Invited paper

Studies on Hashing PART-1: A Comparison of Hashing Algorithms with Key Deletion

TAKAO GUNJI* and EIICHI GOTO*,**

Four concepts σ (storage utility factor), U (average number of probes for Unsuccessful searching), I (that for Insertion), and S (that for Successful searching) are introduced to compare speed and memory tradeoffs of various hashing algorithms. An open addressing hashing scheme suited for frequent deletion of nonrelocatable keys is proposed and analyzed in terms of the four concepts.

1. Introduction

Hashing as a fast matching technique now has many new sophisticated applications beyond the simple symbol-searching in compilers and assemblers. For example, hashing is employed in programming languages, such as LEAP (Feldman and Rovner [5]) and Interlisp (Teitelman [13], Bobrow [4]), to implement associative data structures. McCarthy [11] proposed a "hash-Lisp" scheme, where *cons* operation computes a hash address to store the new pair in order to represent all S-expressions uniquely; in HLISP, which is a "hash-Lisp" implemented at the University of Tokyo, not only are the S-expressions of McCarthy's schema represented, but sets are also represented uniquely by hashing, thus demonstrating an application to formula manipulation (Goto and Kanada [7]).

Hashing algorithms are divided into two classes according to the methods for resolving collisions (or conflicts): *open addressing* and *chaining*. While the subsequent address is re-computed after each collision in the former, it is obtsined by a chained link in the latter. Well-known representatives for these classes are *uniform hashing* (Algorithm D in Knuth [10]) for the foemer, and *coalesced chaining* (Algorithm C in Knuth [10]) for the latter.

Although numerous hashing algorithms have been proposed and/or analyzed in the past two decades (140 references are listed by Knott [9]), we find some insufficiency with regard to the new applications. In this paper we will treat two mutually-interrelated problems.

The first problem is *the basis on which to compare various hashing algorithms* Traditionally, probe numbers have been used as measures of efficiency, since they correspond to numbers of memory accesses. While we will also follow this line, we propose, in the first place, that *three* conceptually different probe numbers be distinguished; U for unsuccessful searching, I for insertion after the search is unsuccessful, and S for successful searching. Though each probe number has its own significance, one (or even two) of them has sometimes been neglected. Typically, U and I are often mixed up, partly because unsuccessful searching is often followed by insertion. However, unsuccessful searching *not* followed by insertion is also a fundamental operation in hashing. For example, unsuccessful searching of tables for the reserved words in compilers is not followed by insertion once all the reserved words are inserted at an initial stage. We will treat these numbers of probes with equal emphasis. In the second place, we propose to use *storage utility factor* σ instead of the *load factor* α when different hashing algorithms are compared with one another. σ is defined as

$$\sigma=\frac{\text{(storage occupied by inserted items)}}{\text{(total storage allotted for table organization)}}. \quad (1.1)$$

The numerator is the indispensable storage required to store the items, and the denominator includes, besides occupied and unoccupied table positions, all the extra storage (such as linkage area for chaining) to make a hashing algorithm work effectively. Although α, as defined by

$$\alpha=\frac{\text{(number of inserted items)}}{\text{(total number of hash table positions)}}, \quad (1.2)$$

is helpful for analysis *within* a specific algorithm, it is quite powerless once different algorithms are to be collated. A very misleading feature of α is that the denominator of (1.2) counts no extra storage required for some algorithms; hence α, in such cases, has little to do with the actual storage utility in spite of its literal meaning. In Section 2 we shall compare several algorithms with these new standpoints.

The second problem is *how to treat deletion in open addressing*. Deletion of unused items is troublesome in open addressing, because we have to mark unoccupied table positions that are in collisions as "deleted" so as to prevent fallacious, unsuccessful searching. Thus a table position takes one of three states: *empty*, *occupied*, or *deleted* (Morris [12], Knuth [10]). However, probe num-

*Department of Information Science, University of Tokyo, Tokyo 113, Japan.

**Institute for Physical and Chemical Research, Wako-shi, Saitama 351, Japan.

bers are not reduced even though the number of inserted items is decreased, and if deletion is frequent, the table will have no empty position, because a deleted position will never be empty again, unless the table is updated somehow. Updates of a hash table by rehashing, that is, by reconstructing the whole table, would not be practical, because it involves (1) relocation of items, which is often undesirable owing to mutual referencing of items in complex data structures, and (2) the use of secondary storage, as McCarthy [11] complained. (Though the deleted position is not created by Algorithm R in Knuth [10], it involves relocation of items in order to fill in the deleted position, and it works only for the "slowest" algorithm, called *linear probing*.) In Section 3, we propose an update scheme that does not necessitate relocation or the use of secondary storage. The point is to reclaim as many empty positions as is possible, allowing some deleted positions to remain. Although the remaining deleted positions cause some inefficiency, we have found this scheme to be reasonably efficient in certain cases. A comparative argument and a detailed analysis are disclosed in Sections 3 and 4.

2. Comparison of Algorithms

In order to make the discussion concrete, we have organized a hash table with respect to storage, as follows. An item to be stored in the table will be identified hereafter with its *key*, which is k-bits long. A *hash table* consists of M cells, each of which is k-bits long in open addressing, and $(k+p)$-bits long in chaining (cf. Fig. 1). The additional p bits in chaining are for linkage pointers. Hence p is at least $\log_2 M$. Let $\kappa = k/p$. We use this "reduced" key length in the later analysis. Let n be the number of keys in the table. Then the load factor α becomes n/M in both open addressing and chaining, and σ becomes $kn = kM = \alpha$ in open addressing, and $kn = (k+p)M = (\kappa/(\kappa+1))\alpha$ in chaining, as seen in Fig. 1 (cf. (1.1)). In coalesced chaining, a new key is inserted, after encountering an end of a chain (denoted by Γ), into an empty cell, which is found by searching the table sequentially, so that the keys need not be relocated, though chains may coalesce.

In this section we treat three more schemes of hashing. One is a modification of open addressing proposed by Furukawa [6] to reduce the value of U, and the other two are concerned with what Morris [12] called "scatter index table technique."

In the traditional open addressing, U and I are "degenerate," because searching cannot be terminated until an empty cell is encountered. This degeneracy is removed in Furukawa's *collision (conflict) flag method* by attaching to each cell a one-bit flag (possibly with almost no burden to storage) to indicate whether collisions have once occurred at the cell. The flags are all set to 0 initially, and if a cell is probed on the way of insertion of a new key, its flag is set to 1, so that flag 0 indicates that no probe sequence for successful searching for

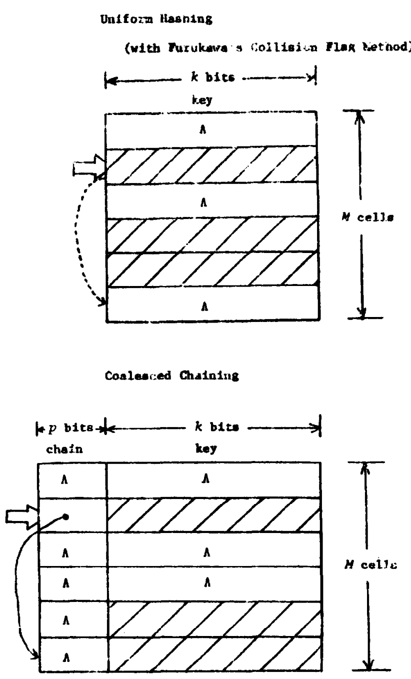


Fig. 1 Organization of hash tables.

another key includes that cell. Thus, searching can be terminated if a cell with its flag at 0 is found, possibly before encountering an empty cell. The analysis in Appendix A shows that U becomes smaller than I by a factor $1/(1-\ln(1-\alpha))$. Because $1 \ll \log_2 M \leq p$ in practical cases, we neglect the additional bit in counting σ, so that $\sigma = \alpha$.

The scatter index table technique is particularly suited for long (κ being large) keys, though the upper bound of σ may be rather severe. The hash table of size M stores only indices (pointers) to a separate key table of size N. Thus each cell of the hash table is p-bits long, and the key table k-bits long (open addressing) or $(k+p)$-bit long (chaining) (cf. Fig. 2). Collisions are resolved either by open addressing within the hash table or by chaining through the key table. Either method is almost as efficient, as we shall see, in spite of Morris's [12] statement. The advantage is to be able to keep N just as small as is enough to store all the keys and thus make M as large as possible. Since $\alpha = n/M$, not n/N, the possibility of collisions decreases. As can be seen in Fig. 2, σ's for the scatter index table techniques are, respectively,

$$\sigma = \frac{kn}{pM+kN} = \frac{k\tau}{\tau+\kappa}\alpha \quad \text{(open addressing)}, \tag{2.1}$$

$$\sigma = \frac{kn}{pM+(k+p)N} = \frac{\kappa\tau}{\tau+\kappa+1}\alpha \quad \text{(chaining)}, \tag{2.2}$$

where $\tau = M/N$.

Note that α in (2.2) has completely lost its literal meaning as "load" factor, since the hash table stores only the

Table 1 Average numbers of probes and σ for various algorithms.

Algorithm	U	I	S	σ	σ_m
U*	$\frac{1}{1-\alpha}$	$\frac{1}{1-\alpha}$	$-\frac{1}{\alpha}\ln(1-\alpha)$	α	1
UF**	$\frac{1}{1-\alpha}\frac{1}{1-\ln(1-\alpha)}$	$\frac{1}{1-\alpha}$	$-\frac{1}{\alpha}\ln(1-\alpha)$	α	1
CC***	$1+\frac{1}{4}(e^{2\alpha}-1-2\alpha)$	$1+\frac{1}{4}(e^{2\alpha}-1-2\alpha)+\alpha e^{\alpha}$	$1+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)+\frac{1}{4}\alpha$	$\frac{\kappa}{\kappa+1}\alpha$	$\frac{\kappa}{\kappa+1}$
SO†	$\frac{1}{1-\alpha}-\alpha$	$\frac{1}{1-\alpha}$	$-\frac{1}{\alpha}\ln(1-\alpha)$	$\frac{\kappa\tau}{\tau+\kappa}\alpha$	$\frac{\kappa}{\tau+\kappa}$
SC††	$e^{-\alpha}+\alpha$	$1+\alpha$	$1+\frac{1}{2}\alpha$	$\frac{\kappa\tau}{\tau+\kappa+1}\alpha$	$\frac{\kappa}{\tau+\kappa+1}$

*Uniform hashing
**Uniform hashing with Furukawa's collision flag method
***Coalesced Chaining
†Scatter index table technique with Open addressing
††Scatter index table technique with Chaining

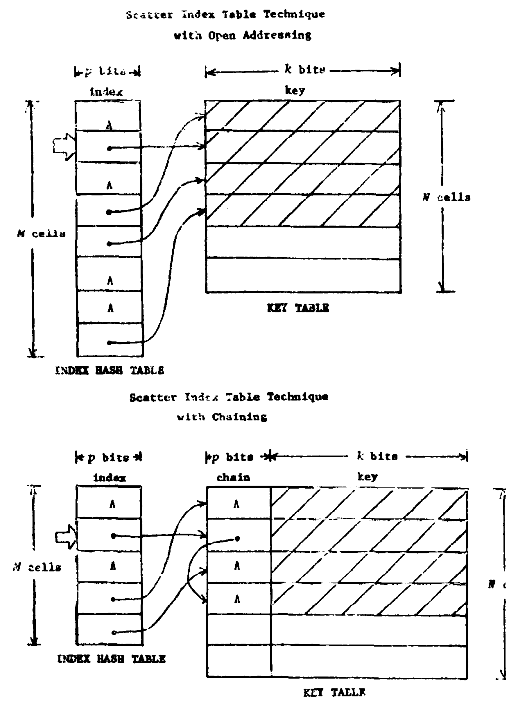


Fig. 2 Organization of scatter index tables.

heads of chains and α can be greater than unity. As for σ's in (2.1) and (2.2), they have coherent meanings.

Now U, I, and S for the algorithms mentioned above are tabulated in Table 1 (cf. Appendix B). Let us take examples to get an idea how comparison based on σ looks.

Example 1 Short keys ($\kappa=1$). In coalesced chaining $\sigma=(1/2)\alpha\leq 1/2$: thus only half the storage, at most, can be used to store keys. In other words, if the total available storage is the same, and the number of keys is the same, α of coalesced chaining becomes twice as large as that of uniform hashing. Hence, a comparison of these two algorithms should be made by substituting $\alpha=\sigma$ for uniform hashing, and $\alpha=2\sigma$ for coalesced chaining, as done in Table 2. (Because the scatter index table techniques are not advantageous for short keys, they are omittec.) We can see that uniform hashing is undoubtedly superior to coalesced chaining when keys are short; even the naive scheme without collision flage is more efficient as far as I and S are concerned.

Table 2 Average numbers of probes in terms of σ ($\kappa=1$).

	σ	0.2	0.3	0.4	0.5	0.6	0.8	0.9
U	U	1.250	1.429	1.667	2.000	2.500	5.000	10.000
	UF	1.022	1.053	1.103	1.181	1.305	1.916	3.028
	CC	1.106	1.280	1.588	2.097	—	—	—
I	U, UF	1.250	1.429	1.667	2.000	2.500	5.000	10.000
	CC	1.703	2.373	3.369	4.816	—	—	—
S	U, UF	1.116	1.189	1.277	1.386	1.527	2.012	2.558
	CC	1.233	1.383	1.568	1.799	—	—	—

Example 2 Long keys ($\kappa=4$). In Table 3, $\kappa=4$ is substituted for the five algorithms. Thus $\alpha=(5/4)\sigma$ for coalesced chaining, and we can see that it is now almost comparable to uniform hashing. As for the scatter index table techniques, parameter τ must be fixed. This value is determined by considering space/time tradeoffs; the larger τ, the smaller σ_m, maximum storage utility factor, but the more efficient. In Table 3, $\tau=2$ in the open scatter index table, and $\tau=1$ in the chained scatter index table, making both algorithms work up to $\sigma=2/3$. Thus $\alpha=(3/4)\sigma$ with open addressing, and $\alpha=(3/2)\sigma$ with chaining. We can see that the scatter index table techniques are the most advantageous in this example. Though these σ_m, techniques result in a rather small they have advantages other than efficiency; the key table can be expanded without affecting the hashing strategy, and deletion of keys is trivial (Morris [12]). We will discuss the latter point again in Section 3.

Table 3 Average numbers of probes in terms of σ ($\kappa=4$).

	σ	0.2	0.3	0.4	0.5	0.6	0.8	0.9
	U	1.250	1.429	1.667	2.000	2.500	5.000	10.000
	UF	1.022	1.053	1.103	1.181	1.305	1.916	3.028
U	CC	1.037	1.092	1.180	1.311	1.495	2.097	—
	SO	1.026	1.065	1.129	1.225	1.368	—	—
	SC	1.041	1.088	1.149	1.222	1.307	—	—
	U, UF	1.250	1.429	1.667	2.000	2.500	5.000	10.000
I	CC	1.358	1.638	2.004	2.479	3.083	4.816	—
	SO	1.176	1.290	1.429	1.600	1.818	—	—
	SC	1.300	1.450	1.600	1.750	1.900	—	—
	U, UF	1.116	1.189	1.277	1.386	1.527	2.012	2.558
S	CC	1.137	1.216	1.305	1.404	1.518	1.799	—
	SO	1.083	1.133	1.189	1.253	1.329	—	—
	SC	1.150	1.225	1.300	1.375	1.450	—	—

Note: $\tau=2$ in SO and $\tau=1$ in SC

Before closing this section, it should be mentioned that the motivation of Bays' idea ([1]) of introducing bilateral transformation between load factors for open addressing and chaining seems to be similar to ours. However, our scheme is an intrinsically multilateral transformation of various load factors. Because the storage utility factor crucially depends on the key length κ, it offers, so to speak, a unified viewpoint, regardless of algorithms used.

3. Deletion in Open Addressing

In Section 1 we mentioned the three occupancy states of a cell—*empty*, *occupied*, and *deleted*. The first one is an unoccupied state which is not in collisions; the second is an occupied state which is or is not in collisions; and the last is an unoccupied state which is in collisions. Generally, the cells are classified into $2M$ states in a similar but more minute way as below. Let a_i and b_i denote the fractions of cells in state A_i and B_i, respectively ($0 \le i \le M-1$).

		occupied	
		yes	no
	none	A_0	B_0
	once	A_1	B_1
collisions	twice	A_2	B_2
	$\vdots$	$\vdots$	$\vdots$
	$M-1$ times	A_{M-1}	B_{M-1}

Also let A_c and B_c denote the "unions" of A_i ($1 \le i \le M-1$) and B_i ($1 \le i \le M-1$), respectively. If empty and deleted states are indicated by the respective reserved values that are not identical with any key, these reserved values corr spond to state B_0 (*empty*) and B_c (*deleted*), respectively. Note that in Furukawa's collision flag method, we can distinguish B_c from B_0 by checking the collision flag. Thus no reserved value is necessary. Moreover, A_c also is distinguished from A_0 by the flag. Therefore, we can distinguish four states by the flag and occupancy of cells.

The deleted state B_c is double-faced in the sense that, on searching, it acts as an occupied state, and on insertion, as an empty state, so that the average number of probes for unsuccessful searching is not reduced. What is worse is, though a cell in state B_c may sometime become occupied again, it will never automatically be empty again, even if all the colliding keys are deleted from the table.

Therefore, when deleted cells pile up to cause too inefficient searching, the hash table should somehow be updated in order to reclaim empty cells. The following update scheme is different from the so-called "rehash" scheme, because no keys are relocated, and some deleted cells may remain.

UPDATE SCHEME When deleted cells pile up to a certain prespecified amount, all the deleted cells are first marked as "empty." Then for each remaining key, the probe sequence is traced, and empty cells encountered on the way are marked as "deleted" (by putting a reserved value or by setting the collision flag to 1). This UPDATE SCHEME can be part of a garbage collector, if any, for the system.

Thus only the cells now in state B_c are marhed as "deleted". Since each probe sequence is one for successful searching, nS probes are necessary for a total update. Because deleted cells still remain after an update, average numbers of probes are not simply related to α. In fact, they are not determined by α alone; they depend on the past history of insertion and deletion.

Because mathematical settings for the analysis of average numbers of probes are rather complicated, the next section is dedicated to such an analysis, and we use only the results obtained, in this section.

According to Section 4, if the system always uses the hash table within a certain maximum load factor α_m, the average numbers of probes do not exceed the following upper bounds.

In the scatter index table techniques mentioned in the previous section, deletion causes no serious problem, since we can "rehash" the index hash table without affecting the location of keys in the separate key table. Now these techniques are compared with uniform hashing as follows. Table 5 shows values of the maximum storage utility factor σ_m to realize values of U, I, and S

Table 4 Upper bounds of the average numbers of probes.

	U	I	S
UD*	e^{β_m}	e^{β_m}	$\frac{1}{1-\alpha_m}$
UFD**	$\frac{1}{F(\beta_m)+e^{-\beta_m}}$	$\frac{1}{F(\beta_m)+e^{-\beta_m}}+\frac{1}{1-\alpha_m}-\frac{1}{1-\alpha_m+F(\beta_m)}$	$\frac{1}{1-\alpha_m}$

*Uniform hashing with Deletion

**Uniform hashing with Furukawa's collision flag method and Deletion

Note: $\beta_m=\frac{\alpha_m}{1-\alpha_m}$, $F(x)=x\sum_{n=0}^{\infty}\frac{(-x)^n}{n!}\frac{1}{x+n+1}$

Table 5(a) σ_m when U_m is given.

	U_m	2.0	3.0	4.0	5.0	6.0	7.0	8.0
UD*	σ_m	0.409	0.523	0.581	0.617	0.642	0.661	0.675
UFD**	σ_m	0.571	0.650	0.689	0.712	0.72	0.742	0.752
SO***	τ	1.618	1.366	1.264	1.208	1.171	1.145	1.127
	$\kappa=1$	0.382	0.422	0.442	0.453	0.461	0.466	0.470
σ_m	$\kappa=2$	0.553	0.594	0.613	0.623	0.631	0.636	0.640
	$\kappa=3$	0.650	0.687	0.703	0.712	0.719	0.724	0.727
	$\kappa=4$	0.712	0.745	0.760	0.768	0.774	0.777	0.780
SC***	τ	0.543	0.339	0.251	0.200	0.167	0.143	0.125
	$\kappa=1$	0.393	0.427	0.444	0.454	0.462	0.467	0.471
σ_m	$\kappa=2$	0.564	0.599	0.615	0.625	0.632	0.636	0.640
	$\kappa=3$	0.660	0.691	0.706	0.714	0.720	0.724	0.727
	$\kappa=4$	0.722	0.749	0.762	0.769	0.774	0.778	0.780

*Uniform hashing with Deletion
**Uniform hashing with Furukawa's collision flag method and Deletion
***Scatter index table technique with Open addressing
****Scatter index table technique with Chaining

Table 5(c) σ_m when S_m is given.

	S_m	2.0	3.0	4.0	5.0	6.0	7.0	8.0
UD	σ_m	0.500	0.667	0.750	0.800	0.833	0.857	0.875
UFD	σ_m	0.500	0.667	0.750	0.800	0.833	0.857	0.875
SO	τ	1.255	1.064	1.020	1.007	1.003	1.001	1.000
	$\kappa=1$	0.443	0.485	0.495	0.498	0.499	0.500	0.500
σ_m	$\kappa=2$	0.614	0.653	0.662	0.665	0.666	0.666	0.667
	$\kappa=3$	0.705	0.738	0.746	0.749	0.749	0.750	0.750
	$\kappa=4$	0.761	0.790	0.797	0.799	0.800	0.800	0.800
SC	τ	0.500	0.250	0.167	0.125	0.100	0.083	0.071
	$\kappa=1$	0.400	0.444	0.462	0.471	0.476	0.480	0.483
σ_m	$\kappa=2$	0.571	0.615	0.632	0.640	0.645	0.649	0.651
	$\kappa=3$	0.667	0.706	0.720	0.727	0.732	0.735	0.737
	$\kappa=4$	0.727	0.762	0.774	0.780	0.784	0.787	0.789

Table 5(b) σ_m when I_m is given.

	I_m	2.0	3.0	4.0	5.0	6.0	7.0	8.0
UD	σ_m	0.409	0.523	0.581	0.617	0.642	0.661	0.675
UFD	σ_m	0.449	0.587	0.652	0.689	0.713	0.729	0.742
SO	τ	2.000	1.500	1.333	1.250	1.200	1.167	1.142
	$\kappa=1$	0.333	0.400	0.429	0.444	0.455	0.461	0.467
σ_m	$\kappa=2$	0.500	0.571	0.600	0.615	0.625	0.632	0.637
	$\kappa=3$	0.600	0.667	0.692	0.706	0.714	0.720	0.724
	$\kappa=4$	0.667	0.727	0.750	0.762	0.769	0.774	0.778
SC	τ	1.000	0.500	0.333	0.250	0.200	0.167	0.142
	$\kappa=1$	0.333	0.400	0.429	0.444	0.455	0.461	0.467
σ_m	$\kappa=2$	0.500	0.571	0.600	0.615	0.625	0.632	0.637
	$\kappa=3$	0.600	0.667	0.692	0.706	0.714	0.720	0.724
	$\kappa=4$	0.667	0.727	0.750	0.762	0.769	0.774	0.778

that are not greater than the prespecified values of U_m, I_m, and S_m respectively: that is, in order to keep U, I, and S smaller than U_m, I_m, and S_m, respectively, the hash table must be used within the specified σ_m. As for UD and UFD, the derivation of σ_m can be trivial. SO and SC depend on the values of τ, since $\sigma_m = \kappa/(\tau+\kappa)$ (for SO) and $\sigma_m = \kappa/(\tau+\kappa+1)$ (for SC) (cf. Table 1). The appropriate τ can be determined as $\tau = 1/\alpha_m$, where α_m's are what make values of U, I, and S equal to U_m, I_m, and S_m, respectively (cf. Appendix B). Now, since σ_m shows how full the storage can be used by keys, the larger the σ_m, the more favorable. From Table 5 we can conclude that even if deletion is frequent, uniform hashing with or without collision flags is still more favorable than the scatter index table techniques, if $\kappa \le 2$, namely, for short keys. If keys are longer, we might resort to the latter schemes, unless full utility of the storage is more important than speed. Note that UD (or UFD) is the only algorithm that permits σ to reach 1.

4. Mathematical Analysis

4.1 Markovian Model

Let t be a discrete time variable that increases, one by one, starting with 0, and let either insertion or deletion of a single key take place at each time instant. Then a_i and b_i at time $t+1$ are determined by those at time t in the following way. First, we assume that the hash table is initially empty at $t=0$ so that;

$$a_i(0)=0 \text{ for } i \ge 0,\quad b_0(0)=1,\quad b_i(0)=0 \text{ for } i \ge 1. \tag{4.1}$$

If insertion takes place at time t,

$$a_i(t+1)=a_i(t)+\frac{1}{M-n(t)}b_i(t) - \frac{1}{M-n(t)+1}(a_i(t)-a_{i-1}(t)), \tag{4.2}$$

with the convention $a_{-1}(t) \leqq 0$,

$$b_i(t+1)=b_i(t)-\frac{1}{M-n(t)}b_i(t), \tag{4.3}$$

and if deletion takes place at time t,

$$a_i(t+1)=a_i(t)-\frac{i}{n(t)}a_i(t)+\frac{i+1}{n(t)}a_{i+1}(t)-\frac{1}{n(t)}a_i(t), \tag{4.4}$$

$$b_i(t+1)=b_i(t)-\frac{i}{n(t)}b_i(t)+\frac{i+1}{n(t)}b_{i+1}(t)+\frac{1}{n(t)}a_i(t), \tag{4.5}$$

where $n(t)$ is the number of keys in the table at time t.

(4.2) and (4.3) are derived as follows. On insertion, an inserted new key collides, on the average, with $(M+1)/(M-n(t)+1)-1$ occupied cells (cf. Knuth [10]). Hence, the probability that an occupied cell increases its number of collisions is $((M+1)/(M-n(t)+1)-1)/n(t) = 1/(M-n(t)+1)$. Because the new key is stored in one of $M-n(t)$ unoccupied cells, the probability that an unoccupied cell turns into an occupied cell with the same number of collisions is $1/(M-n(t))$. As for (4.4) and (4.5), consider a cell in state A_i (or B_i). Since one of $n(t)$ keys is deleted, the probability that one of i keys colliding with this cell is deleted is $i/n(t)$. This is equal to the probability that this cell turns into state A_{i-1} (or B_{i-1}). Since the new deleted cell was one of $n(t)$ occupied cells before the deletion, the probability that a cell in state A_i turns into B_i is $1/n(t)$.

From the definition of a_i and b_i, we have

$$\sum_{i=0}^{M-1} a_i(t)=\frac{n(t)}{M}=\alpha(t), \quad \sum_{i=0}^{M-1} b_i(t)=1-\alpha(t). \tag{4.6}$$

$U(t)$, $I(t)$, and $S(t)$ are related to $a_i(t)$ and $b_i(t)$, as below, when the collision flag method is employed.

$$U(t)=\frac{1}{a_0(t)+b_0(t)}, \tag{4.7}$$

$$I(t)=\frac{1}{\alpha_0(t)+b_0(t)}+\frac{1}{1-\alpha(t)}-\frac{1}{a_0(t)+1-\alpha(t)}, \tag{4.8}$$

$$S(t)=\frac{1}{\alpha(t)}\sum_{i=0}^{M-1}\{(i+1)a_i(t)+ib_i(t)\}. \tag{4.9}$$

(4.7) and (4.8) are proved in Appendix A. (4.9) is obvious since the total number of probes is $n(t)S(t)=M\alpha(t)S(t)$, and this is equal to the total number of collisions plus $n(t)$; i.e.,

$$M\sum_{i=0}^{M-1}\{ia_i(t)+ib_i(t)\}+M\sum_{i=0}^{M-1}a_i(t).$$

Using (4.2)–(4.5) with (4.9) we have, if insertion takes place at time t (note that $a_i(t)=0$ for $i<M-2$, and $b_i(t)=0$ for $i<M-1$ by the initial conditions),

$$S(t+1)=\frac{1}{n(t)+1}\left\{n(t)S(t)+\frac{M+1}{M-n(t)+1}\right\}, \tag{4.10}$$

and if deletion takes place at time t,

$$S(t+1)=S(t). \tag{4.11}$$

If the collision flag method is not employed, U and I differ from (4.7) and (4.8). By denoting the average numbers of probes for this case by U^*, I^* and S^*,

$$U^*(t)=I^*(t)=\frac{1}{b_0(t)}, \tag{4.12}$$

since searching cannot be stopped until an empty cell is found. S^* is the same as (4.9), or (4.10) and (4.11): i.e.,

$$S^*(t)=S(t).$$

4.2 Loading History Functions

It is obvious that $n(t+1)=n(t)>1$; $n(t+1)=n(t)+1$ indicating insertion at time t and $n(t+1)=n(t)-1$ indicating deletion at time t. In this subsection we consider general properties of such functions.

Definition A *loading history function n* is a function of discrete variable $t\in\{0, 1, 2, \ldots, T\}$ such that,

$$n(0)=0, \quad n(t+1)=n(t)\pm 1 \text{ for all } t,$$
$$\text{and } N\leqq \max_t n(t)<M, \text{ or } \alpha_m\leqq N/M<1. \tag{4.13}$$

The value of N is the prespecified maximum number of keys; the table is updated when the number of keys reaches this value so as to recover empty cells. The set of n's is obviously finite, though T may be arbitrarily large.

Definition Let n_1 and n_2 be two loading history functions. We write $n_1\subseteq n_2$ and read "n_1 is *under* n_2" iff $n_1(t)\leq n_2(t)$ for all t $(0\leq t\leq T)$. Let us write $n_1=n_2$ iff $n_1\subseteq n_2$ and $n_2\subseteq n_1$, and $n_1\subset n_2$ iff $n_1\subseteq n_2$ and $n_1\neq n_2$.

It is easily verified that the relation $\subseteq$ is reflexive, antisymmetric, and transitive. Hence the set of loading history functions forms a finite partially-ordered set. We denote this partially-ordered set as $\mathscr{L}$

Definition We writen $n_1\subset_p n_2$ and read "n_1 is *prime under* n_2" iff $n_1\subset n_2$ and there exists no $n_3\in\mathscr{L}$ such that $n_1\subset n_3\subset n_2$.

The following lemma may be obvious.

Lemma4.1 If $n_1\subset_p n_2$, then there exists one and only one time t_p such that,

$$n_1(t)=n_2(t) \text{ for } t\text{—}t_p, \quad n_1(t_p)=n_1(t_p-1)-1,$$
$$\text{and } n_2(t_p)=n_2(t_p-1)+1, \tag{4.14}$$

and vice versa (cf. Fig. 3).

Definition Let a subset $\{n_{10}, n_{11}, \cdots, n_{1k}\}$ of $\mathscr{L}$ be totally ordered and its elements be arranged in ascending order. This subset is said to be a *connected chain* between n_{10} and n_{1k} iff $n_{10}\subset_p n_{11}\subset_p\cdots\subset_p n_{1k}$. k is said to be the *length* of the connected chain.

Lemma 4.2 Let $n_1\subset n_2$. Then there is at least one connected chain between n_1 and n_2. Moreover, the length k of the connected chain between n_1 and n_2 is determined by n_1 and n_2, independently of the chain taken, as

$$k=\sum_{t=0}^{T}\{n_2(t)-n_1(t)\}. \tag{4.15}$$

Proof Appendix C.

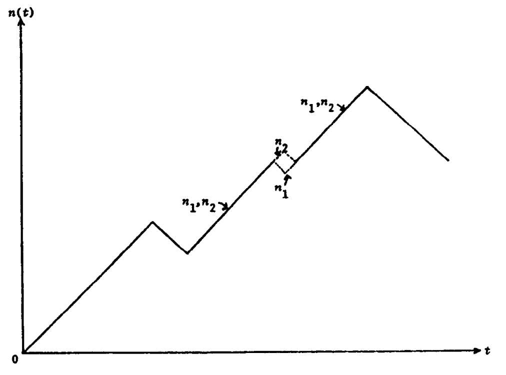


Fig. 3 n_1 is prime under n_2.

4.3 Average Numbers of Probes as Functionals of *n*

Since the loading history function n determines the whole past history, we can calculate the values of a_i, b_i, U, I, and S at any given time t, if n is specified. In other words, the values depend directly on the loading history functions. Hence a_i, b_i, U, I, and S are functionals of n, rather than being functions of time t itself. We hereafter denote these quantities at time t as functionals of n, like $a_i[n; t]$.

According to Appendix C, $\mathscr{L}$ forms a finite modular lattice. It is clear that n_m, as defined below, is the maximum element of the finite lattice $\mathscr{L}$.

$$n_m(t)\pm t \text{ for } 0\le t\le N, \quad n_m(N+t)=N \text{ for event } t>0,$$
$$\text{and } n_m(N+t)=N-1 \text{ for odd } t>0. \tag{4.16}$$

Definition A functional $F[n; t]$ is said to be *monotone increasing* if it satisfies the following:

$$\text{If } n_1\subset n_2 \text{ then } F[n_1; t]\le F[n_2; t] \text{ for all } t. \tag{4.17}$$

Conjecture 4.1 U, U^*, I, and I^* are monotone increasing functionals.

Unfortunately, we find it hard to prove the conjecture mathematically. Instead, we performed numerous numerical calculations and confirmed that the conjecture holds with very high numerical accuracy.

As for S, it is not a monotone increasing functional of the loading history function. However, we have the following lemma:

Lemma 4.3 Let n be an arbitrary loading history function and t_1 be an arbitrary time instant, and let n_m be defined in terms of n, t_1, and n_m as follows (cf. Fig. 4):

$$n'_m(t)=n_m(t) \text{ for } 0\le t\le t_1+n(t_1)-N+1,$$
$$n'_m(t)=n(t_1)+t_1-t \text{ for } t_1+n(t_1)-N+1<t\le t_1,$$

(i.e., deletion takes place from time $t_1+n(t_1)-N+1$ to t_1-1).

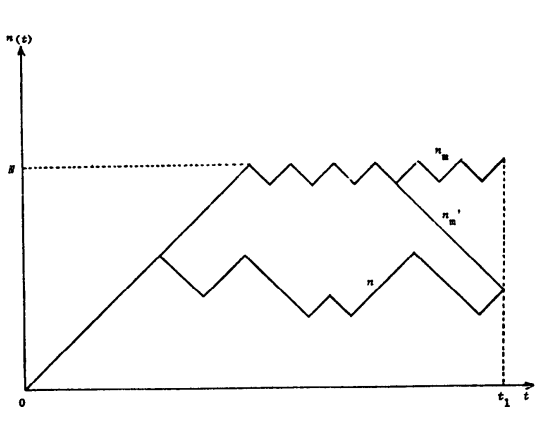


Fig. 4 n, n'_m and n_m in Lemma 4.3.

Then,

$$S[n; t_1]<S[n_m; t_1], \quad (\text{we assume } n\neq{}'_m) \tag{4.18a}$$

and

$$S[n'_m; t_1]\le S[n_m; t_1]. \tag{4.18b}$$

Proof (4.18a): By virtue of LEMMA 4.2, there is a connected chain between n and n'_m such that $n=n_0\subset_p n_1\subset_p\cdots\subset_p n_k=n'_m$. Hence we have only to prove:

$$S[n_j; t_1]<S[n_{j+1}; t_1], \quad \text{where } n_j\subset_p n_{j+1}.$$

By Lemma 4.1, there is a time instant $t_p<t_1$ such that $n_j(t)=n_{j+1}(t)$ for $t\neq t_p$, $n_j(t_p)=n_j(t_p-1)-1$, $n_{j+1}(t_p)=n_{j+1}(t_p-1)+1$. Then

$$S[n_j; t]=S[n_{j+1}; t] \quad \text{for } t<t_p.$$

Let $K=n_j(t_p-1)=n_{j+1}(t_p-1)$ and $s=S[n_j; t_p-1]=S[n_{j+1}; t_p-1]$. Then,

$$S[n_j; t_p]=S[n_j; t_p-1]=s,$$
$$S[n_{j+1}; t_p]=\frac{1}{K+1}\left\{KS[n_{j+1}; t_p-1]+\frac{M+1}{M-K+1}\right\}$$
$$=\frac{1}{K+1}\left(Ks+\frac{M+1}{M-K+1}\right).$$

On the other hand,

$$S[n_j; t_p+1]=\frac{1}{K}\left\{(K-1)S[n_j; t_p]+\frac{M+1}{M-K+2}\right\}$$
$$=\frac{1}{K}\left((K-1)s+\frac{M+1}{M-K+2}\right),$$
$$S[n_{j+1}; t_p+1]=S[n_{j+1}; t_p]=\frac{1}{K+1}\left(Ks+\frac{M+1}{M-K+1}\right).$$

Hence

$$S[n_{j+1}; t_p+1]-S[n_j; t_p+1]$$
$$=\frac{1}{K(K+1)}\left\{s+\frac{(2K-M-1)(M+1)}{(M-K+1)(M-K+2)}\right\}>0,$$

since $s\ge 1$ and the second term in the braces is greater than -1 for $0\le K\le M$. Since $n_j(t)=n_{j+1}(t)$ for $t<t_p$, by virtue of (4.10) and (4.11),

$$S[n_j; t_1]<S[n_{j+1}; t_1].$$

(4.18b): Since $S[n_m; t_1]=S[n_m; t_1+n(t_1)-N+1]=S[n_m; t_1+n(t_1)-N+1]$, we have only to prove:

$$S[n_m; t_1+n(t_1)-N+1]\le S[n_m; t_1],$$

or more simply,

$$S[n_m; t]\le S[n_m; t+1].$$

Since the case of $t>N$ is trivial, we assume $t\ge N$. Then there are two cases according as $n_m(t)=N$ or $N-1$.

Case 1 $n_m(t)=N$. Then

$$S[n_m; t+1]=S[n_m; t].$$

Case 2 $n_m(t)=N-1$. Then

$$S[n_m; t+1]=\frac{1}{N}\left\{(N-1)S[n_m; t]+\frac{M+1}{M-N+2}\right\}.$$

Hence if $S[n_m; t] > (M+1)/(M-N+2)$, then

$$S[n_m; t] < S[n_m; t+1] < \frac{M+1}{M-N+2}.$$

On the other hand, since $n_m(t) = t$ (for $0 \le t \le N$),

$$S[n_m; N] = \frac{1}{N} \sum_{k=0}^{N-1} \frac{M+1}{M-k+1} < \frac{M+1}{M-N+2}.$$

Hence

$$S[n_m; t] < S[n_m; t+1].$$

4.4 Upper Bound of the Average Numbers of Probes

By Conjecture 4.1 for U, U^*, I, and I^*, and by Lemma 4.3 for S, we have the following corollary.

Corollary 4.1 The values of U, U^*, I, I^*, and S at time t are bounded by those of $U[n_m; t]$, $U^*[n_m; t]$, $I[n_m; t]$, $I^*[n_m; t]$, and $S[n_m; t]$ respectively.

In case of U^* and I^* (cf. (4.12)), we can set a smaller upper bound owing to the simple form of eq. (4.3). Suppose insertion repeatedly takes place from time t to time $t+s-1$. By the repeated use of (4.3), we have

$$b_0(t+s) = \frac{M-n(t+s)}{M-n(t)} b_0(t). \tag{4.19}$$

Hence,

$$U^*[n; t] = \frac{M-n(t+s)}{M-n(t)} U^*[n; t+s]. \tag{4.20}$$

(Since U^* and I^* are always the same, we omit the formulae for I^*.)

By virtue of Corollary 4.1

$$U^*[n; t+s] \le U^*[n_m; t+s]. \tag{4.21}$$

Hence, by (4.21) and (4.20)

$$U^*[n; t] \le \frac{M-n(t+s)}{M-n(t)} U^*[n_m; t+s]. \tag{4.22}$$

Let $s = N - n(t)$. Then $n(t+s) = n(t) + s = N$ and $n_m(t+s) = N$. Since $(M - n_m(t+s))U^*[n_m; t+s] = (M - n_m(t))U^* \cdot [n_m; t]$ by virtue of (4.20),

$$U^*[n; t] \le \frac{M-N}{M-n(t)} U^*[n_m; t+s] = \frac{M-n_m(t)}{M-n(t)} U^*[n_m; t]. \tag{4.23}$$

For $t \to \infty$, $U[n_m; t]$, $U^*[n_m; t]$, $I[n_m; t]$, $I^*[n_m; t]$, and $S[n_m; t]$ are shown to be asymptotically constant independently of time t.

Let U_m, I_m, and S_m be the asymptotic values of $U[n_m; t]$, $I[n_m; t]$, and $S[n_m; t]$, respectively, as $t \to \infty$. In order to estimate these values, it is convenient to treat (4.2)–(4.5) as transitions of a vector. Let $c(t)$ be a vector whose components are

$$c_{2i}(t) = a_i(t) \quad \text{and} \quad c_{2i+1}(t) = b_i(t) \qquad (0 \le i \le N-1). \tag{4.24}$$

By virtue of (4.6), $c(t)$ is a probability vector in the sense:

$$\sum_{i=0}^{2N-1} c_i(t) = 1. \tag{4.25}$$

(4.2) and (4.3) can be expressed as a formula:

$$c(t+1) = P(n(t))c(t), \tag{4.26}$$

where $P(n(t))$ is a transition matrix that corresponds to (4.2) and (4.3) as

$$P_{2i,2i}(n(t)) = 1 - \frac{1}{M-n(t)+1},$$

$$P_{2i,2i-2}(n(t)) = \frac{1}{M-n(t)+1},$$

$$P_{2i,2i+1}(n(t)) = \frac{1}{M-n(t)}$$

$$P_{2i+1,2i+1}(n(t)) = 1 - \frac{1}{M-n(t)},$$

and all others are zero $(0 \le i \le N-1)$.

$P(n(t))$ is a non-negative Markov matrix in the sense:

$$P_{ij}(n(t)) \ge 0 \text{ for } 0 \le i, j \le 2N-1,$$

$$\text{and } \sum_{i=0}^{2N-1} p_{ij}(n(t)) = 1 \text{ for } 0 \le j \le 2N-1, \tag{4.27}$$

(4.4) and (4.5) can be expressed similarly:

$$c(t+1) = Q(n(t))c(t), \tag{4.28}$$

where

$$q_{2i,2i}(n(t)) = 1 - \frac{i}{n(t)} - \frac{1}{n(t)},$$

$$q_{2i-2,2i}(n(t)) = \frac{i}{n(t)},$$

$$q_{2i+1,2i+1}(n(t)) = 1 - \frac{i}{n(t)},$$

$$q_{2i-1,2i+1}(n(t)) = \frac{i}{n(t)},$$

$$q_{2i+1,2i}(n(t)) = \frac{1}{n(t)},$$

and all others are zero $(0 \le i \le N-1)$.

$Q(n(t))$ is also a non-negative Markov matrix.

Now consider n_m. For $t < N$, the transition is in the form of $c(t+1) = P(N-1)c(t)$ and $c(t+1) = Q(N)c(t)$ alternately. Hence

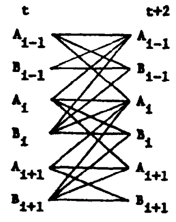
$$c(t+2) = P(N-1)Q(N)c(t), \tag{4.29}$$

or

$$c(t+2) = Q(N)P(N-1)c(t). \tag{4.30}$$

Lemma 4.4 $P(N-1)Q(N)$ is irreducible in the sense: $(P(N-1)Q(N))^k < 0$ for some $k < 0$. Similarly, $Q(N)$ $P(N-1)$ is irreducible.

Proof In the diagram below, a state at time t and a



state at time $t+2$ are connected by a line if there is a positive matrix-element in $P(N-1)Q(N)$, which allows a transition between the two states. Since $A_i(t+2)$ is connected with $A_{i-1}(t)$, $A_i(t)$ and $A_{i+1}(t)$, all the A_i are connected in $(P(N-1)Q(N))^k$ if $k \geq N$. Although $B_i(t+2)$ is not connected with $B_{i-1}(t)$, it is easy to see that $B_i(t+6)$ is connected with $B_{i-1}(t)$ in $(P(N-1)Q(N))^3$ via $A_{i-1}(t+2)$ and $A_i(t+4)$. Hence all the B_i's are connected in $(P(N-1)Q(N))^k$ if $k \geq 3N$. Likewise, all the A_i's and the B_i's are mutually connected with one another in $(P(N-1)Q(N))^k$, if k is at least $3N$. As for $Q(N)P(N-1)$, the proof is quite similar. □

By virtue of Lemma 4.3, $(P(N-1)Q(N))^k$ and $(Q(N)P(N-1))^k$ are positive Markov matrices for a sufficiently large k. Let $R^{(1)}(N)$ and $R^{(2)}(N)$ denote the positive Markov matrices which are thus constructed from powers of $P(N-1)Q(N)$ and $Q(N)P(N-1)$ respectively. Then (4.29) and (4.30) take the form:

$$c(t+2k)=R^{(j)}(N)c(t) \quad j=1, 2 \text{ for some } k>0. \quad (4.31)$$

As for the asymptotic behavior of the equation like (4.31), the following result has been obtained (Bellman [2, p. 266]):

$\lim_{t\to\infty} c(t)=d^{(j)}$, where $d^{(j)}$ is a probability vector which is independent of $c(0)$,
and $d^{(j)}$ is a characteristic vector of $R^{(j)}(N)$ with associated characteristic root 1.

Hence we can conclude that in (4.31), vector $c(t)$ asymptotically approaches a steady state $d^{(j)}$, which is a characteristic vector of $R^{(j)}$: i.e., $d^{(j)}=R^{(j)}(N)d^{(j)}$. The specific form of each component of $d^{(j)}$ can easily be obtained in the following manner.

Let $f(x, t)$ and $g(x, t)$ be generating functions of $a_i(t)$ and $b_i(t)$ respectively:

$$f(x, t)=\sum_{i=0}^{N-1} x^i a_i(t), \quad g(x, t)=\sum_{i=0}^{N-1} x^i b_i(t). \quad (4.32)$$

Then (4.2) and (4.3) for insertion are expressed in terms of $f(x, t)$ and $g(x, t)$ as

$$f(x, t+1)=f(x, t)+\frac{1}{M-n(t)}g(x, t) - \frac{1-x}{M-n(t)+1}f(x, t), \quad (4.33)$$

$$g(x, t+1)=g(x, t)-\frac{1}{M-n(t)}g(x, t). \quad (4.34)$$

Similarly, (4.4) and (4.5) for deletion become

$$f(x, t+1)=f(x, t)+\frac{1-x}{n(t)}\frac{\partial f(x, t)}{\partial x}-\frac{1}{n(t)}f(x, t), \quad (4.35)$$

$$g(x, t+1)=g(x, t)+\frac{1-x}{n(t)}\frac{\partial g(x, t)}{\partial x}+\frac{1}{n(t)}f(x, t). \quad (4.36)$$

In the equation: $d^{(1)}=R^{(1)}(N)d^{(1)}$, since $R^{(1)}(N)$ is at least $3N$-th power of $P(N-1)Q(N)$, $P(N-1)Q(N)R^{(1)}\cdot(N)\mathscr{L}R^{(1)}(N)$ if $N \gg 1$. Hence

$$d^{(1)}=P(N-1)Q(N)d^{(1)}. \quad (4.37)^\dagger$$

If deletion takes place at time t and then insertion takes place at time $t+1$, we have from (4.33)–(4.36) (by putting $n(t)=N$),

$$f(x, t+2)=f(x, t)+\frac{1-x}{N}\frac{\partial f(x, t)}{\partial x}-\frac{1}{N}f(x, t)+\frac{1}{M-N+1}\left\{g(x, t)+\frac{1-x}{N}\frac{\partial g(x, t)}{\partial x}+\frac{1}{N}f(x, t)\right\}$$
$$-\frac{1-x}{M-N+2}\left\{f(x, t)+\frac{1-x}{N}\frac{\partial f(x, t)}{\partial x}-\frac{1}{N}f(x, t)\right\}, \quad (4.38)$$

$$g(x, t+2)=g(x, t)+\frac{1-x}{N}\frac{\partial g(x, t)}{\partial x}+\frac{1}{N}f(x, t)-\frac{1}{M-N+1}\left\{g(x, t)+\frac{1-x}{N}\frac{\partial g(x, t)}{\partial x}+\frac{1}{N}f(x, t)\right\}. \quad (4.39)$$

Let $f^{(1)}(x)$ and $g^{(1)}(x)$ be generating functions of the components of $d^{(1)}$, which is independent of t. Then we have from (4.37)–(4.39),

$$(1-x)\{1-\gamma(1-x)\}\frac{d^2f^{(1)}(x)}{dx^2}-\{\beta+2-(\beta+3-N)\gamma(1-x)\}\frac{df^{(1)}(x)}{dx}+\gamma(N-1)(\beta+1)f^{(1)}(x)=0, \quad (4.40)$$

$$(1-x)\{1-\gamma(1-x)\}\frac{d^2g^{(1)}(x)}{dx^2}-\{\beta+2-(\beta+2-N)\gamma(1-x)\}\frac{dg^{(1)}(x)}{dx}+\gamma(N-1)\beta g^{(1)}(x)=0, \quad (4.41)$$

where $\gamma=\dfrac{1}{M-N+2}$ and $\beta=\dfrac{N}{M-N}$.

(4.40) and (4.41) can be solved in terms of Gaussian hypergeometric functions. The ones that do not diverge at $x=1$ (since $f^{(1)}(1)=N/M$ and $g^{(1)}(1)=1-N/M$) are

†Strictly speaking, $d^{(1)}$ is the characteristic vector of $(P(N-1)Q(N))^k$. This asymptotic equation corresponds to counting only the terms of $O(1/N^2)$, neglecting $O(1/N^3)$.

$$f^{(1)}(x)=\frac{N}{M}F(-(N-1),\ \beta+1,\ \beta+2;\ \gamma(1-x)), \tag{4.42}$$

$$g^{(1)}(x)=\left(1-\frac{N}{M}\right)F(-(N-1),\ \beta,\ \beta+2;\ \gamma(1-x)), \tag{4.43}$$

where

$$F(p,q,r;z)=\frac{\Gamma(r)}{\Gamma(p)\Gamma(q)}\sum_{i=0}^{\infty}\frac{\Gamma(p+i)\Gamma(q+i)}{\Gamma(r+i)}\frac{z^i}{i!}. \tag{4.44}$$

Similarly, generating functions $f^{(2)}(x)$ and $g^{(2)}(x)$ of components of $d^{(2)}$, which satisfies $d^{(2)}=Q(N)P(N-1)d^{(2)}$, are

$$f^{(2)}(x)=\frac{N-1}{M}F(-(N-2),\ \beta+1,\ \beta+2;\ \gamma(1-x)), \tag{4.45}$$

$$g^{(2)}(x)=\left(1-\frac{N-1}{M}\right)F(-(N-1,\ \beta,\ \beta+2;\ \gamma(1-x)). \tag{4.46}$$

The components of $d^{(j)}$ are obtained from $f^{(j)}(x)$ and $g^{(j)}(x)$ as

$$a_i^{(j)}=\left[\frac{1}{i!}\frac{d^if^{(j)}(x)}{dx^i}\right]_{x=0},\quad b_i^{(j)}=\left[\frac{1}{i!}\frac{d^ig^{(j)}(x)}{dx^i}\right]_{x=0}. \tag{4.47}$$

Note that $f^{(1)}(x)$, $g^{(1)}(x)$, and $g^{(2)}(x)$ are polynomials of degree $N-1$, and $f^{(2)}(x)$ of degree $N-2$. Hence no components with a higher suffix than $N-1$ or $N-2$, appear.

Since U, U^*, I, I^*, and S sre related to a_i and b_i as in (4.7)–(4.9), or (4.12), asymptotic values U_m, U_m^*, I_m, I_m^*, and S_m can be calculated using (4.47). In particular, U_m^* (and I_m^*) for uniform hashing without collision flags can be expressed in a simpler form. Notice that $b_0^{(1)}$ $(=g^{(1)}(0))$ and $b_0^{(2)}$ $(=g^{(2)}(0))$ are almost equal if M, $N\gg 1$. By replacing $(N-1)/M$ and N/M by α_m,

$$U_m^*=I_m^*\sim\frac{1}{(1-\alpha_m)F(-(N-1),\ \beta,\ \beta+2;\ \gamma)}. \tag{4.48}$$

When $N\to\infty$ and $N\gamma\simeq\alpha_m/(1-\alpha_m)$ remains finite, the right-hand side of (4.48) becomes $e^{\alpha_m/(1-\alpha_m)}$. (See Appendix D for the proof.) Thus

$$U_m^*=I_m^*\sim e^{\alpha_m/(1-\alpha_m)}. \tag{4.49}$$

Hence by virtue of (4.23) and (4.49), we have the following upper bound.

$$U^*[n;t]=I^*[n;t]\leq\frac{1-\alpha_m}{1-\alpha(t)}U_m^*\sim\frac{1-\alpha_m}{1-\alpha(t)}e^{\alpha_m(1-\alpha_m)}, \tag{4.50}$$

where $\alpha(t)=n(t)/M$.

As for S_m, while we can calculate the asymptotic value by (4.9), it can be derived directly from (4.10) and (4.11):

$$S_m\sim\frac{M+1}{M-N+2}=\frac{1}{1-\alpha_m}. \tag{4.51}$$

Hence

$$S[n;t]\leq S_m\sim\frac{1}{1-\alpha_m}. \tag{4.52}$$

Table 6 Maximum average numbers of probes when deletion is frequent.

	α_m	0.2	0.3	0.4	0.5	0.6	0.7	0.8
U_m	UD*	1.824	1.535	1.948	2.718	4.881	10.312	54.60
	UFD**	1.049	1.129	1.281	1.582	2.272	4.428	18.83
I_m	UD	1.284	1.535	1.948	2.718	4.881	10.312	54.60
	UFD	1.273	1.487	1.795	2.274	3.151	5.429	19.57
S_m	UD, UFD	1.250	1.429	1.667	2.000	2.500	3.333	5.00

*Uniform hashing with Deletion

**Uniform hashing with Furukawa's collision flag method and Deletion

U_m, I_m, and S_m are tabulated in Table 6.

5. Conclusion

In this paper we have introduced a new vehicle σ, the storage utility factor, for use in the analysis of hashing. Although we are not necessarily against chaining, we have seen that open addressing is often favorable in the light of storage and efficiency. Remember that, except for open addressing, every other algorithm (including chaining and the scatter index table techniques) has an upper bound of σ less than 1. The fact that α, the load factor, can become equal to 1 (coalesced chaining), or even greater than 1 (scatter index chaining), by no means implies that the storage is utilized quite effectively; in terms of the storage utility factor, they are valid within a rather limited range of σ. In this regard we suspect that comparisons based on the load factor (such as the figures in Knuth [10, p. 539]) are misleading.

In Section 2 we have treated only the cases where "bucket" size b is 1. However, σ for $b<1$ may easily be obtained by (1.1). For instance, σ for coalesced chaining becomes $kn/((kb+p)M)=(\kappa/(\kappa+1/b))\alpha$, where $\alpha=n/Mb$. Needless to say, comparison of various algorithms using buckets should be based on such σ's.

As for deletion, most of the previous treatments seem to have been only qualitative. By virtue of our model of loading a hash table, and the update scheme presented in Section 3, we have seen that open addressing is not so bad if keys are short, in spite of the troublesome "deleted cells". Since the favorable algorithm greatly depends on the key length, an implementor of a hash-table system might consult Table 6, or similar tables.

Unfortunately, the analysis of deletion in open addressing is not complete; a mathematical proof that U and I are monotone-increasing has not been established. In fact, there remains another open problem: namely, the analysis of deletion in coalesced chaining when keys are non-relocatable. If relocation is not permitted, the same problem of "deleted cells" also arises in this algorithm. The analysis of such a case would be an interesting theme for future research.

Appendix A: Average Numbers of Probes for the Collision Flag Method

With no Deletion Furukawa [6] derived a formula of U in terms of M and n, though its dependence on $\alpha = n/M$ is rather unclear because it involves several iterations. Here we derive a formula of U in terms of α by considering a probability distribution of probe numbers.

Lemma A.1 Let p be a probability of an event P occurring at a random trial. Then the average number of trials, N_P, until P takes place for the first time, is $1/p$.

Proof If P takes place at a certain trial, then no more trial is necessary. Otherwise, N_P more trials are necessary on the average. Thus we have the following recursive formula on N_P:

$$N_P = p \cdot 1 + (1-p)(1+N_P).$$

Hence $N_P = 1/p$. □

Now let a_0 be the probability of a cell being occupied and not in collisions. Since totally nS probes are made, and the probability of a cell being probed at one probe is $1/M$, the probability of a cell being probed i times totally is $\binom{nS}{i}(1/M)^i(1(1/M)^{nS-i}$, or by approximating with the Poisson distribution: $(\alpha S)^i e^{-\alpha S}/i!$. Hence

$$a_0 = \alpha S\, e^{-\alpha S} = -(1-\alpha)\ln(1-\alpha) \quad \text{(cf. TABLE 2.1).} \tag{A·1}$$

Since $1-\alpha+a_0$ is the probability that a search terminates in the collision flag method, and by virtue of Lemma A.1,

$$U = \frac{1}{1-\alpha+a_0} = \frac{1}{1-\alpha}\,\frac{1}{1-\ln(1-\alpha)}. \tag{A·2}$$

With Deletion

Lemma A.2 Let p and q be, respectively, probabilities of vents P Q occurring at a random trial. Also let r be the probability of either event P or Q occurring at a random trial. Then the average number of trials, N_{PQ}, until both P and Q (not necessarily simultaneously) take place, is $1/p + 1/q - 1/r$.

Proof If both P and Q take place simultaneously at a certain trial, of which the probability is $p+q-r$, then no more trial is necessary. If P alone takes place at a certain trial, of which the probability is $p-(p+q-r) = r-q$, then $1/q$ more trials are necessary until Q takes place (cf. Lemma A.1). If Q alone takes place at a certain trial, of which the probability is $r-p$, then $1/p$ more trials are necessary until P takes place. Otherwise, if neither P nor Q takes place, N_{PQ} more trials are necessary. Hence,

$$N_{PQ} = (p+q-r)1 + (r-q)(1+1/q) + (r-p)(1+1/p) + (1-r)(1+N_{Q}),$$

resulting in $N_{PQ} = 1/p + 1/q - 1/r$. □

As for (4.7), the key to be searched is assured to be new by encountering a cell in state A_0 or B_0, of which the probability is $a_0(t)+b_0(t)$. Hence we have (4.7) by Lemma A.1. As for (4.8), we must know that the key to be inserted is new and must find an unoccupied cell. The latter happens with the probability $1-\alpha(t)$. Hence, by Lemma A.2, $I(t)$ is given by (4.8).

Appendix B: Notes on Table 1

Most of the formulae of U, I, and S are found in Knuth [10]. Here we just make some additional remarks. U of UF is derived in Appendix A. As for I of CC, the additional αe^{α} probes are for finding an empty cell by sequentially searching the table (cf. Knuth [10, p. 517]). For U of SO, note that one probe of SO consists of an access to the index hash table, and an access to the key table for a key comparison if the cell in the index hash table is not empty. Since SO is suited for long keys, the most essential portion of probe time is shared by accesses to the key table and key comparisons. Thus we count $U=1$ when the first probed cell in the index hash table is empty, because in the latter case, the key table is probed only once. Hence $U = (1-\alpha)1 + \alpha(1-\alpha)1 + \sum_{i=2}^{\infty} \alpha^i(1-\alpha)i = 1/(1-\alpha) - \alpha$. I of SO is $1/(1-\alpha)$ since, if a search terminates at the index hash table, one more access to the key table is inevitable. The same convention for counting U and I is made in SC also. As for σ_m, that of CC is obtained Sy putting $\alpha = 1$, and those of SO and SC are obtained by putting $\alpha = 1/\tau$, since $\alpha_m = N/M = 1/\tau$.

Appendix C: Proof of Lemma 4.2

The first part of Lemma 4.2 is obvious, since $\mathscr{L}$ is finite. As for the remainder, we need some preparations.

Definition A finite partially-ordered set $\mathscr{P}$ is said to be *upper semimodular* when a, b, and $c \in \mathscr{P}$ satisfy:

If $a \neq b$, $c \subset_p a$ and $c \subset_p b$, then there exists $d \in \mathscr{P}$ such that

$$a \subset d \quad \text{and} \quad b \subset d. \tag{C·1}$$

Lower semimodularity is defined dually.

Lemma C.1 $\mathscr{L}$ is upper and lower semimodular.

Proof Let $n_1 \neq n_2$, $n_3 \subset_p n_1$ and $n_3 \subset_p n_2$. Since $n_3 \subset_p n_1$, there is a time instant t_{p_1} such that $n_3(t) = n_1(t)$ for $t \neq t_{p_1}$, $n_3(t_{p_1}) = n_3(t_{p_1}-1)-1$, and $n_1(t_{p_1}) = n_1(t_{p_1}-1)+1$. Similarly, there exists a time instant t_{p_2} such that $n_3(t) = n_2(t)$ for $t \neq t_{p_2}$, $n_3(t_{p_2}) = n_3(t_{p_2}-1)-1$ and $n_2(t_{p_2}) = n_2(t_{p_2}-1)+1$. Since $n_1 \neq n_2$, $t_{p_1} \neq t_{p_2}$. Let us define n_4 as follows. $n_4(t) = n_3(t)$ $(= n_1(t) = n_2(t))$ for $t \neq t_{p_1}$ and $t \neq t_{p_2}$, $n_4(t_{p_1}) = n_1(t_{p_1})$, and $n_4(t_{p_2}) = n_2(t_{p_2})$. Obviously $n_1 \subset_p n_4$ and $n_2 \subset_p n_4$. Lower semimodularity is proved dually.

It is shown that, in any (upper or lower) semimodular finite partially-ordered set, all connected chains between the same end-point elements have the same length

(Birkhoff [3]).

Since $\sum_{t=0}^{T}\{n_2(t)-n_1(t)\}=2$ if $n_1 \subset_p n_2$, $k=1/2\sum_{t=0}^{T}\{n_2(t)-n_1(t)\}$ for any n_1 and n_2 such that $n_1 \subset n_2$. This proves the second part of Lemma 4.2. □

In fact, n_4 in the proof of Lemma C.1 is the least upper bound of n_1 and n_2 in the sense $n_1 \subset n_4$, $n_2 \subset n_4$, and there exists no n' such that $n_1 \subset n'$, $n_2 \subset n'$, and $n' \subset n_4$. Moreover, there exists a least upper bound between *any* two $n_1, n_2 \in \mathscr{L}$, since $n_1 \cup n_2$ defined as

$$(n_1 \cup n_2)(t)=\max\{n_1(t), n_2(t)\} \quad \text{for all } t, \qquad (\text{C}\cdot 2)$$

is obviously the least upper bound of n_1 and n_2. A greatest lower bound $n_1 \cap n_2$ is defined dually. Hence $\mathscr{L}$ forms a finite modular lattice, since it is both upper and lower semimodular (Birkhoff [3]).

Appendix D: Proof of (4.49)

It suffices to prove

$$\lim_{N\to\infty}(1-\alpha)F(-(N-1), \beta, \beta+2; \gamma)=e^{-\beta}.$$

First, from the definition (4.44), it is straightforward that if $N\gamma$ remains finite,

$$\lim_{N\to\infty} F(-(N-1), \beta, \beta+2; \gamma)=\lim_{N\to\infty} {}_1F_1(\beta, \beta+2; -N\gamma), \qquad (\text{D}\cdot 1)$$

where ${}_1F_1$ is the confluent hypergeometric function:

$${}_1F_1(p, q; z)=\sum_{i=0}^{\infty}\frac{\Gamma(p+i)\Gamma(q)}{\Gamma(p)\Gamma(q+i)}\frac{z^i}{i!}. \qquad (\text{D}\cdot 2)$$

Note that $N\gamma \to \beta$ if $N\to\infty$. Hence we have only to prove

$${}_1F_1(\beta, \beta+2; -\beta)=(1+\beta)e^{-\beta}. \qquad (\text{D}\cdot 3)$$

From (D.S), we have the following equations, where ${}_1F_1'$ stands for $(\partial/\partial v){}_1F_1$:

$${}_1F_1'(p, q; z)={}_1F_1(p, q; z)+\frac{p-q}{q}{}_1F_1(p, q+1; z), \qquad (\text{D}\cdot 4)$$

$$p\,{}_1F_1(p+1, q; z)=p\,{}_1F_1(p, q; z)+z\,{}_1F_1(p, q; z). \qquad (\text{D}\cdot 5)$$

By substituting $p=\beta$, $q=\beta+1$ and $v=-\beta$,

$${}_1F_1'(\beta, \beta+1; -\beta)={}_1F_1(\beta, \beta+1; -\beta)-\frac{1}{\beta+1}{}_1F_1(\beta, \beta+2; -\beta), \qquad (\text{D}\cdot 6)$$

$$\beta\,{}_1F_1(\beta+1, \beta+1; -\beta)=\beta\,{}_1F_1(\beta, \beta+1; -\beta)-\beta\,{}_1F_1(\beta, \beta+1; -\beta). \qquad (\text{D}\cdot 7)$$

Hence, by (D.6) and (D.7),

$$\frac{1}{\beta+1}{}_1F_1(\beta, \beta+2; -\beta)={}_1F_1(\beta+1, \beta+1; -\beta)=e^{-\beta}. \qquad (\text{D}\cdot 8)$$

References

1. Bays, C. A Note When to Chain Overflow Items within a Direct-Access Table, *Comm. ACM* **16**, 1 (Jan. 1973) 46–47.
2. Bellman, R. *Introduction to Matrix Analysis*, 2nd ed., McGraw-Hill, New York, N.Y., (1970).
3. Birkhoff, G. *Lattice Theory*, 3rd ed., American Mathematical Society, Providence, Rhode Island, (1967).
4. Bobrow, D. A Note on Hash Linking, *Comm. ACM* **18**, 7 (Jul. 1975) 413–415.
5. Feldman, J. A. and Rovner, P. D. An Algol-Based Associative Language, *Comm. ACM* **12**, 8 (Aug. 1969) 439–449.
6. Furukawa, K. Hash Addressing with Conflict Flag (in Japanese), *Johoshori* **13**, 8 (Aug. 1972) 533–539, Information Processing Society of Japan.
7. Goto, E. and Kanada, Y. Hashing Lemmas on Time Complexities with Applications to Formula Manipulation, in R. D. Jenks (ed.) *Processing of the 1976 ACM Symposium on Symbolic and Algebraic Computation*, Yorktown Heights, N.Y., (Aug. 1976) 154–158.
8. Gunji, T. *Analysis of Hash Addressing Methods*, Technical Report 76–03, Department of Information Science, University of Tokyo, Tokyo, (1976).
9. Knott, G. D. Hashing Functions, *The Computer Journal* **18**, 3 (1975) 265–278.
10. Knuth, D. E. *Sorting and Searching, The Art of Computer Programming*, vol. 3, Addison Wesley, Reading, Mass., (1973).
11. McCarthy, J. in D. Bobrow (ed.) *Symbol Manipulation Languages and Techniques*, North Holland, Amsterdam, (1971) 151–152.
12. Morris, R. Scatter Storage Techniques, *Comm. ACM* **11**, 1 (Jan. 1968) 38–44.
13. Teitelman, W. *Interlisp Reference Manual*, Xerox Palo Alto Research Center, Palo Alto, Calif., (1974).