Symbolic Gray Code as a Perfect Multiattribute Hashing Scheme for Partial Match Queries

C. C. CHANG, R. C. T. LEE, MEMBER, IEEE, AND M. W. DU, MEMBER, IEEE

Abstract—In this paper, we shall show that the symbolic Gray code hashing mechanism is not only good for best matching, but also good for partial match queries. Essentially, we shall propose a new hashing scheme, called bucket-oriented symbolic Gray code, which can be used to produce any arbitrary Cartesian product file, which has been shown to be good for partial match queries. Many interesting properties of this new multiattribute hashing scheme, including the property that it is a perfect hashing scheme, have been discussed and proved.

Index Terms—Bucket-oriented symbolic Gray code, Cartesian product file, multiattribute file organization, partial match query, perfect hashing, symbolic Gray code.

I. The Partial Matching Problem

In this paper, we are concerned with partial match query systems [1], [3], [5], [6], [10], [18]-[21], [23]. We assume that we are dealing with a multiattribute file consisting of a set of multiattribute records. Each record is characterized by attributes $A_1, A_2, \cdots, A_N$. A partial match query is a query of the following form: retrieve all records where $A_{i_1} = a_{i_1}, \cdots, A_{i_k} = a_{i_k}$ where $0 < k < N$.

We shall assume that all of the records are divided into buckets and stored in disks. Each time a partial match query is processed, one or more disk accesses are performed. Since the disk accessing is much more time-consuming than any other processing in the internal main memory, we shall measure the performance of our file system by the number of buckets necessary to be examined.

Let us consider Tables I and II. In both tables, a query $(A_1 = a, A_2 = *)$ denotes a partial match query which retrieves all of the records with A_1 equal to a and A_2 can be any value, i.e., a don't care condition. It can be seen that the average number of buckets to be examined, over all possible partial match queries, is 2 for the file system in Table II and 4 for file system in Table I.

Thus, our multiattribute file system design problem for partial match queries can be stated as follows: given a set of multiattribute records, the problem is to arrange the records in such a way that the average number of buckets to be examined, over all possible partial match queries, is minimized.

Unfortunately, a solution to the above stated problem is still at large. In other words, given an arbitrary set of multiattribute records, there is no efficient algorithm to find an optimal arrangement of records into buckets. However, if all records are present, under certain conditions, it is possible to have an optimal solution.

We shall elaborate this in the following section.

TABLE I
(A) An Arrangement of 16 Records into Four Buckets.
(B) Buckets to be Examined for All Possible Queries for (A)

(A)	Bucket 1	Bucket 2	Bucket 3	Bucket 4
	(a,a)	(a,b)	(a,c)	(a,d)
	(b,b)	(b,c)	(b,d)	(b,a)
	(c,c)	(c,d)	(c,a)	(c,b)
	(d,d)	(d,a)	(d,b)	(d,c)

(B) Queries	Buckets to be examined
(a,*)	1,2,3,4
(b,*)	1,2,3,4
(c,*)	1,2,3,4
(d,*)	1,2,3,4
(*,a)	1,2,3,4
(*,b)	1,2,3,4
(*,c)	1,2,3,4
(*,d)	1,2,3,4

TABLE II
(A) Another Arrangement of 16 Records into Four Buckets.
(B) Buckets to be Examined for All Possible Queries for (A)

(A)	Bucket 1	Bucket 2	Bucket 3	Bucket 4
	(a,a)	(a,c)	(c,a)	(c,c)
	(a,b)	(a,d)	(c,b)	(c,d)
	(b,a)	(b,c)	(d,a)	(d,c)
	(b,b)	(b,d)	(d,b)	(d,d)

(B) Queries	Buckets to be examined
(a,*)	1,2
(b,*)	1,2
(c,*)	3,4
(d,*)	3,4
(*,a)	1,3
(*,b)	1,3
(*,c)	2,4
(*,d)	2,4

II. The Cartesian Product File Concept

Before presenting the Cartesian product file concept, let us assume that each record is characterized by N attributes $A_1, A_2, \cdots, A_N$. Each A_i is associated with a set D_i, which is the domain of A_i. The size of domain D_i is denoted as q_i. The domain of the file F, consisting of all possible records, is

Manuscript received April 24, 1981; revised July 8, 1981.

C. C. Chang and M. W. Du are with the Institute of Computer Engineering, National Chiao-Tung University, Hsinchu, Taiwan.

R.C.T. Lee is with the Institute of Computer and Decision Sciences, National Tsing Hua University, Hsinchu, Taiwan.

0098-5589/82/0500-0235$00.75

thus $D_1 \times D_2 \times \cdots \times D_N$. The total number of records, or the size of F, denoted as NR, is $q_1 q_2 \cdots q_N$. We shall assume that the entire file is divided into NB buckets: $B_1, B_2, \cdots, B_{NB}$.

Definition: A file system is called a Cartesian product file if all records in every bucket are in the form of $D_{1s_1}, D_{2s_2}, \cdots, D_{Ns_N}$ where D_{js_j} is a subset of D_j. This bucket is denoted as $[s_1, s_2, \cdots, s_N]$.

Example 2.1: Let $D_1 = \{a, b, c, d\} = D_2$. Let $D_{11} = D_{21} = \{a, b\}$ and $D_{12} = D_{22} = \{c, d\}$. Then the following file system is a Cartesian product file:

Bucket [1, 1] $= D_{11} \times D_{21} = \{(a, a), (a, b), (b, a), (b, b)\}$
Bucket [1, 2] $= D_{11} \times D_{22} = \{(a, c), (a, d), (b, c), (b, d)\}$
Bucket [2, 2] $= D_{12} \times D_{22} = \{(c, c), (c, d), (d, c), (d, d)\}$
Bucket [2, 1] $= D_{12} \times D_{21} = \{(c, a), (c, b), (d, a), (d, b)\}$.

The reader should note that the above file system is exactly the same as that shown in Table II. This is not accidental. As first pointed out by Lin *et al.* [19], many good file systems, such as those designed by Rivest [21], Rothnie and Lozano [23], as well as Liou and Yao [20], are all Cartesian product files. Aho and Ullman [1] explored the problem of designing optimal multiattribute file systems whose probabilities of an attribute being specified are not equal. This file system is also a Cartesian product file. In [6], more properties of Cartesian product files were discussed.

The physical meaning of a Cartesian product file discussed in Example 2.1 can be visualized by considering Fig. 1 where each dot represents a record. In Fig. 1, each bucket corresponds to a square. In this case, it is easy to see that this Cartesian product file divides records into natural clusters. If we want to retrieve all records with A_1 equal to a, only two buckets (Bucket [1, 1] and Bucket [1, 2]) have to be accessed. Similarly, if we want to retrieve all records with $A_2 = b$, again, only two buckets (Bucket [1, 1] and Bucket [2, 1]) have to be retrieved.

If we do not use the Cartesian product file concept and instead we use the file system shown in Table I, the reader can verify for himself that within each bucket, records are not similar to one another at all, as shown in Fig. 2.

In [19], it was pointed out that good multiattribute file systems all exhibit some kind of clustering property. That is, within each bucket, records should be similar to one another. It just happens that Cartesian product files do cluster similar records together.

Example 2.2: Cartesian product files do not necessarily group records into squares, as shown in Fig. 2. For the case in Example 2.1, we may also have the following Cartesian product file:

$D_{11} = \{a\}$, $D_{12} = \{b\}$, $D_{13} = \{c\}$, $D_{14} = \{d\}$.
$D_{21} = D_2 = \{a, b, c, d\}$.
Bucket [1, 1] $= \{(a, a), (a, b), (a, c), (a, d)\}$
Bucket [2, 1] $= \{(b, a), (b, b), (b, c), (b, d)\}$
Bucket [3, 1] $= \{(c, a), (c, b), (c, c), (c, d)\}$
Bucket [4, 1] $= \{(d, a), (d, b), (d, c), (d, d)\}$.

The above file system is shown in Fig. 3, where each long strip corresponds to a bucket. This file system performs very well if the user specifies A_1 and very poorly if the user specifies A_2.

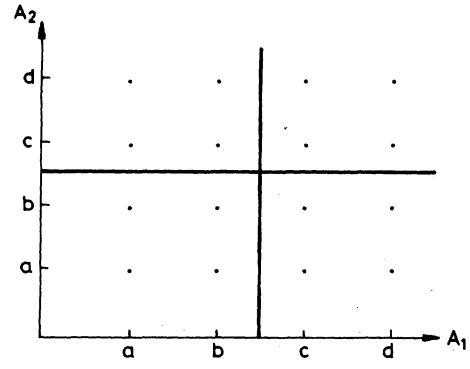


Fig. 1. Simple geometry representation of Table II(A).

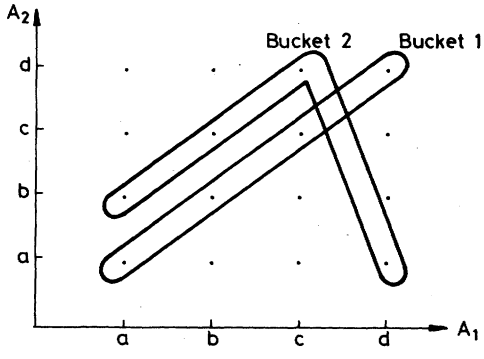


Fig. 2. Geometry representation showing that records not similar to one another are within each bucket.

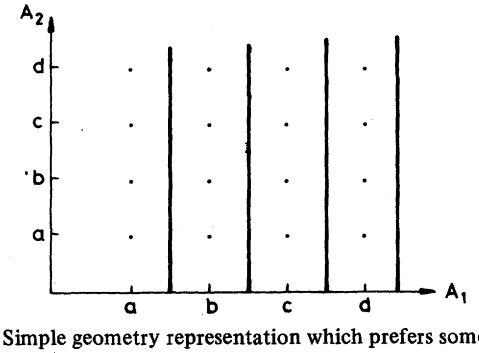


Fig. 3. Simple geometry representation which prefers some attribute.

Because of the clustering properties of Cartesian product files, they are also useful for nearest neighbor searching [2], [4], [9], [14], [17], [22], [24]. Du and Sobolewski [10] used the Cartesian product files for parallel processing in multiple disk systems.

Given two records $R_1 = (r_{11}, r_{12}, \cdots, r_{1N})$ and $R_2 = (r_{21}, r_{22}, \cdots, r_{2N})$, the Hamming distance between R_1 and R_2 is defined as follows:

$$d(R_1, R_2) = \sum_{i=1}^{N} \delta(r_{1i}, r_{2i})$$

where

$$\delta(r_{1i}, r_{2i}) = 1 \quad \text{if } r_{1i} \neq r_{2i}$$
$$= 0 \quad \text{if } r_{1i} = r_{2i}.$$

One of the most important properties of Cartesian product files is that it is possible to arrange records in a Cartesian product file such that the Hamming distance [19] between

TABLE III
USING INDEX PAIRS TO DENOTE EACH BUCKET NUMBER OF TABLE II(A)

Bucket [1,1]	(a,a)
	(a,b)
	(b,b)
	(b,a)
Bucket [1,2]	(b,c)
	(b,d)
	(a,d)
	(a,c)
Bucket [2,2]	(c,c)
	(c,d)
	(d,d)
	(d,c)
Bucket [2,1]	(d,a)
	(d,b)
	(c,b)
	(c,a)

TABLE IV
USING INDEX PAIRS TO DENOTE EACH BUCKET NUMBER

Bucket [1,1]	(a,a)
	(a,b)
	(a,c)
	(a,d)
Bucket [2,1]	(b,d)
	(b,c)
	(b,b)
	(b,a)
Bucket [3,1]	(c,a)
	(c,b)
	(c,c)
	(c,d)
Bucket [4,1]	(d,d)
	(d,c)
	(d,b)
	(d,a)

two consecutive records in the file is equal to one. For example, consider the two files in Examples 2.1 and 2.2, respectively. They can be displayed in Tables III and IV. The reader may verify that in both files, for any pair of consecutive records, the Hamming distance between them is equal to one. Thus Cartesian product files exhibit the consecutive retrieval property advocated by Ghosh [16].

Note that the Cartesian product file concept is only a method to organize records physically. We still need an indexing scheme to locate the records. Since this indexing scheme occupies memory space, it will be desirable to eliminate it. This can only be achieved by using some kind of multiattribute hashing scheme [9], [23] which maps a record directly to its address space without the help of any indexing file.

In this paper, we shall show that we have a multiattribute hashing method for Cartesian product files. That is, for every Cartesian product file, we can easily design a multiattribute hashing which produces such a file. This hashing method has the property of being a minimum perfect hashing method [8], [11], [25] in the sense that no collision occurs and no memory space is wasted. Our hashing scheme is based upon symbolic Gray code [9] which will be discussed in the next section.

TABLE V
USING SYMBOLIC GRAY CODE TO HASH ALL OF THE POSSIBLE RECORDS OF EXAMPLE 3.1

Records	Location
(a,a)	1
(a,b)	2
(a,c)	3
(a,d)	4
(b,d)	5
(b,c)	6
(b,b)	7
(b,a)	8
(c,a)	9
(c,b)	10
(c,c)	11
(c,d)	12
(d,d)	13
(d,c)	14
(d,b)	15
(d,a)	16

III. SYMBOLIC GRAY CODE

The symbolic Gray code concept was proposed by Du and Lee [9]. While the exact algorithm of this hashing function is rather complicated, its meaning is easy to understand. Consider the following example.

Example 3.1: Let us assume that $D_1 = D_2 = \{a, b, c, d\}$. The symbolic Gray code will always hash all of the possible records as shown in Table V.

It was shown in [9] that the symbolic Gray code has the following interesting properties.

Property 1–Address to Key Transformation Property: All hashing functions provide a key to address transformation. But symbolic Gray code hashing also provides the address to key transformation. That is, given an address in the address space, we can calculate the record stored in that location. For instance, for location equal to 5, we can easily show that the record stored there is (b, a).

Property 2–The One-to-One Correspondence: Let us denote KAT and AKT as the key to address transformation and the address to key transformation, respectively. Property 2 means that if $\mathrm{KAT}(R) = i$, then $\mathrm{AKT}(i) = R$.

Property 3–No Collision in the Table: This is a consequence of Property 2.

Property 4–No Waste of Memory Space: This is a consequence of Properties 1 and 2.

Property 5–The Nearest Neighbor Property: If symbolic Gray code is used, the Hamming distance between every pair of two consecutive records in the resulting file is always equal to one. This means that they are nearest neighbors to each other.

Property 6–The Multiattribute Tree Property: For a detailed discussion of this property, consult [9].

From the above properties, one can easily see that the symbolic Gray code hashing is a minimal perfect hashing [25]

(perfect means no collision and minimal means no waste of memory space).

In spite of the above desirable properties, the symbolic Gray code nevertheless suffers from one disadvantage—it is not good for partial match queries. Consider Table V. Let us assume that every four records are stored in one bucket. Then, if our query specifies the first attribute, only one bucket has to be examined. On the other hand, if our query specifies the second attribute, all buckets have to be examined. We may say that the consecutive retrieval properties among the attributes are not balanced.

If there are three attributes, this imbalance is even more pronounced. A typical file produced by symbolic Gray code involving three attributes is now shown in Table VI.

Note that the symbolic Gray code does produce Cartesian product files. The unfortunate thing is that it cannot be used to produce a Cartesian product file specified by us. For instance, it cannot produce the file shown in Table III.

In this paper, we shall propose a new symbolic Gray code, called bucket-oriented symbolic Gray code as a multiattribute hashing scheme to produce any Cartesian product file that we want, in particular, a Cartesian product file suitable for partial match queries.

IV. Bucket-Oriented Symbolic Gray Code

We indicated before that symbolic Gray code always produces a special kind of Cartesian product file. This can be modified. Note that for a Cartesian product file, each bucket is associated with an index and the index itself can be considered as a record. For instance, in Example 2.1, the indexes associated with the four buckets are

(1, 1)
(2, 2)
(2, 1)
(1, 2).

If we consider the above 2-tuples as multiattribute records, we can use symbolic Gray code to order them into the following sequence:

(1, 1)
(1, 2)
(2, 2)
(2, 1).

For the first bucket, there are four records:

(a, a)
(b, b)
(a, b)
(b, a).

We can again use symbolic Gray code to order them into the following sequence:

(a, a)
(a, b)
(b, b)
(b, a).

TABLE VI
A Three-Attribute File Produced by Symbolic Gray Code

Record $R_L(A_{L1}, A_{L2}, A_{L3})$	Address L
(A_{11}, A_{21}, A_{31})	1
(A_{11}, A_{21}, A_{32})	2
(A_{11}, A_{21}, A_{33})	3
(A_{11}, A_{22}, A_{33})	4
(A_{11}, A_{22}, A_{32})	5
(A_{11}, A_{22}, A_{31})	6
(A_{12}, A_{22}, A_{31})	7
(A_{12}, A_{22}, A_{32})	8
(A_{12}, A_{22}, A_{33})	9
(A_{12}, A_{21}, A_{33})	10
(A_{12}, A_{21}, A_{32})	11
(A_{12}, A_{21}, A_{31})	12
(A_{13}, A_{21}, A_{31})	13
(A_{13}, A_{21}, A_{32})	14
(A_{13}, A_{21}, A_{33})	15
(A_{13}, A_{22}, A_{33})	16
(A_{13}, A_{22}, A_{32})	17
(A_{13}, A_{22}, A_{31})	18

Now consider the second bucket: Bucket [1, 2]. The four records are as follows:

(a, c)
(a, d)
(b, c)
(b, d).

We can use symbolic Gray code to order them into the following sequences:

(a, c)
(a, d)
(b, d)
(b, c).

For reasons that will become obvious later, we may reverse the above ordering without affecting the consecutive retrieve property. The reversed order will be

(b, c)
(b, d)
(a, d)
(a, c).

The above process can be applied to each bucket and finally we shall obtain the file in Table VII.

The reader can now see that we can use the symbolic Gray code to obtain a Cartesian product file specified by us. Since it is not the symbolic Gray code as originally proposed by Du and Lee [9], we shall call the new code bucket-oriented symbolic Gray code.

TABLE VII
USING INDEX PAIRS TO DENOTE EACH BUCKET NUMBER OF TABLE II(A)

Bucket [1,1]	(a,a) (a,b) (b,b) (b,a)
Bucket [1,2]	(b,c) (b,d) (a,d) (a,c)
Bucket [2,2]	(c,c) (c,d) (d,d) (d,c)
Bucket [2,1]	(d,b) (d,a) (c,a) (c,b)

V. BUCKET-ORIENTED SYMBOLIC GRAY CODING AS A MULTIATTRIBUTE HASHING FUNCTION

In the previous section, we gave an outline about how symbolic Gray code can be used as a hashing to create any arbitrary Cartesian product file. In this section, we show the exact algorithm.

We are given a set of records characterized by N attributes $A_1, A_2, \cdots, A_N$. The domain of A_i is denoted as D_i. The domain size of A_i is q_i. Each record is denoted as $R = (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$ where $1 \leqslant b_i \leqslant q_i$. Each D_i is divided into t_i subdomains and the size of each subdomain is denoted as z_i. Each bucket is in the form of

$$D_{1s_1}, D_{2s_2}, \cdots, D_{Ns_N}$$

where D_{is_i} is a subdomain of D_i. This bucket will be denoted as

$$\text{Bucket } [s_1, s_2, \cdots, s_N].$$

Our algorithm to hash a record $R = (A_{1i_1}, A_{2i_2}, \cdots, A_{Ni_N})$ to an address consists of two main steps. In the first step, we determine the order of the bucket which will contain this record. In the second step, the exact location of this record inside the bucket is determined.

Given a record $R = (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$, the bucket $[s_1, s_2, \cdots, s_N]$ which will contain this record is determined by the following formula:

$$s_i = \left\lceil \frac{b_i}{z_i} \right\rceil$$

for $1 \leqslant i \leqslant N$ where z_i is the size of a subdomain of D_i and $\lceil x \rceil$ is the smallest integer greater than or equal to x.

The order of $[s_1, s_2, \cdots, s_N]$ is determined through Algorithm A which is essentially the key to address transformation algorithm of the symbolic Gray code discussed in [9].

Algorithm A: The Algorithm which Determines the Order of a Bucket $[s_1, s_2, \cdots, s_N]$

Input:

$[s_1, s_2, \cdots, s_N]$
$[t_1, t_2, \cdots, t_N]$

where t_i is the number of subdomains of the domain of attribute A_i.

Output: The order of the bucket $[s_1, s_2, \cdots, s_N]$.

Step 1: Determine an N-tuple $(a_1, a_2, \cdots, a_N)$ through the following rules.

a) For $i = 1$, let $a_i = s_i - 1$. That is, $a_1 = s_1 - 1$.

b) For $1 < i \leqslant N$, let

$$L_2 = a_1 + 1$$
$$L_3 = a_1 t_2 + a_2 + 1$$
$$\vdots$$
$$L_i = a_1(t_2 t_3 \cdots t_{i-1}) + a_2(t_3 t_4 \cdots t_{i-1}) + \cdots + a_{i-1} + 1$$
$$\vdots$$
$$L_N = a_1(t_2 t_3 \cdots t_{N-1}) + a_2(t_3 t_4 \cdots t_{N-1}) + \cdots + a_{N-1} + 1;$$

if L_i is odd, $a_i = s_i - 1$;

if L_i is even, $a_i = t_i - s_i$.

Step 2: The order of Bucket $[s_1, s_2, \cdots, s_N]$ is now calculated as follows:

$$P = a_1(t_2 \cdots t_N) + a_2(t_3 t_4 \cdots t_N) + \cdots + a_{N-1} t_N + a_N + 1.$$

Example 5.1: Consider Example 2.1 again. In this case we have four buckets:

Bucket [1, 1]
Bucket [1, 2]
Bucket [2, 2]

and

Bucket [2, 1]

Let us now determine the order of Bucket [1, 1]. Since

$$s_1 = s_2 = 1 \quad \text{and} \quad t_1 = t_2 = 2$$

we have

$$a_1 = s_1 - 1 = 1 - 1 = 0$$
$$L_2 = a_1 + 1 = 0 + 1 = 1 \quad \text{is odd.}$$

Therefore,

$$a_2 = s_2 - 1 = 1 - 1 = 0$$
$$(a_1, a_2) = (0, 0)$$
$$P = a_1 t_2 + a_2 + 1 = 0 \times 2 + 0 + 1 = 1.$$

The order of Bucket [1, 1] is 1. For Bucket [2, 1], we have

$$s_1 = 2$$

and

$$s_2 = 1$$

$$a_1 = s_1 - 1 = 2 - 1 = 1$$

$$L_2 = a_1 + 1 = 1 + 1 = 2 \quad \text{is even.}$$

Therefore,

$$a_2 = t_2 - s_2 = 2 - 1 = 1$$

$$(a_1, a_2) = (1, 1)$$

$$P = a_1 t_2 + a_2 + 1 = 1 \times 2 + 1 + 1 = 4.$$

The order of Bucket [2, 1] is 4.

Having determined the order of the bucket which will contain the record, we can determine the relative address of the record inside the bucket. Again, we apply the symbolic Gray code to all of the records inside the bucket. If the order of the bucket is even, the ordering is reversed. The following algorithm determines the relative address of a record inside the bucket containing it. After determining both the order of the bucket and the relative address, the absolute address of the record can be easily determined.

Algorithm B: The Algorithm which Calculates the Absolute Address of a Record

Input: Record

$$R = (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$$

$$(z_1, z_2, \cdots, z_N)$$

$$(t_1, t_2, \cdots, t_N)$$

where z_i is the size of each subdomain of D_i and t_i is the number of subdomains of D_i.

Output: The absolute address of R.

Step 1: For $i = 1$ to N,

$$s_i = \left\lceil \frac{b_i}{z_i} \right\rceil.$$

Step 2: For $i = 1$ to N,

$$b_i' = b_i - (s_i - 1) \times z_i.$$

Step 3:

a) For $i = 1$, let $a_i = b_i' - 1$. That is, $a_1 = b_1' - 1$

b) For $1 < i \leqslant N$, let

$$L_2 = a_1 + 1$$

$$L_3 = a_1 z_2 + a_2 + 1$$

$$\vdots$$

$$L_i = a_1(z_2 z_3 \cdots z_{i-1}) + a_2(z_3 z_4 \cdots z_{i-1}) + \cdots + a_{i-1} + 1$$

$$\vdots$$

$$L_N = a_1(z_2 z_3 \cdots z_{N-1}) + a_2(z_3 z_4 \cdots z_{N-1}) + \cdots + a_{N-1} + 1;$$

if L_i is odd, $a_i = b_i' - 1$;

if L_i is even, $a_i = z_i - b_i'$

c) $m = a_1(z_2 z_3 \cdots z_N) + a_2(z_3 z_4 \cdots z_N) + \cdots + a_{N-1} z_N + a_N + 1$.

Step 4: Apply Algorithm A to $(s_1, s_2, \cdots, s_N)$ and $(t_1, t_2, \cdots, t_N)$ to determine the order P of Bucket $[s_1, s_2, \cdots, s_N]$.

Step 5: If P is even, $m' = z_1 z_2 \cdots z_N - m + 1$. If P is odd, $m' = m$. (Note that m' is the relative address of R inside Bucket $[s_1, s_2, \cdots, s_N]$ which will contain R.)

Step 6: The absolute address L of R is determined by the following formula:

$$L = (z_1 z_2 \cdots z_N) \times (P - 1) + m'.$$

The above procedure is called the key to address transformation (KAT) of the bucket-oriented symbolic Gray code.

Example 5.2: Consider Example 5.1 again; in this case we have four buckets:

Bucket [1, 1]
Bucket [1, 2]
Bucket [2, 2]

and

Bucket [2, 1]

Let us now determine the absolute address of some record in this file system.

Case 1: For the record $R = (a, c) = (A_{11}, A_{23})$, $(b_1, b_2) = (1, 3)$.

After applying Algorithm B (or AKT), since $z_1 = z_2 = 2$ and $t_1 = t_2 = 2$, we have

$$s_1 = \left\lceil \frac{b_1}{z_1} \right\rceil = \left\lceil \frac{1}{2} \right\rceil = 1$$

and

$$s_2 = \left\lceil \frac{b_2}{z_2} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2.$$

So we know that the record will be contained in Bucket [1, 2].

Next, we have

$$b_1' = b_1 - (s_1 - 1) \times z_1 = 1 - (1 - 1) \times 2 = 1$$

and

$$b_2' = b_2 - (s_2 - 1) \times z_2 = 3 - (2 - 1) \times 2 = 3 - 2 = 1.$$

From Step 3, we have

$$a_1 = b_1' - 1 = 1 - 1 = 0$$

$$L_2 = a_1 + 1 = 1 \quad \text{is odd.}$$

Therefore,

$$a_2 = b_2' - 1 = 1 - 1 = 0$$

$$m = a_1 z_2 + a_2 + 1 = 0 \times 2 + 0 + 1 = 1.$$

For Bucket [1, 2], we have $(s_1, s_2) = (1, 2)$.

By applying Step 4 (or Algorithm A) to $(s_1, s_2) = (1, 2)$ and $(t_1, t_2) = (2, 2)$, the order of Bucket [1, 2] is determined to be 2. That is, $P = 2$.

For P is even, the relative address of R inside Bucket [1, 2] is

$$m' = z_1 z_2 - m + 1 = 2 \times 2 - 1 + 1 = 4.$$

Hence, the absolute address L of R is

$$\begin{aligned} L &= z_1 z_2 \times (P - 1) + m' \\ &= 2 \times 2 \times (2 - 1) + 4 \\ &= 4 + 4 = 8. \end{aligned}$$

That is, Record (a, c) is stored at the eighth address after applying this KAT technique.

Case 2: For record $R = (c, d) = (A_{13}, A_{24})$, $(b_1, b_2) = (3, 4)$.

After applying Algorithm B (or KAT), since

$$z_1 = z_2 = 2 \quad \text{and} \quad t_1 = t_2 = 2,$$

we have

$$s_1 = \left\lceil \frac{b_1}{z_1} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2$$

and

$$s_2 = \left\lceil \frac{b_2}{z_2} \right\rceil = \left\lceil \frac{4}{2} \right\rceil = 2.$$

So we know that the record will be contained in Bucket [2, 2]. Next we have

$$b_1' = b_1 - (s_1 - 1) \times z_1 = 3 - (2 - 1) \times 2 = 1$$

and

$$b_2' = b_2 - (s_2 - 1) \times z_2 = 4 - (2 - 1) \times 2 = 2.$$

For Step 3, we have

$$a_1 = b_1' - 1 = 1 - 1 = 0$$

$$L_2 = a_1 + 1 = 0 + 1 = 1 \quad \text{is odd.}$$

Therefore,

$$a_2 = b_2' - 1 = 2 - 1 = 1$$

$$m = a_1 z_2 + a_2 + 1 = 0 \times 2 + 1 + 1 = 2.$$

For Bucket [2, 2], we have $(s_1, s_2) = (2, 2)$.

By applying Step 4 (or Algorithm A) to $(s_1, s_2) = (2, 2)$ and $(t_1, t_2) = (2, 2)$, the order of Bucket [2, 2] is determined to be 3. That is, $P = 3$.

For $P = 3$ is odd, the relative address of R inside Bucket [2, 2] is $m' = m = 2$.

Hence, the absolute address L of R is

$$L = z_1 z_2 \times (P - 1) + m' = 2 \times 2 \times (3 - 1) + 2 = 10.$$

That is, Record (c, d) is stored at the tenth address after applying this KAT technique.

If the reader consults Example 2.1 with Table III, he will discover that the addresses of (a, c) and (c, d) are just the same as those in Table III. In fact, if Algorithm B is used, all records in $D_1 \times D_2$ will be organized as shown in Table III.

Let us now conclude this section by stating the fact again that given an arbitrary Cartesian product file, we can apply the bucket-oriented symbolic Gray code to determine the address of every record in the file. In other words, the bucket-oriented symbolic Gray code can be used as a multiattribute hashing function to produce any arbitrary Cartesian product file.

VI. Some Properties of Using Bucket-Oriented Symbolic Gray Code to Organize any Cartesian Product File

In this section, we shall present some interesting properties of using the bucket-oriented symbolic Gray code to produce Cartesian product file systems.

Property 1–The Address to Key Transformation: While most hashing functions provide "key to address transformation" only, our bucket-oriented symbolic Gray code also provides an "address to key transformation" (AKT) which maps an address to a unique record. Let us note that the total number of possible records of a file $D_1 \times D_2 \times \cdots \times D_N$ is $q_1 q_2 \cdots q_N$, where q_i is the size of D_i. Suppose all of the records are stored in NB buckets. In the following, we shall show that we have an address to key transformation.

Algorithm C: To Convert an Address to its Associated Record

Input:

$(z_1, z_2, \cdots, z_N)$

$(q_1, q_2, \cdots, q_N)$

$L, 1 \leqslant L \leqslant q_1 q_2 \cdots q_N$

where q_i and z_i are the sizes of D_i and each subdomain of D_i, respectively.

Output:

$(A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$

where $R = (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$ is a record which is associated with the address L.

Step 1: Calculate

$$m' = L - \left(\left\lceil \frac{L}{z_1 z_2 \cdots z_N} \right\rceil - 1 \right) \times z_1 z_2 \cdots z_N$$

where m' is the relative address of the record R_L inside the bucket containing it.

Step 2: Calculate

$$P = \left\lceil \frac{L}{z_1 z_2 \cdots z_N} \right\rceil;$$

if P is odd, $m = m'$;

if P is even, $m = z_1 z_2 \cdots z_N - m' + 1$.

P is the order of the bucket containing R_L.

Step 3:

a) Determine an N-tuple $(a_1, a_2, \cdots, a_N)$ through the following equation:

$$P = a_1(t_2 t_3 \cdots t_N) + a_2(t_3 t_4 \cdots t_N) + \cdots + a_{N-1} t_N + a_N + 1$$

where $t_i = q_i/z_i$ and P is obtained in Step 2.

b) For $i = 1$ to N, determine as follows:

$$s_i = a_i + 1, \quad \text{if } \left\lceil \frac{P}{t_i t_{i+1} \cdots t_N} \right\rceil \text{ is odd.}$$

$$s_i = t_i - a_i, \quad \text{if } \left\lceil \frac{P}{t_i t_{i+1} \cdots t_N} \right\rceil \text{ is even.}$$

$[s_1, s_2, \cdots, s_N]$ is the bucket containing R_L.

Step 4:

a) Determine an N-tuple $(a_1, a_2, \cdots, a_N)$ through the following equation:

$$m = a_1(z_2 z_3 \cdots z_N) + a_2(z_3 z_4 \cdots z_N) + \cdots + a_{N-1} z_N + a_N + 1$$

where $a_i's$ are all integers and m is obtained in Step 2.

b) For $i = 1$ to N, determine b_i' as follows:

$$b_i' = a_i + 1, \quad \text{if } \left\lceil \frac{m}{z_i z_{i+1} \cdots z_N} \right\rceil \text{ is odd.}$$

$$b_i' = z_i - a_i, \quad \text{if } \left\lceil \frac{m}{z_i z_{i+1} \cdots z_N} \right\rceil \text{ is even.}$$

Step 5: For $i = 1$ to N,

$$b_i = z_i \times (s_i - 1) + b_i'.$$

Step 6:

$(A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$ is R_L.

The above procedure is called the address to key transformation (AKT) of the bucket-oriented symbolic Gray code.

Now, let us show how the AKT can be applied to the data in Table III.

Example 6.1: Consider the case where $L = 10$, $q_1 = q_2 = 4$ and $z_1 = z_2 = 2$.

Step 1:

$$m' = 10 - \left(\left\lceil \frac{10}{2 \times 2} \right\rceil - 1 \right) \times 2 \times 2 = 10 - 8 = 2.$$

Step 2:

$$P = \left\lceil \frac{10}{2 \times 2} \right\rceil = 3.$$

$m' = m = 2$ because P is odd.

Step 3:

a) $t_1 = q_1/z_1 = 2$ and $t_2 = q_2/z_2 = 2$; we have

$$3 = a_1 t_2 + a_2 + 1 = 2a_1 + a_2 + 1$$

This gives $(a_1, a_2) = (1, 0)$.

b) Because

$$\left\lceil \frac{P}{t_1 t_2} \right\rceil = \left\lceil \frac{3}{2 \times 2} \right\rceil = 1 \text{ is odd, } s_1 = a_1 + 1 = 1 + 1 = 2.$$

Because

$$\left\lceil \frac{P}{t_2} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2 \text{ is even, } s_2 = t_2 - a_2 = 2 - 0 = 2.$$

$[2, 2]$ is the bucket in which R_{10} is stored.

Step 4:

a) $m = a_1 z_2 + a_2 + 1$.

We have

$$2 = a_1 \cdot 2 + a_2 + 1$$

This gives $(a_1, a_2) = (0, 1)$.

b) Because

$$\left\lceil \frac{m}{z_1 z_2} \right\rceil = \left\lceil \frac{2}{2 \times 2} \right\rceil = 1 \text{ is odd, } b_1' = a_1 + 1 = 0 + 1 = 1.$$

Because

$$\left\lceil \frac{m}{z_2} \right\rceil = \left\lceil \frac{2}{2} \right\rceil = 1 \text{ is odd, } b_2' = a_2 + 1 = 1 + 1 = 2.$$

Step 5:

$$b_1 = z_1 \cdot (s_1 - 1) + b_1' = 2 \cdot (2 - 1) + 1 = 3$$

$$b_2 = z_2 \cdot (s_2 - 1) + b_2' = 2 \cdot (2 - 1) + 2 = 4.$$

Therefore, we have $R_{10} = (A_{13}, A_{24})$.

Property 2–The One-to-One Correspondence Property: We have shown the key to address transformation (KAT) mechanism of the bucket-oriented symbolic Gray code hashing function. We have also shown the address to key transformation (AKT) in the hashing function. We now ask: What is the relationship between these two transformations?

The following theorem depicts that there is a one to one correspondence relationship between the addresses and records.

Theorem 6.1: For every record $R \in D_1 \times D_2 \times \cdots \times D_N$, if $\mathrm{KAT}(R) = L$, then $\mathrm{AKT}(L) = R$. (The proof of this theorem can be found in Appendix A.)

The reader can verify this point by checking into Table III. If he applies the AKT to any address $1 \leqslant L \leqslant 16$, he will obtain the record stored in that location and if he applies the KAT to the record already stored there, he will obtain exactly the same address.

Property 3–No Collision in the Hash Table: For most hashing functions, there will be collisions in the hash table because two distinct records may be hashed into the same address. From Property 2, we know that if R_1 is different from R_2, the address of R_1 will be different from that of R_2. Thus there will be no collisions in the hash table. We may say that the bucket-oriented symbolic Gray code can be considered as a perfect hash function [8], [11], [25].

Property 4–No Waste of Memory Space: For most hashing functions, if we know that the total number of records to be stored is M and some kind of hashing function is used, we usually must reserve more than M locations. It is not the case when this hashing function is used. Because of Property 2 and Property 1, we only have to reserve exactly M locations. Thus, the bucket-oriented symbolic Gray code is a minimal perfect hash function.

Property 5–The Nearest Neighbor Property: For $1 \leqslant L < NR = q_1 q_2 \cdots q_N$, the Hamming distance between the record R_L stored at location L and the record R_{L+1} stored at location $L+1$ is always 1. Because of this special property, every pair of consecutive records in the hash table are nearest neighbors to each other. This is a very desirable property for organizing records for a best match searching system [2], [4], [9], [12]-[14], [22].

Theorem 6.2: Let there be N sets: $D_1, D_2, \cdots, D_N$ where $D_i = \{A_{i1}, A_{i2}, \cdots, A_{iq_i}\}$ and $q_i > 1$. Let R_L denote the record associated with L by applying Algorithm B (KAT) to L. Let NR denote the total number of records in $D_1 \times D_2 \times \cdots \times D_N$. Then the Hamming distance between R_i and R_{i+1} is 1, for $1 \leqslant i < NR$. (The proof of this theorem can be found in Appendix B.)

Property 6–Appropriate for Partial Match Searching: It was shown in [6] that Cartesian product files were suitable for partial match searching. Since any Cartesian product file can be produced by using the bucket-oriented symbolic Gray code, we shall say that this hashing scheme is good for partial match searching.

Property 7–For any Partial Match Query, It is Easy to Determine All Buckets Necessary to Be Examined: Assume a partial match query is of the following form. Retrieve all records where $A_{i_1} = A_{i_1 b_{i_1}}$, $A_{i_2} = A_{i_2 b_{i_2}}, \cdots, A_{i_j} = A_{i_j b_{i_j}}$ and $i_1 \neq i_2 \neq \cdots \neq i_j$. Assume each $A_{i_k b_{i_k}}$ is in $D_{i_k s_{i_k}}$, $1 \leqslant k \leqslant j$. The buckets we have to examine are $[s_1, s_2, \cdots, s_N]$'s, where s_k is any value ranged from 1 to t_k, (t_k is the number of subdomains of the domain of attribute A_k) if $k \neq i_p$, $1 \leqslant p \leqslant j$ and $s_k = s_{i_p}$, if otherwise.

For instance, in Table III, consider the query $(A_1 = c, A_2 = *)$. That is, the query is $(A_{13}, *)$. Since A_{13} is in D_{12}, $s_1 = 2$ and s_2 can be from 1 to 2. So there are two buckets [2, 1] and [2, 2] to be examined. By applying Algorithm A to these two buckets, we have the order of these buckets being 4 and 3, respectively. Hence we can conclude that bucket 3 and bucket 4 must be examined for the query $(A_1 = c, A_2 = *)$.

Property 8–The Multiattribute Tree Property: Assume that we have a sequence of buckets, $BK_1, BK_2, \cdots$ and BK_{NB} produced by the bucket-oriented symbolic Gray code. This sequence of buckets can be viewed as a tree whose structure is explained as follows.

1) The top node of the tree corresponds to all buckets.

2) For level 1 of the tree, there are t_1 nodes: $B_1, B_2, \cdots, B_{t_1}$. Each node corresponds to a set of $t_2 t_3 \cdots t_N$ buckets. Thus the first node on level 1 consists of buckets ordered from 1 to $t_2 t_3 \cdots t_N$. The second node consists of buckets ordered from $t_2 t_3 \cdots t_N + 1$ to $2(t_2 t_3 \cdots t_N)$, etc.

3) Each node on level 1 is split into t_2 nodes on level 2. Thus there are $t_1 t_2$ nodes on level 2. Each node corresponds to $t_3 t_4 \cdots t_N$ buckets. Thus the first node on level 2 consists of buckets ordered from 1 to $t_3 t_4 \cdots t_N$. The second node corresponds to buckets ordered from $t_3 t_4 \cdots t_N + 1$ to $2(t_3 t_4 \cdots t_N)$, etc.

4) In general, there are $t_1 t_2 \cdots t_i$ nodes on level i of the tree. Each node corresponds to $t_{i+1} t_{i+2} \cdots t_N$ buckets.

5) Within each node on level i, $s_1, s_2, \cdots, s_i$ assume the same value, for all buckets in this node.

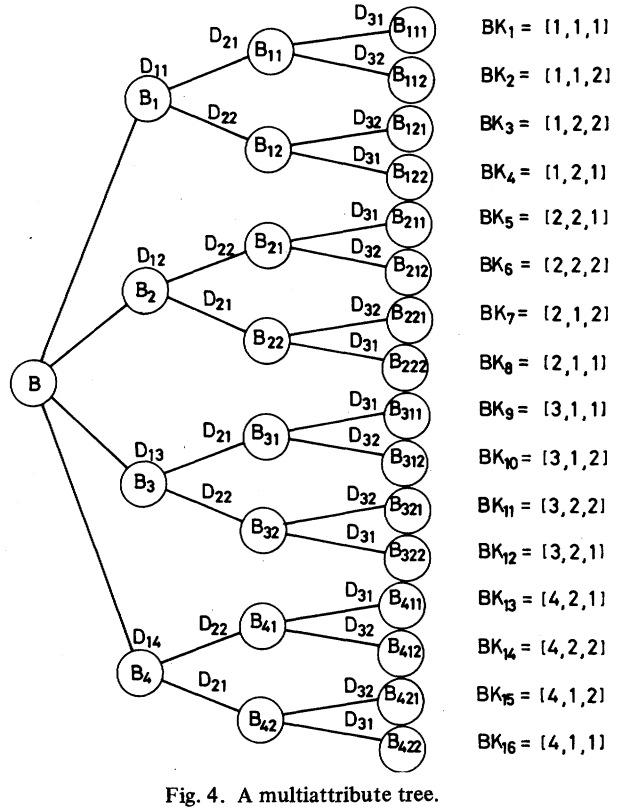


Fig. 4. A multiattribute tree.

Example 6.2: Consider a three-key records set:

$$D_1 = \{A_{11}, A_{12}, A_{13}, A_{14}, A_{15}, A_{16}, A_{17}, A_{18}\}$$
$$D_2 = \{A_{21}, A_{22}, A_{23}, A_{24}\}$$

and

$$D_3 = \{A_{31}, A_{32}, A_{33}, A_{34}, A_{35}, A_{36}\}.$$

Let $D_{11} = \{A_{11}, A_{12}\}$, $D_{12} = \{A_{13}, A_{14}\}$, $D_{13} = \{A_{15}, A_{16}\}$, $D_{14} = \{A_{17}, A_{18}\}$, $D_{21} = \{A_{21}, A_{22}\}$, $D_{22} = \{A_{23}, A_{24}\}$, $D_{31} = \{A_{31}, A_{32}, A_{33}\}$, and $D_{32} = \{A_{34}, A_{35}, A_{36}\}$.

$$NR = q_1 \cdot q_2 \cdot q_3 = 8 \times 4 \times 6 = 192$$

$$BZ = z_1 \cdot z_2 \cdot z_3 = 2 \times 2 \times 3 = 12$$

$$NB = t_1 \cdot t_2 \cdot t_3 = \frac{q_1}{z_1} \times \frac{q_2}{z_2} \times \frac{q_3}{z_3} = 4 \times 2 \times 2 = 16.$$

The tree corresponding to the buckets in which all the records in this case is stored is now depicted as in Fig. 4.

For level 1 of the tree, the first node B_1 corresponds to buckets ordered from 1 to 4. On the second level of the tree, the second node B_{12} corresponds to buckets ordered in 3 and 4. For records in B_1, the first key is in $D_{11} = \{A_{11}, A_{12}\}$ for all records. In B_2, the first key is in D_{11} and the second key is in $D_{22} = \{A_{23}, A_{24}\}$ for all records.

This kind of structure is called multiple-attribute tree [7], [15].

VII. Concluding Remarks

In this paper, we proposed the bucket-oriented symbolic Gray code as a multiattribute minimal perfect hash function.

This hashing function can be used to produce any arbitrary Cartesian product file. Since Cartesian product file systems

have been shown to be appropriate for partial match queries, our bucket-oriented symbolic Gray code is appropriate for partial match queries. We would like to emphasize here that hashing is good in this case because no index file is necessary.

Our next job is to investigate how our multiattribute hashing function can be used to organize files where some possible records are missing.

Appendix A
Proof of Theorem 6.1

Definition: Let

$$1 \leqslant L \leqslant q_1 q_2 \cdots q_N$$

$$L = a_1(q_2 q_3 \cdots q_N) + a_2(q_3 q_4 \cdots q_N) + \cdots + a_{N-2}(q_{N-1} q_N) + a_{N-1} q_N + a_N + 1,$$

where

$$a_i \text{ is an integer}, \quad 0 \leqslant a_i < q_i.$$

This N-tuple $(a_1, a_2, \cdots, a_N)$ is called the q_i representation of L. For example, consider the case where $q_1 = 3$, $q_2 = 2$, $q_3 = 3$, and $L = 6$. Then $q_2 q_3 = 2 \times 3$ and $q_3 = 3$. Therefore, $6 = 0(2 \times 3) + 1(3) + 2 + 1$. This means $(a_1, a_2, a_3) = (0, 1, 2)$.

The q_i representation of 6 is (0, 1, 2).

Lemma 1: Let $A = (a_1, a_2, \cdots, a_N)$ be the q_i representation of L and $B = (b_1, b_2, \cdots, b_N)$ be the q_i representation of $L + 1$, then the Hamming distance between A and B is $m \geqslant 1$ and

a) $a_i = b_i$, for $1 \leqslant i \leqslant N - k, k \geqslant m$

b) $a_{N-k+1} + 1 = b_{N-k+1}$

c) $a_i = q_i - 1$ for $N - k + 2 \leqslant i \leqslant N$ and $b_i = 0$

Proof: For any integer L, $1 \leqslant L < q_1 q_2 \cdots q_N$, there is only one N-tuple $(a_1, a_2, \cdots, a_N)$ such that $L = a_1 q_2 q_3 \cdots q_N + a_2 q_3 \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1$, where $0 \leqslant a_i < q_i$. (1)

For integer $L + 1$, $1 \leqslant L + 1 \leqslant q_1 q_2 \cdots q_N$, there is only one N-tuple $(b_1, b_2, \cdots, b_N)$, such that $L + 1 = b_1 q_2 q_3 \cdots q_N + b_2 q_3 \cdots q_N + \cdots + b_{N-1} q_N + b_N + 1$, where $0 \leqslant b_i \leqslant q_i$. So, $L = b_1 q_2 q_3 \cdots q_N + b_2 q_3 \cdots q_N + b_{N-1} q_N + b_N$ (2)

Compare (1) and (2)—there are two possibilities.

Case 1:

$$0 \leqslant a_N < q_N - 1, \quad 0 \leqslant a_N + 1 < q_N.$$

Assume $b_N = a_N + 1$ and $b_i = a_i$, $i = 1, 2, \cdots, N - 1$. In this case, the Hamming distance between $(a_1, a_2, \cdots, a_N)$ and $(b_1, b_2, \cdots, b_N)$ is 1.

Case 2:

$$a_N = q_N - 1, a_{N-1} = q_{N-1} - 1, \cdots, a_{N-k+2} = q_{N-k+2} - 1$$

and

$$a_{N-k+1} < q_{N-k+1} - 1, \quad \text{for some } k, 1 < k \leqslant N.$$

In this case,

$$b_{N-k+1} = a_{N-k+1} + 1,$$
$$b_{N-k+2} = b_{N-k+3} = \cdots = b_{N-1} = b_N = 0$$

and

$$b_i = a_i, \quad i = 1, 2, \cdots, N - k.$$

The Hamming distance between $(a_1, a_2, \cdots, a_N)$ and $(b_1, b_2, \cdots, b_N)$ is m, where $1 \leqslant m \leqslant k$.

In general, the Hamming distance between $(a_1, a_2, \cdots, a_N)$ and $(b_1, b_2, \cdots, b_N)$ is $m \geqslant 1$ and

$$a_i = b_i \quad \text{for} \quad 1 \leqslant i \leqslant N - k,$$

and

$$a_{N-k+1} + 1 = b_{N-k+1}. \qquad \text{Q.E.D.}$$

Lemma 2: Let $A = (a_1, a_2, \cdots, a_N)$ be the q_i representation of L where $1 \leqslant L \leqslant q_1 q_2 \cdots q_N$. Then

$$\left\lceil \frac{L}{q_i q_{i+1} \cdots q_N} \right\rceil$$

is equal to

$$a_1(q_2 q_3 \cdots q_{i-1}) + a_2(q_3 q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1.$$

Proof: Since $A = (a_1, a_2, \cdots, a_N)$ is the q_i representation of L, we have $L = a_1 q_2 q_3 \cdots q_N + a_2 q_3 \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1$,

$$\left\lceil \frac{L}{q_i q_{i+1} \cdots q_N} \right\rceil$$

$$= \left\lceil \frac{a_1 q_2 q_3 \cdots q_N + a_2 q_3 \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1}{q_i q_{i+1} \cdots q_N} \right\rceil$$

$$= a_1(q_2 q_3 \cdots q_{i-1}) + a_2(q_3 q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + \left\lceil \frac{a_i q_{i+1} \cdots q_N + a_{i+1} q_{i+2} \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1}{q_i q_{i+1} \cdots q_N} \right\rceil.$$

Since $a_i < q_i$, we have $a_i \leqslant q_i - 1$ and

$$1 \leqslant a_i q_{i+1} \cdots q_N + a_{i+1} q_{i+2} \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1$$
$$\leqslant (q_i - 1) q_{i+1} \cdots q_N + (q_{i+1} - 1) q_{i+2} \cdots q_N + \cdots + (q_{N-1} - 1) q_N + (q_N - 1) + 1$$
$$= q_i q_{i+1} \cdots q_N.$$

Hence

$$\left\lceil \frac{a_i q_{i+1} \cdots q_N + a_{i+1} q_{i+2} \cdots q_N + \cdots + a_{N-1} q_N + a_N + 1}{q_i q_{i+1} \cdots q_N} \right\rceil = 1.$$

That is,

$$\left\lceil \frac{L}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ is } a_1(q_2 q_3 \cdots q_{i-1}) + a_2(q_3 q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1.$$

We have the proof. Q.E.D.

Lemma 3: Let $A = (a_1, a_2, \cdots, a_N)$ be the q_i representation of u and $B = (b_1, b_2, \cdots, b_N)$ be the q_i representation of v. Let $X = (x_1, x_2, \cdots, x_N)$ and $Y = (y_1, y_2, \cdots, y_N)$ be two N-tuples which are defined as follows:

$$x_i = a_i + 1, \quad \text{if } \left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ is odd,}$$

$$x_i = q_i - a_i, \quad \text{if } \left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ is even,}$$

and

$$y_i = b_i + 1, \quad \text{if } \left\lceil \frac{v}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ is odd,}$$

$$y_i = q_i - b_i, \quad \text{if } \left\lceil \frac{v}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ is even.}$$

If $|u - v| = 1$, the Hamming distance between X and Y is 1.

Proof: Because $|u - v| = 1$, we may suppose that $v = u + 1$. Let $A = (a_1, a_2, \cdots, a_N)$ be the q_i representation of u and $B = (b_1, b_2, \cdots, b_N)$ be the q_i representation of $u + 1$. By Lemma 1, the Hamming distance between A and B is $m \geqslant 1$ and

$$a_i = b_i \quad \text{for } 1 \leqslant i \leqslant N - k, k \geqslant m. \tag{1}$$

$$a_{N-k+1} + 1 = b_{N-k+1} \tag{2}$$

By Lemma 2, we have

$$\begin{aligned}\left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil &= a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{i-1} + 1 \\ &= b_1 q_2 \cdots q_{i-1} + b_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + b_{i-1} + 1 \\ &= \left\lceil \frac{v}{q_i q_{i+1} \cdots q_N} \right\rceil,\end{aligned}$$

$$\text{for } 1 \leqslant i \leqslant N - k + 1. \tag{3}$$

Equations (1)-(3) imply that

$$y_i = x_i \quad \text{for } 1 \leqslant i \leqslant N - k$$

and

$$y_{N-k+1} \neq x_{N-k+1}. \tag{4}$$

Consider

$$\left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil \text{ and } \left\lceil \frac{v}{q_i q_{i+1} \cdots q_N} \right\rceil$$

$$\text{for } N - k + 2 \leqslant i \leqslant N.$$

In this case,

$$\begin{aligned}\left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil &= a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{i-1} + 1 \\ &= (a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{N-k} q_{N-k+1} \cdots q_{i-1}) \\ &\quad + a_{N-k+1} q_{N-k+2} \cdots q_{i-1} \\ &\quad + [(q_{N-k+2} - 1) q_{N-k+3} \cdots q_{i-1} \\ &\quad + (q_{N-k+3} - 1) q_{N-k+4} \cdots q_{i-1} \\ &\quad + \cdots + (q_{i-1} - 1)] + 1 \\ &= a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{N-k+1} q_{N-k+2} \cdots q_{i-1} \\ &\quad + q_{N-k+2} q_{N-k+3} \cdots q_{i-1}.\end{aligned}$$

$$\begin{aligned}\left\lceil \frac{v}{q_i q_{i+1} \cdots q_N} \right\rceil &= b_1 q_2 \cdots q_{i-1} + b_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + b_{i-1} + 1 \\ &= [a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{N-k} q_{N-k+1} \cdots q_{i-1}] \\ &\quad + (a_{N-k+1} + 1) q_{N-k+2} \cdots q_{i-1} + 1 \\ &= a_1 q_2 \cdots q_{i-1} + a_2 q_3 \cdots q_{i-1} \\ &\quad + \cdots + a_{N-k} q_{N-k+1} \cdots q_{i-1} \\ &\quad + a_{N-k+1} q_{N-k+2} \cdots q_{i-1} \\ &\quad + q_{N-k+2} \cdots q_{i-1} + 1 \\ &= \left\lceil \frac{u}{q_i q_{i+1} \cdots q_N} \right\rceil + 1.\end{aligned}$$

Case 1: $\lceil u/q_i q_{i+1} \cdots q_N \rceil$ is even. In this case, $x_i = q_i - a_i = q_i - (q_i - 1) = 1$, since $a_i = q_i - 1$. Then $\lceil v/q_i q_{i+1} \cdots q_N \rceil$ is odd. Therefore, $y_i = b_i + 1 = 0 + 1 = 1$, since $b_i = 0$. We have $x_i = y_i = 1$.

Case 2: $\lceil u/q_i q_{i+1} \cdots q_N \rceil$ is odd. In this case, $x_i = a_i + 1 = (q_i - 1) + 1 = q_i$, since $a_i = q_i - 1$.

$\lceil v/q_i q_{i+1} \cdots q_N \rceil$ is odd. Therefore, $y_i = q_i - b_i = q_i - 0 = q_i$. We have $x_i = y_i = q_i$ in this case.

Hence, $x_i = y_i$ for $N - k + 2 \leqslant i \leqslant N$. Combining (4) and (5), we conclude that the Hamming distance between X and Y is 1. Q.E.D.

Lemma 4: Let F be a function, F: $(f_1, f_2, \cdots, f_n) \to K$, $f_i < q_i$, defined by following equation:

$$K = a_1(q_2 q_3 \cdots q_N) + a_2(q_3 q_4 \cdots q_N) + \cdots + a_{N-1} q_N + a_N + 1$$

where the N-tuple $(a_1, a_2, \cdots, a_N)$ is determined through the following rules:

a) $a_1 = f_1 - 1$.

b) For $1 < i \leqslant N$

$$\begin{aligned}L_2 &= a_1 + 1 \\ L_3 &= a_1(q_2) + a_2 + 1 \\ &\vdots \\ L_i &= a_1(q_2 q_3 \cdots q_{i-1}) + a_2(q_3 q_4 \cdots q_{i-1}) \\ &\quad + \cdots + a_{i-1} + 1 \\ &\vdots \\ L_N &= a_1(q_2 q_3 \cdots q_{N-1}) + a_2(q_3 q_4 \cdots q_{N-1}) \\ &\quad + \cdots + a_{N-1} + 1;\end{aligned}$$

if L_i is odd, $a_i = f_i - 1$

if L_i is even, $a_i = q_i - f_i$.

Let G be a function, G: $m \to (g_1, g_2, \cdots, g_N)$, $1 \leqslant m \leqslant$

$q_1q_2 \cdots q_N$, $g_i = q_i - b_i$, if $\lceil m/q_iq_{i+1} \cdots q_N \rceil$ is even, $g_i = b_i + 1$, if $\lceil m/q_iq_{i+1} \cdots q_N \rceil$ is odd, where the N-tuple $(b_1, b_2, \cdots, b_N)$ is determined through the following equation:

$$m = b_1(q_2q_3 \cdots q_N) + b_2(q_3q_4 \cdots q_N) + \cdots + b_{N-1}q_N + b_N + 1.$$

Then $G = F^{-1}$.

Proof: For any N-tuple $(f_1, f_2, \cdots, f_N)$, $F: (f_1, f_2, \cdots, f_N) \to K$,

$$K = a_1(q_2q_3 \cdots q_N) + a_2(q_3q_4 \cdots q_N) + \cdots + a_{N-1}q_N + a_N + 1$$

where $a_1, a_2, \cdots, a_N$ are determined by

a) $a_1 = f_1 - 1$

b) for $1 < i \leqslant N$,

if $a_1(q_2q_3 \cdots q_{i-1}) + a_2(q_3q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1$ is odd, $a_i = f_i - 1$; (1)

if $a_1(q_2q_3 \cdots q_{i-1}) + a_2(q_3q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1$ is even, $a_i = q_i - f_i$. (2)

We shall show that

$$G(F(f_1, f_2, \cdots, f_N)) = (f_1, f_2, \cdots, f_N)$$

or

$$G(K) = (f_1, f_2, \cdots, f_N).$$

Suppose $G(K) = (g_1, g_2, \cdots, g_N)$. For $i = 1$, since $K \leqslant q_1q_2 \cdots q_N = q_iq_{i+1} \cdots q_N$, $[K/q_iq_{i+1} \cdots q_N] = 1$ is odd, we have $g_1 = a_1 + 1$ or $a_1 = g_1 - 1$. By Lemma 2, since $(g_1, g_2, \cdots, g_N)$ is the q_i representation of K, we have

$$\left\lceil \frac{K}{q_iq_{i+1} \cdots q_N} \right\rceil = a_1(q_2q_3 \cdots q_{i-1}) + a_2(q_3q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1.$$

Because of the conditions of the G function, if $[K/q_iq_{i+1} \cdots q_N]$ is odd, $g_i = a_i + 1$. If $[K/q_iq_{i+1} \cdots q_N]$ is even, $g_i = q_i - a_i$. That is, if $a_1(q_2q_3 \cdots q_{i-1}) + a_2(q_3q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1$ is odd,

$$g_i = a_i + 1 \quad \text{or} \quad a_i = g_i - 1; \tag{3}$$

if $a_1(q_2q_3 \cdots q_{i-1}) + a_2(q_3q_4 \cdots q_{i-1}) + \cdots + a_{i-1} + 1$ is even,

$$g_i = q_i - a_i \quad \text{or} \quad a_i = q_i - g_i \tag{4}$$

Comparing (1) and (3), (2) and (4), we have $g_i = f_i$. So, $G(F(f_1, f_2, \cdots, f_N)) = G(K) = (f_1, f_2, \cdots, f_N)$. That is, $G = F^{-1}$. Q.E.D.

Theorem 6.1: For every record $R \in D_1 \times D_2 \times \cdots \times D_N$, if $\mathrm{KAT}(R) = L$, then $\mathrm{AKT}(L) = R$.

Proof: Let $R = (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$ where the address associated with R is determined by the KAT (Algorithm B). Let $\mathrm{KAT}(R) = L$. We shall now show that the record $(r_{L1}, r_{L2}, \cdots, r_{LN})$ associated with L determined through the use of AKT (Algorithm C) is exactly $(A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N})$.

From KAT, we finally have $L = z_1z_2 \cdots z_N(P - 1) + m'$. Let $z_1z_2 \cdots z_N = C$. Then $L = C(P - 1) + m'$, where m' is either $C - m + 1$ or m. By Step 3,

$$m = a_1(z_2z_3 \cdots z_N) + a_2(z_3z_4 \cdots z_N) + \cdots + a_{N-1}z_N + a_N + 1,$$

where $0 \leqslant a_i < z_i$ and a_i, z_i are integers.

We have

$$0(z_2z_3 \cdots z_N) + 0(z_3z_4 \cdots z_N) + \cdots + 0z_N + 0 + 1 \leqslant m \leqslant (z_1 - 1)(z_2z_3 \cdots z_N) + (z_2 - 1)(z_3z_4 \cdots z_N) + \cdots + (z_{N-1} - 1)z_N + (z_N - 1) + 1.$$

So $1 \leqslant m \leqslant z_1z_2 \cdots z_N$, i.e., $1 \leqslant m \leqslant C$. Hence, $1 \leqslant m' \leqslant C$.

Consider $L = C(P - 1) + m'$, and $1 \leqslant m' \leqslant C$. We have $L - C(P - 1) = m'$, $1 \leqslant L - C(P - 1) \leqslant C$. Therefore,

$$\frac{L - 1}{C} + 1 \geqslant P \geqslant \frac{L}{C}.$$

Since P is an integer,

$$\left\lfloor \frac{L - 1}{C} \right\rfloor + 1 \geqslant P \geqslant \left\lceil \frac{L}{C} \right\rceil.$$

Because

$$\left\lfloor \frac{L - 1}{C} \right\rfloor + 1 = \left\lceil \frac{L}{C} \right\rceil,$$

we have

$$\left\lceil \frac{L}{C} \right\rceil \geqslant P \geqslant \left\lceil \frac{L}{C} \right\rceil.$$

So

$$P = \left\lceil \frac{L}{C} \right\rceil \quad \text{if} \quad L = C(P - 1) + m'$$

and

$$m' = L - \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C.$$

In this case,

$$m = z_1z_2 \cdots z_N - m' + 1 = C - L + \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C + 1, \quad \text{if } \left\lceil \frac{L}{C} \right\rceil \text{ is even.}$$

$$m = m' = L - \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C, \quad \text{if } \left\lceil \frac{L}{C} \right\rceil \text{ is odd.}$$

Consider the AKT procedures. For an address L, we have

$$m'_L = L - \left(\left\lceil \frac{L}{z_1z_2 \cdots z_N} \right\rceil - 1 \right) \cdot z_1z_2 \cdots z_N = L - \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C$$

$$P_L = \left\lceil \frac{L}{z_1z_2 \cdots z_N} \right\rceil = \left\lceil \frac{L}{C} \right\rceil;$$

if $\left\lceil \frac{L}{C} \right\rceil$ is odd, $m_L = m'_L = L - \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C.$

if $\left\lceil \frac{L}{C} \right\rceil$ is even, $m_L = z_1 z_2 \cdots z_N - m'_L + 1$

$$= C - L + \left(\left\lceil \frac{L}{C} \right\rceil - 1 \right) \cdot C + 1$$

$$= \left\lceil \frac{L}{C} \right\rceil \cdot C - L + 1.$$

So, $m_L = m$ and $P_L = P$.

Let F: $(b'_1, b'_2, \cdots, b'_N) \rightarrow m$, by using the procedures of Step 3 in Algorithm B and G: $m_L \rightarrow (b'_{L1}, b'_{L2}, \cdots, b'_{LN})$ by using the procedures of Step 4 in Algorithm C.

By Lemma 4, we have $G = F^{-1}$. Since $m_L = m$, we have $(b'_{L1}, b'_{L2}, \cdots, b'_{LN}) = (b'_1, b'_2, \cdots, b'_N)$. In similar manner, let H: $(s_1, s_2, \cdots, s_N) \rightarrow P$ by using the procedures of Step 4 in Algorithm B and T: $P_L \rightarrow (s_{L1}, s_{L2}, \cdots, s_{LN})$ by using the procedures of Step 3 in Algorithm C. By Lemma 4, we have $T = H^{-1}$.

Since $P_L = P$, we have $(s_{L1}, s_{L2}, \cdots, s_{LN}) = (s_1, s_2, \cdots, s_N)$. For Step 1 and Step 2 in Algorithm B, we have

$$s_i = \left\lceil \frac{b_i}{z_i} \right\rceil \quad \text{and} \quad b'_i = b_i - (s_i - 1) z_i, \text{ respectively.}$$

That is,

$$b_i = b'_i + (s_i - 1) \cdot z_i. \tag{1}$$

For Step 5 in Algorithm C, we have

$$b_{Li} = z_i (s_{Li} - 1) + b'_{Li}. \tag{2}$$

Since $b'_i = b'_{Li}$ and $s_i = s_{Li}$, comparing (1) with (2), we have $b_{Li} = b_i$. In other words,

$$(r_{L1}, r_{L2}, \cdots, r_{LN}) = (A_{1b_{L1}}, A_{2b_{L2}}, \cdots, A_{Nb_{LN}})$$
$$= (A_{1b_1}, A_{2b_2}, \cdots, A_{Nb_N}). \quad \text{Q.E.D.}$$

Appendix B
Proof of Theorem 6.2

Theorem 6.2: Let there be N sets: $D_1, D_2, \cdots, D_N$ where $D_i = \{A_{i1}, A_{i2}, \cdots, A_{iq_i}\}$ and $q_i > 1$. Let R_L denote the record associated with L. Let NR denote the total number of records in $D_1 \times D_2 \times \cdots \times D_N$. Then the Hamming distance between R_i and R_{i+1} is 1, for $1 \leq i < NR$.

Proof: $(z_1 z_2, \cdots, z_N)$ and $(q_1, q_2, \cdots, q_N)$ are given. Let $C = z_1 z_2 \cdots z_N$. Let $R_i = (r_{i1}, r_{i2}, \cdots, r_{iN})$, and $R_{i+1} = (r_{(i+1)1}, r_{(i+1)2}, \cdots, r_{(i+1)N})$.

We want to show that $d(R_i, R_{i+1}) = \sum_{j=1}^{N} \delta(r_{ij}, r_{(i+1)j}) = 1$. By applying AKT (or Algorithm C), we have

$$m'_i = i - \left(\left\lceil \frac{i}{C} \right\rceil - 1 \right) \cdot C$$

$$m'_{i+1} = (i+1) - \left(\left\lceil \frac{i+1}{C} \right\rceil - 1 \right) \cdot C$$

$$P_i = \left\lceil \frac{i}{C} \right\rceil$$

and

$$P_{i+1} = \left\lceil \frac{i+1}{C} \right\rceil.$$

For $\lceil i/C \rceil$ and $\lceil i + 1/C \rceil$, we have two cases to consider.

Case 1:

$$\left\lceil \frac{i}{C} \right\rceil = \left\lceil \frac{i+1}{C} \right\rceil = \text{integer}.$$

In this case, $m'_{i+1} = m'_i + 1$ and $P_{i+1} = P_i$. If $P_i = P_{i+1}$ is odd, $m_i = m'_i$ and $m_{i+1} = m'_{i+1}$, we have $m_{i+1} = m_i + 1$.

For $P_i = P_{i+1}$ is even, $m_i = C - m'_i + 1$ and $m_{i+1} = C - m'_{i+1} + 1$, we have $m_i = m_{i+1} + 1$.

So we can conclude that

$$|m_{i+1} - m_i| = 1.$$

If $P_{i+1} = P_i$, we have

$$(s_{i1}, s_{i2}, \cdots, s_{iN}) = (s_{(i+1)1}, s_{(i+1)2}, \cdots, s_{(i+1)N})$$

where s_{ij}'s and $s_{(i+1)j}$'s are defined as follows:

$$s_{ij} = a_{ij} + 1, \quad \text{if } \left\lceil \frac{P_i}{t_j t_{j+1} \cdots t_N} \right\rceil \text{ is odd}$$

$$s_{ij} = t_j - a_{ij}, \quad \text{if } \left\lceil \frac{P_i}{t_j t_{j+1} \cdots t_N} \right\rceil \text{ is even}$$

and

$$s_{(i+1)j} = a_{(i+1)j} + 1, \quad \text{if } \left\lceil \frac{P_{i+1}}{t_j t_{j+1} \cdots t_N} \right\rceil \text{ is odd}$$

$$s_{(i+1)j} = t_j - a_{(i+1)j}, \quad \text{if } \left\lceil \frac{P_{i+1}}{t_j t_{j+1} \cdots t_N} \right\rceil \text{ is even}$$

then $(a_{i1}, a_{i2}, \cdots, a_{iN})$ is the t_j representation of P_i and $(a_{(i+1)1}, a_{(i+1)2}, \cdots, a_{(i+1)N})$ is the t_j representation of P_{i+1}. Since $|m_{i+1} - m_i| = 1$, by Lemma 3, we have $(b'_{i1}, b'_{i2}, \cdots, b'_{iN})$ and $(b'_{(i+1)1}, b'_{(i+1)2}, \cdots, b'_{(i+1)N})$ as two N-tuples which are defined as below and the Hamming distance between them is 1.

$$b'_{ij} = a_{ij} + 1, \quad \text{if } \left\lceil \frac{m_i}{z_j z_{j+1} \cdots z_N} \right\rceil \text{ is odd}$$

$$b'_{ij} = z_j - a_{ij}, \quad \text{if } \left\lceil \frac{m_i}{z_j z_{j+1} \cdots z_N} \right\rceil \text{ is even}$$

and

$$b'_{(i+1)j} = a_{(i+1)j} + 1, \quad \text{if } \left\lceil \frac{m_{i+1}}{z_j z_{j+1} \cdots z_N} \right\rceil \text{ is odd}$$

$$b'_{(i+1)j} = z_j - a_{(i+1)j}, \quad \text{if } \left\lceil \frac{m_{i+1}}{z_j z_{j+1} \cdots z_N} \right\rceil \text{ is even}$$

where $(a_{i1}, a_{i2}, \cdots, a_{iN})$ is the z_j representation of m_i and $(a_{(i+1)1}, a_{(i+1)2}, \cdots, a_{(i+1)N})$ is the z_j representation of m_{i+1}.

For $b_{ij} = z_j \cdot (s_{ij} - 1) + b'_{ij}$ and $b_{(i+1)j} = z_j \cdot (s_{ij} - 1) + b'_{(i+1)j}$,

$j = 1, 2, \cdots, N$, since the Hamming distance between $(b'_{i1}, b'_{i2}, \cdots, b'_{iN})$ and $(b'_{(i+1)1}, b'_{(i+1)2}, \cdots, b'_{(i+1)N})$ is 1, there exists a k, such that $b'_{ik} \neq b'_{(i+1)k}$ and $b'_{ij} = b'_{(i+1)j}$ for all $j = 1, 2, \cdots, N, j \neq k$.

Since $s_{ij} = s_{(i+1)j}$, for all $j = 1, 2, \cdots, N$, we have only one $b_{ik} \neq b_{(i+1)k}$ and $b_{ij} = b_{(i+1)j}$, for all $j = 1, 2, \cdots, N$, $j \neq k$. Because $r_{ij} = A_{jb_{ij}}$ and $r_{(i+1)j} = A_{jb_{(i+1)j}}$, we have $r_{ij} = r_{(i+1)j}$ for all $j = 1, 2, \cdots, N, j \neq k$ and $r_{ik} \neq r_{(i+1)k}$. Therefore,

$$d(R_i, R_{i+1}) = \sum_{j=1}^{N} \delta(r_{ij}, r_{(i+1)j}) = 1.$$

Case 2:

$$\left\lceil \frac{i+1}{C} \right\rceil = e + 1 \quad \text{and} \quad \left\lceil \frac{i}{C} \right\rceil = e.$$

In this situation, for $\lceil i/C \rceil = e$, we have

$$e - 1 < \frac{i}{C} \leqslant e.$$

If $i/C < e$, or $i < Ce$, we have

$$i \leqslant Ce - 1, \quad i + 1 \leqslant Ce, \quad \frac{i+1}{C} \leqslant e.$$

So $\lceil i + 1/C \rceil \leqslant e$, which is contradict to $\lceil i + 1/C \rceil = e + 1$. Hence we have $i/C = e$, that is $i = Ce$.

In this case, we have two possibilities.

1) If $P_i = e$ is odd, then $P_{i+1} = e + 1$ is even, we have

$$m_i = m'_i = i - (e - 1) \cdot C = i - eC + C = C$$

$$m_{i+1} = C - m'_{i+1} + 1 = C - [(i + 1) - eC] + 1$$

$$= C - i + eC = C.$$

That is, $m_i = m_{i+1} = C$.

2) If $P_i = e$ is even, then $P_{i+1} = e + 1$ is odd, we have

$$m_i = C - [i - (e - 1) \cdot C] + 1 = C - i + (e - 1) \cdot C + 1 = 1$$

and

$$m_{i+1} = m'_{i+1} = i + 1 - eC = 1.$$

That is, $m_i = m_{i+1} = 1$.

So we have concluded that if $\lceil i/C \rceil = e$ and $\lceil i + 1/C \rceil = e + 1$, $m_i = m_{i+1}$ and $P_{i+1} = P_i + 1$.

For $P_{i+1} = P_i + 1$, by Lemma 3, we have $(s_{i1}, s_{i2}, \cdots, s_{iN})$ and $(s_{(i+1)1}, s_{(i+1)2}, \cdots, s_{(i+1)N})$ or two N-tuples which are defined as below and the Hamming distance between them is 1:

$$s_{ij} = a_{ij} + 1, \quad \text{if} \quad \left\lceil \frac{P_i}{t_j t_{j+1} \cdots t_N} \right\rceil \quad \text{is odd}$$

$$s_{ij} = t_j - a_{ij}, \quad \text{if} \quad \left\lceil \frac{P_i}{t_j t_{j+1} \cdots t_N} \right\rceil \quad \text{is even}$$

and

$$s_{(i+1)j} = a_{(i+1)j} + 1, \quad \text{if} \quad \left\lceil \frac{P_{i+1}}{t_j t_{j+1} \cdots t_N} \right\rceil \quad \text{is odd}$$

$$s_{(i+1)j} = t_j - a_{(i+1)j}, \quad \text{if} \quad \left\lceil \frac{P_{i+1}}{t_j t_{j+1} \cdots t_N} \right\rceil \quad \text{is even}$$

where $(a_{i1}, a_{i2}, \cdots, a_{iN})$ is the t_j representation of P_i and $(a_{(i+1)1}, a_{(i+1)2}, \cdots, a_{(i+1)N})$ is the t_j representation of P_{i+1}.

For $m_{i+1} = m_i$, we have $(b'_{i1}, b'_{i2}, \cdots, b'_{iN}) = (b'_{(i+1)1}, b'_{(i+1)2}, \cdots, b'_{(i+1)N})$, where b'_{ij}'s and $b'_{(i+1)j}$'s are defined as follows:

$$b'_{ij} = a_{ij} + 1, \quad \text{if} \quad \left\lceil \frac{m_i}{z_j z_{j+1} \cdots z_N} \right\rceil \quad \text{is odd}$$

$$b'_{ij} = z_j - a_{ij}, \quad \text{if} \quad \left\lceil \frac{m_i}{z_j z_{j+1} \cdots z_N} \right\rceil \quad \text{is even}$$

and

$$b'_{(i+1)j} = a_{(i+1)j} + 1, \quad \text{if} \quad \left\lceil \frac{m_{i+1}}{z_j z_{j+1} \cdots z_N} \right\rceil \quad \text{is odd}$$

$$b'_{(i+1)j} = z_j - a_{(i+1)j}, \quad \text{if} \quad \left\lceil \frac{m_{i+1}}{z_j z_{j+1} \cdots z_N} \right\rceil \quad \text{is even}$$

where $(a_{i1}, a_{i2}, \cdots, a_{iN})$ is the z_j representation of m_i and $(a_{(i+1)1}, a_{(i+1)2}, \cdots, a_{(i+1)N})$ is the z_j representation of m_{i+1}.

For $b_{ij} = z_j(s_{ij} - 1) + b'_{ij}$ and $b_{(i+1)j} = z_j(s_{(i+1)j} - 1) + b'_{(i+1)j}$, since the Hamming distance between $(s_{i1}, s_{i2}, \cdots, s_{iN})$ and $(s_{(i+1)1}, s_{(i+1)2}, \cdots, s_{(i+1)N})$ is 1, there exists a w, such that $A_{iw} \neq A_{(i+1)w}$ and $s_{ij} = s_{(i+1)j}$, for all $j = 1, 2, \cdots, N, j \neq w$.

Since $b'_{ij} = b'_{(i+1)j}$, for all $j = 1, 2, \cdots, N$, we have $b_{iw} \neq b_{(i+1)w}$ and $b_{ij} = b_{(i+1)j}$, for all $j = 1, 2, \cdots, N, j \neq w$. Since $r_{ij} = A_{jb_{ij}}$ and $r_{(i+1)j} = A_{jb_{(i+1)j}}$, $r_{iw} \neq r_{(i+1)w}$ and $r_{ij} = r_{(i+1)j}$, for all $j = 1, 2, \cdots, N, j \neq w$. Hence, $d(R_i, R_{i+1}) = \sum_{j=1}^{N} \delta(r_{ij}, r_{(i+1)j}) = 1$. Q.E.D.

REFERENCES

[1] A. V. Aho and J. D. Ullman, "Optimal partial-match retrieval when fields are independently specified," *ACM Trans. Database Syst.*, vol. 4, pp. 168-179, June 1979.

[2] J. L. Bentley and J. H. Friedman, "Data structures for range searching," *Comput. Surveys*, vol. 11, pp. 397-409, Dec. 1979.

[3] W. A. Burkhard, "Partial-match hash coding: Benefits of redundancy," *ACM Trans. Database Syst.*, vol. 4, pp. 228-239, June 1979.

[4] W. A. Burkhard and R. M. Keller, "Some approaches to best-match file searching," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 230-236, Apr. 1973.

[5] C. C. Chang and R.C.T. Lee, "Optimal Cartesian product files for partial match queries and partial match patterns," in *Proc. NCS 1979 Conf.*, Taipei, Taiwan, Dec. 1979, pp. 5-27-5-37.

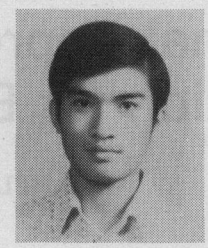
[6] C. C. Chang, R.C.T. Lee, and H. C. Du, "Some properties of Cartesian product files," in *Proc. ACM-SIGMOD 1980 Conf.*, Santa Monica, CA, May 1980, pp. 157-168.

[7] J. M. Chang and K. S. Fu, "On the retrieval time and the storage space of doubly-chained multiple-attribute tree data base organization," *Policy Anal. Inform. Syst.*, vol. 1, pp. 22-48, Jan. 1978.

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

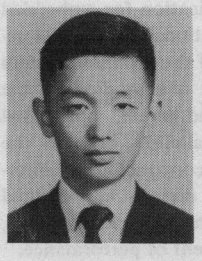
[9] H. C. Du and R.C.T. Lee, "Symbolic gray code as a multikey hashing function," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-2, pp. 83-90, Jan. 1980.

[10] H. C. Du and J. S. Sobolewski, *Disk Allocation for Cartesian Product Files on Multiple Disk Systems.* Seattle: Univ. Washington, 1980.
[11] M. W. Du, K. F. Jea, and D. W. Shieh, "The study of a new perfect hash scheme," in *Proc. COMPSAC 1980*, pp. 341–347.
[12] J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 1000–1006, Oct. 1975.
[13] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, pp. 209–226, Sept. 1977.
[14] K. Fukunaga and P. M. Narenda, "A branch and bound algorithm for computing *k*-nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 750–753, July 1975.
[15] R. L. Kashyap, S.K.C. Subas, and S. B. Yao, "Analysis of the multiple-attribute-tree data-base organization," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 451–567, Nov. 1977.
[16] S. P. Ghosh, *Data Base Organization for Data Management.* New York: Academic, 1977.
[17] R.C.T. Lee, Y. H. Chin, and S. C. Chang, "Application of principal component analysis to multikey searching," *IEEE Trans. Software Eng.*, vol. SE-2, pp. 185–193, Sept. 1976.
[18] R.C.T. Lee and S. H. Tseng, "Multi-key sorting," *Policy Anal. Inform. Syst.*, vol. 3, pp. 1–20, Dec. 1979.
[19] W. C. Lin, R.C.T. Lee, and H. C. Du, "Common properties of some multi-attribute file systems," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 160–174, Mar. 1979.
[20] J. H. Liou and S. B. Yao, "Multi-dimensional clustering for data base organizations," *Inform. Syst.*, vol. 2, pp. 187–198, 1977.
[21] R. L. Rivest, "Analysis of associative retrieval algorithms," Ph.D. dissertation, Dep. Comput. Sci., Stanford Univ., Stanford, CA, 1974.
[22] ——, "Partial-match retrieval algorithms," *SIAM J. Comput.*, vol. 15, No. 1, pp. 19–50, Mar. 1976.
[23] J. B. Rothnie and T. Lozano, "Attribute based file organization in a paged memory environment," *Commun. Ass. Comput. Mach.*, vol. 17, pp. 63–69, Feb. 1974.
[24] C. W. Shen and R.C.T. Lee, "A nearest neighbor search technique with short zero-in-time," *IEEE Trans. Software Eng.*, to be published.
[25] R. Sprugnoli, "Perfect hashing functions: A single probe retrieving method for static sets," *Commun. Ass. Comput. Mach.*, vol. 20, pp. 841–850, Nov. 1977.



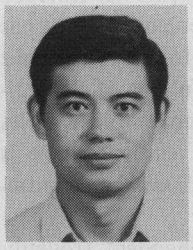
C. C. Chang was born in Taiwan in 1954. He received the B.S. degree in applied mathematics in 1977 and the M.S. degree in computer and decision sciences in 1979, both from the National Tsing Hua University.

He is presently an instructor as well as a Ph.D. student of the Institute of Computer Engineering in National Chiao-Tung University, Hsinchu, Taiwan. His research interests are in database design, algorithm analysis, and statistics.



R.C.T. Lee (A'74–M'75) received the Ph.D. degree from the University of California, Berkeley, in 1967.

He is currently the Director of the Institute of Computer and Decision Sciences, National Tsing Hua University, Hsinchu, Taiwan. He previously worked for NCR (California), National Institutes of Health (Bethesda, MD), and the Naval Research Laboratory (Washington, DC) before joining the National Tsing Hua University in 1975. He is the coauthor of *Symbolic Logic and Mechanical Theorem Proving* (New York: Academic), which has been translated into both Japanese and Russian. His research in clustering analysis will appear as a chapter entitled "Clustering Analysis and its Applications" in *Advances in Information Systems Science* (New York: Plenum). He is the author of more than 50 papers on mechanical theorem proving, database design, and pattern recognition.



M. W. Du (S'70–M'72) was born in Chung-King, China, in 1944. He received the B.S.E.E. degree from the National Taiwan University in 1966 and the Ph.D. degree from The Johns Hopkins University, Baltimore, MD, in 1972.

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