1978

Pseudo chaining in Hash Tables

Halatsis, Constantin

Nuclear Research Center "Democritos"

http://hdl.handle.net/11728/6514

Downloaded from HEPHAESTUS Repository, Neapolis University institutional repository
Open addressing and chaining are the two fundamental collision-resolution methods in hashing or scatter storage techniques. Many hash table storage and retrieval algorithms have been developed and are successfully employed in assemblers, compilers, file-handling routines, and other applications. Most of these techniques have been thoroughly surveyed [6, 7]. Recent developments, especially in open addressing, may be found in [1–5].

The open addressing approach has a single hash table \( T \) of \( n \) contiguous cells for item storage. There are several variations of open addressing where stored items do not always remain in the cell they initially occupy, but may be displaced when other items are stored (see for example [4]). These variations improve upon what we now describe as basic open addressing in several ways, but they are not considered further here.

Let \( H \) denote the hashing function being used in basic open addressing. When an item \( x \) with key value \( x.key = k \) is to be stored, its home address \( H(k) \) is computed and if the cell whose address is \( H(k) \) is not occupied the item is stored there and the process terminates; otherwise a sequence of cells is scanned until an unoccupied cell is found (in which case \( x \) is stored) or until the sequence is exhausted (in which case the hash table is effectively full).

The sequence of addresses or indices that specify the sequence of cells to be scanned is called the probe sequence. The probe sequence for a key value \( k \) depends in general upon \( k \) and is determined by a collision-resolution, probe-sequence generating function, \( p(i, k), i = 0, 1, 2, \ldots, n - 1. \) The hashing function being used, \( H, \) is called in \( p \) so that \( p(0, k) = H(k). \) The main difference between distinct basic open addressing schemes is in the collision-resolution function which they use.

To retrieve an item, given a key value \( k, \) the probe sequence for \( k \) is followed and each referenced cell is tested to see if it contains an item with key value \( k. \) If so, an item with key value \( k \) has been found, but if a “vacant” address is found during probing it is certain that no item with key value \( k \) has been stored and the search may terminate. An address is said to be vacant if its corresponding cell is not occupied even by a now-deleted item. Initially all table addresses are vacant.

The collision problem is solved with chaining techniques by maintaining items which have identical home addresses in a chain using simple list-processing techniques. Each cell of a chained hash table has an item field and a link field. There are two basic chaining techniques: Direct chaining and indirect chaining. The former uses a single hash table while the latter uses a separate table for storing items which collide (overflow) due to the occurrence of identical home addresses.

In direct chaining, when an item \( x \) is to be stored, its home address, \( H(k), \) is computed. If \( H(k) \) is unoccupied the item is stored there, otherwise an empty cell has to be found by any technique whatever. If \( H(k) \) is occupied by a key with an identical home address, then item \( x \) is stored in the empty cell found and it is linked into the appropriate chain; otherwise the item in cell \( H(k) \) is moved into the empty cell found, its chain is updated, and item \( x \) is stored in its home location. Such item displacements are avoided by the indirect chaining approach.

In either of these chaining methods the probe sequence, at item retrieve time, is directly given by the link fields of the cells in the appropriate chain.

The pseudochaining method presented here combines characteristics of open addressing and chaining. The scheme employs open addressing for storing items, but each cell of the hash table has an associated link field as in direct chaining, and whenever the first overflow of a given home location occurs, the link field of the home location may be used to store the “address” of
that overflow item. This link value may then be used when retrieving that item to short circuit the successive probes which were used when storing that item. Actually, unlike a chaining method, the address of the overflow item is not stored but the probe number, \( j \), is stored instead. That is, if \( p(i, k) \) is the underlying probe-sequence generating function, and if the item in question is stored in cell \( p(j, k) \) on the \((j + 1)\)th probe, then \( j \) is stored in the link field of the home location, and it may be used to recreate the address \( p(j, k) \) when needed. Indeed, the link field need not be so large as to hold the probe number of the first overflow item, but may hold only a divisor of the probe number. This is another reason for storing probe numbers rather than addresses: The number of probes needed to store an item is generally smaller than an index to the hash table, so the link fields can be quite modest in size.

Pseudochaining employs a probe-sequence generating function, \( p_a \), which is adaptive (i.e., which is not a true function, but rather a process with memory). The memory involved is the array of link fields of the hash table. An ordinary nonadaptive probe-sequence generating function of a basic open addressing scheme is used in defining \( p_a \).

In the following sections algorithms for storing, retrieving, and deleting items using pseudochaining are given. Also, the performance of the method in terms of the mean number of probes needed to retrieve an item is determined theoretically. It is found that the performance of pseudochaining is between that of direct chaining and that of the best known basic open addressing method. Since this performance over basic open addressing is attained at the expense of extra space for the link fields, the advantage area of pseudochaining is determined along the lines developed in [1].

**Description of Pseudochaining**

Pseudochaining is described below by means of procedures for storage, retrieval and deletion. The various global symbols used are:

\[ T[1:n] \] (integer key, data): \( T \) is the hash table. Each cell of \( T \) holds an item having a key field and a data field. Presumably \( n \) is at least as large as the number of items to be stored. We assume that all key values, taken as integers, are positive. (This assumption is not at all vital, but merely establishes a particular situation so that explicit algorithms may be given.) Then \( T_j.key = 0 \) means that \( T_j \) is empty and \( T_j.key < 0 \) means that \( T_j \) is empty but has been occupied by a now-deleted item. Initially \( T_j.key = 0 \) for \( 1 \leq j \leq n \).

\[ L[1:n] \]: \( L \) is an array of \( s \)-bit integers and is used as the list of link fields for \( T \). Initially \( L_i \) is zero for \( 1 \leq i \leq n \).

\[ \text{gbd}(i, w) \]: \( \text{gbd}(i, w) \) is the greatest divisor of \( i \) less than \( w \) and relatively prime to \( n \).

\[ p(i, k) \]: \( p \) is a collision-resolution, probe-sequence generating function. \( p \) takes a key value, \( k \), and a nonnegative integer, \( i \), as arguments and returns an index to \( T \). We have \( 1 \leq p(i, k) \leq n \). The sequence \( p(0, k), p(1, k), \ldots, p(n-1, k) \) is the probe sequence for \( k \). \( i \) is the number of previous probes which have been made following this probe sequence.

The hashing function being used, \( H \), is invoked in \( p \) so that \( p(0, k) = H(k) \). Thus there is no explicit reference to \( H \) in the programs below. The adaptive probe sequence generating function \( p_a \) used by pseudochaining is defined in terms of \( p \) as follows:

\[ p_a(i, k) = p(i * \text{max}(1, L_{p(i,k)}) \mod n, k) \]

Whenever an item \( x \) with key value \( k \) is stored in cell \( T_{p_a(i,k)} \), under the condition that \( T_{p(i,k)} \) is occupied and \( L_{p(i,k)} = 0 \), (i.e., item \( x \) is a first overflow item), then \( L_{p(i,k)} \) is set equal to \( \text{gbd}(i, 2^k) \). As a result, the value of \( p_a(i, k) \) is now changed, and this "adapted" probe-sequence generator will be used henceforth.

Store (item \( x \) (integer key, data)):

begin integer \( i, h, j \);
\[ \text{comment the item } x \text{ is stored in table } T; \]
\[ i \leftarrow 0; \]
\[ h = p(0, x.key); \]
\[ \text{while } T_{p(i, h)} \neq 0 \text{ and } L_{p(i, h)} > 0 \text{ do} \]
\[ \text{if } i = n - 1 \text{ then table-full else } i \leftarrow i + 1; \]
\[ T_j.key = x; \]
\[ \text{if } j = 0 \text{ and } L_h = 0 \text{ then } L_h \leftarrow \text{gbd}(i, 2^h) \]
end Store.

Retrieve (integer \( k, j \)):

begin integer \( i, h \);
\[ \text{comment } k \text{ is an item key value, } j \text{ is returned as an index to } T \text{ such that } T_j \text{ is an item with key value } k, \text{ or else } j \text{ is set to } -1 \text{ if no such item exists in } T; \]
\[ i \leftarrow 0; \]
\[ h = p(0, k); \]
\[ \text{while } T_{p(i, h)} \neq 0 \text{ and } L_{p(i, h)} = 0 \text{ do} \]
\[ \text{if } T_j.key = 0 \text{ or } i = n - 1 \text{ then } (j \leftarrow -1; \text{return}) \text{ else } i \leftarrow i + 1 \]
end Retrieve.

Delete (integer \( k \)):

begin integer \( j \);
\[ \text{comment a cell of } T, \text{ holding an item with key value } k \text{ (if any such cell exists) has its key field reset to } -1 \text{ and thus marked as deleted.}; \]
\[ i \leftarrow 0; \]
\[ h = p(0, k); \]
\[ \text{while } T_{p(i, h)} \neq 0 \text{ and } L_{p(i, h)} = 0 \text{ do} \]
\[ \text{if } T_j.key = 0 \text{ or } i = n - 1 \text{ then } (j \leftarrow -1; \text{return}) \text{ else } i \leftarrow i + 1 \]
end Delete.

Observe that when an item is deleted the corresponding link field is left intact. This implies that a large amount of insertion-deletion activity will eventually revert pseudochaining to open addressing.

**Performance Analysis**

In the following discussion, pseudochaining is analyzed for the case where the base probe sequence function, \( p(i, k) \), used to define \( p_a(i, k) \), defines an independent nonduplicating random probing called uniform probing [6].
Table I. Mean Number of Probes Required by Pseudochaining for Successful Lookup (n = 997).

<table>
<thead>
<tr>
<th>α</th>
<th>0.20</th>
<th>0.40</th>
<th>0.60</th>
<th>0.70</th>
<th>0.80</th>
<th>0.90</th>
<th>0.95</th>
<th>0.99</th>
<th>1.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>s = 2</td>
<td>1.102</td>
<td>1.219</td>
<td>1.392</td>
<td>1.529</td>
<td>1.740</td>
<td>2.107</td>
<td>2.595</td>
<td>3.749</td>
<td>5.134</td>
</tr>
<tr>
<td>s = 3</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.439</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 4</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 5</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 6</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 7</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 8</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 9</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
<tr>
<td>s = 10</td>
<td>1.101</td>
<td>1.211</td>
<td>1.347</td>
<td>1.438</td>
<td>1.607</td>
<td>1.934</td>
<td>2.313</td>
<td>3.306</td>
<td>4.458</td>
</tr>
</tbody>
</table>

Table II. Advantage of Pseudochaining over Uniform Probing for Several Values of α and s/t (n = 997, s = 10).

<table>
<thead>
<tr>
<th>α</th>
<th>0.01</th>
<th>0.05</th>
<th>0.10</th>
<th>0.15</th>
<th>0.20</th>
<th>0.25</th>
<th>0.30</th>
<th>0.40</th>
</tr>
</thead>
<tbody>
<tr>
<td>s/t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td>0.013</td>
<td>0.008</td>
<td>0.002</td>
<td>-0.003</td>
<td>-0.007</td>
<td>-0.012</td>
<td>-0.015</td>
<td>-0.016</td>
</tr>
<tr>
<td>0.40</td>
<td>0.062</td>
<td>0.048</td>
<td>0.032</td>
<td>0.018</td>
<td>0.005</td>
<td>0.006</td>
<td>0.016</td>
<td>0.006</td>
</tr>
<tr>
<td>0.60</td>
<td>0.170</td>
<td>0.136</td>
<td>0.098</td>
<td>0.067</td>
<td>0.039</td>
<td>0.015</td>
<td>0.006</td>
<td>-0.041</td>
</tr>
<tr>
<td>0.70</td>
<td>0.266</td>
<td>0.210</td>
<td>0.152</td>
<td>0.104</td>
<td>0.063</td>
<td>0.028</td>
<td>0.002</td>
<td>-0.052</td>
</tr>
<tr>
<td>0.80</td>
<td>0.419</td>
<td>0.320</td>
<td>0.222</td>
<td>0.146</td>
<td>0.084</td>
<td>0.032</td>
<td>0.012</td>
<td>0.078</td>
</tr>
<tr>
<td>0.90</td>
<td>0.701</td>
<td>0.483</td>
<td>0.297</td>
<td>0.164</td>
<td>0.062</td>
<td>0.018</td>
<td>0.083</td>
<td>0.184</td>
</tr>
<tr>
<td>0.95</td>
<td>0.978</td>
<td>0.577</td>
<td>0.285</td>
<td>0.096</td>
<td>0.040</td>
<td>0.144</td>
<td>0.226</td>
<td>0.349</td>
</tr>
<tr>
<td>0.99</td>
<td>1.353</td>
<td>0.388</td>
<td>-0.089</td>
<td>-0.356</td>
<td>-0.535</td>
<td>-0.665</td>
<td>-0.765</td>
<td>-0.911</td>
</tr>
<tr>
<td>1.00</td>
<td>1.378</td>
<td>-0.084</td>
<td>-0.637</td>
<td>-0.935</td>
<td>-1.125</td>
<td>-1.261</td>
<td>-1.368</td>
<td>-1.520</td>
</tr>
</tbody>
</table>

It is evident that storing an item x using pseudochaining based on uniform probing requires the same expected number of probes as uniform probing alone, even though the "step size" depends upon \( L_{x(key)} \).

Retrieving an item also requires the same expected number of probes as does uniform probing, except when the item being retrieved is a first overflow item (i.e., an item which when stored, hashes initially to an already occupied cell, \( i \), whose link field \( L_i \) is 0). In this case, item x is stored in some unoccupied cell after some number, \( j \), of additional probes; but then \( L_i \) is set to \( gbd(j, 2^s) \). When the item x is later retrieved, the use of \( L_i \) in \( P_a \) will result in the saving of some probes. In general, something less than \( j - 1 \) probes will be saved. Thus pseudochaining will, in some cases, save some probes over those required by uniform probing.

Let \( S_{k+1} \) be the number of probes saved over uniform probing for the retrieval of the \((k + 1)\)st item, \( x_{k+1} \), which is stored. Then

\[
E(S_{k+1}) = \sum_{i \leq k+1} f_{k+1} P_{i,k+1} (i - g(i))
\]

where

\[
f_{k+1} = P(x_{k+1} \text{ is a first overflow item}),
\]

\[
P_{i,k+1} = P(x_{k+1} \text{ is stored using } i \text{ probes} | x_{k+1} \text{ is a first overflow item}),
\]

\[
g(i) = \text{(number of probes used to retrieve } x_{k+1} | x_{k+1} \text{ is a first overflow item stored with } i \text{ probes}).
\]

Now, \( f_{k+1} \) may be obtained as follows.\(^1\) Observe that \( f_1 = 0 \), and that \( n f_k \) is the number of nonempty cells in \( T \) whose \( L \)-fields are zero with \( k - 1 \) items stored. Then this number changes only when \( x_k \) is not a first overflow item, in which case \( n f_{k+1} = n f_k + 1 \). Thus,

\[
f_{k+1} = f_k(n f_k + (1 - f_k)(n f_k + 1)) \quad \text{for } k \geq 1.
\]

Solving this recursion equation, we have:

\[
f_{k+1} = 1 - ((n - 1)/n)^k \quad \text{for } 0 \leq k \leq n - 1.
\]

Now, \( P_{i,k+1} = 0 \) and for \( i > 1 \), \( P_{i,k+1} \) is easily seen to be the probability that storing \( x_{k+1} \) requires \( i - 2 \) further probes of occupied cells following the initial probe at an occupied cell, terminated by a final probe at an unoccupied cell. But, since independent nonduplicating random probing is used, we have:

\[
P_{i,k+1} = \frac{k - 1}{n - 1} \frac{k - 2}{n - 2} \cdots \frac{k - i + 2}{n - i + 2} \left( \frac{k - k + 1}{n - i + 1} \right)
\]

or,

\[
P_{i,k+1} = \frac{n(n - k)}{k} \times \frac{k!}{n^i}
\]

where the underscore notation is defined by

\[
a^b = \prod_{0 \leq i < b} (a - i)
\]

Also, for \( i > 1 \),

\[
g(i) = \frac{i - 1}{gbd(i - 1, 2^s) + 1}
\]

since this is the number of probes which will be required to fetch \( x_{k+1} \) following its storage using \( i \) probes in total.
Now, let \( C_{nk} \) be the number of probes needed to retrieve a present item chosen randomly from among \( k \) items stored using pseudochaining based upon uniform probing in a table of size \( n \). Then the mean value of \( C_{nk} \) is given by:

\[
E(C_{nk}) = E(U_{nk}) - \frac{1}{k} \sum_{j=1}^{k} E(S_{j+1})
\]  

(5)

where \( E(U_{nk}) \) is the corresponding mean number of probes using uniform probing only. \( E(U_{nk}) \) is given by:

\[
E(U_{nk}) = \frac{n + 1}{k} \sum_{j=1}^{k} \frac{1}{n + 1 - j} \approx \frac{n}{k} \log \frac{n}{n-k}
\]  

(6)

\( E(C_{nk}) \) may be computed from (5) for various values of \( k \), \( n \), and \( s \), the size of the link fields. Table I lists \( E(C_{nk}) \) for \( n = 997 \) and various values of the hash table load factor, \( \alpha = k/n \), and \( s = 2, 3, \ldots, 10 \). For comparison \( E(U_{nk}) \) (the mean number of probes using uniform probing), \( E(B_{nk}) \) (the cost of Brent's method of open addressing [4]), and \( E(D_{nk}) \) (the cost for direct chaining) are also included in Table I. Sample values of \( E(C_{nk}) \) have also been computed for larger values of \( n \). Dependence of \( E(C_{nk}) \) on \( n \) was not notable.

It can be seen from Table I that \( E(C_{nk}) \), considered as a function of \( s \), becomes nearly constant long before \( s \) approaches \([\log_n n]\). This implies that the size of the link fields, \( L[1:n] \), can be smaller than that needed to hold an arbitrary address value in \( \{1, 2, \ldots, n\} \). As a result, pseudochaining has a space advantage over chaining methods, which require full-sized link fields. At the same time pseudochaining has a time advantage over basic open addressing schemes. It is also seen that pseudochaining is comparable to Brent's method.

This advantage may actually vanish if the link field space can be used as extra item space by an open addressing scheme with a lower load factor, \( \alpha' \), instead. If \( \alpha \) denotes the load factor of a pseudochained hash table, then \( \alpha' \) for a corresponding "linkless" table is given in [1] as \( \alpha' = \alpha/(1 + (s/t)) \) where \( t \) is the number of bits required to store an item and \( s \) the number of bits of the link field. Thus, in order to determine the region of actual advantage of pseudochaining over uniform probing, it is more appropriate to compare \( E(U_{nk}) \) to \( E(C_{nk}) \), where \( r = n(1 + (s/t)) \) is the number of items which can be accommodated by uniform probing if all space is used for item storage.

Table II lists the advantage \( A = E(U_{nk}) - E(C_{nk}) \) of pseudochaining over uniform probing for various values of \( s/t \) and load factor \( \alpha \) of the pseudochained hash table, and for \( n = 997 \) and \( s = [\log_2 n] \). It is seen that, for very small values of \( s/t \), pseudochaining is always advantageous over uniform probing. For \( 0.05 \leq s/t \leq 0.25 \), \( A \) attains a positive peak value at some value of \( \alpha \). For \( s/t \geq 0.3 \) pseudochaining is never advantageous over uniform probing \( (A < 0) \). The positive peak value of \( A \) is retained for values of \( s \) less than \([\log_2 n]\), as it can be deduced from Table I. Obviously, pseudochaining is always advantageous over basic open addressing when a link field can be accommodated in a cell at no extra cost (i.e., when the key length is several bits shorter than the memory cell length).

Acknowledgment. The authors wish to express their gratitude to the referees for their valuable comments on the initial manuscript and their help in improving the quality of the final presentation.

Received February 1976; revised August 1977

References
1. Bays, C. A note on when to chain overflow items within a direct-access table, Comm. ACM 16, 1 (Jan. 1973), 46–47.