Programming Techniques

R. L. Rivest, S.L. Graham
Editors

Perfect Hashing Functions: A Single Probe Retrieving Method for Static Sets

Renzo Sprugnoli
Istituto di Elaborazione della Informazione
del Consiglio Nazionale delle Ricerche

A refinement of hashing which allows retrieval of an item in a static table with a single probe is considered. Given a set I of identifiers, two methods are presented for building, in a mechanical way, perfect hashing functions, i.e. functions transforming the elements of I into unique addresses. The first method, the "quotient reduction" method, is shown to be complete in the sense that for every set I the smallest table in which the elements of I can be stored and from which they can be retrieved by using a perfect hashing function constructed by this method can be found. However, for nonuniformly distributed sets, this method can give rather sparse tables. The second method, the "remainder reduction" method, is not complete in the above sense, but it seems to give minimal (or almost minimal) tables for every kind of set. The two techniques are applicable directly to small sets. Some methods to extend these results to larger sets are also presented. A rough comparison with ordinary hashing is given which shows that this method can be used conveniently in several practical applications.

Key Words and Phrases: hashing, hashing methods, hash coding, direct addressing, identifier-to-address transformations, perfect hashing functions, perfect hash coding, reduction, scatter storage, searching

CR Categories: 3.7, 3.74, 4.34

Author's address: Istituto di Elaborazione della Informazione del Consiglio Nazionale delle Ricerche, Via S. Maria 46, I56100 Pisa, Italy.

Introduction

This paper is devoted to a refinement of the well-known technique of *hashing*, which allows retrieval of an item in a static table with a single probe. Let I be a given set of identifiers; if we wish to know whether an identifier w belongs to I, it is common practice to use an identifier-to-address function h to store the elements of I in a *hash table* and then to use the same function h to retrieve w in the table. In general, several probes are necessary to locate w in the table or to be convinced that w is not there (i.e. $w \notin I$).

However, if h transforms the identifiers in I into unique addresses, a single probe is sufficient. Such a transformation will be called a *perfect hashing function*. In [4] Knuth defines an "amusing puzzle," to find a perfect hashing function for a given set I. He points out that a slight modification of I may change h completely so that the tedious calculations to find h become useless and everything has to be started over from scratch.

Here we show how the problem of finding a perfect hashing function for a given set I can be mechanized. We claim that the use of perfect hashing functions can be useful in many applications.

As a very simple example, let us consider an Assembler/370 program dealing with a lot of dates, in which the month is abbreviated to the first three characters. A fast routine is to be designed to recognize the month, to check if it is correctly spelled, and to point to some related information. Since the set is so small, a linear search is usually preferred. The month abbreviations are stored in the table TABLE in the last three bytes of a full word, with the first byte set to zero; the month to be searched for is right adjusted in the full word MONTH, located at the end of TABLE, i.e. MONTH equals TABLE + 48.

Figure 1 gives a possible piece of coding, where, on the right, we have written the units of time (see Knuth [3]) relative to each instruction. C is the number of times the loop is executed. Assuming C = 6.5, we have a total score of 34. We remark that it is useless to unroll the loop because we wish to know explicitly the index in the table for further use.

Now, according to the theory developed in Section 3, we can set up a perfect hash table with an appropriate perfect hashing function (Table I) and use the piece of coding given in Figure 2.

The perfect hashing function is performed by the four instructions in the shaded area. Summing up the units of time we get a total score of 26, an improvement of about 24 percent over the linear search.

As an example, let us suppose that MONTH contains 'FEB'. In register 3 we load the characters 'EB' corresponding to the decimal number 50626; adding 9 and multiplying by $2^8 = 256$ we get 12962560; dividing by 23, in register 2 we find the remainder 13, so that the last shift gives 6 in register 3. In fact, 'FEB' is the month at location 6 in TABLE.

Fig. 1.

```
          L       1,MONTH          2
          LA      2,LOOP           1
          LA      3,4              1
          LNR     3,3              1
LOOP      LA      3,4(3)           C
          C       1,TABLE(3)       2C
          BNER    2                C
          C       3,=F'48'         2
          BE      FAILURE          1
```

Fig. 2.

```
  L       1,MONTH          2
  LA      3,1              1
  N       3,=X'FFFF'       2
  SR      2,2              1
~~LA      3,9(3)~~         1
~~SLA     3,8~~            2
~~D       2,=F'23'~~      10
~~SRDA    2,33~~           2
  SLA     3,2              2
  C       1,TABLE(3)       2
  BNE     FAILURE          1
```

Table I.

TABLE	MAR
+1	OCT
+2	JUN
+3	SEP
+4	AUG
+5	JAN
+6	FEB
+7	APR
+8	DEC
+9	NOV
+10	JUL
+11	MAY

We have developed direct methods to derive perfect hashing functions for small sets, say with at most 10–12 elements, and we have extended our results to larger sets.

In Section 1 we give some general definitions and a more formal approach to the problem; in Section 2 we present the "quotient reduction" method for constructing perfect hashing functions. Section 3 is devoted to the "remainder reduction" method, and, finally, in Section 4, we extend our results to larger sets.

1. General Considerations

Essentially we are interested in functions defined on sets of identifiers. However, because of the representation of characters in the computer memory, we may consider functions on integers without any loss of generality.

Let $\mathbf{N}$ denote the set $\{0, 1, 2, 3, \ldots\}$ of natural numbers and $\mathbf{Z}$ the set of integers; for $n, n', m \in \mathbf{Z}$, "n mod m" is the remainder (≥ 0) of the integer division of n by m, and $n \equiv n'$ (modulo m) is the congruence relation defined by: $n \bmod m = n' \bmod m$. Furthermore, if $n < m$, $[n, m]$ is the interval $\{n, n + 1, \ldots, m - 1, m\}$, the *length* of which is $m - n + 1$. Z_m denotes the set of residues modulo m, i.e. the interval $[0, m - 1]$; G_m is the set of integers q such that $0 < q < |m|$ and q is prime to m; the number of the elements in G_m is $\phi(m)$, the Euler totient function applied to m. Occasionally we use the group structure of Z_m and G_m imposed by addition and multiplication modulo m, respectively. Finally, $\lfloor x \rfloor$ and $\lceil x \rceil$ denote the *floor* and *ceiling* functions applied to the (real) number x.

Given a set $I = \{w_1, w_2, \ldots, w_n\}$ of natural numbers and $w \in \mathbf{N}$, we consider the problem of determining in a practical way whether $w \in I$. Usually the elements of I are stored in a table with m ($\geq n$) locations, and w is to be searched for in the table. Two reviews of a variety of techniques usable for this purpose can be found in [4] and [7].

One of the most efficient techniques is *hashing* (see also [2] and [5]). The method consists in using a *hashing function* $h: \mathbf{N} \rightarrow Z_m$ and storing an element $w_i \in I$ at location $h(w_i)$ in the table. The same function h is then used to check whether a given $w \in \mathbf{N}$ is present in the table. Since several elements in I can hash to the same address, different methods have been developed to store and locate colliding elements. Therefore in general more than one probe (i.e. access to the table) is necessary to verify that $w \in I$ or to be convinced that $w \notin I$. The average number of probes can be made close to 1 at the cost of having sparse tables, that is, tables with a *loading factor* n/m much less than 1.

The set I may be *static*, that is, it does not change during the execution of a program. In this case, the distribution of the elements of I can be used to design a hashing function h which improves on the usual performance of hashing.

The best situation is given by a function h such that h on I is injective and $\max h(I) = n - 1$. The first condition assures that a single probe is sufficient to retrieve an element, and the second assures that the table is full. Given the set I, we are going to show that it is possible to construct a function h (depending on I) which satisfies the first condition and allows us to use a table with a loading factor very close to 1.

A *perfect hashing function* (phf for short) for I is a function h: $\mathbf{N} \rightarrow \mathbf{N}$ such that h on I is injective, $\min h(I) = 0$, and $\max h(I) = m - 1$ for some $m \geq n$ (m is the *length of the table*). A *minimal* perfect hashing function for I is one for which $m = n$.

The elements in I are assumed to be in ascending order; thus the set I is determined by its first element w_1 and the ordered set of its 1st-*differences* $\delta_i = w_{i+1} - w_i$ for every $i \in [1, n - 1]$. Obviously any difference $w_j - w_i$ can be expressed in terms of the differences δ_i.

2. Quotient Reduction Method

The first, and perhaps the only, previous systematic attempt to construct perfect hashing functions is given in [1]. However, if we consider the set $I = \{1, 3, 8,$

14, 17, 23}, given there as an example, the function $h(w) = \lfloor (w + 3)/5 \rfloor$ is a far simpler minimal perfect hashing function for I than the function described by Greniewski and Turski.

This example illustrates the first method we present, the *quotient reduction* method. Given a finite set $I \subset \mathbf{N}$, the method consists in translating the set I and then taking the integer quotients by some divisor $N \in \mathbf{N}$: $(\forall w \in I)$ more formally, $h(w) = \lfloor (w + s)/N \rfloor$ for some integer s depending on I. A function of this form will be called a *quotient reduction* perfect hashing function.

The translation term s can be decomposed $s = qN + s'$ for some integers q and s' $(0 \le s' < N)$. The term qN allows us to have $h(w_1) = 0$, while s' is used to adjust the elements of I to different intervals $[kN, (k + 1)N - 1]$ so that

$$h(w_i) \neq h(w_j) \tag{2.1}$$

for every $i, j \in [1, n]$ and $i \neq j$.

The perfect hashing functions generated by this method are very simple and work well for uniformly distributed sets. Furthermore the results obtained will be used later to develop more general techniques.

Algorithm Q (Quotient reduction): Given a set I as above, this algorithm finds the best quotient reduction phf for I.

```
(a) [find upper bound for $N$] $N_0 \leftarrow \min\{\lfloor (w_j - w_i - 1)/(j - i - 1) \rfloor \mid i, j \in [1, n-1]$ and $j > i + 1\}$;
(b) [initialize $\Delta$] $\Delta \leftarrow [1, N_0]$ (at the end of step (c) the set $\Delta$ will contain all the possible values for $N$);
(c) [scan $I$] $\forall i, j \in [1, n-1]$ such that $j > i$ and $\delta_i + \delta_j \le N_0$ do:
    (c1) [initialize $D$] $d \leftarrow \delta_i + \delta_j - 1$; $D \leftarrow [1, d]$ (at the end of step (c3) the set $D$ will contain all the values $N$ satisfying (2.1) relative to $i$ and $j$);
    (c2) [find $\theta'$ and $\theta''$] $m \leftarrow (w_j + 1) - w_{i+1}$; $M \leftarrow w_{j-1} - (w_i + 1)$; $\theta' \leftarrow \lceil m/N_0 \rceil$; $\theta'' \leftarrow \lceil m/(d+1) \rceil$;
    (c3) [determine $D$] $\forall \theta \in \mathbf{Z}$ $(\theta' \le \theta \le \theta'')$, do: $D \leftarrow D \cup [\lceil m/\theta \rceil, \lfloor M/\theta \rfloor]$;
    (c4) [update $\Delta$] $\Delta \leftarrow \Delta \cap D$;
(d) [find $N$] $N \leftarrow \max\Delta$ ($N$ is taken as large as possible in order to get the smallest table);
(e) [initialize $J$] $J \leftarrow Z_N$ (at the end of step (f) the set $J$ will contain the integers $s (0 \le s < N)$ satisfying (2.1) for every $i \neq j$);
(f) [determine $J$] $\forall i \in [1, n-1]$ such that $\delta_i < N$, do: $J \leftarrow J \cap \{(t - w_{i+1}) \bmod N \mid 0 \le t < \delta_i\}$;
(g) [a smaller $N$, if necessary] if $J = \varnothing$, drop $N$ from $\Delta$ and go to step (d);
(h) [choose best $t$] let $t$ be any element in $J$ minimizing $(w_1 + t) \bmod N$ (this condition assures that the table is of minimal length);
(i) [find $s$] $s \leftarrow t - N\lfloor (w_1 + t)/N \rfloor$.
```

Now let us show that this algorithm is correct.

First we prove that N_0 is an upper bound for N. By (2.1), for $j > i + 1$ we have $h(w_j) - h(w_i) > j - i - 1$; hence $(w_j + s) > (w_i + s) + (j - i - 1)N$, and so:

$$N \le \lfloor (w_j - w_i - 1)/(j - i - 1) \rfloor \tag{2.2}$$

for $i, j \in [1, n]$ and $j > i + 1$.

As an example throughout this section, let us consider the set $I = \{17, 138, 173, 294, 306, 472, 540, 551, 618\}$. We compute N_0 by means of Table II, which contains the differences of the elements in I. On each row we have circled the least difference $w_j - w_i$, and on the right we have computed $\lfloor (w_j - w_i - 1)/(j - i - 1) \rfloor$. Thus we have $N_0 = 72$ and, by step (b), $\Delta = [1, 72]$.

Table II.

	17		138		173		294		306		472		540		551		618	
1st-diff.		121		$\boxed{35}$		121		$\boxed{12}$		166		68		$\boxed{11}$		67		
2nd-diff.			156		156		133		178		234		79		$\textcircled{78}$			··· $\lfloor 77/1 \rfloor = 77$
3rd-diff.				277		168		299		246		245		$\textcircled{146}$				$\lfloor 145/2 \rfloor = 72$
4th-diff.					289		334		367		$\textcircled{257}$		312					$\lfloor 256/3 \rfloor = 85$
5th-diff.						455		402		378		$\textcircled{324}$						$\lfloor 323/4 \rfloor = 80$
6th-diff.							523		$\textcircled{413}$		445							$\lfloor 412/5 \rfloor = 82$
7th-diff.								534		$\textcircled{480}$								$\lfloor 479/6 \rfloor = 79$
8th-diff.									$\textcircled{601}$									$\lfloor 600/7 \rfloor = 85$

Now, before explaining step (c), we have to analyze the role played by the set J. So let us suppose that N has already been found and consider steps (e) and (f). For every $w_i \in I$, an *admissible increment* for w_i is any integer t for which condition (2.1) is satisfied for $j = i + 1$:

$$\lfloor (w_{i+1} + t)/N \rfloor \neq \lfloor (w_i + t)/N \rfloor. \tag{2.3}$$

In other words, an admissible increment for w_i is any translation value which adjusts w_i and w_{i+1} to two different intervals $[kN, (k + 1)N - 1]$. Clearly a quotient reduction phf for I can be found if and only if there exists an admissible increment which works for every $w_i \in I$.

The set of all the admissible increments for w_i is given by $J^*(w_i) = \{u - w_{i+1} + kN \mid 0 \le u < \delta_i$ and $k \in \mathbf{Z}\}$. In fact, let $t = u - w_{i+1} + kN$; then we have $w_i + t = w_i + u - w_{i+1} + kN = u - \delta_i + kN$ and $w_{i+1} + t = w_{i+1} + u - w_{i+1} + kN = u + kN$, and relation (2.3) holds if and only if $0 \le u < \delta_i$.

We can ignore the term kN in the expression for $J^*(w_i)$ and define the set $J(w_i)$ of the *reduced* admissible increments for w_i:

$$J(w_i) = \{(t - w_{i+1}) \bmod N \mid 0 \le t < \delta_i\}. \tag{2.4}$$

Obviously, if $\delta_i \ge N$, $J(w_i) = Z_N$ and no computation is necessary. Thus steps (e) and (f) determine the set $J = \bigcap_{i=1}^{n-1} J(w_i)$ correctly, and, as we remarked above, there exists a quotient reduction phf for I if and only if $J \neq \varnothing$.

In our example, let us suppose $N = 64$ $(\le 72 = N_0)$. In Table II we put in a box the 1st differences less than 64. It is:

$$J(138) = \{(t - 173) \bmod 64 \mid 0 \le t < 35\} = [19, 53],$$
$$J(294) = \{(t - 306) \bmod 64 \mid 0 \le t < 12\} = [14, 25],$$
$$J(540) = \{(t - 551) \bmod 64 \mid 0 \le t < 11\} = [25, 35],$$

and $J = \{25\}$. The reader can verify that for $N = 65$ and $N = 63$, $J = \varnothing$ and $J = [16, 21]$, respectively. Thus there exist quotient reduction phf's for $N = 63$ and $N = 64$, but not for $N = 65$.

Now, looking at the thing from the other side, we can determine the set Δ of possible values of N for which the associated set J is nonempty. We can prove that if $J \neq \varnothing$ (relative to N) then $\forall w_i, w_j \in I$ $(i < j)$

there exists a multiple of N in the interval $T_{ij} = [m_{ij}, M_{ij}]$, where $m_{ij} = (w_j + 1) - w_{i+1}$ and $M_{ij} = w_{j+1} - (w_i + 1)$. In fact, by (2.4), $J(w_i) \cap J(w_j) \neq \varnothing$ if and only if there exist two elements $a \in A = \{w_{i+1} - t \mid 0 \leq t < \delta_i\}$ and $b \in B = \{w_{j+1} - t \mid 0 \leq t < \delta_j\}$ such that $a \equiv b$ (modulo N). This means that the set of all the differences between the elements in B and the elements in A must contain a multiple of N. But this set is just T_{ij}, because its limits are the cross-differences of the limits of B and A.

The converse is not true because J can be empty although the $J(w_i)$'s are not pairwise disjoint. Thus $\max\Delta$ is a better approximation to N than N_0, but Δ may contain elements which do not correspond to any quotient reduction phf for I. Actually step (c) can be eliminated; however, as shown by the example below, when N is smaller than N_0, step (c) can save a lot of computations relative to the $J(w_i)$'s.

Now, if we call D_{ij} the set of positive integers with a multiple in the interval T_{ij}, we have to choose N in the set $\Delta = [1, N_0] \cap \bigcap_{i<j} D_{ij}$. By definition, it is $D_{ij} = \bigcup_{1 \leq \theta \leq m_{ij}} [\lceil m_{ij}/\theta \rceil, \lfloor M_{ij}/\theta \rfloor]$; however, we can remark that:

(a) since $N \leq N_0$, the useful lower limit for θ is $\theta' = \lceil m_{ij}/N_0 \rceil$;

(b) the length of the interval T_{ij} is $d = w_{j+1} - (w_i + 1) - (w_j + 1) + w_{i+1} + 1 = \delta_i + \delta_j - 1$; so every integer in $[1, d]$ has a multiple in T_{ij}. This implies that (i) every couple (i, j) for which $d = \delta_i + \delta_j - 1 \geq N_0$ can be ignored in the computation of Δ, and (ii) the upper limit for θ is $\theta'' = \lceil m_{ij}/(d + 1) \rceil$.

So we have $T_{ij} = [1, d] \cup \bigcup_{\theta' \leq \theta \leq \theta''} [\lceil m_{ij}/\theta \rceil, \lfloor M_{ij}/\theta \rfloor]$, and step (c) determines the set Δ correctly.

In our example we have to consider the following intervals:

$T_{24} = [122, 167]$ for which $d = 46, \theta' = 2, \theta'' = 3$;
$T_{27} = [368, 412]$ for which $d = 45, \theta' = 6, \theta'' = 8$;
$T_{47} = [235, 256]$ for which $d = 22, \theta' = 4, \theta'' = 11$;

so:

$D_{24} = [1, 55] \cup [61, 72]$;
$D_{27} = [1, 51] \cup [53, 58] \cup [62, 68]$;
$D_{47} = [1, 25] \cup [27, 28] \cup [30, 32] \cup [34, 36] \cup [40, 42] \cup [47, 51] \cup [59, 64]$;
$\Delta = [1, 25] \cup [27, 28] \cup [30, 32] \cup [34, 36] \cup [40, 42] \cup [47, 51] \cup [62, 64]$.

Now, every $N \in \Delta$ can possibly determine one or more quotient reduction phf's. It is possible to show that if $N > N'$ the table length for N is not longer than any table length for N'. Hence we choose N as large as possible (steps (d) and (g)). In our example, for $N = 64 = \max \Delta$, it is $J = \{25\}$.

Finally, let us come to steps (h) and (i). Up to this moment, we have found N and the set J of the reduced admissible increments relative to N. Every $t \in J$ determines a quotient reduction phf for I; in order to have $h(w_1) = 0$, the translation value s_t is given by $s_t = t - \lfloor (w_1 + t)/N \rfloor$. Since we are looking for the shortest table, $h(w_n)$ is to be as small as possible. This occurs when $w_1 + s_t$ (which is non-negative and less than N) takes its nearest value to 0. However, since $w_1 + t \equiv w_1 + s_t$ (modulo N), the quantity $(w_1 + t) \bmod N$ is to be as small as possible, and this condition determines the value t of the "best" reduced admissible increment.

In our example, it is $s = t = 25$, and the best quotient reduction phf is $h(w) = \lfloor (w + 25)/64 \rfloor$. It is not minimal, as shown by the following table:

w	17	138	173	294	306	472	540	551	618
$h(w)$	0	2	3	4	5	7	8	9	10

Now, let us consider a possible improvement of the quotient reduction method. Obviously, in order to have an (almost) full table for a given set I, we should have $N \approx (w_n - w_1)/(n - 1)$. However, if the elements in I are not uniformly distributed, N can be considerably smaller than this best value; so the table is sparse. Often it is possible to obtain shorter tables introducing the concept of a *cut*. A cut consists in translating all the elements in I larger than a certain value (the *cut value*) before performing the quotient reduction. Since the cut value can be identified with some $w_t \in I$, the index t will be called the *cut point* and the perfect hashing function h is defined by:

$$h(w_i) = \lfloor (w_i + s)/N \rfloor, \qquad \forall i \leq t,$$
$$h(w_i) = \lfloor (w_i + s + r)/N \rfloor, \quad \forall i > t, \qquad (2.5)$$

for some integer r.

The problem consists in finding the cut value (or, equivalently, the cut point) and the integer r for which the table may be of minimal length. In order to satisfy condition (2.3), the integer r should produce an admissible increment common to all the elements in I. An obvious approach is to try successively all the elements $w_1, w_2, \ldots, w_{n-1}$ as possible cut values. This leads to the following Algorithm C. An important point in this algorithm is the evaluation of N; we ignore the differences of elements in I on opposite sides of the cut point. Actually the values of these differences will be determined only when the integer r is known.

Algorithm C (Quotient reduction with a cut). Given a set I, this algorithm determines the cut point t and the integer r, allowing us to obtain the best cut reduction phf of the form (2.5).

```
(a) [initialize t] $t \leftarrow 1$;
(b) [upper bound for $N_t$] $N_0 \leftarrow \min \{\lfloor (w_j - w_i - 1)/(j - i - 1) \rfloor \mid (i, j \in [1, t]$
    or $i, j \in [t+1, n])$ and $j > i+1\}$ (ignore couples $w_i, w_j$ such that $i \leq t < j$);
(c) [find $\Delta$] evaluate $\Delta$ as in steps (b) and (c) of Algorithm Q;
(d) [find $N_t$] $N_t \leftarrow \max\Delta$;
(e) [admissible increments] $J_L \leftarrow \bigcap_{i=1}^{t-1} \{(v - w_{i+1}) \bmod N_t \mid 0 \leq v < \delta_i\}$;
    $J_R \leftarrow \bigcap_{i=t+1}^{n-1} \{(v - w_{i+1}) \bmod N_t \mid 0 \leq v < \delta_i\}$;
(f) [a smaller $N_t$, if necessary] if either $J_L$ or $J_R$ is empty, drop $N_t$
    from $\Delta$ and go to step (d);
(g) [lower bound for $\delta_t'$] $\delta_0 \leftarrow \min \{(j - i - 1) N_t + 1 - w_j + w_i + \delta_t \mid i \leq t < j\}$;
(h) [find $s'$, $\delta_t'$] perform the following steps:
    (h1) $\hat{p} \leftarrow \min \{p \in [1, N_t] \mid (-w_t - p) \bmod N_t \in J_L\}$;
    (h2) let $s'$ be the element in $J_L$ corresponding to the value of $\hat{p}$;
    (h3) $\delta_t' \leftarrow \min \{\delta \geq \delta_0 \mid \exists j'' \in J_R$ such that $\delta \equiv w_{t+1} + j'' + \hat{p}$ (modulo $N_t$)$\}$;
(i) [determine $r, s$] $r_t \leftarrow \delta_t' - \delta_t$; $s_t \leftarrow s' - N_t \lfloor (w_1 + s')/N_t \rfloor$;
(j) [length of the table] $L_t \leftarrow \lfloor (w_n + s_t + r_t)/N_t \rfloor - 1$;
```

```
(k) [loop on $t$] $t \leftarrow t+1$ and go to step (b) if $t<n$;
(l) [table of minimal length] let $z$ any index for which $L_z$ is minimal
    and output $s_z, N_z, w_z, r_z$.
```

Now let us show that this algorithm is correct. For fixed t, we consider the set $I_t = \{w_i | 1 \leq i \leq t\} \cup \{w_i + r | t < i \leq n\}$; then the function (2.5) relative to I is equivalent to a quotient reduction phf for I_t. However, the 1st differences of I_t equal the 1st differences of I, except for $\delta_t' = \delta_t + r$. Thus the determination of r is equivalent to the evaluation of δ_t', and the most important steps of the algorithm are (g) and (h).

Step (g) determines a lower bound for δ_t' by applying relation (2.2) to the set of differences $w_j - w_i$ ($i \leq t < j$). By steps (h1) and (h2) we have $\hat{p} = -w_t - s' + k'N$, and by (h3), $\delta_t' = w_{t+1} + j'' + \hat{p} + k''N = \delta_t + j'' - s' + kN$, where $k = k' + k''$; hence, for every $w_i' \in I_t$ ($i > t$), $w_{i+1}' + s' = w_{i+1} + \delta_t' - \delta_t + s' = w_{i+1} + j'' + kN$. This proves that s' is a reduced admissible increment for w_i'. Since $\delta_t' \geq \delta_0$, s' is a reduced admissible increment for every element in I_t. Actually $\hat{p}$ has been defined in such a way that s' is the "best" reduced admissible increment. Thus Algorithm C determines the best cut reduction phf (2.5) correctly.

The run time of Algorithm C, however, is exceedingly high, at least of order Kn^3, where K is the largest value of $\theta'' - \theta' + 1$ (step (c3)). If we content ourselves with a near-to-optimal function of the form (2.5), we can find a far better algorithm. In fact, N is usually very close to N_0, and, by construction, a cut adjusts w_t and w_{t+1} to two consecutive intervals $[kN, (k+1)N - 1]$. Thus we may assume that the value $(w_n - w_1 - \delta_t)/N_0 + 3$ approximates the length of the table obtained using t as a cut point. This leads to the following:

Algorithm S (Simplified quotient reduction with a cut). Given a set I, this algorithm first determines the cut point t corresponding to a nearly optimal perfect hashing function of the form (2.5) and then finds the other parameters of the function.

```
(a) [initialize $t$] $t \leftarrow 1$;
(b) [upper bound for $N$] $N_t^0 \leftarrow \min\{\lfloor (w_j-w_i-1)/(j-i-1)\rfloor | (i, j \in [1, t]$ or $i, j \in [t+1, n])$ and $j>i+1\}$;
(c) [approximate length of the table] $L_t \leftarrow (w_n-w_1-\delta_t)/N_t^0$;
(d) [loop on $t$] let $t \leftarrow t+1$ and go to step (b) if $t<n$;
(e) [shortest table] let $z$ be any index for which $L_z$ is minimal;
(f) [initialize $N$] $N \leftarrow N_z^0+1$;
(g) [decrement $N$] $N \leftarrow N-1$;
(h) [admissible increments] $J_L \leftarrow \bigcap_{i=1}^{z-1} \{(v-w_{i-1}) \bmod N | 0 \leq v < \delta_i\}$;
    $J_R \leftarrow \bigcap_{i=z+1}^{n-1} \{(v-w_{i+1}) \bmod N | 0 \leq v < \delta_i\}$;
(i) [lower bound for $\delta_z'$] $\delta_0 \leftarrow \min\{(j-i-1)N+1-w_j+w_i+\delta_z | i \leq z < j\}$;
(j) [find $s'$, $\delta_z'$] perform the following steps:
    (j1) $\hat{p} \leftarrow \min\{p \in [1, N] | (-w_z-p) \bmod N \in J_L\}$;
    (j2) let $s'$ be the element in $J_L$ corresponding to the value of $\hat{p}$;
    (j3) $\delta_z' \leftarrow \min\{\delta \geq \delta_0 | \exists j'' \in J_R$ such that $\delta \equiv w_{z+1}+j''+\hat{p}$ (modulo $N$)$\}$;
(k) [determine $r, s$] $r \leftarrow \delta_z'-\delta_z$; $s \leftarrow s'-N\lfloor (w_1+s')/N \rfloor$;
(l) [output results] output $s, N, w_z, r$.
```

Let us apply Algorithm S to our example; we get:

t	1	2	3	4	5	6	7	8
N_0	72	72	72	72	72	77	83	78
L	6.67	7.86	6.67	8.18	6.04	6.92	7.11	6.85

so $z = 5$. Considering the differences in the shaded area of Table II, we obtain $\delta_0 = 131$. Now we have $J_L = J(138) \cap J(294) = [54, 65]$ and $J_R = J(472) \cap J(540) \cap J(551) = [30, 31]$; so $\hat{p} = \min\{p \in [1, 72] | (-306 - p) \bmod 72 \in [54, 65]\} = 61$, and $s' = 65$. Hence $\delta_5' = \min\{\delta \geq 131 | \exists j'' \in [30, 31]$ such that $\delta \equiv 472 + j'' + 61$ (modulo 72)$\}$, and it is simple to see that $\delta \equiv 59$ or $\delta \equiv 60$ (modulo 72). However, $131 \equiv 59$ (modulo 72); so $\delta_5' = 131$, $r = -35$, $s = -7$. The perfect hashing function obtained is minimal and can be implemented by the two Fortran statements:

```
IF(W.GT.306) W = W - 35
H = (W - 7)/72
```

where H and W are INTEGER variables.

We conclude this section with three remarks:

(i) The programmer has to consider the fact that if $w > w_n$ (and $h(w) > m$), w is compared to something outside the table. Thus some care has to be taken when allocating the table or an extra comparison between $h(w)$ and m must be introduced.

(ii) It is possible to consider cut reduction phf's with more than one cut. An algorithm similar to Algorithm S is easily devised to get a near-to-optimal function of this new type.

(iii) All the functions considered in this section are monotonic.

3. Remainder Reduction Method

As we have remarked, the quotient reduction method works well when the set I is uniformly distributed. However, this is not always the case, especially when the elements in I are derived from identifiers coded in EBCDIC. In fact, the collating sequence for letters and digits contains three large gaps—between I and J, R and S, and Z and 0.

When the set I is not uniformly distributed, we can scramble its elements to get a more uniform distribution and try to apply a quotient reduction phf to the scrambled set. To scramble the elements of I in a 1–1 manner, we take the moduli of the elements in I by some appropriate integer divisor M. This method will be called the *remainder reduction* method.

In order that the *scrambled* set $I_M = \{w_i \bmod M | w_i \in I\} \subset Z_M$ be of interest to us, we should have:

$$w_i \not\equiv w_j \text{ (modulo } M), \quad \forall i, j \in [1, n] \text{ and } i \neq j. \tag{3.1}$$

If $\mathbf{D}$ is the set of divisors of some difference $w_j - w_i$ ($j > i$), condition (3.1) is verified if and only if $M \notin \mathbf{D}$. In what follows, we suppose $M \notin \mathbf{D}$ if not otherwise stated.

We are interested in obtaining *remainder reduction* perfect hashing function of the form:

$$h(w) = \lfloor ((d + wq) \bmod M)/N \rfloor. \tag{3.2}$$

In order to do this, the values d, q, N, and M must be suitably chosen. In particular, to chose N and d, we need to consider some properties of the scrambled set I_M with respect to a simpler form of function, which we call a *rotation reduction* perfect hashing function:

$$h(w) = \lfloor((d + w) \bmod M)/N\rfloor. \tag{3.3}$$

A (scrambled) set I_M will be called *rotationally reducible* by $N \in \mathbf{Z}$ iff there exists a rotation reduction phf (3.3) for I_M.

Since $(d + w) \bmod M = (d + (w \bmod M) \bmod M)$, the set $d + I_M = \{(d + w_i) \bmod M | w_i \in I_M\}$ is a rotation of I_M; we call d the *rotation value*. Thus, in other words, I_M is rotationally reducible by N iff there exists a rotation value d such that $d + I_M$ has a quotient reduction phf of the form $h(w) = \lfloor w/N\rfloor$.

Therefore, in a sense, a rotation reduction phf is analogous to a quotient reduction phf in which the translation has become a rotation. However, a rotation reduction phf has a remarkable advantage over a quotient reduction phf. In fact, the rotation allows us to ignore the 1st difference between the two elements which become the last and the first elements of the rotated set. We can choose this difference as the one which causes trouble for determining N. Thus a rotation reduction phf is more like a cut reduction phf than it is like a simple quotient reduction phf.

It is possible to verify the validity of these remarks by an example. Let $I_{19} = \{0, 1, 4, 6, 8, 12, 14\}$; it is rotationally reducible by 3, and a suitable rotation is $14 + I_{19} = \{1, 3, 7, 9, 14, 15, 18\}$.

The simplest method to find whether a set I_M is rotationally reducible is to look successively at all the M possible rotation values; however, we can look at them in parallel. Let us consider the *extended set* of I_M, defined as the set of $2n - 1$ elements $X_M = I_M \cup (M + (I_M \setminus \{w\})) \subset Z_{2M}$, where w is the largest element in I_M. We always suppose that the elements in X_M are arranged in ascending order. The following theorem is the basis for finding the rotation reduction phf's for I_M; moreover, it tells the whole story about the connections between rotation and quotient reduction phf's.

THEOREM A. *Let I_M and X_M be as above. I_M is rotationally reducible by N if and only if there exists a subset $X_M^{(i)} = \{w_{i+1}, w_{i+2}, \ldots, w_{i+n}\}$ of n consecutive elements of X_M which has a quotient reduction phf $h(w) = \lfloor(w + s)/N\rfloor$ such that $w_{i+n} + s < M$.*

In fact, if I_M is rotationally reducible, then w_{i+1} is the element which becomes the first after the rotation. Since w_{i-n} becomes the last element, the condition $w_{i+n} + s < M$ is verified. Conversely, if there exists a set $X_M^{(i)}$ satisfying the conditions of the theorem, then either $i = 0$ and $s \geq 0$ so that the rotation value d equals s, or $s < 0$ and then $d = s + M$.

For the set I_{19}, we have $X_{19} = \{0, 1, 4, 6, 8, 12, 14, 19, 20, 23, 25, 27, 31\}$ and we try to find a quotient reduction phf for $N = 3$ as shown in Table III.

In order that a quotient reduction phf exist for a subset $X_M^{(i)}$, condition (2.2) must hold. We have circled the differences violating the condition; so the differences in the dark shaded area can be ignored during

Table III.

	0	1	4	6	8	12	14	19	20	23	25	27	31	N(k-1)+1
1st-diff.	1	3	2	2	4	2	5	1	3	2	2	4		1
2nd-diff.	4	5	4	6	6	7	6	4	5	4	6			4
3rd-diff.	6	7	8	8	11	8	9	6	7	8				7
4th-diff.	8	11	10	13	12	11	11	8	11					10
5th-diff.	12	13	15	14	15	13	13	12						13
6th-diff.	14	18	16	17	17	15	17							16
7th-diff.	19	19	19	19	19	19								19

the computation. Moreover, in the tuple of kth differences there must be $n - k$ consecutive differences satisfying (2.2); so the differences in the light shaded area can also be ignored. Usually these properties provide a fast way to decide that a set is not rotationally reducible.

In our example, we have three candidate sets $X_{19}^{(2)}$, $X_{19}^{(3)}$, and $X_{19}^{(4)}$, corresponding to the $(n - 1)$th differences not enclosed in the shaded areas. We have the following sets of admissible increments: $J(4) = \{0, 1\}$, $J(6) = J(12) = \{1, 2\}$, $J(19) = \{1\}$; so 1 is the only admissible increment for all the three sets.

For $X_{19}^{(2)}$ we have $s = 1$; however, $w_{2+7} + s = 20 > 19$, and the last condition of Theorem A is violated. For $X_{19}^{(3)}$ it is $s = -2$ and $w_{3+7} + s = 18 < 19$; so we have the reducible rotation $(19 - 2) + I_{19} = 17 + I_{19}$. Finally, for $X_{19}^{(4)}$, it is $s = -5$ and $w_{4+7} + s = 18 < 19$; so we have the reducible rotation $(19 - 5) + I_{19} = 14 + I_{19}$, which is the rotation we have considered already.

At this point we can state an algorithm for finding rotation reduction phf's.

Algorithm T (Rotation reduction): Given a (scrambled) set I_M, this algorithm finds all the rotation values d corresponding to rotation reduction phf's for I_M by a given integer N.

```
(a) [extended set] $X_M \leftarrow I_M \cup (M+(I_M \setminus \{\max I_M\}))$;
(b) [initialize V] $V \leftarrow [1, n]$ (at the end of step (c), $V$ will contain
    the indices $i$ corresponding to sets $X_M^{(i-1)}$ for which the first
    condition of Theorem A is satisfied);
(c) [determine V] $\forall k \in [2, n-1]$ do:
    (c1) [kth differences] compute the set $\Delta$ of $k$th differences ($\Delta$
         contains $2n-k-1$ elements);
    (c2) [verify condition (2.2)] let $D$ be the set of indices $t$ of $\delta_t \in \Delta$
         such that $\delta_t > N(k-1)$;
    (c3) [update V] $V \leftarrow V \cap \{i \in [1, n] | [i, i+n-k] \subset D\}$ ($D$ has to
         contain $(n-k)$ consecutive differences satisfying (2.2));
    (c4) [no reduction] if $V = \varnothing$, exit (the set $I_M$ does not have any
         rotation reduction phf by $N$);
(d) [initialize i] let $i$ be the first element in $V$ (we consider $X_M^{(i-1)}$);
(e) [find J] $J \leftarrow \bigcap_{j=i}^{i+n-2} \{(t-w_{j+1}) \bmod N | w_j \in X_M \text{ and } 0 \leq t < \delta_j\}$;
(f) [no common admissible increment] if $J = \varnothing$, go to step (k);
(g) [find t] let $t$ be the element in $J$ corresponding to the least value
    of $\{(w_i + t) \bmod N | t \in J\}$;
(h) [determine s] $s \leftarrow t - N\lfloor(w_i + t)/N\rfloor$;
(i) [no rotation value] if $w_{i+n-1} + s \geq M$, go to step (k) (the last
    condition of Theorem A is violated);
(j) [find rotation value] output $s \leftarrow t \bmod N$;
(k) [loop on V] let $i$ be the next element in $V$; if there are no more
    elements, exit; else go to step (e).
```

We remark that the rotation reduction phf by N can be (almost) minimal if $M \approx Nn$; so we can consider N as determined by M and n.

Given the set I_M, Algorithm T may fail in finding a rotation reduction phf for I_M. However, if q is any element in G_M, the set $qI_M = \{qw \bmod M | w \in I_M\}$ is a subset of Z_M with n distinct elements because q is prime to M. In general, if $q, q' \in G_M$ and $q \neq q'$, the distributions of qI_M and $q'I_M$ in Z_M may be different, and we have the possibility of investigating many sets qI_M for a rotation reduction phf. Actually, by the following theorem, at most $\phi(M)/2$ sets qI_M have different distributions.

THEOREM B. *The set qI_M is rotationally reducible by N if and only if $(M - q)\, I_M$ is also.*

Intuitively the differences of $(M - q)I_M$ coincide with the differences of qI_M, but in reverse order. This justifies Theorem B, but a formal proof is rather intricate. We sketch it here: If qI_M is rotationally reducible, let $X_M^{(i)}$ be the set satisfying the conditions of Theorem A. Depending on I, it is possible to find a set $X_M^{(j)}$ satisfying the first condition of Theorem A, but not necessarily the last one ($w_{j+n} + s < M$). Passing to $(M - q)\, I_M$, we may consider the sets corresponding to $X_M^{(i)}$ and $X_M^{(j)}$ and prove that either satisfies all the conditions of Theorem A. Hence $(M - q)\, I_M$ is rotationally reducible.

We may now return to remainder reduction phf's. Formula (3.2) can be written:

$$h(w) = \lfloor (((w + d')q) \bmod M)/N \rfloor \qquad (3.4)$$

where $d' = dq^{-1} \in Z_M$ and q^{-1} is the inverse of q in the multiplicative group G_M, that is $qq^{-1} \equiv 1$ (modulo M). This formula requires two divisions and one multiplication. Computationally this is usually worse than the one comparison, two sums, and one division required by a cut reduction phf. However, it is possible to improve the performance of (3.4) on binary computers with shifting operations. In fact, if M is odd, the integer 2 is prime to M and we can consider the elements $q \in G_M$ belonging to the subgroup generated by 2. In this case we have $q \equiv 2^{q'}$ (modulo M) for some q' ($0 \leq q' < \phi(M)$), and the multiplication by q can be replaced by a shifting of q' positions.

Obviously the shift cannot be of indefinite length. For example, in System/370 double shifting can be used, but the subsequent division by M may not be allowed to produce a fixed-point divide exception. Thus, in what follows, we consider an upper bound K for the length of shifting operations. K depends on the particular computer on which the remainder reduction phf has to be implemented.

Now, if in (3.4) we have $N = 2^{m'}$ for some integer m', the last division also can be replaced by a shift. We say that the set I has a *computational remainder reduction* phf iff there exists a perfect hashing function (3.2) for I such that M is odd, $N = 2^{m'}$ for some integer m', and $q \equiv 2^{q'}$ (modulo M) for some integer q' ($0 \leq q' \leq K < \phi(M)$).

For example, the set of the abbreviations of the twelve months considered in the Introduction has the following computational remainder reduction phf:

$$\begin{aligned} h(w) &= \lfloor ((3w + 4) \bmod 23)/2 \rfloor \\ &= \lfloor (((w + 9)2^8) \bmod 23)/2^1 \rfloor \end{aligned} \qquad (3.5)$$

where w is obtained from the EBCDIC coding of the 2nd and 3rd characters of every month.

Now, in order that the elements in I can be mapped onto different values, it must be that $N(n - 1) < M$. Hence, for any $M \notin \mathbf{D}$, there is only a finite number of values for q, d, N corresponding to possible computational remainder reduction phf's for I. Thus a procedure can be devised to perform an exhaustive search for such parameters. However, we are looking for an almost minimal hashing function; that is, the resulting table must have a loading factor not less than a given value α. A sufficient condition is to have $M \leq N \lfloor n/\alpha \rfloor$; so we propose:

Algorithm R (Remainder reduction). Given a set I, this algorithm is looking for a computational remainder reduction phf for I which gives a table with a loading factor greater than or equal to α. The algorithm stops unsuccessfully if m' becomes larger than a predefined value $\hat{m}$.

```
(a) [initialize N] $m' \leftarrow 0$; $N \leftarrow 1$;
(b) [initialize M] $M \leftarrow N\,(n-1)+1$;
(c) [check whether $M \in \mathbf{D}$] if $\forall w_i, w_j \in I\ (i \neq j)$, $w_i \not\equiv w_j$ (modulo $M$),
    then go to step (f) (as a by-product we get the scrambled set $I_M$);
(d) [increment M] $M \leftarrow M+1+(m'>0)$ (increment by 1 if $N=1$, by 2
    if $N>1$);
(e) [loop on M] if $M<N\lfloor n/\alpha \rfloor$, then go to step (c), else go to step (k);
(f) [initialize q] $q \leftarrow 1$; $q' \leftarrow 0$;
(g) [rotation reduction phf] apply Algorithm T to $qI_M$; if a rotation
    reduction if found, output $M, N, d$, and $q$ and exit, else continue;
(h) [update q] $q \leftarrow (2q) \bmod M$; $q' \leftarrow q'-1$;
(i) [loop on q] if $q \neq 1$ and $q \neq M-1$ and $q' \leq K$, go to step (g);
(j) [update N] $m' \leftarrow m' + 1$; $N \leftarrow 2N$;
(k) [loop on N] if $m' \leq \hat{m}$ go to step (b); else exit unsuccessfully.
```

Several remarks may be made about this algorithm:

(i) Algorithm R may fail in finding a computational remainder reduction phf for some set I. Actually we have not been able to prove that the algorithm will eventually stop if we do not introduce an upper limit $\hat{m}$ to the loops on m'. Many experiments were made selecting at random 12 items out of a set of 93 identifiers coded in EBCDIC. Algorithm R always found a minimal ($\alpha = 1$) function. This is by no means conclusive, and presumably there is an upper bound on the size of the sets I for which a minimal (computational) remainder reduction phf can be found.

(ii) It is necessarily a rather complex procedure to evaluate the set $\mathbf{D}$. Hence we have introduced step (c), which applies relation (3.1).

(iii) When $m' = 0$, it is not necessary to apply Algorithm T. In fact, we try to reduce I by $M = n, n + 1, \ldots, \lfloor n/\alpha \rfloor$, and whenever condition (3.1) is satisfied, we get the simple function $h(w) = w \bmod M$. If $M = n$ the table is minimal. If $M = n + 1$, we can always make the table minimal by a rotation, taking the empty position to the end of the table. If $M = n + 2$, there are two empty positions in the table, say positions λ and μ. It is possible to take them to the last two positions in the table if there exists q such that

$q\lambda \equiv q\mu + 1$ (modulo M). Obviously such a q exists if $(\lambda - \mu)$ is prime to M, so that $q = (\lambda - \mu)^{-1}$ in G_M.

(iv) When $m' = 1$ (i.e. $N = 2$), the application of Algorithm T can be greatly simplified by using the following result as a substitute for Theorem A:

THEOREM A′. *Given a (scrambled) set I_M, let us consider the set of its 1st differences $\{\delta_1, \delta_2, \ldots, \delta_n\}$, where $\delta_n = M + w_1 - w_n$; I_M is rotationally reducible by $N = 2$ if and only if there is an index $t \in [1, n]$ such that:*

(1) *$\forall i < t$, the w_i's with $\delta_i = 1$ have the same parity P, and $\forall i > t$ the w_i's with $\delta_i = 1$ have the same parity P', with $P \neq P'$, and*
(2) *if $\delta_t = 1$ then $\delta_{t+1} \neq 1$.*

The proof is rather obvious; we rotate I_M in such a way that w_t becomes the last element and every w_i ($i \neq t$) such that $\delta_i = 1$ becomes odd. This is possible because M is odd, $P \neq P'$, and, by (2), if w_{t+1} is changed into 0 by the rotation, it does not influence the positions of the elements w_i with $\delta_i = 1$.

The example concerning the abbreviations of the months can be developed by hand. If we consider the EBCDIC coding of the second and third characters of every month, we get the set I = {49621, 50626, 49625, 55257, 49640, 58581, 58579, 58567, 50647, 50147, 55013, 50627}. For better referencing we have arranged the elements in their natural order, rather than in ascending sequence. With $\alpha = 0.85$ we have the situation of Table IV after step (c) of Algorithm R. Now we have to multiply I_{23} successively by $q = 1$, $q = 2$, $q = 4$ and so on. By Theorem B we have at most 11 different distributions, as shown in Table V. On the right and up to $q = 13$ we have written the elements with $\delta_i = 1$ which violate the first condition of Theorem A′ concerning parity. When $q = 3 \equiv 2^8$ (modulo 23), the condition is verified for $w_t = 17$ and $w_t = 18$. However, the last condition of the same theorem excludes $w_t = 18$. Thus the set has to be rotated in such a way that $w_{t+1} = 19$ becomes the first element and 3, 9, 17 remain odd. This implies $d = 4$, and we get the minimal computational remainder reduction phf (3.5).

Obviously $3^{-1} = 8$ because $3 \times 8 = 24 \equiv 1$ (modulo 23). In general, since q is prime to M, q^{-1} can be computed by means of the Euclidean Algorithm as the solution of the congruence $qx \equiv 1$ (modulo M); see, e.g. [4] and [6]. Also Euler's Theorem:

$$a^{\phi(M)} \equiv 1 \text{ (modulo } M) \quad \text{if } a, M \text{ are coprime}$$

can be used because $a = 2$ and M is odd. If we set $q = 2^{q'}$, it is $q^{-1} \equiv 2^{\phi(M)-q'}$ (modulo M). In our example, we have by construction that $3 \equiv 2^8$ (modulo 23); 23 is a prime number; hence $\phi(23) = 22$ and $3^{-1} \equiv 2^{22-8} \equiv (2^7)^2 \equiv 13^2 \equiv 8$ (modulo 23).

Finally, we point out that it is not necessary to check the addresses obtained by a perfect hashing function generated by the remainder reduction method since these addresses always point into a well-defined limited interval.

4. Segmentation

When the set I is very small, the problem of determining whether $w \in I$ is usually solved by means of a linear search. Hashing with a perfect hashing function is an improvement over this direct method. However, when we try to extend our results to larger sets, we are faced with two major problems:

(i) As the number n of the elements in I increases, the probability of a reasonably uniform distribution of I or of the sets qI_M decreases very rapidly (in the case of I_M, Knuth calls this the "birthday paradox").

(ii) The run time of our algorithms is always greater than $O(n^2)$; so they may become impractical when n becomes large.

So let us consider the following approach, which will be called *segmentation*. In fact, the set I will be divided into "segments," something like buckets in ordinary hashing.

Let $I = \{w_1, w_2, \ldots, w_n\}$. A *grouping function* for I is a function $\rho:\mathbf{N} \to \mathbf{N}$ such that ρ on I takes its values in Z_K for some positive integer $K < n$. For

Table IV.

	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC	
$M = 12$	1	10	5	9	8	9							
$M = 13$	0	4	4										$m' = 0$
$M = 14$	5	2	9	13	10	5							
$M = 23$	10	3	14	11	6	0	21	9	1	7	20	4	$m' = 1$

Table V.

$q = 1$	10	3	14	11	6	0	21	9	1	7	20	4	0/1, 3/4, 9/10/11
$q = 2$	20	6	5	22	12	0	19	18	2	14	17	8	5/6, 17/18/19/20
$q = 4$	17	12	10	21	1	0	15	13	4	5	11	16	0/1, 4/5, 10/11/12/13
$q = 8$	11	1	20	19	2	0	7	3	8	10	22	9	0/1/2/3
$q = 16$	22	2	17	15	4	0	14	6	16	20	21	18	14/15/16/17/18
$q = 9$	21	4	11	7	8	0	5	12	9	17	19	13	4/5, 7/8/9
$q = 18$	19	8	22	14	16	0	10	1	18	11	15	3	0/1, 10/11, 14/15/16, 18/19
$q = 13$	15	16	21	5	9	0	20	2	13	22	7	6	5/6/7, 15/16
$q = 3$	7	9	19	10	18	0	17	4	3	21	14	12	3/4, 9/10, 17/18/19

every $i \in Z_K$, the ith *segment* of I is the set $S_i = \{w \in I \mid \rho(w) = i\} = I \cap \rho^{-1}(i)$. Let us suppose that for every $i \in Z_K$ the segment S_i contains a small number of elements (say, at most 10–12) and there exists a perfect hashing function for S_i. If we define the *base* of the ith segment as $b_i = i + \sum_{j=0}^{i-1} \max h_j(S_j)$, where an empty summation evaluates to zero, we get the perfect hashing function:

$$h(w) = b_i + h_i(w) \quad \text{where } i = \rho(w).$$

As will be clear from the examples below, it may be convenient to apply ρ and h_i to some functions of w. If we denote these functions by Ψ and ψ_i, respectively, we have the more general expression for h:

$$h(w) = b_i + h_i(\psi_i(w)) \quad \text{where } i = \rho(\Psi(w)).$$

Now let us remark that every hashing function can be used as a grouping function for I. In fact, if two or more items are mapped onto the same address i, they constitute the ith segment. This allows a rough comparison between ordinary hashing and our method, where: (a) the ordinary hashing function becomes the grouping function; (b) the function used to resolve collisions is replaced with the perfect hashing functions h_i; (c) as soon as we have obtained $h(w)$, we are sure that if $w \in I$ it is located at position $h(w)$ in the table; this is not true of ordinary hashing.

Hence our method can be worse than ordinary hashing only if the cost of a probe is sufficiently low or the average number of probes (for ordinary hashing) is very close to 1. Usually this is obtained at the cost of sparse tables. Besides, in external searching, it may be useful to know in advance that a single probe is sufficient to locate the relevant record or page of records.

One possible form of grouping function is $\rho(w) = \lfloor (w + S)/R \rfloor$, where R and S are two integers. Just as for the integer s in quotient reduction phf's, a suitable value of S can give an optimal distribution of the segment lengths. The remainder $(w + S) \bmod R$ is obtained at no extra cost and can be used as the unique function $\psi = \psi_i$, $\forall i \in Z_K$. Note that if the perfect hashing functions are all quotient reduction phf's, then the function h is monotonic.

A second kind of grouping function is $\rho(w) = w \bmod R$ for some integer R. In this case the quotient $\lfloor w/R \rfloor$ can be used as the unique function $\psi = \psi_i$, $\forall i \in Z_K$.

There are several variants of these techniques. As an example, let us consider the set of the 31 most frequent words in English, as given in [4]. Using the collating sequence for characters given by Knuth, we developed this example by hand in about two hours. For the function Ψ we used the first character in the words and for the grouping function ρ we used $\rho(w) = w \bmod 4$. This creates four segments: in the first segment there are grouped the words beginning with H, O, Y; in the second, words beginning with A, I; in the third, words beginning with B, F, W; in the fourth, words beginning with N, T.

We used a single function ψ_i for $i = 0, 1, 2, 3$, that is the value of the first three characters in each word. Then, we looked for four computational remainder reduction phf's with the same N to simplify implementation. We obtained four minimal perfect hashing functions:

$$h_0(w) = [(((w + 33)2^{10}) \bmod 75)/2^3]$$
$$h_1(w) = [(((w + 68)2^{0}) \bmod 71)/2^3]$$
$$h_2(w) = [(((w + 51)2^{4}) \bmod 67)/2^3]$$
$$h_3(w) = [(((w + 38)2^{1}) \bmod 41)/2^3]$$

The bases of the segments are 0, 9, 18, and 26, respectively, and the hash table is given in Table VI. By using the packing capabilities and the instructions of a binary MIX computer, it is possible to code h very efficiently and achieve a remarkable saving of space and computing time over [4].

Finally, as a third kind of grouping function, we mention *folding*. It can be applied to integers of any precision (i.e. to identifiers of any length) and is known to produce good results. The function Ψ is the identity. Moreover, in each segment S_i there are only a few identifiers. Usually two (or at most three) characters are sufficient to identify an item in its segment. Then, for every segment S_i, we can record the positions of the relevant characters and use as the functions ψ_i the value obtained selecting these characters.

Table VI.

TABLE	HE
+1	HAVE
+2	OF
+3	HAD
+4	HER
+5	ON
+6	HIS
+7	YOU
+8	OR
+9	IS
+10	I
+11	AND
+12	AT
+13	AS
+14	A
+15	IN
+16	ARE
+17	IT
+18	WITH
+19	FOR
+20	FROM
+21	BY
+22	WHICH
+23	WAS
+24	BUT
+25	BE
+26	NOT
+27	TO
+28	THAT
+29	THE
+30	THIS