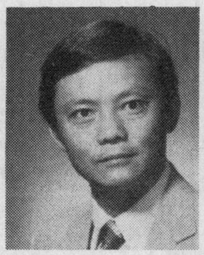
[13] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Syst. J.*, vol. 9, pp. 78-117, 1970.
[14] J. B. Morris, "Demand paging through the use of working sets on the MANIAC II," *Commun. Ass. Comput. Mach.*, vol. 15, pp. 867-872, Oct. 1972.
[15] J.-F. Pâris, personal communication, Feb. 1981.
[16] B. G. Prieve, "A page partition replacement algorithm," Ph.D. dissertation, Univ. California, Berkeley, 1974.
[17] J. Rodriguez-Rosell and J. P. Dupuy, "The design, implementation, and evaluation of a working set dispatcher," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 247-253, Apr. 1973.
[18] A. J. Smith, "A modified working set paging algorithm," *IEEE Trans. Comput.*, vol. C-25, pp. 907-914, Sept. 1976.
[19] A. J. Smith, "Two simple methods for the efficient analysis of memory address trace data," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 94-101, Jan. 1977.
[20] J. R. Spirn, "Program locality and dynamic memory management," Ph.D. dissertation, Princeton Univ., Mar. 1973.

Domenico Ferrari (M'67), for a photograph and biography, see p. 79 of the January 1983 issue of this TRANSACTIONS



Yiu-Yo Yih received the B.E. degree in computer engineering from the University of Michigan, Ann Arbor, and the M.S. degree in computer science from the University of California, Berkeley.

He joined Bell Laboratories, Holmdel, NJ, as a member of the technical staff in May 1980. His interests include operating systems and pattern recognition. He is currently working towards a Ph.D. degree in computer science at Rutgers University.

The Study of a New Perfect Hash Scheme

M. W. DU, MEMBER, IEEE, T. M. HSIEH, K. F. JEA, AND D. W. SHIEH

Abstract—A new approach is proposed for the design of perfect hash functions. The algorithms developed can be effectively applied to key sets of large size. The basic ideas employed in the construction are rehash and segmentation. Analytic results are given which are applicable when problem sizes are small. Extensive experiments have been performed to test the approach for problems of larger size.

Index Terms—Hashing, perfect hash functions, rehash, segmentation.

I. INTRODUCTION

HASHING has been considered as an effective means to organize and retrieve data in program design and has been widely used in database management, compiler construction, and many other applications. In order to use hashing techniques in a specific application, one has to first choose a suitable hash function, then select a method for collision resolution. Quite a number of ways have been proposed to design hash functions [2], [13], [16], [17], [21]. Two collision resolving methods, chaining and open addressing, have also been explored in many papers [12], [15] [17]-[20], [23].

If a hash function can be found which is one-to-one from the set of keys in the key space to the address space then that hash function will become much easier to use since the bothersome key collision problem can be avoided. There exist a number of methods for the design of a one-to-one hash function [1], [3], [8]-[10], [22] or called perfect hashing function in [22]. In suitable situations their methods may yield good hash functions as far as the memory space used or the execution time are concerned.

In this paper an entirely new approach is proposed for the design of perfect hash functions. An indicator table is used in the construction of a perfect hash function with table size in linear proportion to the number of keys. Compared to other methods proposed in the literature for designing perfect hash functions [1], [3], [8]-[10], [22], the construction procedures here have the advantages that they are easy to implement and can be effectively applied to key sets of large size.

The basic ideas employed in our construction are rehash and segmentation. In Sections II-IV we will show how random hash functions are organized by using a hash indicator table to construct a desired perfect hash function. Analytic results are given which are applicable when problems sizes are small. In Section V, two algorithms are designed for the construction of the hash indicator table. Extensive experiments have been performed to test the new approach for problems of large size. The results are presented in Section VI.

Formulas for calculating probabilities and expectations discussed in the context are listed in the Appendix.

II. RANDOM HASH FUNCTIONS

We shall consider random hash function first. Fig. 1 shows the basic model which will be considered throughout this paper. In the model, a set of n nonequal keys $K_1, K_2, \cdots, K_n$ in the key space is mapped into an address space with m entries

Manuscript received April 20, 1981; revised December 27, 1982. This work was supported in part by the National Science Council of R.O.C. under Grant NSC-68E-0201-04(05). Part of this paper was presented at COMPSAC'80, Chicago, IL, October 1980.

M. W. Du is with the Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

T. M. Hsieh is with the Institute of Electronic Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C. and the Department of Electronic Engineering, Chung Yuan University, Chung Li, Taiwan, R.O.C.

K. F. Jea is with the Department of Computer Science, University of Wisconsin, Madison, WI 53706.

D. W. Shieh is with the System Development Center, Institute for Information Industry, Taiwan, R.O.C.

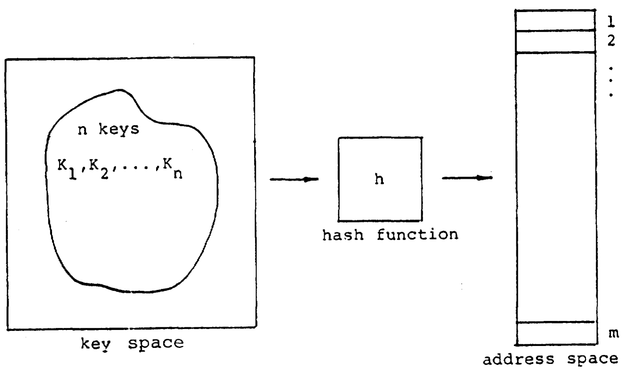


Fig. 1. Basic hash model.

by a hash function h. No interrelations will be assumed to exist among the n keys. The keys can be thought of as just arbitrary binary strings. Therefore, the hash function can be fully characterized by the listing of the values $h(K_i)$ for all $1 \leqslant i \leqslant n$, where $1 \leqslant h(K_i) \leqslant m$. h is called a random hash function whenever all the $h(K_i)$'s are selected randomly from $\{1, 2, \cdots, m\}$.

We say that a collision occurs if for some $i \neq j, h(K_i) = h(K_j)$. In such situation, we also say that K_i and K_j are collided keys under h.

Definition 1: h is a perfect hash function iff no collision occurs in h.

We define below a probability which will be used to measure the likelihood of constructing perfect hash functions.

Definition 2: Let F be a set of possible distinct functions obtainable from a construction procedure P, with the functions mapping from a set of keys into an address space. Assume that there are n_p perfect hash functions among the n_F functions in F. If a hash function h is selected randomly from F, the probability of h being perfect, denoted as pbp(h), is equal to n_p/n_F.

It should be noted that h is a formal name representing a function selected randomly from F in Definition 2. It can also be thought of as the formal name representing a function constructed by the procedure P. Clearly, pbp(h) depends entirely on how procedure P constructs perfect hash functions.

As an example, assume that h is a mapping function selected randomly from $F_{n \times m}$, the set of all functions that map n keys into an address space with m entries. The chances that h is a perfect hash function are ordinarily quite small. Actually, $\text{pbp}(h) = 0$ if $n > m$, and is equal to $m!/(m-n)! \times (1/m^n)$ if $n \leqslant m$. When $n = m = 10$, $\text{pbp}(h) = 0.0003629$. Even when m is much larger than n, the probability is still very small. When $n = 23$ and $m = 365$, we have the famous "birthday paradox" [7], [13]. The probability is only 0.4927, still less than half!

We say that a hash function h has k singletons if there are k entries in the address space with a single key hashed by h to each of them. In general, we can find the probability for h having exactly k singletons by the following theorem.

Theorem 1: Assume that h is a random hash function from n keys to an address space of size m. Let us denote the probability distribution of the number of singletons $k (0 \leqslant k \leqslant \min(m, n))$ as $P_k(n, m)$. Then

$$P_k(n,m) = \frac{e_k(n,m)}{m^n} \tag{1}$$

where

$$e_k(n,m) = n! \binom{m}{k} \sum_{r=0}^{n-k} (-1)^r \binom{m-k}{r} \frac{(m-r-k)^{n-r-k}}{(n-r-k)!}.$$

This theorem can be proved by first finding the number of possible h functions having at least j singletons, using the ordinary enumerator $(x_1 + x_2 + \cdots + x_m)^n$ of the number of ways in which n distinct objects can be distributed into m distinct cells, then applying the principle of inclusion and exclusion [6], [14] to get $p_k(n, m)$.

By using the distribution function in Theorem 1, closed form for the mean of $P_k(n, m)$ can be found [4], [11] as

$$E[k] = n \times \left(1 - \frac{1}{m}\right)^{n-1}. \tag{2}$$

Let us assume that $n = \alpha m$, the formula $n \times (1 - (1/m))^{n-1}$ approaches $n \times e^{-\alpha}$ when n becomes very large. Therefore, is h is selected randomly from $F_{n \times m}$, ordinarily it will be far from being a perfect hash function. To improve this situation, some information about the keys should be used to construct the hash function. In the next section we use rehash to construct a new hash function from a number of hash functions selected from $F_{n \times m}$. Information related to the keys and the hash functions selected is kept in a table called hash indicator table (HIT).

III. First Level Rehash

Fig. 2 shows the first-level rehash model where the hash function h is constructed from a number of hash functions, $h_1, h_2, \cdots, h_s$, selected randomly from $F_{n \times m}$. In the figure, the HIT has the same number of entries as that of the address space. In fact, entry d in HIT corresponds to entry d in the address space.

The contents in HIT can be defined by the following Pidgin Algol program.

Procedure 1 [Procedure to Construct First-Level HIT]:

```
begin
    KEYSET := $\{K_1, K_2, \cdots, K_n\}$;
    clear all entires of HIT;
    for $j$ := 1 step 1 until $s$ do
    begin
        for all elements in KEYSET do
            HIT($d$) := $j$ if HIT($d$) = 0 and $h_j(K_r) = d$
                            for one and only one $K_r$ in KEYSET;
        KEYSET := KEYSET − $\{K_r | K_r$ satisfies the above
                                          conditions$\}$,
    end
end
```

The first-level composite hash function h can be defined as follows:

$$h(K) = h_i(K) = d \quad \text{if HIT}(h_r(K)) \neq r \text{ for } r < i \text{ and HIT}(h_i(K)) = i, \qquad = \text{undefined} \quad \text{otherwise.} \tag{3}$$

It can be seen that if q hash functions are selected for composing h, a table HIT of width $\lceil \log_2 (q+1) \rceil$ bits is required.

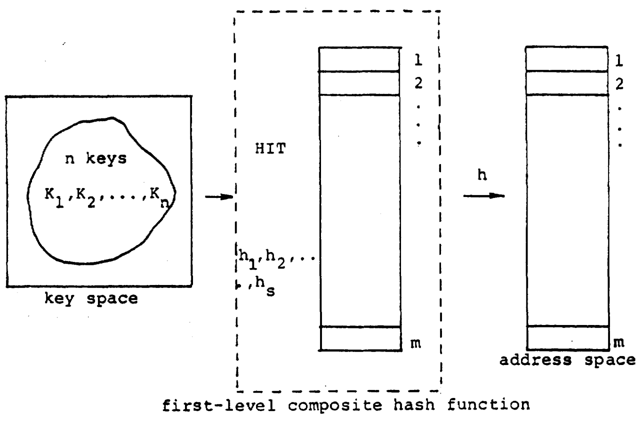


Fig. 2. First-level rehash model.

TABLE I
HASH FUNCTIONS h_1, h_2, h_3 FOR EXAMPLE 1

hash function \ Key	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8
h_1	4	⑤	4	6	2	6	2	⑨
h_2	5	8	9	②	4	4	9	2
h_3	⑦	6	5	5	2	④	9	8

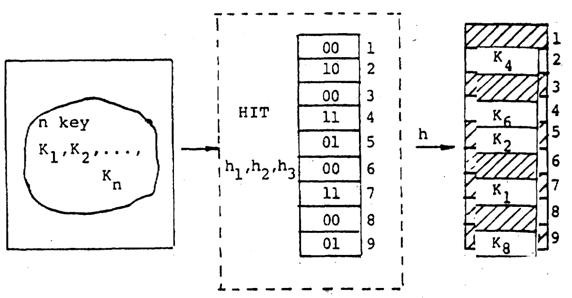


Fig. 3. HIT constructed by Procedure 1 of Example 1.

To find $h(K)$ for a key K in the key space, we apply the hash functions $h_1, h_2, \cdots, h_s$ on K in turn until $\mathrm{HIT}(h_i(K)) = i$. If the search fails, $h(K)$ is undefined.

Example 1: Assume that the key set is $\{K_1, K_2, \cdots, K_8\}$, and that the address space is from 1 to 9. The hash functions selected are h_1, h_2, h_3, as defined by Table I. The circles in each row h_j in the table indicate where the h_j contributes singletons.

The HIT constructed by Procedure 1 is given in Fig. 3. We can see that K_2 and K_8 find their right places by h_1. K_4 by h_2, K_1 and K_6 by h_3. Note that $h_2(K_1) = 5$ is a singleton in the second row of h_2, however, since 5 has been already occupied by K_2 through the mapping of h_1, K_1 does not find its right position by h_2. The mapping of the composed function h is undefined on the remaining keys K_3, K_5, and K_7.

Let us see how $h(K_6)$ can be calculated. When h_1, h_2, h_3 are applied on K_6 in turn, we have $\mathrm{HIT}(h_1(K_6)) = \mathrm{HIT}(6) = (00)_2 = 0 \neq 1$, $\mathrm{HIT}(h_2(K_6)) = \mathrm{HIT}(4) = (11)_2 = 3 \neq 2$, $\mathrm{HIT}(h_3(K_6)) = \mathrm{HIT}(4) = (11)_2 = 3 = 3$. Therefore, $h(K_6) = 4$.

Suppose that k keys have been hashed successfully into the address space after s hash functions of h_j's are applied in the way described above. Given any one key K of these k keys, we can apply the $h_1, h_2, \cdots, h_s$ successively, consulting the HIT table, to find $h(K)$. We define $NE1$ (n, m, s, k) as the expected number of times it is required to apply h_i's for calculating $h(K)$ with K among these k keys.

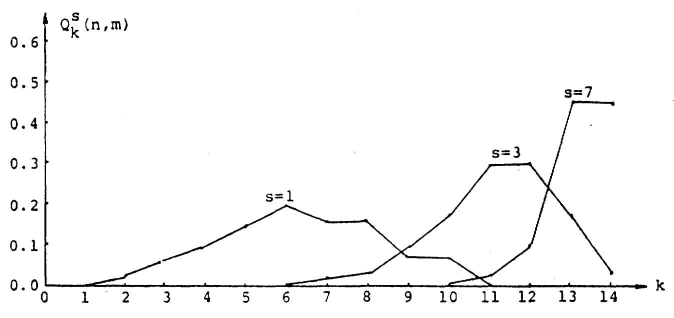


Fig. 4. Distribution diagram of $Q_k^s(n, m)$ with $n = 14$, $m = 17$, and $s = 1, 3, 7$.

Let $Q_k^s(n, m)$ be the probability of getting k singletons by applying $h_1, h_2, \cdots, h_s$ in the way described previously, where each h_i maps from the n given keys to the address space of size m. Formulas for calculating $NE1$ and Q_k^s are given in the Appendix.

Example 2: The $Q_k^s(n, m)$ values for $n = 14$, $m = 17$, $s = 1, 3, 7$, are calculated and plotted in Fig. 4.

It can be seen that $Q_k^1(n, m)$ is exactly the probability distribution of the number of singletons of a random hash function and is equal to the distribution function $P_k(n, m)$ in (1).

By comparing the values of $Q_k^1(14, 17)$ with those of $Q_k^7(14, 17)$, one can observe that he may find more singletons in a first-level composite hash function than in a random hash function. The expectation of k for $Q_k^1(14, 17)$ is 6.673471 while that for $Q_k^7(14, 17)$ is 13.280872.

When n singletons have been obtained by the above construction, we get a perfect hash function. The expectation of the number of times of applying h_i's in calculating $h(K)$ is $NE1(14, 17, s, 14)$. They are 1, 1.579916, 2.163818 with $s = 1, 3, 7$, respectively.

The probability of being perfect pbp for the first-level hash function composed from seven random hash function is $Q_{14}^7(14, 17)$. It is much larger than $Q_{14}^1(14, 17)$, the pbp of a random hash function.

When more and more random hash functions are selected to compose a first-level composite hash function, the probability of being perfect of the composite function will be certainly increased, but it will be increased very slowly. The reason is that the more singletons we already have, the fewer chances are there for the remaining keys to be hashed on unused entries in the address space. It has been found that dividing the address space into segments can be a more effective means to increase the pbp than using indefinitely many random hash functions, as will be described in the next section.

IV. SECOND LEVEL REHASH

Fig. 5 shows the second-level rehash scheme.

The address space is divided into q segments. Corresponding to segment A_i, which is of size m_i, we have a first-level composite hash function h^i which maps $\{K_1, K_2, \cdots, K_n\}$ into $\{1, \cdots, m_i\}$. HIT_i is the hash indicator table of h^i for indicating which h_j^i is applied to get the value of h^i. $\mathrm{HIT}_1, \cdots, \mathrm{HIT}_q$ compose the hash indicator table HIT of the second-level composite hash function H.

The second-level composite hash function H is defined as

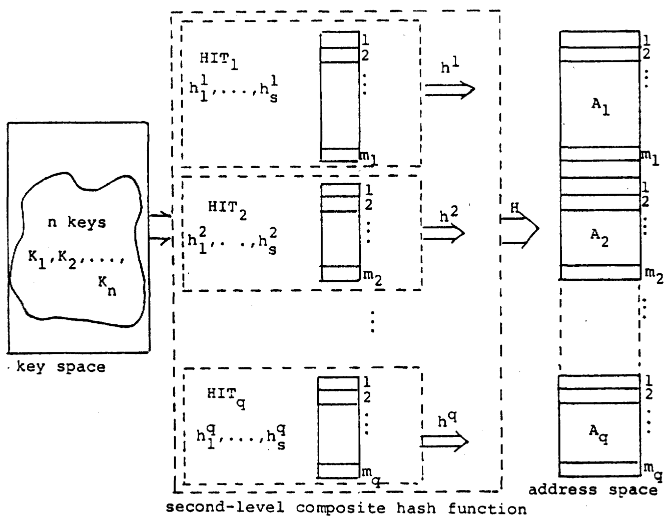


Fig. 5. Second-level rehash.

follows:

$$H(K) = h_j^i(K) + \sum_{t=0}^{i-1} m_t \quad \text{if } i, j \text{ can be found such that } \mathrm{HIT}_i(h_j^i(K)) = j, \text{ and } \mathrm{HIT}_i(h_r^i(K)) \neq r \text{ for } r < j, \text{ and } \mathrm{HIT}_a(h_b^a(K)) \neq b \text{ for } a < i,\ 1 \leqslant b \leqslant s, \text{ where it is assumed that } m_0 = 0,$$
$$= \text{undefined} \quad \text{otherwise.}$$

That is, if a key K can not find its right position in the first segment by h^1, it will try the second segment by h^2, and so forth.

The following program is an implementation of the hash function H.

Procedure 2 [Procedure for Computing H(K)]:

```
begin
   $t := 0$;
   for $i := 1$ step 1 until $q$ do
      begin
         for $j := 1$ step 1 until $s$ do
            begin
               $z := h_j^i(K)$;
               if $\mathrm{HIT}_i(z) = j$ then return $H(K) := t + z$
            end;
         $t := t + m_i$
      end;
   return $H(K)$ := undefined
end
```

We use $R_k^s(n, \overline{m}, q)$ to denote the probability distribution of getting k singletons by the second-level composite hash function H. The vector $\overline{m}$ is the partition on the address space. It can also be represented as $(m_1, m_2, \cdots, m_q)$. We shall also represent $(m_1, m_2, \cdots, m_r)$ as $\overline{rm}$. $\overline{qm} = \overline{m}$ by our convention.

Formulas for calculating $R_k^s(n, \overline{m}, q)$ are also given in the Appendix.

Example 3: To make things comparable to the results shown in Fig. 4, we let $n = 14$, $m = 17$, $\overline{m} = (13, 4)$, $s = 1, 3, 7$, and get three second-level composite hash functions which map 14 keys into two segments of size 13 and 4 each. The $R_k^s(n, \overline{m}, q)$ values are calculated and plotted on Fig. 6.

Let us examine the $R_k^7(14, (13, 4), 2)$ case. The probability

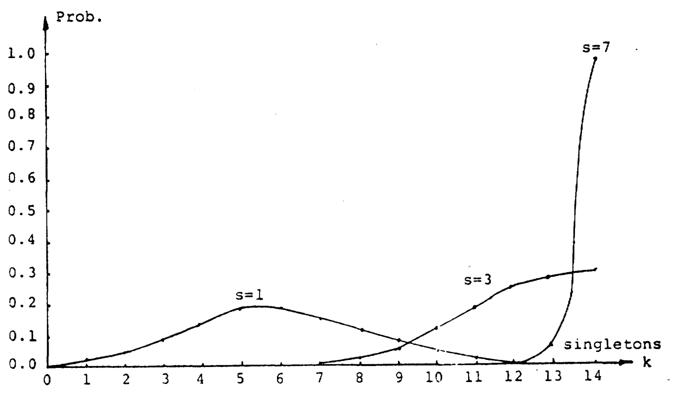


Fig. 6. Distribution diagram of $R_k^s(n, \overline{m}, q)$ with $n = 14$, $\overline{m} = (13, 4)$, $s = 1, 3, 7$.

for the second-level hash function H to be perfect is $R_{14}^7(14, (13, 4), 2)$, which is greater than 97.8 percent!

Like in the first-level analysis, suppose that k keys have been hashed successfully into the address space after the construction of the second-level hash function. Given any one key K of these k keys, we can apply the $h_1^1, h_2^1, \cdots, h_s^1, h_1^2, h_2^2, \cdots, h_s^2, \cdots$ successively, consulting the second-level HIT table, to find $H(K)$. Similarly, we define $NE2(n, \overline{m}, q, s, k)$ as the expected number of times it is required to apply h_j^i's for calculating $H(K)$ with K among these k keys. Formulas for calculating $NE2$ are given in the Appendix.

Example 3 (Continued): $NE2(n, \overline{m}, q, s, n)$ is the expected number of times to apply the h_j^i's functions to calculate $H(K)$ for any key K, if the second-level composite hash function is perfect. Using (A9) in the Appendix to calculate $NE2(14, (13, 4), 2, s, 14)$, we get 1.275607, 2.286545, 3.273526 for $s = 1, 3, 7$.

V. Construction of the Hash Indicator Table

According to the way the hash indicator table is defined in Section III, a multipass procedure can be designed for the construction of the HIT. Let thet set of keys be $\{K_1, K_2, \cdots, K_n\}$. Assume that the partition of the address space $(A_1, A_2, \cdots, A_q)$ of size $(m_1, m_2, \cdots, m_q)$ and the hash functions $h_1^1, \cdots, h_s^1, h_1^2, \cdots, h_s^2, \cdots, h_1^q, \cdots, h_s^q$ are all given. The following is a HIT construction procedure.

Procedure 3 (Static Procedure to Construct HIT's):

```
begin
   KEYSET := $\{K_1, K_2, \cdots, K_n\}$;
   clear all entries of $\mathrm{HIT}_i$;
   for $i := 1$ step 1 until $q$ do
      for $j := 1$ step 1 until $s$ do
      begin
         for all elements in KEYSET do
            $\mathrm{HIT}_i(d) := j$ if $\mathrm{HIT}_i(d) = 0$ and $h_j^i(K_r) = d$
                        for one and only one $K_r$
                        in KEYSET;
         KEYSET := KEYSET - $\{K_r | K_r$ satisfies the above
                                    conditions$\}$;
      end;
   if KEYSET = empty then HIT of a perfect hash function
   has been assigned else it fails to find a perfect hash
   function
end
```

TABLE II
HASH FUNCTIONS FOR EXAMPLE 4

Key / hash function	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9
h_1^1	4	⑤	4	6	2	6	2	⑨	①
h_2^1	5	8	9	②	4	4	1	2	8
h_3^1	⑦	6	1	5	2	④	9	8	5
h_1^2	1	3	2	4	2	3	2	5	3
h_2^2	2	4	5	2	5	5	③	1	3
h_3^2	5	2	①	1	⑤	4	1	3	2

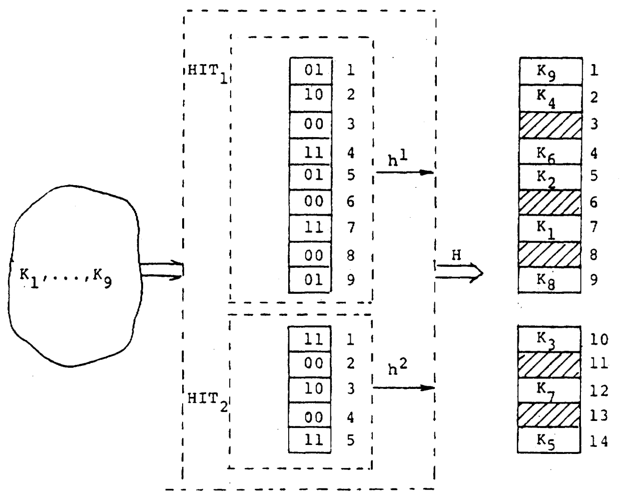


Fig. 7. HIT constructed by Procedure 3 of Example 4.

Example 4: Assume that nine keys $K_1, K_2, \cdots, K_9$ are to be hashed perfectly to an address space of size 14. The address space is partitioned into (A_1, A_2) of size (9, 5). The hash functions used are $\{h_1^1, h_2^1, h_3^1\}$, $\{h_1^2, h_2^2, h_3^2\}$ as defined by Table II. Again, as in Example 1, the circles in each row h_j^i in the table indicate where the h_j^i contributes singletons.

Fig. 7 shows the HIT constructed by Procedure 3.

Procedure 3 works on an address space with the segments preset to $(m_1, \cdots, m_q)$. In contrast to this, the size of the address space and the segments can be set in a dynamic manner, as the following procedure suggests.

Procedure 4 [Dynamic Procedure to Construct HIT (or H)]:

```
; assume that the size limitation of the address space is $m$.
begin
  KEYSET := $\{K_1, \cdots, K_n\}$;
  for $i$ := 1 step 1 until a large number $M$ do
    begin
      let $m_i$ be the size of KEYSET;
      $m := m - m_i$;
      if $m < 0$, then it fails to find a perfect hash
      function, return;
      reserve a table HIT$_i$ of length $m_i$ and clear it;
      generate $s$ hash functions $h_1^i, \cdots, h_s^i$ that map from
      KEYSET into $\{1, 2, \cdots, m_i\}$;
      for $j$ := 1 step 1 until $s$ do
        begin
          for all elements in KEYSET do
            HIT$_i(d) := j$ if HIT$_i(d) = 0$ and $h_j^i(K_r) = d$
                 for one and only one $K_r$ in
                 KEYSET;
          KEYSET := KEYSET - $\{K_r | K_r$ which satisfies
          the above conditions$\}$;
          if KEYSET = empty then HIT's of a perfect hash
          function has been assigned, return
        end
    end
end
```

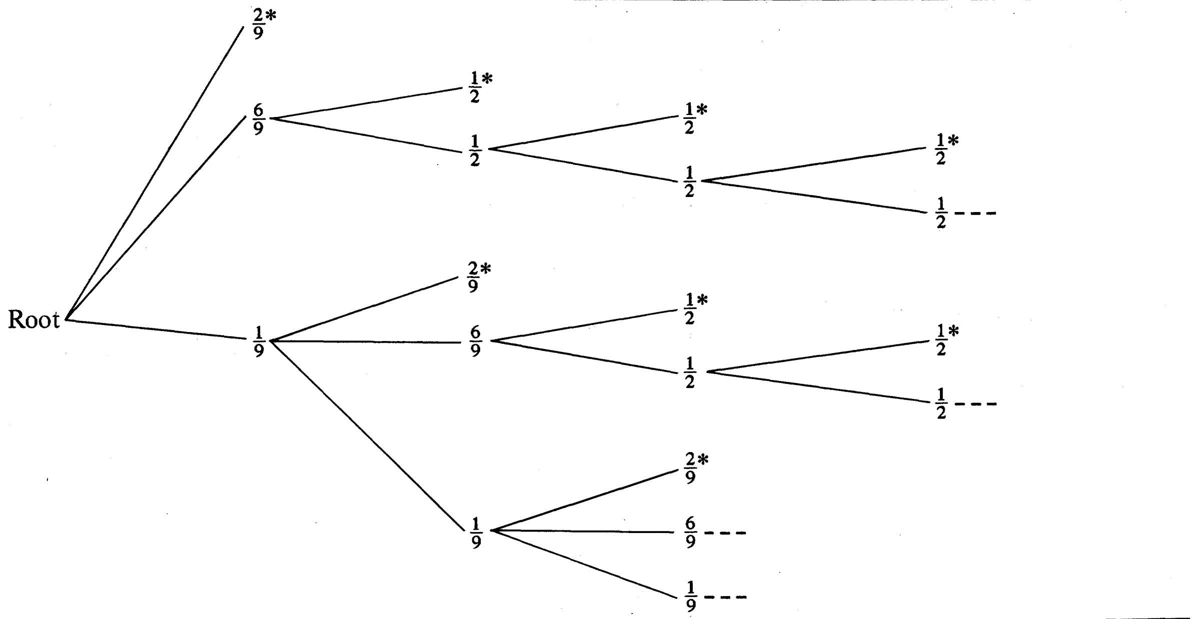
The expectation of the length of the HIT (or the length of the address space) and the expectation of the number of times it is required to apply h_j^i's to find $H(K)$ for K = any key K_j, with the HIT constructed by Procedure 4, are defined as $L(n, s)$ and $NED(n, s)$ respectively. They can be calculated by formulas in the Appendix.

Example 5: Let $s = 1$, $n = 3$. The probabilities of all possible cases constructed by Procedure 4 are depicted by the following probability tree:

$$\begin{tikzcd}[column sep=large, row sep=small]
 & Q_3^1(3,3)^* & & \\
 & Q_2^1(3,3) \arrow[r, no head] \arrow[dr, no head] & Q_1^1(1,1)^* & \\
 & & Q_0^1(1,1) \arrow[r, no head] \arrow[dr, no head] & Q_1^1(1,1)^* \\
 & & & Q_0^1(1,1)\text{---} \\
 & & Q_2^1(2,2)^*A & \\
\text{Root} \arrow[uuuuur, no head] \arrow[uuuur, no head] \arrow[r, no head] \arrow[dddr, no head] & Q_1^1(3,3) \arrow[ur, no head] \arrow[r, no head] \arrow[dr, no head] & Q_1^1(2,2)\text{---} & \\
 & & Q_0^1(2,2)\text{---} & \\
 & & Q_3^1(3,3)^* & \\
 & Q_0^1(3,3) \arrow[ur, no head] \arrow[r, no head] \arrow[dr, no head] \arrow[ddr, no head] & Q_2^1(3,3)\text{---} & \\
 & & Q_1^1(3,3)\text{---} & \\
 & & Q_0^1(3,3)\text{---} &
\end{tikzcd}$$

The path from the root to each terminal node (indicated by an asterisk) represents a possible configuration $(l_1, \cdots, l_q)$, with $l_1 + l_2 + \cdots + l_q = 3$. For example, the path from the root to *A in the figure represents the configuration (1, 2). The probability to get $l_1 = 1, l_2 = 2$ is $Q_1^1(3,3) \times Q_2^1(2,2)$—the product of the $Q_{l_i}^1$ on the path from the root to the terminal node.

There are $3^3 = 27$ distinct functions which map three keys into $\{1, 2, 3\}$. Six of these contain 3 singletons, 18 of these contain 1 singletons, and 3 of these contain 0 singletons. Therefore, $Q_3^1(3,3) = 6/27 = 2/9$, $Q_2^1(3,3) = 0$, $Q_1^1(3,3) = 18/27 = 2/3$, $Q_0^1(3,3) = 3/27 = 1/9$. Similarly, $Q_2^1(2,2) = 1/2$, $Q_1^1(2,2) = 0$, $Q_0^1(2,2) = 1/2$. The probability tree can be simplified to the following tree by noting that $Q_2^1(3,3) = 0$ and $Q_1^1(2,2) = 0$:



By (A6) we have

$$L(3,1) = \tfrac{2}{9} \times 3 + \tfrac{6}{9} \times \tfrac{1}{2} \times (3+2) + \tfrac{6}{9} \times \tfrac{1}{2} \times \tfrac{1}{2} \times (3+2+2) + \cdots + \tfrac{1}{9} \times \tfrac{2}{9} \times (3+3) + \tfrac{1}{9} \times \tfrac{6}{9} \times \tfrac{1}{2} \times (3+3+2) + \cdots + \tfrac{1}{9} \times \tfrac{1}{9} \times \tfrac{2}{9} \times (3+3+3) + \cdots = 6\tfrac{3}{8}.$$

Similarly, by (A8), we have

$$NED(3,1) = \tfrac{2}{9} \times \tfrac{3}{3} + \tfrac{6}{9} \times \tfrac{1}{2} \times [\tfrac{1}{3} \times (1 + 2 \times 2)] + \cdots + \tfrac{1}{9} \times \tfrac{2}{9} \times [1 + \tfrac{3}{3}] + \tfrac{1}{9} \times \tfrac{6}{9} \times \tfrac{1}{2} \times [1 + \tfrac{1}{3} \times (1 + 2 \times 2)] + \cdots + \tfrac{1}{9} \times \tfrac{1}{9} \times \tfrac{2}{9} \times [1 + 1 + \tfrac{3}{3}] + \cdots = 2\tfrac{1}{8}.$$

VI. Experimental Results

The formulas derived in the previous sections can be used to find important measures to characterize the construction of second-level perfect hash functions. However, they can be applied only when the problem sizes are small. This is due to the fact that the formulas get involved with the enumeration of all possible partitions of a number of integers. The complexity of such enumeration is exponential. Computing by recursive formulas eases the problem to a certain extent. When the integers (parameters) becomes larger, it will be still extremely time-consuming to do the calculations. For instance, it takes 36.237 s for a program implemented on a Cyber 170/720 system to calculate $NE2(14, (14, 3), 2, 7, 14) = 3.273526$. Therefore, extensive experiments have been designed to test the approach of constructing perfect hash functions introduced in this paper.

Quite a number of strategies can be adopted for the partitioning of a given address into segments in the experiments. Here we introduce a strategy called geometric partition strategy.

Assume that the size of the address space is m, and that it is to be partitioned into segment $(m_1, m_2, \cdots)$. In the geometric partition strategy all m_{i+1}/m_i are kept close to a constant, called the reduction ratio. If the reduction ratio is $\gamma, 0 \leq \gamma < 1$, the following formula can be applied.

$$\begin{aligned} m_1 &= \lfloor (1-\gamma)m + \tfrac{1}{2} \rfloor \\ m_i &= \lfloor \gamma \times m_{i-1} + \tfrac{1}{2} \rfloor \quad \text{if } \sum_{j=1}^{i} m_j \leq m \\ &= m - \sum_{j=1}^{i-1} m_j \quad \text{otherwise.} \end{aligned} \tag{4}$$

For example, if $m = 125$, $\gamma = 0.3$, m will be partitioned into 5 segments according to the geometric partition strategy as $(m_1, m_2, m_3, m_4, m_5) = (88, 26, 8, 2, 1)$.

In all the experiments, 36-bit random numbers are generated for the key values. The hash functions h_j^i's are also generated randomly.

Some more comments on the commonly used terminology, loading factor, are needed here. In an experiment, if the number of keys is n, the loading factor is given as τ, then the size of the address space is calculated as $\lfloor n/\tau + 1/2 \rfloor$.

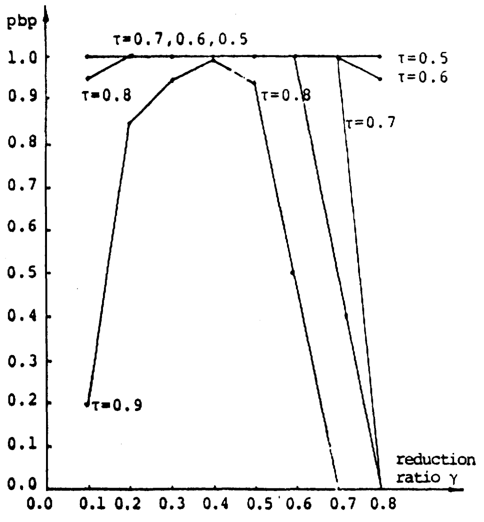


Fig. 8. Diagram of the pbp values of Experiment 1.

Experiment 1–Explanation:

1) Geometric partition strategy is tested.

2) Reduction ratio γ varies from 0.1 to 0.8 in steps of 0.1.

3) Loading factor τ varies from 0.5 to 0.9 in steps of 0.1.

4) Number of random hash functions for each h^i is 7.

5) 100 independent sets with 100 nonequal keys in each are tested for each combination of γ and τ.

6) For each combination of γ and τ, we calculate the probability of being perfect pbp as the number of succeeded trials divided by the number of total trials (which is 100).

The results of this experiment are plotted in Fig. 8.

From Fig. 8 we may observe immediately that we may have better chance to find a perfect hash function when the loading factor is smaller, which is quite natural. Also, all the curves corresponding to different τ values appear concave downward. It appears that there exists one and only one peak in each of these curves.

Experiment 2–Explanation:

1) Dynamic procedure (Procedure 4) for the construction of a perfect hash function is tested.

2) Number of keys n varies from 40 to 50, in steps of 1, then from 50 to 500 in steps of 50.

3) For each n, 100 independent sets with n nonequal keys are tested.

4) The number of hash functions for constructing each h^i is 1, 3, 7, i.e., $s = 1, 3, 7$.

The results of Experiment 2 are depicted in Figs. 9 and 10. Fig. 9 shows the average size of the address space assigned for each (n, s) in the experiment. Note that there are 100 trials for each combination of n and s. Fig. 10 shows the average number of times to apply h_j^i's for calculating $H(K)$ for each combination of n and s. For $n \leqslant 40$, these two figures, $L(n, s)$ and $NED(n, s)$, are calculated through applying (A7) and (A9), respectively.

Example 6: Suppose that Procedure 4 will be used to construct a perfect hash function to map 300 nonequal keys into an address space. From Figs. 9 and 10, if 7 hash functions are used in constructing each h^i, the length of the address space will be about equal to 348. The total size of the HIT table created will be about 348×3 bits. Which is about 131 bytes. The expected number of times it is required to apply h_j^i's for calculating $H(K)$ will be about equal to 3.48.

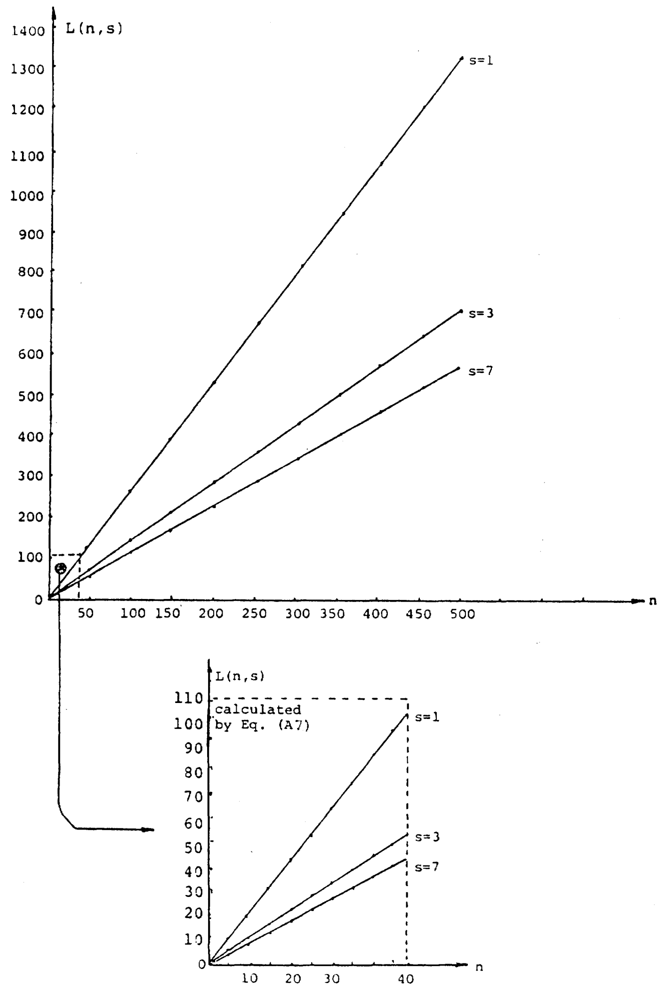


Fig. 9. The average size of the address space assigned for each pair (n, s) in Experiment 2.

VII. Discussion

This paper essentially proposed two procedures for constructing perfect hash functions. In the first procedure, the segmentation of the address space is preset. From Experiment 1 we can see that the way the address space is partitioned has great effect on the probability that a perfect hash function can be obtained. An interesting but unsolved problem is to show how to partition the address space so that we may maximize the probability of obtaining a perfect hash function by the construction procedure.

In the second procedure, the segmentation of the address space is obtained dynamically. If the address space is unlimited, the procedure can certainly construct a perfect hash function eventually. From Experiment 2, we observe that the ratio of the size of the address space constructed to the number of keys will approach a constant for each s value, as the number of keys goes large. For example, when $s = 7$, $L(n, s) \approx 7/6n$. This provides us a very simple guide rule in applying Procedure 4.

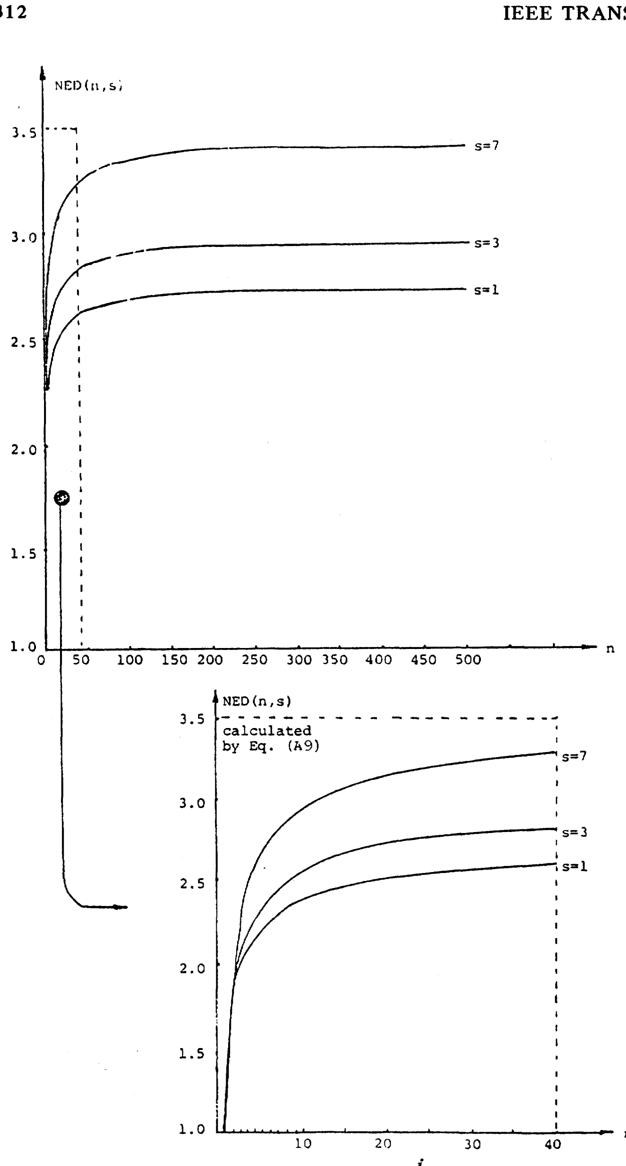


Fig. 10. Average number of times to apply h_j^i's for $H(K)$ for each pair (n, s) in Experiment 2.

APPENDIX

This Appendix lists formulas for calculating Q_k^s, R_k^s, $NE\,1$, $NE2$, $L(n, s)$, and $NED(n, s)$ described in the context. $NE1$, $NE2$, and NED can be referred as the expected number of internal probes for finding the hash value of a key.

Suppose that i singletons have been obtained by the successive application of r random hash functions in the way described before. We define $\Delta P_k(n, m, i)$ as the probability of getting k more singletons by applying the $(r+1)$th random hash function.

Theorem A1: The probability

$$\Delta P_k(n,m,i) = \left(\frac{i}{m}\right)^{n-i} \times \sum_{j=k}^{n-i} \frac{\binom{n-i}{j}}{i^j} e_k(j, m-i) \qquad \text{(A1)}$$

where the function e_k was defined in Theorem 1.

Theorem A2: The probability distribution $Q_k^s(n,m)$ can be calculated by

$$Q_k^{s+1}(n,m) = \sum_{j=0}^{k} Q_{k-j}^s(n,m)\,\Delta P_j(n,m,k-j) = \sum_{r=0}^{k} Q_r^s(n,m)\,\Delta P_{k-r}(n,m,r) \qquad \text{(A2)}$$

with $Q_k^1(n,m) = P_k(n,m)$.

Theorem A3:

$$NE1(n,m,s+1,k) = \sum_{i_{s+1}=0}^{k} \left\{ \left[\frac{k - i_{s+1}}{k} NE1(n,m,s,k-i_{s+1}) + \frac{i_{s+1}}{k}(s+1) \right] \frac{Q_{k-i_{s+1}}^s(n,m)\cdot \Delta P_{i_{s+1}}(n,m,k-i_{s+1})}{Q_k^{s+1}(n,m)} \right\}$$

with

$$NE1(n,m,l,k) = 1 \quad \text{for } 1 \leqslant k \leqslant n. \qquad \text{(A3)}$$

Assuming that all h_j^i's are independent random hash functions, we have the following.

Theorem A4:

$$R_k^s(n,\overline{qm},q) = \sum_{k_q=0}^{k} \sum^{k_1+k_2+\cdots+k_{q-1}=k-k_q} \cdot \prod_{i=1}^{q} Q_{k_i}^s\left(n - \sum_{t=0}^{i-1} k_t, m_i\right) = \sum_{k_q=0}^{k} R_{k-k_q}^s(n, \overline{(q-1)m}, q-1) \cdot Q_{k_q}^s(n-(k-k_q), m_q) \qquad \text{(A4)}$$

with $R_k^s(n, \overline{1m}, 1) = Q_k^s(n, m_1)$.

Theorem A5:

$$NE2(n,\overline{qm},q,s,k) = \left\{ \sum_{q=0}^{k} \left(\frac{k-l_q}{k}\right) NE2(n, \overline{(q-1)m}, q-1, s, k-l_q) + \frac{l_q}{k}\left[(q-1)s + NE1\overline{(n-(k-l_q)}, m_q, s, l_q)\right] \right\} \frac{R_{k-lq}^s(n,\overline{(q-1)m}, q-1)\cdot Q_{lq}^s(n-(k-lq), mq)}{R_k^s(n,\overline{m},q)} \qquad \text{(A5)}$$

with $NE2(n, \overline{1m}, 1, s, k) = NE1(n, m_1, s, k)$ for all $0 \leqslant k \leqslant n$.

Theorem A6:

$$L(n,s) = \sum_{q=1}^{\infty} \left\{ \sum^{l_1+\cdots+l_q=n \text{ and } l_q\neq 0} \prod_{i=1}^{q} Q_{l_i}^s(n_i,n_i) \cdot \left(\sum_{i=1}^{q} n_i\right) \right\} \qquad \text{(A6)}$$

when

$$n_i = n - \sum_{t=0}^{i-1} l_t, l_0 = 0.$$

Theorem A7:

$$L(n,s) = \frac{1}{1 - Q_0^s(n,n)} \left[n + \sum_{k=1}^{n-1} L(k,s)\, Q_{n-k}^s(n,n) \right] \qquad \text{(A7)}$$

with $L(0,s) = 0$, $L(1,s) = 1$.

Theorem A8:

$$NED(n,s) = \sum_{q=1}^{\infty} \left\{ \sum^{l_1+\cdots+l_q=n \text{ and } l_q\neq 0} \cdot \prod_{i-1}^{q} Q_{l_i}^s(n_i,n_i)\, NE2(n,\overline{m},q,s,n) \right\} \qquad \text{(A8)}$$

where

$$n_i = n - \sum_{t=0}^{i-1} l_t, l_0 = 0, \quad \text{and} \quad \overline{m} = (n_1, n_2, \cdots, n_q).$$

Theorem A9:

$$NED(n,s) = \frac{1}{1 - Q_0^s(n,n)} \cdot s \cdot Q_0^s(n,n) + NE1(n,n,s,n)\, Q_n^s(n,n) + \sum_{k=1}^{n-1}\left[\left(\frac{k}{n}\right) NE1(n,n,s,k) + \left(\frac{n-k}{n}\right)(s + NED(n-k,s))\right] \cdot Q_k^s(n,n)\Bigg\} \tag{A9}$$

References

[1] M. R. Anderson and M. G. Anderson, "Comments on perfect hashing functions: A single probe retrieving method for static sets," *Commun. Ass. Comput. Mach.*, vol. 22, p. 104, Feb. 1979.

[2] W. Buchholz, "File organization and addressing," *IBM Syst. J.*, vol. 2, pp. 86–111, June 1963.

[3] R. J. Cichelli, "Minimal perfect hash functions made simple," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 17–19, Jan. 1980.

[4] F. N. Davis and D. E. Barton, *Combinational Chances*. London: Griffin, 1962.

[5] M. W. Du, K. F. Jea, and D. W. Shieh, "The study of new perfect hash schemes," in *Proc. COMPSAC'80*, Chicago, IL, Oct. 1980, pp. 341–347.

[6] M. W. Du and H. C. Lin, "The design of a memory saving Chinese input/output system," in *Proc. NCS*, Taipei, Dec. 1979, pp. 8.1–8.10.

[7] W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 1. New York: Wiley, 1950.

[8] S. P. Ghosh, *Data Base Organization for Data Management*. New York: Academic, 1977.

[9] G. Jaeschke and G. Osterburg, "On Chichelli's minimal perfect hash functions method," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 728–729, Dec. 1980.

[10] G. Jaeschke, "Reciprocal hashing: A method for generating minimal perfect hashing functions," *Commun. Ass. Comput. Mach.*, vol. 24, pp. 829–833, Dec. 1981.

[11] K. F. Jea, "The study of the static properties of a new perfect hash scheme," Master thesis, Nat. Chiao Tung Univ., Hsinchu, Taiwan, R.O.C., May 1980.

[12] L. R. Johnson, "An indirect chaining method for addressing on secondary keys," *Commun. Ass. Comput. Mach.*, vol. 4, pp. 218–222, May 1981.

[13] D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*. Reading, MA: Addison-Wesley, 1973.

[14] C. L. Liu, *Introduction to Combinatorial Mathematic*. New York: McGraw-Hill, 1968.

[15] V. Y. Lum, P.S.T. Yuen, and M. Dodd, "Key-to-address transform techniques: A fundamental performance study on large existing formatted files," *Commun. Ass. Comput. Mach.*, vol. 14, pp. 228–239, Apr. 1971.

[16] W. D. Maurer and T. G. Lewis, "Hash table method," *Comput. Surveys*, vol. 7, pp. 5–19, Mar. 1975.

[17] R. Morris, "Scatter storage techniques," *Commun. Ass. Comput. Mach.*, vol. 11, pp. 38–44, Jan. 1968.

[18] C. A. Olson, "Random access file organization for indirectly accessed records," in *Proc. ACM 24th Nat. Conf.*, 1969, pp. 539–549.

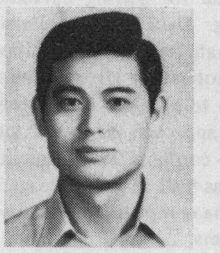
[19] W. W. Peterson, "Addressing for random-access storage," *IBM J. Res. Develop.*, vol. 1, pp. 130–146, Apr. 1957.

[20] G. Schay and W. G. Spruth, "Analysis of a file addressing method," *Commun. Ass. Comput. Mach.*, vol. 5, pp. 459–462, Aug. 1962.

[21] D. G. Severance, "Identifier search mechanisms: A survey and generalized model," *Comput. Surveys*, vol. 6, pp. 175–194, Sept. 1974.

[22] R. Sprugnoli, "Perfect hashing functions: A single probe retrieving method for static sets," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 841–850, Nov. 1977.

[23] M. Tainiter, "Addressing for random-access storage with multiple bucket capacities," *J. Ass. Comput. Mach.*, vol. 10, pp. 307–315, July 1963.



M. W. Du (S'70–M'72) received the B.S.E.E. degree from the National Taiwan University in 1966 and the Ph.D. degree from The Johns Hopkins University, Baltimore, MD, in 1972.

He is currently the Director of the Institute of Computer Engineering, National Chiao Tung University, Hsinchu, Taiwan. His research interests include fault diagnosis, automata theory, algorithm design and analysis, database design, and Chinese I/O design.

T. M. Hsieh was born in Taiwan, on November 15, 1947. He received the B.S. degree in electrical engineering from the Chung Yuan University, Chung-Li, Taiwan, in 1970 and the M.S. degree in electrical engineering from the Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, in 1974.

He is now working towards the Ph.D. degree in the Institute of Electronics, National Chiao Tung University. He previously worked for Telecommunication Laboratories, Chung-Li, Taiwan, before joining the Department of Electronic Engineering at the Chung Yuan University in 1975, where he is currently an Associate Professor. His research interests include database systems, fault-tolerant computing, microprocessor/microcomputer based systems.

K. F. Jea was born in Taiwan on May 13, 1956. He received the B.S. and M.S. degrees in computer science from the National Chiao Tung University, Hsinchu, Taiwan, in 1978 and 1980, respectively.

From 1977 to 1980, he served as a Research Assistant in the Department of Computer Science, National Chiao Tung University. He is currently working towards the Ph.D. degree in computer science at the University of Wisconsin, Madison. His research interests include database design, programming language and compiler design, algorithm design, and analysis.

D. W. Shieh was born in Taiwan, Republic of China, on January 1, 1952. He received the B.S. and M.S. degrees in computer science from the National Tung University, Taiwan, Republic of China, in 1976 and 1980, respectively.

Currently, he is a System Engineer at the System Development Center, Institute for Information Industry, Taiwan, Republic of China. His primary job is software system development.