sum_{0\le k\le t}\sum_{0\le s\le (t-r)c+T} C(i-k+(t-k)c+T-s,i-k)\;C((c+1)k+s,k)$$

which simplifies to equation (3.8). ■