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* $h: \mathbf{N} \to Z_m$ 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 $h: \mathbf{N} \to \mathbf{N}$ 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] $N_0 \leftarrow \min\{\lfloor (w_j - w_i - 1)/(j - i - 1) \rfloor \mid i, j \in [1, n-1] \text{ and } j > i+1\}$;
(b) [initialize Δ] $\Delta \leftarrow [1, N_0]$ (at the end of step (c) the set Δ 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 θ' and θ''] $m \leftarrow (w_j + 1) - w_{i+1}$; $M \leftarrow w_{j+1} - (w_i + 1)$; $\theta' \leftarrow \lceil m/N_0 \rceil$; $\theta'' \leftarrow \lfloor m/(d+1) \rfloor$;
  (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 Δ] $\Delta \leftarrow \Delta \cap D$;
(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 Δ 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	(35)	121	(12)	166	68	(11)	67		
2nd-diff.		156	156	133	178	234	79	(78)		⌊77/1⌋ =77
3rd-diff.		277	168	299	246	245	(146)			⌊145/2⌋ =72
4th-diff.			289	334	367	(257)	312			⌊256/3⌋ =85
5th-diff.				455	402	378	(324)			⌊323/4⌋ =80
6th-diff.					523	(413)	445			⌊412/5⌋ =82
7th-diff.						534	(480)			⌊479/6⌋ =79
8th-diff.							(601)			⌊600/7⌋ =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 $J^*(w_i) = \{u - w_{i+1} + kN \mid 0 \le u < \delta_i \text{ and } k \in \mathbf{Z}\}$. 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), $J(w_i) \cap J(w_j) \neq \emptyset$ 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 $\theta'' = \lfloor m_{ij}/(d + 1) \rfloor$.

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 $s_t = t - [(w_1 + t)/N]$. 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:

$$\begin{aligned} h(w_i) &= [(w_i + s)/N], \quad \forall i \leq t, \\ h(w_i) &= [(w_i + s + r)/N], \quad \forall i > t, \end{aligned} \tag{2.5}$$

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 Δ] evaluate Δ 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 Δ and go to step (d);
(g) [lower bound for δ_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', δ_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 $I_t = \{w_i | 1 \le i \le t\} \cup \{w_i + r | t < i \le n\}$; 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 $w_j - w_i$ ($i \le t < j$). 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 $\delta'_t \ge \delta_0$, 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] $N_t^0 \leftarrow \min \{\lfloor (w_j - w_i - 1)/(j-i-1) \rfloor | (i, j \in [1, t] \text{ or } i, j \in [t+1, n]) \text{ and } j > i+1\}$;
(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] $J_L \leftarrow \bigcap_{i=1}^{z-1} \{(v - w_{i+1}) \bmod N | 0 \le v < \delta_i\}$; $J_R \leftarrow \bigcap_{i=z+1}^{n-1} \{(v - w_{i+1}) \bmod N | 0 \le v < \delta_i\}$;
(i) [lower bound for δ'_z] $\delta_0 \leftarrow \min\{(j-i-1)N + 1 - w_j + w_i + \delta_z | i \le z < j\}$;
(j) [find s', δ'_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) $\delta'_z \leftarrow \min \{\delta \ge \delta_0 | \exists j'' \in J_R \text{ such that } \delta \equiv w_{z+1} + j'' + \hat{p} \text{ (modulo } N)\}$;
(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 $\delta'_5 = \min \{\delta \ge 131 | \exists j'' \in [30, 31] \text{ such that } \delta \equiv 472 + j'' + 61 \text{ (modulo } 72)\}$, 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:

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

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

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.