Short Communications

Programming Techniques

Comment on Bell’s Quadratic Quotient Method for Hash Code Searching

Leslie Lamport
Applied Data Research, Inc., Wakefield, Massachusetts

Key Words and Phrases: hashing, hash code, scatter storage, calculated address, clustering, search, symbol table, keys, table look-up

CR Categories: 3.74, 4.9

In a recent paper [1], James R. Bell gave a method for resolving collisions in a hash coded table search which generalized the quadratic search method of W. D. Maurer [2]. However, he ignored an important anomaly of Maurer’s method, as well as one singular case.

In the quadratic search method, table locations \( k + ai + bi^2 \) modulo \( p \) are examined sequentially for \( i = 0, 1, 2, \ldots \); where \( p \) is the table size which is a prime number, \( k \) is the hash code of the entry’s key, and \( a \) and \( b \) are constants with \( b \neq 0 \). The same table location is examined for \( i = i_1 \) and \( i = i_2 \); \( i_1 \neq i_2 \), if and only if

\[
k + ai_1 + bi_1^2 \equiv k + ai_2 + bi_2^2 (p)
\]

which is true if and only if

\[
(i_2 - i_1)[a + b(i_1 + i_2)] = 0.
\]

This in turn is true if and only if

\[
i_1 = i_2 \quad \text{or} \quad i_1 + i_2 = (p - a)/b,
\]

where \( /p \) denotes division in the field of integers modulo \( p \).

It is easy to see from this that the procedure examines \( (p + 1)/2 \) different table locations, if \( p > 2 \). However, it will examine them all on its first \( (p + 1)/2 \) tries only if \( a = 0 \). Therefore, for an efficient procedure we should take

\[
a \equiv “\text{some fixed constant”} + Q/p^2
\]

\[
b \equiv Q/p^2,
\]

where \( Q \) is a function of the entry’s key, and \( p > 2 \). (Although Bell states that \( a \) is a constant, this is not true for the algorithm he describes.) The algorithm considers the search a failure (table “full” and entry not in it) when it returns to the first location examined. This occurs on the \( i \)th try, where \( 1 < i < p + 1, i - 1 = (p - a)/b \). At that time, every other location which it has examined will
have been examined twice. Clearly, half the table will be searched only if we replace the "fixed constant" by a number congruent to \(-Q/2\). However, even if this is done, there is still the problem that when \(Q = 0\), only one table location is examined!

To correct these problems, replace steps (3) and (4) of Bell’s algorithm with:

(3) Initialize \(A\) with \(C\), where \(C\) is defined below.
(4) Increment \(A\) by \(2Q\).

For this algorithm, we have \(a = Q + C\), \(b = Q\). We must then choose \(C\) so that \(C = -Q\) if \(Q \neq 0\) and \(C \neq -Q\) if \(Q = 0\). The algorithm will then search \((p + 1)/2\) locations if \(Q \neq 0\), and will search all \(p\) locations if \(Q = 0\).

The trouble with this algorithm is that it requires testing for \(Q = 0\), which means performing an extra division. A seemingly possible way out is to observe that if \((p - a)/b = -j\), \(b \neq 0\), then the algorithm searches \(j\) fewer locations before it starts re-examining locations. We can then try to choose \(C\) so that we get \(j\) to be small, thereby examining nearly half the table before repeating. However, this requires that we make \(C = -(j + 1)Q\). There does not appear to be any simple algorithm for choosing a \(C\) satisfying this congruence for a small \(j\) when \(Q \neq 0\), and choosing \(C \neq -Q\) when \(Q = 0\). It seems that the division is necessary.

The corrected version of Bell’s algorithm still contains a gross inefficiency. For \(Q \neq 0\), it decided that the search is a failure after \(p\) tries, instead of the necessary \((p + 1)/2\) tries. This is easily corrected by changing the criterion for failure.

In summary, Bell’s algorithm requires a correction which adds an extra division to the initialization procedure. This must be considered in evaluating its efficiency. Bell’s table comparing the efficiency of his method with that of Maurer’s indicates that this extra initialization cost is justified only if checking a single entry is a relatively time consuming operation.

**REFERENCES**


**REPLY BY BELL.** Before discussing Lamport’s comment in detail, let us consider the correct observation on which it is based: Although any quadratic search (including quadratic quotient) hits half of the table entries, sometimes some entries are hit twice while others are hit once.

In other words, \(K + a_i + b_i^2\) may not have maximum period for an arbitrary \(a\) and \(b\). The author proves that forcing \(a\) to zero will guarantee maximal period.

A much simpler constraint is to let the constant in step (3) of the original algorithm be zero. Then \(h_1(K) = R + (Q/2)i + (Q/2)^2\) and we first return to our original hash address when \(R = R + (Q/2)i + (Q/2)^2\), that is, when \(i = -1\) or \(i = 0\) or \(Q = 0\).

The first two cases state that \(h(K)\) has a maximum periodicity. The third case is the degenerate one where the quotient is congruent to zero. We could use a division to spot the degenerate case. But by adopting the suggestion of paragraph 3 of Section 3c of the original article we can use \((Q \land \text{lowbitmask}) + 1\) in lieu of \(Q\) to guarantee that this case does not occur.

Lamport has taken a more complicated approach.