A file organization scheme based on composite perfect hashing, which guarantees single access retrieval from external files has been proposed recently. The ideal retrieval performance is achieved by using an auxiliary internal table and direct perfect hashing. In this paper, we explore systematic methods of finding direct perfect hashing functions. Experimental results indicate that the proposed methods are practical.

A hashing function is said to be ‘perfect’ for a given key set and table, if it causes no overflows. Perfect hashing yields ideal retrieval performance since there are no overflows to be handled. Perfect hashing was originally defined and investigated by Sprugnoli (1977). He gave algorithms for finding two different kinds of perfect hashing functions. The other early work in this regard are reported Jaeschke (1981) and Cichelli (1980). Because of the complexity of the computations involved, all these methods considered the storage of small static sets (10 to 20 elements), such as a table of reserved words in a compiler. The difficulty encountered in, and the complexity of finding perfect hashing functions led to theoretical investigations in this regard. Mairson (1983, 1984) proved that $\Omega(n)$ is the lower bound for the program size of searching a table with guaranteed single access retrieval. Fredman, Kolmos and Szemeredi (1982) gave a data structure which required $O(n)$ space and enables retrievals is $O(1)$ time.

Ramakrishna and Larson (1989) considered using perfect hashing for organizing external files. The perfect hashing schemes were classified into two categories: (a) Direct Perfect Hashing; and (b) Composite Perfect Hashing. Composite Perfect Hashing uses auxiliary storage and retrievals involve two levels of access. They proposed a file organization scheme based on composite perfect hashing. This scheme guarantees single access retrieval from external files at the cost of storage space in an internal memory and increased cost of insertions. The basic idea is to partition the large file into a number of small groups and store each group separately using direct perfect hashing. The details of individual groups are stored in internal table. The computational and I/O cost of the scheme have been shown to be competitive with other traditional hashing schemes. A trial and error method of finding direct perfect hashing functions was proposed and investigated. Investigation of systematic methods of finding direct perfect hashing functions was left as open problem. This is the main problem addressed in this paper.
Outline of the Paper

In the next section we give the QR algorithm for finding Quotient Reduction perfect hashing functions for external files. It forms a part of the Remainder Reduction method of finding direct perfect hashing functions, presented in the labeled section. Both methods are extensions of Sprugnoli's methods to external files. Experimental results presented at the end of the section titled Remainder Reduction Method indicate that the cost of finding perfect hashing functions and the resulting storage utilization are in acceptable ranges. The dynamic behavior of the proposed methods is also investigated.

Quotient Reduction Method

Let \( I = \{ x_1, x_2, \ldots, x_n \} \) denote the set of keys to be stored in a hash table, \( m \) the number of buckets in the table, and \( b \) the bucket size. Without loss of generality, we assume that the set \( I \) is in sorted order. We use \([p, q]\) to denote the interval \([p, p+1, p+2, \ldots, q-1, q]\). (\( p < q \).) Let \(\alpha\), the ratio \(n/mb\), often expressed as a percentage.

Sprugnoli defined functions of the form \( h(x) = \lfloor (x+s)/N \rfloor \), \( s \) and \( N \) are constants, as Quotient Reduction hashing functions. He gave a method of determining \( s \) and \( N \) for any given set key set \( I \) such that the Quotient Reduction hashing function defined is perfect. His algorithm gives the optimal values of \( s \) and \( N \) to achieve the maximum storage utilization possible. His algorithm is restricted to the special case of \( b = 1 \) (and could handle 10 - 15 element sets only). In the following we develop algorithms which can handle key sets having a few hundred elements, when the bucket size is large. It forms a part of the Remainder Reduction method, a practical method of finding direct perfect hashing functions, discussed in the next section. We first define a set of admissible increments of the Quotient Reduction hashing function for a given \( I \) and \( b \). The bounds for the quotient \( N \) is analyzed, and algorithms QR and Efficient QR (EQR) are then introduced.

Admissible increments

In order that a hashing function is perfect for the key set \( I \) the keys \( x_i \) and \( x_{i+b} \) should not hash into the same bucket, for \( 1 \leq i \leq n-b \). For given a quotient \( N \), an integer \( t_i \) is said to be an admissible increment of key \( x_i \) if \( \lfloor (x_i + t_i)/N \rfloor < \lfloor (x_i + t_i + kN)/N \rfloor \), \( 1 \leq i \leq n-b, t_i \in J_i \). In other words, \( t_i \) is a translation value which adjusts \( x_i \) and \( x_{i+b} \) into two different intervals \([p-1]N \ldots pN-1\) and \([q-1]N \ldots qN-1\], where \( p \) and \( q \) are integers, \( p < q \). We use \( J_i \) to denote the set of all admissible increments of \( x_i \). In all, there are \( n-b \) sets of admissible increments for a given \( I \) and \( b \).

Proposition 1

There exists a quotient reduction perfect hashing function for a given \( I, b, \) and \( N \) if and only if the set of admissible increments \( J_i, 1 \leq i \leq n-b \), have at least one element in common: \( \{ n-b \} \cap \bigcap_{i=1}^{n-b} J_i \neq \emptyset \).

Proof

The “only if” part is true, since if \( h(x) = \lfloor (x+s)/N \rfloor \) is a perfect hashing function for the key set, then for every \( x_i \), \( \lfloor (x_i + s)/N \rfloor < \lfloor (x_{i+b} + s)/N \rfloor \). (Since, no more than \( b \) keys hash into any bucket.) Thus, \( s \in J_i \) for \( i = 1,2,\ldots,n-b \), and hence \( s \in \bigcap J_i \).

The “if” part can be proved as follows. Let \( s \) be a common element of all \( J_i \)'s, \( i = 1,2,\ldots,n-b \). For every \( x_i \), \( \lfloor (x_i + s)/N \rfloor < \lfloor (x_{i+b} + s)/N \rfloor \). And hence no more than \( b \) keys hash into any bucket under the hashing function \( h(x) = \lfloor (x+s)/N \rfloor \).

Proposition 2

For a given \( I, b, \) and \( N \), the sets of admissible increments \( J_i, 1 \leq i \leq n-b \), are given by

\[
J_i = \{ t_i | t_i = u_i - x_{i+b} \mod N \} 
\]

(2.1)

where \( \delta_i = x_{i+b} - x_i \) and \( u_i \) defined by \( 0 \leq u_i < \sum_{j=i}^{i+b-1} \delta_j \).

(Note: the idea of \( u_i \) becomes more clear in example 1.)

Proof

The \( J_i \) defined above is equivalent to

\[
J_i = \{ t_i | t_i = u_i - x_{i+b} + kN \text{ and } 0 \leq u_i < \sum_{j=i}^{i+b-1} \delta_j \} \}
\]

(2.2)

where \( k \) is a constant.

Let \( t_i \) be an integer, and let \( k \lfloor (x_{i+b} + t_i)/N \rfloor \). \( t_i \) satisfies the condition \( x_{i+b} + t_i \geq kN \).

Proposition 3

If \( t_i \) satisfies the condition \( x_i + t_i < kN \), then \( x_i \) and \( x_{i+b} \) hash into two different buckets under the hashing function \( h(x) = \lfloor (x_i + t_i)/N \rfloor \). Hence, any \( t_i \) satisfying (2.3) and (2.4) is an admissible increment of \( x_i \). Consider an element of the set \( J_i \) as defined, \( (u_i - x_{i+b} + kN) \). This element satisfies condition (2.3), since \( x_{i+b} + (u_i - x_{i+b} + kN) \).
Related Content

Methodology of Schema Integration for New Database Applications: A Practitioner’s Approach
Joseph Fong, Kamalakar Karlapalem, Qing Li and Irene Kwan (1999). Journal of Database Management (pp. 2-18).
www.igi-global.com/article/methodology-schema-integration-new-database/51209?camid=4v1a

Antecedents of Online Game Dependency: The Implications of Multimedia Realism and Uses and Gratifications Theory
www.igi-global.com/chapter/antecedents-online-game-dependency/63667?camid=4v1a

Designing Document SQL (DSQL): An Accessible yet Comprehensive Ad-Hoc Querying Frontend for Query
www.igi-global.com/article/designing-document-sql-dsql/37211?camid=4v1a

Assuring Database Integrity
www.igi-global.com/article/assuring-database-integrity/51075?camid=4v1a