procedure for large pending event lists. It is shown to be roughly comparable to the time-indexed procedure and to be operationally more useful since the need for external parameter definition is eliminated. The two-level procedure created by Franta and Maly is probably faster than either the time-indexed or the two-list procedure. However, the two-list procedure can be made transparent to the user by the inclusion of an arbitrary initial interval and an adaptive smoothing mechanism. The two-list procedure appears to be ideal for any simulation language or large scale program that does not utilize advanced synchronization procedures. The use of balanced trees to maintain File 1, such as Franta and Maly have done with the TL structure, might be a nice enhancement to the two-list procedure.

Received 3/79; revised 12/80; accepted 6/81

Reference

1. Electric Power Institute, Texas A&M University, Large-scale system effectiveness analysis. Final report on Contract F.D.-77-S-01-S104 to the U.S. Department of Energy.
2. Engelbrecht–Wiggans, R., and Maxwell, W. L. Analysis of the time indexed list procedure for synchronization of discrete event simulations. *Manage. Sci. 24*, 13 (Sept. 1978), 1417–1427.
3. Franta, W. R., and Maly, K. An efficient data structure for the simulation event set. *Comm. ACM 20*, 8 (Aug. 1977), 596–602.
4. Gonnet, G. H., Heaps applied to event driven mechanisms. *Comm. ACM 19*, 2 (July 1976), 417–418.
5. Markowitz, H. M., Karr, H. W., and Hadsner, B. *SIMSCRIPT: A Simulation Programming Language*. Prentice-Hall, Englewood Cliffs, N.J., 1963.
6. Pritsker, A. A. B. *The GASP-IV Simulation Language*. Wiley,New York, 1974.
7. Schriber, T. J. *Simulation Using GPSS*. Wiley, New York, 1974.
8. Vaucher, J. G., and Duvall, P. A comparison of simulation event list algorithms. *Comm. ACM 18*, 4 (April 1975), 223–230.
9. Wyman, P. F. Improved event-scanning mechanisms for discrete event simulations. *Comm. ACM 18*, 6 (June 1975), 350–353.

Programming Techniques and Data Structures

D. McIlroy*
Editor

Reciprocal Hashing: A Method for Generating Minimal Perfect Hashing Functions

G. Jaeschke
Heidelberg Scientific Center

A method is presented for building minimal perfect hash functions, i.e., functions which allow single probe retrieval from minimally sized tables of identifier sets. A proof of existence for minimal perfect hash functions of a special type (reciprocal type) is given. Two algorithms for determining hash functions of reciprocal type are presented and their practical limitations are discussed. Further, some application results are given and compared with those of earlier approaches for perfect hashing.

Key Words and Phrases: searching, hashing methods, quotient reduction, minimal perfect hashing
CR Categories: 3.15, 3.74, 4.34

I. Introduction

A refinement of hashing [5] which allows retrieval of an item (= key) in a static table with a single probe is called "perfect hashing" [2, 7]. Here the hashing function transforms the identifiers of a set of items into unique addresses. If W is the set of identifiers (which may be assumed to be positive integers), then

$$h: W \rightarrow I$$

is called a "perfect hashing function" (or "collision-free hashing function") if it is an injective mapping from W into some interval $I = \{0, 1, \ldots, M\}$ of positive integers.

A simple way of perfect hashing is to determine an integer M such that the residues w mod M for $w \in W$ are

* Former Editor of Programming Techniques and Data Structures of which Ellis Horowitz is the current editor.

Author's Present Address: G. Jaeschke, Heidelberg Scientific Center, Niederlassung Heidelberg, Germany.

all different from each other. However, it turns out that such integers M lead to hash tables with a small loading factor $|W|/M$. In order to avoid sparse hash tables, several methods for generating perfect hashing functions have been developed, mainly by Sprugnoli [7]. In these methods the occurring constants (for the hashing functions) are determined such that the corresponding hash table size is as small as possible but minimum table size cannot be guaranteed.

Consequently, we are interested in hashing functions that are perfect and minimum, i.e., that no collisions occur and that the size of the hash table is equal to the number of identifiers in W (loading factor = 1). We prove in Sec. II that to any given finite set W of positive integers there exist three constants C, D, E such that the function h with

$$h(w) = \lfloor C/(Dw + E) \rfloor \bmod n, \qquad n = |W|$$

is a minimal perfect hashing function. The function h is then a bijective mapping from W onto the set $\{0, 1, \ldots, n-1\}$. In order to determine the constants C, D, E one needs special algorithms which are presented in Secs. III and IV.

Algorithm C of Sec. III determines for a given set $W = \{w_1, \ldots, w_n\}$ of integers an integer C such that the residues $\lfloor C/w_1 \rfloor, \lfloor C/w_2 \rfloor, \ldots, \lfloor C/w_n \rfloor$ are different mod n. Such a number C exists if the $w_1, w_2, \ldots, w_n$ are pairwise relatively prime. In the general case (where the w_2 are not necessarily pairwise relatively prime), it may happen that no such C exists (actually out of 3000 observed cases with various distributions of the identifiers this happened in only two linearly independent cases (*cf.* [3].)

For such a case Algorithm DE in Sec. IV determines for a given set $W = \{w_1, \ldots, w_n\}$ of integers, two integers D, E such that

$$Dw_1 + E, Dw_2 + E, \ldots, Dw_n + E$$

are pairwise relatively prime.

A general procedure to finding the constants C, D, E is outlined at the end of Sec. IV. Further, some results on the efficiency of the suggested algorithms for reciprocal hashing are given in Secs. III and IV. In Sec. III we also discuss the case of larger identifier sets ($|W| > 20$).

In Sec. V we compare reciprocal hashing with Sprugnoli's methods [7] and Cichelli's approach [1] with respect to the obtained results and running times of the implemented algorithms.

II. Theorems of Reciprocal Hashing

The theorem which guarantees the existence of a minimal perfect hashing function h for a given set W can be formulated as follows.

THEOREM. *Given a finite set $W = \{w_1, w_2, \ldots, w_n\}$ of positive integers, there exist three integer constants C, D, E such that h defined by*

$$h(w) = \lfloor C/(Dw + E) \rfloor \bmod n$$

is a minimal perfect hashing function.

This theorem is a consequence of the following lemma. (Throughout the paper we assume the elements w_i of W to be in ascending order: $w_1 < \cdots < w_n$).

LEMMA. *For any set $W = \{w_1, \ldots, w_n\}$ of positive integers there exist two integer constants D, E such that*

$$Dw_1 + E, Dw_2 + E, \ldots, Dw_n + E$$

are pairwise relatively prime.

PROOF. Since the statement of the lemma is obvious for $n = 2$ we assume in the following $n \geq 3$. For $n = 3$ put $D = 1$. Otherwise put

$$D = \prod_{q \in Q_n} q$$

where Q_n is the set of all primes $\leq n/2$. Let further P_n be the set of all primes $> n/2$ which divide the product

$$\Delta = \prod_{i>j}^{n} (w_i - w_j).$$

From a theorem of Chebychev (*cf.* [6, Theorem 6–24]) it follows that P_n is not empty. This theorem states that for any natural number m there exists a prime between m and $2m$. Therefore, for any natural number $n \geq 3$ a prime p exists for which $n/2 < p < n$ holds, and all these primes divide Δ. For any $p \in P_n$ we have $n < 2p$ and therefore not all residues mod p occur more than once among the residues $-Dw_1, \ldots, -Dw_n \bmod p$. So we take r_p to be the smallest nonnegative residue mod p which does not occur twice or more among the residues $-Dw_i \bmod p$.

Now let E be a solution of the following system of simultaneous congruences

$$\begin{aligned} E &\equiv 1 \bmod D \\ E &\equiv r_p \bmod p \qquad \text{for all } p \in P_n. \end{aligned} \tag{1}$$

This system is solvable since its moduli are pairwise relatively prime (Chinese remainder theorem, *cf.* [6, theorem 3–13].) Then E has the desired property as we now demonstrate. Suppose $Dw_i + E$ and $Dw_j + E$ are not coprime for some pair i, j with $i \neq j$, and let g_{ij} be the greatest common divisor of $Dw_i + E$ and $Dw_j + E$. If now p is a prime which divides g_{ij}, then p must belong to the set P_n because from

$$Dw_i + E \equiv Dw_j + E \equiv 0 \bmod p \tag{2}$$

it follows that $D(w_i - w_j) \equiv 0 \bmod p$, and by $E \equiv 1 \bmod D$, $D \equiv 0 \bmod p$ cannot hold; hence $\Delta \equiv 0 \bmod p$ and $p > n/2$.

Since by Eq. (2) we have $Dw_i \equiv Dw_j \bmod p$ for $i \neq j$ we have by construction of r_p for $p \in P_n$ the incongruence $r_p \not\equiv -Dw_i \bmod p$ which implies by Eq. (1) $Dw_i + E \not\equiv 0 \bmod p$, a contradiction. Thus $g_{ij} = 1$ and $Dw_i + E$, $Dw_j - E$ are relatively prime which proves the lemma. □

PROOF OF THE THEOREM. Assume now that D, E have been determined according to the proof of the lemma. Since Eq. (1) has infinitely many solutions, E can be chosen such that

$$Dw_1 + E > n.$$

Then all $Dw_i + E$ are greater than n and consequently there exist numbers a_i such that

$$(i-1)(Dw_i + E) \le a_i < i(Dw_i + E),$$

$$a_i - a_j \equiv 0 \bmod n \qquad \text{for all } i, j = 1, \ldots, n.$$

Then by a generalization of the Chinese remainder theorem for nonpairwise relatively prime moduli [6, Theorem 3–12] there exists a number C such that C satisfies all congruences (Eq. 3):

$$C \equiv a_i \bmod n(Dw_i + E), \qquad i = 1, \ldots, n. \tag{3}$$

For this number C we have with integers λ_i

$$C = \lambda_i n(Dw_i + E) + a_i \qquad \text{for } i = 1, \ldots, n.$$

Therefore

$$C/(Dw_i + E) = \lambda_i n + a_i/(Dw_i + E)$$

$$\lfloor C/(Dw_i + E) \rfloor = \lambda_i n + i - 1 \equiv i - 1 \bmod n.$$

This means that the residues $\lfloor C/(Dw_i + E) \rfloor \bmod n$ are all different and $\lfloor C/(Dw + E) \rfloor \bmod n$ is a minimal perfect hashing function for W. This proves our theorem. □

III. Algorithm C

Let $W = \{w_1, w_2, \ldots, w_n\}$ consist of positive integers with $w_1 < \cdots < w_n$. Then Algorithm C tries to find an integer C such that the following condition is satisfied

$$\lfloor C/w_i \rfloor \not\equiv \lfloor C/w \rfloor \bmod n$$
$$\text{for all } i, j, \text{ with } 1 \le i < j \le n. \tag{4}$$

The algorithm starts with an arbitrary positive integer $C = C_0$ (see below). Then the residues of $\lfloor C/w_i \rfloor \bmod n$ are calculated. If they are all different from each other the algorithm terminates successfully. Otherwise the actual C is increased conveniently (see below) by a certain amount $\alpha(C, W)$, and the new C is examined in the same way. Algorithm C terminates unsuccessfully if C exceeds a prescribed limit L.

Now let us discuss which increment $\alpha(C, W)$ and which values C_0 and L are convenient for Algorithm C. In most cases it suffices to take $C_0 = 1$. However, for identifier sets W with a small difference $w_n - w_1$ possibly unnecessary C values are inspected if we put $C_0 = 1$, since $\lfloor C/w_1 \rfloor - \lfloor C/w_n \rfloor \ge n - 1$ is a necessary condition for Eq. (4). Therefore C must satisfy this inequality, which means that

$$C_0 = \left\lceil \frac{(n-2)w_1 w_n}{w_n - w_1} \right\rceil \tag{5}$$

is a reasonable start.

We make the following observations with respect to the increment $\alpha(C, W)$. We need examine only such integers C that are multiples of at least one element w_i of W. This is clear because a C value that is not a multiple of any element of W gives a remainder

$$r_i \equiv C \bmod w_i \qquad (0 < r_i < w_i)$$

and by taking the minimum of these r_i, say r_{i_0}, the quotients $\lfloor C/w_i \rfloor$ equal the quotients $\lfloor C'/w_i \rfloor$ where $C' = C - r_{i_0}$; C' is a multiple of w_{i_0}. This means that $\alpha(C, W)$ should be one of the numbers

$$w_1 - r_1, w_2 - r_2, \ldots, w_n - r_n$$
$$\text{with } r_i = C - \lfloor C/w_i \rfloor \cdot w_i.$$

Our aim is to examine as few as possible C values and to find the smallest one being greater than C_0. If, for example,

$$\lfloor C/w_i \rfloor \equiv \lfloor C/w_j \rfloor \bmod n$$

for some i, j, then no integer C' exists with

$$C < C' < C + \min\{w_i - r_i, w_j - r_j\}$$

such that $\lfloor C'/w_i \rfloor \not\equiv \lfloor C'/w_j \rfloor \bmod n$ as is easy to be seen. If we therefore define

$$\delta_{ij} = \min\{w_i - r_i, w_j - r_j\}$$

$$K(C, W) = \{(i, j) \mid 1 \le i < j \le n \wedge \lfloor C/w_i \rfloor \equiv \lfloor C/w_j \rfloor \bmod n\}$$

$$\alpha'(C, W) = \max_{(i,j) \in K(C,W)} \delta_{ij}$$

then $\alpha'(C, W)$ would be an appropriate increment of C. However, experiments show that it is more efficient to work with

$$\alpha'(C, W) = \delta_{i_0 j_0}$$

as increment of C instead of $\alpha'(C, W)$ where j_0 is the greatest index j for which there exists an i with $(i, j) \in K(C, W)$, and i_0 is the greatest index i with $(i, j_0) \in K(C, W)$. The reason is that i_0, j_0 can be determined in less time than the maximum of possibly many δ_{ij}.

It remains to discuss why a limit L has to be used and how L shall be chosen appropriately. A natural limit for the C values to be inspected is

$$L = n \cdot \mathrm{scm}(w_1, \ldots, w_n) \tag{6}$$

where scm means "smallest common multiple" because if a $C > L$ of the desired kind exists then $C - L$ is also a C value which satisfies Eq. (4) (since $\lfloor (C - L)/w_i \rfloor \equiv \lfloor C/w_i \rfloor \bmod n$). That means if no $C < L$ satisfies Eq. (4), then no C exists at all such that Eq. (4) holds. The number L determined by Eq. (6) is generally very large and therefore not adequate for the termination of Algorithm C. Therefore the user of Algorithm C has to prescribe a limit L for the C values to be examined such that the algorithm terminates in economical time, or such that the C values to be examined will not be too large.

Given $W = \{w_1, \ldots, w_n\}$ with $w_1 < \cdots < w_n$ and integers C_0, L, we perform the following steps.

Step 1. Put $C = C_0$.

Step 2. Determine the residues

$$\lfloor C/w_1 \rfloor \bmod n, \ldots, \lfloor C/w_n \rfloor \bmod n.$$

If they are all different from each other the algorithm terminates successfully.

Step 3. If $C > L$, then the algorithm terminates unsuccessfully.

Step 4. Determine

$$j_0 = \max\{j \mid \exists\, i(\lfloor C/w_i \rfloor \equiv \lfloor C/w_j \rfloor \bmod n)\}$$

$$i_0 = \max\{i \mid \lfloor C/w_i \rfloor \equiv \lfloor C/w_{j_0} \rfloor \bmod n\}$$

$$\alpha(C, W) = \min\{w_{i_0} - C \bmod w_{i_0}, w_{j_0} - C \bmod w_{j_0}\}$$

$C = C + \alpha(C, W)$ and return to step 2.

In the following we discuss the efficiency of the algorithm to compute C. Let $W = \{w_1, \ldots, w_n\}$ with $1 \le w_1 < \cdots < w_n \le b$ where b is a constant, and define $\mathscr{C}_W$ to be the class of all C for which Eq. (4) holds. Since $\mathscr{C}_W$ is possibly empty, $n \cdot \mathrm{scm}\,(w_1, \ldots, w_n)$ is an upper bound for the C values to be tested. The maximal number of iterations (C inspections) to find a value $C \in \mathscr{C}_W$ is therefore bounded by b^{n+1}. This bound does not represent the real complexity of the problem since experiments yield much better results. If the integers w_i are chosen at random from the interval (1, 1,000,000), an average number of about 1.82^n is obtained for $n \le 15$. In detail we have the following results. Let e_n denote the number of experiments with sets $\{w_1, \ldots, w_n\}$, $1 \le w_i \le 1{,}000{,}000$, and let μ_n be the smallest, ν_n the greatest, and δ_n the average number of iterations which were actually necessary. Then we obtain Table I.A for random w_i.

The proof of existence in Sec. II shows that there is not only one congruence system of the form (Eq. 3) which yields C values of the desired kind. Thus one can prove by statistical assumptions for the distribution of the C values that satisfy Eq. (4) for a given set W that the average number of iterations to obtain the smallest C is not greater than $n^n/n!$. It can be seen easily that this bound is better than b^{n+1} but worse than the experimental result 1.82^n.

Table I.B is obtained by choosing logarithmically distributed values w_i. An average of 1.49^n is therefore obtained for the number of iterations if the identifiers w_i are logarithmically distributed.

In the following we discuss the behavior of Algorithm C when the number of identifiers becomes larger. If the number of identifiers w_i is greater than 20, Algorithm C is not efficient in the case of not logarithmically distributed values w_i. For $20 < n \le 40$ one can apply "reciprocal hashing with a cut." This means that we divide W into two sets U_1 and U_2 with m_1 and m_2 elements, respectively, such that all elements of U_1 are smaller than those of U_2. We determine C_1 and C_2 for U_1 and U_2 and put

$$\begin{aligned} h(x) &= \lfloor C_1/x \rfloor \bmod m_1 && \text{for } x \le w_{m_1} \\ h(x) &= m_1 + \lfloor C_2/x \rfloor \bmod m_2 && \text{for } x > w_{m_1}. \end{aligned} \tag{7}$$

Table I. Random and Logarithmically Distributed Values of w_i.

n	e_n	μ_n	ν_n	δ_n
A. Random Values of w_i				
5	500	2	389	21
10	100	15	2628	408
15	100	39	20249	7710
B. Logarithmically Distributed Values of w_i				
5	100	2	15	6
10	100	6	1165	55
15	100	8	1027	380

By this approach we could easily solve the case of the 31 most frequently used English words (see [3].)

If n becomes greater than 40 we apply "reciprocal hashing with grouping" as suggested by Sprugnoli. A grouping function g maps the set W of identifiers $w_1, w_2, \ldots, w_n$ onto a set $\{1, 2, \ldots, m\}$ with $2 \le m < n$. If $g(w_i) = k$, then we say that w_i belongs to the kth group G_k with respect to g and W. Let g_k be the number of elements of W belonging to the kth group G_k. For each group G_k we determine a number C_k with Algorithm C and put

$$h_k(w) = \lfloor C_k/w \rfloor \bmod g_k \qquad \text{for } w \in G_k.$$

If

$$h(w) = h_k(w) + \sum_{j=1}^{k-1} g_j \qquad \text{with } k = g(w)$$

then h is obviously a minimal perfect hashing function for the set W.

The only problem that arises in connection with grouping is to limit the group cardinalities by a number ≤ 15 in order not to be inefficient. Therefore we determine the smallest integer m_0 such that the number of elements w_i which yield the same residue mod m_0 is always ≤ 15, and we then take $g(w) = 1 + w \bmod m_0$ as a grouping function. By reciprocal hashing with grouping we solved (see [3]) several problems up to 1003 identifiers (resulting of coded text) where maximally 163 groups have been used and less than 5000 C iterations were necessary.

For larger identifier sets ($n \ge 2000$) reciprocal hashing with grouping will hardly be efficient with respect to time and storage requirements.

A final remark concerning the constructive proof of existence in Sec. II is made here. Assume the identifiers w_i to be pairwise coprime. Then a C value of the desired kind can be obtained by determining convenient numbers a_i and solving the system of congruences (Eq. 3) by an adequate algorithm. We have programmed such an algorithm which is called the "congruence method" of reciprocal hashing. Its only disadvantage is that it generally returns very large values for C (see [3]). Thus the congruence method seems to be of no importance for practical purposes.

IV. Algorithm DE

Given a set W of positive integers $w_1, w_2, \ldots, w_n$ Algorithm DE determines integers D and E such that the values

$$Dw_1 + E, Dw_2 + E, \ldots, Dw_n + E$$

are pairwise relatively prime.

Let $\mathscr{P}$ be the set of all primes.

Step 1. Determine the set

$$I_n = \{p \mid p \in \mathscr{P} \wedge p \leq n/2\}.$$

Step 2. Determine the set

$$P_2 = \{p \mid p \in I_n \wedge \beta(p) \leq 1\}$$

where

$$\beta(p) = \min_{0 \leq \nu < p} |\{i \mid w_i \equiv \nu \bmod p\}|.$$

Then P_2 contains all primes $\leq n/2$ such that at least one residue ν mod p occurs at most once among the residues w_i mod p.

Step 3. Determine $P_1 = I_n - P_2$ and put

$$D = 1 \text{ if } P_1 = \emptyset; D = \prod_{p \in P_1} p \text{ otherwise.}$$

Step 4. Determine for each $p \in P_2$ the set

$$M(p) = \{-D\nu \bmod p \mid 0 \leq \nu < p \wedge |\{i \mid w_i \equiv \nu \bmod p\}| \leq 1\}.$$

$M(p)$ is not empty for all these p.

Step 5. Examine all integers E for which

$$(E \bmod p) \in M(p) \qquad \text{for } p \in P_2$$

$$(E \bmod p) \neq 0 \qquad \text{for } p \in P_1$$

whether the $Dw_i + E$ are pairwise relatively prime or not. The algorithm terminates successfully before E has reached the upper bound $b = \prod_{p \in T} p$ where $T = I_n \cup \{p \mid \Delta \equiv 0 \bmod p\}$ and $\Delta = \prod_{i>j}^{n} (w_i - w_j)$.

By counting the number of E tests in Algorithm DE when it is applied 100 times for each number n with $5 \leq n \leq 20$ and randomly chosen identifiers w_i smaller than 1,000,000, we obtained Table II.

Table II.

n	Minimum	Average	Maximum
5	1	8	29
6	1	7	19
7	1	10	83
8	1	16	113
9	1	26	245
10	1	17	67
11	1	17	91
12	1	39	157
13	1	41	277
14	1	29	143
15	1	36	163
16	1	39	221
17	1	62	181
18	11	88	241
19	13	113	341
20	11	154	601

V. Comparison with Earlier Approaches of Perfect Hashing

In [7] Sprugnoli offers two methods for generating perfect hash functions: (1) the quotient reduction method, and (2) the remainder reduction method. Neither method guarantees finding minimal perfect hash functions. The quotient reduction method yields no compact tables even for smaller identifier sets W ($n = |W| \leq 12$); however, the load factor is generally > 0.5 and can be improved by introducing a "cut" up to > 0.75. In the case of larger sets W (experiments have been made for $n = 20$) it turns out that load factors $< 1/300$ are not unusual, especially if the numerical keys are not uniformly distributed (*cf.* Tables I.A and B.)

By the remainder reduction method a minimal perfect hash function was found in all analyzed cases (where $n \leq 12$), but there is no proof that this is always the case. For larger $n (n > 12)$ no minimal perfect hash functions could be obtained in acceptable running times.

It is obvious that reciprocal hashing is always optimal with respect to the load factor of the hash tables. With respect to running times it is usually only slightly better than Sprugnoli's methods, but in the case of nonuniformly distributed identifier sets it is essentially better, which means that the ratio of running times lies between 1/10 and 1/1000.

The approach proposed by Cichelli [1] for alphanumerical keys yields simply computable hash codes, but in some cases it requires much time to compute the mapping g from the alphabet into the set of natural numbers such that the codes

h(key) = length (key) + g (first letter of key) + g (last letter of key)

for the keys of a given set of keys yield successive numbers. In fact, the claimed mapping g does not exist in infinitely many nontrivial cases, as is pointed out in [4].

Examples for reciprocal hashing can be found in [3].

Acknowledgment. I wish to thank H.J. Schek for valuable comments and practical advice in editing this paper.

Received 3/80; revised 4/81; accepted 5/81

References

1. Cichelli, R.J. Minimal perfect hash functions made simple. *Comm. ACM, 23*, 1 (Jan. 1980), 17–19.
2. Greniewski, M., and Turski, W. The external language KLIPA for the URAL-2 digital computer. *Comm. ACM, 6*, 6 (June 1963), 322–324.
3. Jaeschke, G. Minimal perfect hashing. Tech. Rept. TR 80.02.003, IBM Heidelberg Scientific Center, Niederlassung Heidelberg, Germany.
4. Jaeschke, G., Osterburg, G. On Cichelli's minimal perfect hash functions method. *Comm. ACM, 23*, 12 (Dec. 1980), 728–729.
5. Knott, G.D. Hashing functions. *Computer J., 18* (Aug. 1975), 265–278.
6. Leveque, W. J. *Topics in Number Theory*. Addison-Wesley, Reading, Mass., 1956.
7. Sprugnoli, R. Perfect hashing functions: a single probe retrieving method for static sets. *Comm. ACM, 20*, 11 (Nov. 1977), 841–850.