On the Design of a Machine-Independent Perfect Hashing Scheme

C. C. Chang, C. Y. Chen, and J. K. Jan

Institute of Computer Science and Information Engineering, National Chung Cheng University, Chia-yi, Taiwan 62107, Republic of China
Department of Electronics, Feng Chia University, Taichung, Taiwan 40724, Republic of China
Department of Applied Mathematics, National Chung Hsing University, Taichung, Taiwan 40227, Republic of China

This paper proposes a new approach to be used for assigning keywords to addresses in the way that no two different keywords are assigned in the same address. In this approach, the assigned address of each keyword is determined as following form: address = \( v_i \) (the keyword's \( i \)th character) + \( v_j \) (the keyword's \( j \)th character), where \( v_i \) and \( v_j \) are two integer-valued functions defined on the set of twenty-six English letters. The approach can be used for searching reserved words in compilers, function names in Operating Systems and so on. In considering heuristics to design the \( v_i \) and \( v_j \) functions, we were led to develop a letter-oriented merging-and-exchanging algorithm that finds the letter value assignments for \( v_i \) and \( v_j \) and achieves the reducing blank spaces in the constructed table.

Received February 1988, revised May 1988

1. INTRODUCTION

The business of developing perfect hashing functions is quite intriguing. The main idea of perfect hashing is to construct a black box that will take a key as its input and issue the address in the list where the key is stored without collisions. In other words, a perfect hashing function is a one-to-one mapping from the set of keys into the address space. A more preferable variation of perfect hashing is the minimal perfect hashing which maps a set of keys one-to-one and onto the address space without collisions. Among the above literatures, the methods of Chang and Lee and Cichelli are suitable for letter-oriented keys.

Cichelli proposed a simple method to construct minimal perfect hashing functions for a number of particular key sets where each key is a character string. In this method, the hashing function is defined as \( h(k) = \text{length of } k + v \) (the first character of \( k \)) + \( v \) (the last character of \( k \)), where \( v \) is an integer-value function defined on the set of twenty-six English letters. The merit of this hashing function lies in that it is simple, efficient and machine-independent that means each assigned value of the associated character is rather small. However, there are two main drawbacks: (1) to find the proper values for different characters, Cichelli used an exhaustive search method and (2) Jaeschke and Osterburg had pointed out that Cichelli's method was unable to find a minimal perfect hashing function for many sets of keys. That is Cichelli's hashing function does not guarantee to act upon a set of keys to minimally sized address space.

Chang and Lee proposed another minimal perfect hash scheme which is also suitable for letter-oriented keys. In Chang and Lee's method, each character is assigned three values and the form of their hashing function is defined as \( h(k) = d(k_1) + (C(k_2) \mod p(k_3)) \) where \( k_1 \) and \( k_2 \) are two characters extracted from \( k \). In Chang and Lee's method, they proposed an algorithm to find appropriate \( d(x) \), \( C(x) \) and \( p(x) \) for all possible twenty-six English letters \( x \). Further, they showed that if all the extracted pairs \( (k_1, k_2) \) are distinct then \( h(k) = d(k_1) + (C(k_2) \mod p(k_3)) \) guarantees minimal perfectness.

But their method still suffered from a very severe weakness: the values of \( C(x) \) become drastically large if the number of keys becomes large, that means some values of \( C(x) \) will run out of the integer words of a conventional computer in case that the number of keys is large.

This paper mainly investigates the design of perfect hashing functions for letter-oriented keys. In the next section, we shall propose our scheme.

2. A NEW LETTER-ORIENTED HASHING SCHEME

We assume that we have a set of keys where each key is a string of characters. Besides, we shall assume that we can pick two characters \( k_{11} \) and \( k_{14} \) from each key \( k_i \) such that all extracted pairs \( (k_{11}, k_{14}) \) are distinct. Our new hashing function is of the form

\[
h(k_i) = v_i(k_{11}) + v_4(k_{14}),
\]

(2.1)

where \( v_i \) and \( v_4 \) are two integer-valued functions defined on the set of distinct \( k_{11} \)s and the set of distinct \( k_{14} \)s respectively.

Example 2.1. The set of twelve month identifiers in English are as shown below:

- **JANUARY**
- **FEBRUARY**
- **MARCH**
- **APRIL**
- **MAY**
- **JUNE**
- **JULY**
- **AUGUST**
- **SEPTEMBER**
- **OCTOBER**
- **NOVEMBER**
- **DECEMBER**

In this case, we may pick the second and the third characters of each identifier as the extracted pair. This will produce twelve distinct pairs \( (k_{11}, k_{14}) \)s as shown below:

- \( (A, N) \)
- \( (E, B) \)
- \( (A, R) \)
- \( (P, R) \)
- \( (A, Y) \)
- \( (U, N) \)
- \( (U, L) \)
- \( (E, G) \)
- \( (E, P) \)
- \( (C, T) \)
- \( (O, V) \)
- \( (E, C) \)

THE COMPUTER JOURNAL, VOL. 34, NO. 5, 1991 469
Let us define $v_i(k_{1i})$s and $v_{ij}(k_{ij})$s as in Table 1.

<table>
<thead>
<tr>
<th>$k_{1i}$</th>
<th>A</th>
<th>C</th>
<th>E</th>
<th>O</th>
<th>P</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_i(k_{1i})$</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$k_{ij}$</th>
<th>B</th>
<th>C</th>
<th>G</th>
<th>L</th>
<th>N</th>
<th>P</th>
<th>R</th>
<th>T</th>
<th>V</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{ij}(k_{ij})$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Then all the extracted pairs of the twelve month identifiers are hashed by the hashing function (2.1) as shown in Table 2.

<table>
<thead>
<tr>
<th>Location</th>
<th>Extracted Pair</th>
<th>Original key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(E, B)</td>
<td>FEBRUARY</td>
</tr>
<tr>
<td>2</td>
<td>(E, C)</td>
<td>DECEMBER</td>
</tr>
<tr>
<td>3</td>
<td>(U, G)</td>
<td>AUGUST</td>
</tr>
<tr>
<td>4</td>
<td>(U, L)</td>
<td>JULY</td>
</tr>
<tr>
<td>5</td>
<td>(A, N)</td>
<td>JANUARY</td>
</tr>
<tr>
<td>6</td>
<td>(U, N)</td>
<td>JUNE</td>
</tr>
<tr>
<td>7</td>
<td>(E, P)</td>
<td>SEPTEMBER</td>
</tr>
<tr>
<td>8</td>
<td>(A, R)</td>
<td>MARCH</td>
</tr>
<tr>
<td>9</td>
<td>(P, R)</td>
<td>APRIL</td>
</tr>
<tr>
<td>10</td>
<td>(C, T)</td>
<td>OCTOBER</td>
</tr>
<tr>
<td>11</td>
<td>(A, Y)</td>
<td>MAY</td>
</tr>
<tr>
<td>12</td>
<td>(O, V)</td>
<td>NOVEMBER</td>
</tr>
</tbody>
</table>

For instance, the hashing value of FEBRUARY is calculated as follows:

$h$(FEBRUARY) = $v_1$(E) + $v_4$(B) = 1 + 0 = 1.

Though (2.1) works for the case of twelve month identifiers in the way of bijection. Unfortunately, as we shall show in the following, it doesn’t always guarantee minimal perfectness.

Let

$$K = (\zeta_1, \zeta_2, \zeta_3, \zeta_4, \zeta_5, \zeta_6, \zeta_7, \zeta_8, \zeta_9, \zeta_{10}, \zeta_{11}, \zeta_{12}),$$

where $\zeta_1, \zeta_2$ and $\zeta_3$ are three English letters. Then the only candidates of distinct pairs extracted from keys of $K$ will be $(\zeta_1, \zeta_2), (\zeta_2, \zeta_3), (\zeta_3, \zeta_4)$, $(\zeta_4, \zeta_5), (\zeta_5, \zeta_6)$ and $(\zeta_6, \zeta_7)$. We abbreviate $v_i(\zeta_i)$ by $\alpha_i$, $v_j(\zeta_j)$ by $\beta_j$ and assume that $h(K) \rightarrow (y, y + 1, \ldots, y + 5)$ is bijective for some positive integer $y$. Then $\alpha_i + \beta_j = \alpha_j + \beta_i$ and $\alpha_i + \beta_j = \alpha_j + \beta_i$ for $j \neq k$. Therefore, without loss of generality, we may assume

$$\alpha_1 < \alpha_2 < \alpha_3$$

and

$$\beta_1 < \beta_2 < \beta_3.$$  

And then

$$\alpha_1 + \beta_1 = y \quad \text{and} \quad \alpha_2 + \beta_2 = y + 5$$  

(2.2)

holds.

From (2.3), we have

$$(\alpha_2 - \alpha_1 + \beta_2 - \beta_1) = 5.$$  

This means, from (2.2), that either

$$\alpha_2 = \alpha_1 + 3\quad \text{and} \quad \beta_2 = \beta_1 + 2$$

or

$$\alpha_2 = \alpha_1 + 2\quad \text{and} \quad \beta_2 = \beta_1 + 3.$$  

Case 1: $\alpha_2 = \alpha_1 + 3$ and $\beta_2 = \beta_1 + 2$

In this case, (2.2) implies that $\beta_3 = \beta_1 + 1$. Further, since $\alpha_1 + \beta_2 = \alpha_2 + \beta_3 = \alpha_1 + \alpha_2 + \beta_1$, we have $\alpha_3 = \alpha_1 + 3$. This also implies that $\alpha_3 = \alpha_2 + 2$ and $\alpha_2 + 1$. But then we will have $\alpha_2 + \beta_2 = (\alpha_3 + 1) + (\beta_3 + 2) = (\alpha_3 + 3) + (\beta_1 + 1) = \alpha_3 + \beta_3$. This contradicts to the fact that $h$ is bijective.

Case 2: $\alpha_2 = \alpha_1 + 2$ and $\beta_2 = \beta_1 + 3$

In this case, (2.2) implies that $\alpha_2 = \alpha_1 + 1$. Further, since $\alpha_1 + \beta_2 = \alpha_2 + \beta_3 = \alpha_1 + \alpha_2 + \beta_1$, we have $\beta_3 = \beta_1 + 1$. This also implies that $\beta_3 = \beta_1 + 2$ and $\beta_2 = \beta_1 + 1$. But then we have $\alpha_1 + \beta_2 = \alpha_2 + \beta_3 = \alpha_1 + \beta_1 + 2$. This also contradicts to the fact that $h$ is bijective.

That means $h$ can not be a minimal perfect hashing function on $K$.

Let

$$K_n = (\zeta_1^{n+1}, \zeta_2^{n+1}, \ldots, \zeta_n^{n+1}), \quad n \geq 1.$$  

Then the distinct extracted pairs are the same as the case for $K$. Therefore $h$ can not be bijective on $K_n$, for $n \geq 1$.

3. AN ALGORITHM FOR DEFINING $v_i$ AND $v_{ij}$

In this section, we propose an efficient algorithm for defining function $v_i$ and $v_{ij}$ such that $h(k_{1i}) = v_i(k_{1i}) + v_{ij}(k_{ij})$ is a perfect hashing function. Before formal describing our algorithm, some definitions are needed.

Let $K$ be a set of keys and $EP$ be the set of distinct extracted pairs $(k_{1i}, k_{ij})$. Let $EP_i = \{k_{1i}, k_{2i}, \ldots, k_{mi}\}$ be the set of distinct $k_{1i}$s where $\phi(k_{1i}) < \phi(k_{1(i+1)})$ for $1 \leq i < m - 1$ and $\phi(x)$ is the lexical order of letter $x$ and $EP = (k_{1i}, k_{2i}, \ldots, k_{s})$ be the set of distinct $k_{1i}$s where $\phi(k_{1i}) < \phi(k_{1(i+1)})$ for $1 \leq i \leq n - 1$. Besides, initially, let $K = (k_{1i})$ and $K_0 = (k_{1i})$ for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$.

Definition 3.1. The matrix $E = (e_i)_{m \times n}$ is called the EP-matrix associated with $EP$ if

$$e_i = \begin{cases} 1 & \text{if } (k_{1i}, k_{2i}) \in EP, \\ 0 & \text{otherwise} \end{cases} \quad i = 1, 2, \ldots, m, \quad j = 1, 2, \ldots, n.$$  

Definition 3.2. Let $E_i$ and $E_j$ denote the $i$th row and the $j$th row of the EP-matrix $E$. Then $E_i$ and $E_j$ are said to be row mergeable if each entry of $E_i + E_j$ is either 0 or 1, here ‘+’ is the usual addition operation of vectors in Euclidean space. Meanwhile, the row merge operation of $E_i$ and $E_j$, represented by $M(E_i, E_j)$, is defined as following:

**Procedure** $M(E_i, E_j)$

begin

$E_i \leftarrow E_i + E_j;$

$E_i \leftarrow 0;$

$K_{1j} \leftarrow K_{1j} \cup K_{1i};$

$K_{1i} \leftarrow \Phi.$

end.

Definition 3.3. The row interchange operation of $E_i$ and $E_j$, represented by $I(E_i, E_j)$, is stated as below:

470 THE COMPUTER JOURNAL, VOL. 34, NO. 5, 1991
Procedure \( I(E_i, E_j) \)
begin
\( T \leftarrow E_i; \)
\( E_i \leftarrow E_j; \)
\( E_j \leftarrow T; \)
\( S \leftarrow K_n; \)
\( K_n \leftarrow K_{n+1}; \)
\( K_{n+1} \leftarrow S; \)
end.

Likewise, let \( E_i \) and \( E_j \) denote the \( i \)th and \( j \)th columns of the EP-matrix \( E \). Then the column merge operation \( M(E_i, E_j) \) and column interchange operation \( I(E_i, E_j) \) are defined as following:

Procedure \( M(E_i, E_j) \)
begin
\( E \leftarrow E + E_j; \)
\( E_j \leftarrow 0; \)
\( K_{n+1} \leftarrow K_{n+1} \cup K_{n+2}; \)
\( K_{n+2} \leftarrow \emptyset; \)
end.

Procedure \( I(E_i, E_j) \)
begin
\( T \leftarrow E_i; \)
\( E_i \leftarrow E_j; \)
\( E_j \leftarrow T; \)
\( S \leftarrow K_n; \)
\( K_n \leftarrow K_{n+1}; \)
\( K_{n+1} \leftarrow S; \)
end.

Definition 3.4. The row compress operation on the EP-matrix \( E \) is a sequence of row merge operations, which is operated as below:

Procedure row compression
begin
\( j \leftarrow m; \)
while \( j > 1 \) do
begin
\( i \leftarrow j - 1; \)
while \( i > 0 \) do
begin
if \( E_i \) and \( E_j \) are row mergeable then
begin
\( M(E_i, E_j); \)
for \( l = j \) to \( m - 1 \)
begin
\( E_l \leftarrow E_{l+1}; \)
\( E_{l+1} \leftarrow 0; \)
\( K_{l+1} \leftarrow K_{l+2}; \)
end;
\( i \leftarrow 0; \)
end else
\( i \leftarrow i - 1; \)
end;
\( j \leftarrow j - 1; \)
end.

Similarly, the column compress operation is a sequence of column merge operations, which is operated as below:

Procedure column compression
begin
\( j \leftarrow n; \)
while \( j > 1 \) do
begin
\( i \leftarrow j - 1; \)
while \( i > 0 \) do
begin
if \( E_i \) and \( E_j \) are column mergeable then
begin
\( M(E_i, E_j); \)
for \( l = j \) to \( m - 1 \)
begin
\( E_l \leftarrow E_{l+1}; \)
\( E_{l+1} \leftarrow 0; \)
\( K_{l+1} \leftarrow K_{l+2}; \)
end;
\( i \leftarrow 0; \)
end else
\( i \leftarrow i - 1; \)
end;
if flag > 0 then go to \( l_1 \)
else \( j \leftarrow j - 1; \)
end.
end.

After performing the above compression operations on the EP-matrix \( E \), let \( p = \max(i \lvert E_i \neq 0 \rvert) \) and \( q = \max(j \lvert E_j \neq 0 \rvert) \). We obtain a more compact (or dense) submatrix \( C = (c_{ij})_{p \times q} \) of \( E \) which contains the same number of 1s as \( E \).

Definition 3.5. The \( p \times q \) submatrix \( C \) of \( E \) obtained by performing the row compression procedure first and then the column compression procedure on \( E \) is called the compressed matrix of \( E \).

Definition 3.6. For each column \( C_i \) in the compressed matrix \( C \), let the first and last non-zero entries be occurred in row \( a_i \) and row \( b_i \) respectively. Define the range of \( C_i \) to be \( \text{Rng}(i) = \{a_i, b_i - a_i\} \) and define the total range of \( C \) to be \( \text{Rng}(C) = \sum_{i=1}^{n} \text{Rng}(i) \).

Definition 3.7. (1) A row interchange operation on the compressed matrix \( C \) is called an improved row interchange operation if it decreases the \( \text{Rng} \) value. (2) A minimal range matrix \( L \) associated with \( C \) is a matrix which can be obtained from \( C \) by performing a sequence of improved row interchange operations such that the \( \text{Rng} \) value of \( L \) cannot be decreased any more.

Actually, \( L \) can be obtained from \( C \) through performing the following procedure:

Procedure minimal range
begin
\( j \leftarrow p, \text{flag} \leftarrow 0; \)
while \( j > 1 \) do
begin
\( i \leftarrow j - 1; \)
while \( i > 0 \) do
begin
if \( I(C_i, C_j) \) is improved then
begin
\( I(C_i, C_j); \)
\( \text{flag} \leftarrow 1; \)
end else
\( i \leftarrow i - 1; \)
end;
if \( \text{flag} > 0 \) then go to \( l_1 \)
else \( j \leftarrow j - 1; \)
end.
Suppose we are given a set of keys and extract two characters $k_1$ and $k_4$ from each key $k$, such that all extracted pairs $(k_1, k_4)$'s are distinct. Then the following algorithm will guarantee to produce function $v_1$ and $v_4$ such that (2.1) is a perfect hashing function.

**Algorithm A.** This algorithm is used to design functions $v_1$ and $v_4$.

**Output.** Two integer-valued functions $v_1$ and $v_4$ defined on the set of twenty six English letters such that $h(k) = v_1(k_1) + v_4(k_4)$ is a perfect hashing function.

**Step 1.** Let $EP$ be the set of all extracted pairs. Let $EP_1$ denote the set of distinct $k_1$'s ordered in their lexical ordering and $EP_2$ denote the set of distinct $k_4$'s ordered in their lexical ordering. Determine $EP_1$, $EP_2$, $m = |EP_1|$ and $n = |EP_2|$.

**Step 2.** Let $K_{1i} = (k_{1i})$ and $K_{4j} = (k_{4j})$ for $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$.

**Step 3.** Construct the EP-matrix $E_{m \times n}$ associated with $EP$.

**Step 4.** Find the compressed matrix $C_{m \times n}$ of $E$.

**Step 5.** Find the minimal range matrix $L_{m \times n}$ associated with $C$. List $K_{1i}$ and $K_{4j}$ for $i = 1, 2, ..., p$ and $j = 1, 2, ..., q$.

**Step 6.** For $x \in K_{1i}$, evaluate $v_1(x)$ from $v_1(x) = i - 1$ for $i = 1, 2, ..., p$.

**Step 7.** Suppose the first and the last non-zero entries of column $j$ in $L$ are occurred in row $a_j$ and row $b_j$ respectively. Determine $a_j$ and $b_j$ for $j = 1, 2, ..., q$.

**Step 8.** (1) For $y \in K_{1j}$, evaluate $v_4(y)$ from

$$(a_j - 1) + v_4(y) = 1.$$  

(2) For $y \in K_{4j}$, evaluate $v_4(y)$ from

$$(a_j - 1) + v_4(y) = v_4(y) + b_j, 1,$$

where $z \in K_{(j-1)j}$ and $j = 2, 3, ..., q$.

**Step 9.** Exit.

The time complexity, in terms of $m$ and $n$, of the above algorithm is essentially dominated by Step 4 and Step 5 which require $O(m^3 + n^3)$ operations according to the Procedure row compression and the Procedure column compression and $O(m!) \times m^n$ operations according to the Procedure minimal range, respectively. Thus, the total computing time for the algorithm is therefore $O(m^3 + n^3)$. However, since $m$ and $n$ denote the number of distinct $k_1$'s and $k_4$'s respectively, where $m$ and $n$ are not greater than twenty six, the computing time of our hashing algorithm is, in fact, bounded by a constant.

It should be pointed out that the method required to obtain all the distinct extracted pairs is essentially artificial. The hashing algorithm proposed by Cichelli is and that proposed by Chang and Lee also used extracted pairs. To the best of our knowledge, there still exists no algorithm that will guarantee to produce distinct pairs for any given set of keys. In fact, when the number of keys is large, two letters may not be sufficient to distinguish all keys. For instance, this occurs in the case of COBOL reserved words. There are roughly 500 COBOL reserved words and they do not exist two characters for each word to form 500 distinct extracted pairs. Thus all hashing algorithms mentioned above are suitable for small key sets only. As a matter of fact, the maximum number of distinct extracted pairs is 512 which is $26 \times 26$.

**Example 3.1 (The Twelve Months)**

Let us now illustrate how can Algorithm A be used to design the functions $v_1$ and $v_4$ for the twelve month identifiers in English.

(1) The set $EP$ of extracted pairs has been shown in Example 2.1. Let

$$EP = ((A, N), (E, B), (A, R), (P, R), (A, Y), (U, N), (U, L), (U, G), (E, P), (C, T), (O, V), (E, C)).$$

(2) Determine


$m = |EP_1| = 6$ and $n = |EP_2| = 10$.

Define $K_{1i}$ and $K_{4j}$, $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$, as shown in Table 3.

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{1i}$</td>
<td>(A)</td>
<td>(C)</td>
<td>(E)</td>
<td>(O)</td>
<td>(P)</td>
<td>(U)</td>
</tr>
<tr>
<td>$K_{4j}$</td>
<td>(B)</td>
<td>(C)</td>
<td>(G)</td>
<td>(L)</td>
<td>(N)</td>
<td>(P)</td>
</tr>
</tbody>
</table>

(3) Construct the EP-matrix associated with $EP$ as follows.

$$E = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ \end{bmatrix}$$

(4) The compressed matrix of $E$ is found as

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \end{bmatrix}$$

(5) The minimal range associated with $C$ is

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \end{bmatrix}$$

And, after this step, $K_{1i}$ and $K_{4j}$, $i = 1, 2$ and $j = 1, 2, 3, 4, 5, 6$, are listed in Tables 4 and 5.

(6) Assign $v_1(A) = 0$ and $v_4(C) = v_4(E) = v_4(O) = v_4(P) = v_4(U) = 1$.

(7) The row numbers $a_j$ and $b_j$, where the first and the last non-zero entries occurred in the $j$th column of the matrix $L$, are found and shown in Table 6.
ON THE DESIGN OF A MACHINE-INDEPENDENT PERFECT HASHING SCHEME

Table 5.

<table>
<thead>
<tr>
<th>j</th>
<th>( K_a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(B)</td>
</tr>
<tr>
<td>2</td>
<td>(C)</td>
</tr>
<tr>
<td>3</td>
<td>(G)</td>
</tr>
<tr>
<td>4</td>
<td>(L)</td>
</tr>
<tr>
<td>5</td>
<td>(N)</td>
</tr>
<tr>
<td>6</td>
<td>(P)</td>
</tr>
<tr>
<td>7</td>
<td>(R)</td>
</tr>
<tr>
<td>8</td>
<td>(T)</td>
</tr>
<tr>
<td>9</td>
<td>(V, Y)</td>
</tr>
</tbody>
</table>

Table 6.

<table>
<thead>
<tr>
<th>j</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_j )</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( b_j )</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

(8) From (3.1) and (3.2), we eventually obtain \( v_1(B) = 0 \), \( v_1(C) = 1 \), \( v_1(G) = 2 \), \( v_1(L) = 3 \), \( v_1(N) = 5 \), \( v_1(P) = 6 \), \( v_1(R) = 8 \), \( v_1(T) = 9 \), \( v_1(V) = v_1(Y) = 11 \).

Therefore, the functions \( v_1 \) and \( v_2 \) depicted in Table 1 are then established. Consequently, by using (2.1), the keys will be hashed as shown in Table 2. It should be noted that, in this example, (2.1) is not only a perfect but also a minimal perfect hashing function, i.e. a one-to-one and onto mapping between the set of keys and the address space.

In the following, we shall present more examples to illustrate our hashing scheme.

Example 3.2 (Pascal's Reserved Words)

The 36 reserved words of Pascal are as follows: ARRAY, AND, BEGIN, CASE, CONST, DOWNTO, DO, DIV, END, ELSE, FUNCTION, FILE, FOR, GOTO, IF, IN, LABEL, MOD, NIL, NOT, OTHERWISE, OR, ORIGIN, PROGRAM, PACKED, REPEAT, RECORD, SET, TYPE, THEN, TO, UNTIL, VAR, WITH, WHILE. We choose the first and last letters if the word contains five or fewer letters the first and the fourth letters if the word contains six or seven letters, and the first and the third letters if otherwise. The corresponding extracted pairs are as follows: (A, Y), (A, D), (B, N), (C, E), (C, T), (D, N), (D, O), (D, Y), (E, D), (E, E), (F, N), (F, E), (F, R), (G, O), (I, F), (I, N), (L, L), (M, D), (N, L), (N, T), (O, H), (O, F), (O, R), (P, O), (P, G), (P, K), (R, E), (R, O), (S, T), (T, E), (T, N), (T, O), (U, L), (V, R), (W, H), (W, E).

Table 7 illustrates the hashing function. Consider the word \( \text{CONST} \) which should be hashed to the last location, we have

\[
h(\text{CONST}) = v_1(C) + v_1(T) = 5 + 32 = 37.
\]

In this case, the loading factor is \( l = 36/37 = 97.3\% \).

Example 3.3 (The identifiers of the Fifty States)

This example considers the identifiers of the fifty states of U.S.A. They are ALASKA, ALABAMA, ARIZONA, ARKANSAS, COLORADO, CONNECTICUT, CALIFORNIA, DELAWARE, FLORIDA, GEORGIA, HAWAII, IOWA, INDIANA, IDAHO, ILLINOIS, KENTUCKY, KANSAS, LOUISIANA, MARYLAND, MAINE, MASSACHUSETTS, MICHIGAN, MONTANA, MINNESOTA, MISSISSIPPI, MISSOURI, NEVADA, NORTH CAROLINA, NORTH DAKOTA, NEW HAMPSHIRE, NEW JERSEY, NEW YORK, NEBRASKA, NEW MEXICO, OREGON, OHIO, OKLAHOMA, PENNSYLVANIA, RHODE ISLAND, SOUTH CAROLINA, SOUTH DAKOTA, TENNESSEE, TEXAS, UTAH, VERMONT, VIRGINIA, WEST VIRGINIA, WYOMING, WASHINGTON, WISCONSIN. Let \( k_i \) denote the \( i \)th state name and \( r_i \) denote its length. The corresponding pair \( (k_{11}, k_{12}) \) is extracted according to the following rule:

1. \( r_i = 9 \). We choose

\[
k_{11} = \text{the last letter in } k_i,
\]
\[
k_{12} = \text{the third letter in } k_i.
\]

2. \( r_i > 9 \). We choose

\[
k_{11} = \text{the first letter in } k_i,
\]
\[
k_{12} = \left\lceil \frac{7 - \left\lfloor \frac{r_i}{2} \right\rfloor}{2} \right\rceil \text{th letter in } k_i \text{ if } 5 \leq r_i \leq 15,
\]
\[
\text{the last letter in } k_i \text{ if otherwise}.
\]

The resulting extracted pairs are: (A, S) (A, B), (A, Z), (A, K), (C, L), (C, O), (C, A), (D, L), (F, R), (G, R), (H, A), (I, A), (I, I), (I, O), (I, L), (K, N), (K, S), (A, U), (M, R), (M, E), (M, M), (M, C), (M, T), (A, N), (M, I), (M, S), (N, A), (N, N), (N, O), (N, E), (Y, W), (N, Y), (N, B), (O, W),

Table 7.

| x | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| \( u_1(x) \) | 2 | 1 | 5 | 4 | 4 | 0 | 0 | 0 | 3 | 0 | 0 | 0 | 2 | 1 | 3 | 2 | 1 | 0 | 3 | 1 | 2 | 1 | 1 | 0 | 0 | 0 |
| \( u_2(x) \) | 0 | 0 | 0 | 0 | 5 | 10 | 13 | 14 | 0 | 0 | 10 | 16 | 0 | 20 | 25 | 0 | 0 | 30 | 0 | 32 | 0 | 32 | 0 | 0 | 0 |

Table 8.

| x | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| \( u_1(x) \) | 3 | 0 | 3 | 2 | 1 | 5 | 4 | 2 | 0 | 0 | 0 | 4 | 1 | 4 | 5 | 0 | 2 | 5 | 2 | 5 | 2 | 5 | 0 | 2 | 0 |
| \( u_2(x) \) | 0 | 5 | 6 | 0 | 11 | 0 | 17 | 11 | 20 | 0 | 24 | 30 | 5 | 17 | 34 | 37 | 0 | 24 | 43 | 37 | 37 | 0 | 48 | 0 | 48 | 48 |
O, (G, (O, O), (O, L), (P, P), (R, H), (S, S), (S, O), (E, N), (T, S), (U, H), (V, M), (V, R), (W, W), (W, C), (W, A), (N, S)). The hashing function is illustrated by Table 8. In this case, the word WESTVIRGINIA is hashed into the last location, which is 53. The loading factor is \( f = \frac{50}{53} = 94.3\% \).

4. CONCLUSIONS

In this paper, we have proposed a new simple and machine independent hashing scheme to handle letter-oriented keys. Although our scheme doesn't guarantee a minimal perfect hashing function, the efforts result in reducing the size of the constructed hash tables. We have successfully applied our hashing function to three nontrivial cases, namely, the twelve month identifiers, Pascal's reserved words, and the identifiers of the fifty states of U.S.A., the loading factor obtained are 1, 97.3% and 94.3% respectively. However, the loading factor still depends heavily on the extracted pairs. Besides, if the number of keys is large, two letters may not be sufficient to distinguish all the keys. A suggested solution is to alter the hashing function in the form

\[ h(k_i) = v_1(k_{i1}) + v_2(k_{i2}) + \ldots + v(r) \]

for some \( r \geq 3 \). The problem is how to evaluate the appropriate values for functions \( v \)'s to make the hash table as small as possible. This remains a very interesting problem of inquiry.

Acknowledgements

The authors wish to express their gratitude to a referee for his very useful comments that greatly improved the presentation of this paper.

REFERENCES


Book Review

H. S. M. ZEDAN (editor)
Distributed Computer Systems
Butterworths, Sevenoaks, Kent, 1990

The title of this book is very misleading. Readers of the book will look in vain for topics such as interprocess communication, remote procedure call, distributed filing systems, transactions and replication that should be in a book purporting to be on Distributed Computer Systems. What they will find is a potpourri of topics with very little connection to each other, doubly surprising as the book owes its origin to a course given at Teesside Polytechnic in 1987. However, having said that, are the individual topic contributions valuable?

The layout of the book is in three sections: Surveys, Special Topics and Annotated Bibliography and Key References in Distributed Computer Systems 1959–1989. The key reference part is over 200 pages, and the references are listed year by year and total over 1,600: this is not what a selection of key references should be. The emphasis should be on the selection of a reasonable number of important papers, possibly annotated, and containing guidance to further reading.

The survey section begins with a cursory overview of Distributed Operating Systems, an area which is the linchpin of distributed computer systems. None of the principles is covered, and interesting systems such as Chorus and Emerald are relegated to a few lines under 'other systems'. This is followed by a chapter on Concurrent Programming Languages, but rather than produce any new material the editor has chosen to publish a ten-year-old paper, giving the reason 'that they couldn't write a better one'. Does this mean that there have been no developments in this area over the last ten years? This chapter, which takes up fifty pages, is followed by five pages of references, none of which is more recent than 1981. Surely not what is expected in a state-of-the-art publication? The final part of the survey section, rather than trying to link up the connections between concurrent languages, distributed operating systems and distributed computer systems, covers load-balancing algorithms in loosely coupled distributed systems, an almost entirely theoretical approach which seems to have little justification in practice.

There are five chapters on special topics by authors covering their particular research areas or interests. The chapter on 'debugging distributed real-time applications: a case study in ADA' is well written and interesting. The other chapters in this section are well written, but with the exception of one on network monitoring, are on the periphery of distributed systems and deal with narrowly defined topics. For instance, the chapter on reliable systems in OCCAM looks like an interesting theoretical topic rather than something to be included in a general book on distributed systems, and the chapter on a distributed model of computation for combinatorial code would be more suitable for inclusion in a book on novel parallel architectures. The last chapter in this section on a simple high-level distributed language for teaching is of marginal interest and presents no new ideas.

This book cannot be recommended to anyone looking for a text on Distributed Computer Systems, especially as there are now a number of excellent books on this subject, for instance: Distributed Systems, edited by Sape Mullender, published by ACM Press, which contains contributions from recognised experts in the field. Nor do I think it would appeal to those interested in current programming languages. This is disappointing, as some of the contributions are good in the area they cover. Regrettably the book is less than the sum of its parts.

D. SHEPERD