An Alternative Analysis of the Open Hashing Algorithm

NINGPING SUN∗, RYOZO NAKAMURA∗∗

Kumamoto National College of Technology,
2659-2 Suya, Nishigoshi,Kikuchi, Kumamoto 861-1102, JAPAN

∗∗Kumamoto University, Kumamoto 860-8555, JAPAN

Abstract: - Hashing is one of the most important techniques for sorting and searching. Two problems that how to design a good hash function and how to deal with the collision must be resolved when hashing is applied. First, we provide an evaluation system of hashing algorithm using some popular hash functions and show some evaluation results. Continuously, we present an analysis of the probability that collision occurs. Introducing the probability distribution of frequency of access on each individual key in the separate chain into the analysis of search cost, we propose a mathematical analysis to exactly analyze and evaluate the performance of open hashing algorithm. Some interesting test results obtained from the proposed formulae are also presented.

Key-Words: - Open hash algorithm, Analysis of algorithm, Hash function, Average search cost, Probability distribution

1 Introduction

Hashing is an important technique widely used to provide fast access to information stored in either main memory or external storage devices. For a given key \( x \) and a table with \( M \) elements, by computing a hash function \( h(x) \) which is some arithmetical calculation on \( x \), the location of \( x \) and the associated data in the table is to result. However, when two keys hash to a same value, the collision happens. How to design a good hash function and approach the collisions are important when the hash is applied.

To evaluate some typical hash functions we built an evaluation system expressed in section 2. This system has a friendly interactive user environment. Some evaluation results are provided. In this analysis, we give an approach to evaluate the probability that collision occurs. Open hashing algorithm is one of the most well-known methods used to resolving the collision. The performance of open hashing algorithm depends on the secondary search within the separate chains. By which order we insert keys into a list and the inserted keys could be located at which position, and how both of the order and the position will be involved in the performance of open hashing algorithm are of most interest. In the analysis of the average search cost of algorithms, the probability distribution of frequency of access to each key plays an crucial role. We propose a mathematical analysis to exactly evaluate the average search cost of the open hashing algorithm according to the probability distribution of the frequency of access to each key. Some experimental results are provided as well.

2 The Evaluation System of Hashing Algorithm

Hashing is one of the best applied search algorithms. Meanwhile it is very difficult for those users who do not know or understand well how to design an appropriate hash function and choose which method to avoid collisions. To help the general hash programming we developed an evaluation system of hashing algorithm by Java. The system provides a friendly interactive user environment including multi-frame such as system control frame shown in Fig.1(a), result display frame of the open hashing of Fig.1(b), result display frame of the closed hashing of Fig.1(c), textual describing frames of the hash functions algorithm and the corresponding C programs.
User chooses the size of hash table, the kind of the hash function, and the kind of hashing method by menus or buttons, inputs the data file with file name on the frame of Fig. 1(a). As the results of evaluation, the hash table occupation ratio of six hash functions are displayed in the result field of Fig. 1(a), the linked lists of the open hashing are shown on the frame of Fig. 1(b), and the rehashing results of the closed hashing are displayed like Fig. 1(c).

In the evaluation system of hashing algorithm we discuss six typical hash functions such as (1)“figure analysis” that chooses appropriate figures of a numeral key to fit the size of hash table, and results the hash value with the selected figures. (2)”cut and fold” that divides a numeral key into some parts then adds these parts to result the hash value. (3) “remainder of division” that divides a key by the size of hash table, the remainder of division is the hash value. (4)"dummy random” that computes \((a \times key + c) \mod p\), \(a\) and \(c\) are arbitrary constants, \(p\) is the size of hash table.(5) “perfect hashing” that works out the sum of the length of the key and the associated value of the first and the last character of the key and all of code values of all characters of the key, then divides the sum by the size of hash table. The remainder of the division is the hash value. (6)"bit computing ” does bit shift and bit logical calculation of the key.

Some experimental results in Table 1 tell us these hash functions have similar performance in the occupation ratio of the hash table.

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Some evaluation results of the hash table occupation ratio of six hash functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>n=50</td>
<td>HASH FUNCTIONS</td>
</tr>
<tr>
<td>figure analysis</td>
<td>0.9</td>
</tr>
<tr>
<td>cut and fold</td>
<td>0.6</td>
</tr>
<tr>
<td>remainder of division</td>
<td>1</td>
</tr>
<tr>
<td>dummy random</td>
<td>1</td>
</tr>
<tr>
<td>perfect hashing</td>
<td>1</td>
</tr>
<tr>
<td>96 bit mix</td>
<td>1</td>
</tr>
<tr>
<td>32 bit mix</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n=100</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH FUNCTIONS</td>
</tr>
<tr>
<td>figure analysis</td>
</tr>
<tr>
<td>cut and fold</td>
</tr>
<tr>
<td>remainder of division</td>
</tr>
<tr>
<td>dummy random</td>
</tr>
<tr>
<td>perfect hashing</td>
</tr>
<tr>
<td>96 bit mix</td>
</tr>
<tr>
<td>32 bit mix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n=150</th>
</tr>
</thead>
<tbody>
<tr>
<td>HASH FUNCTIONS</td>
</tr>
<tr>
<td>figure analysis</td>
</tr>
<tr>
<td>cut and fold</td>
</tr>
<tr>
<td>remainder of division</td>
</tr>
<tr>
<td>dummy random</td>
</tr>
<tr>
<td>perfect hashing</td>
</tr>
<tr>
<td>96 bit mix</td>
</tr>
<tr>
<td>32 bit mix</td>
</tr>
</tbody>
</table>

3 The Probability that Collision Occurs

The hash table has \(M\) positions indexed by \(0,1,\ldots,M-1\) and it contains the headers of \(M\) linked lists. The elements of the \(i\)-th list are a set of records whose value of hash function \(h(x)\) is equal to \(i\), namely, where the elements have the same hash
We assume that \( n \) keys are uniformly mapped into the hash table by a hash function, and each of the \( M^n \) possibilities that hash sequences \( a_1, a_2, \ldots, a_n \) \((0 \leq a_j < M)\) occur is equally likely, where \( a_j \) is the hash table address where the \( j \)-th key to be hashed \( P_{nk} \) denotes the probability that the number of keys in a list is equal to \( k \) \((0 \leq k \leq n)\). There are \( \binom{n}{k} \) possibility to hash \( k \) keys to an identical value \( a_j \), therefore \( k \) is a set of \( j \) \((1 \leq j \leq n)\). Meanwhile, it has \((M - 1)^{n-k}\) ways to assign value to the other \( a'\)'s.

Apply Poisson approximation to the binomial distribution,

\[
P_{nk} = \binom{n}{k} \left( \frac{1}{M} \right)^k \left( 1 - \frac{1}{M} \right)^{n-k} = \frac{\alpha^k e^{-\alpha}}{k!} + o(1)
\]

where \( \alpha = n/M \) that is the load factor of hash table. \( k \) is the number of key in a list. There is no any key or there is only one key in a list means the collision does not happen there, whereas, \( k > 1 \) tells us there are more than one keys hash to the list that is linked to a address of hash table. We give some results from (1) in Fig. 2, 3 and 4 for \( k \geq 0 \) according to some different \( M \).

We see that when \( n \leq M \) the \( P_{nk} \) has its maximum value on \( k=1 \). When \( n > M \) the maximum of \( P_{nk} \) is on the point \( k = \left\lfloor n/M \right\rfloor \), where \( n/M \) is called as the load factor of hash table. The value of \( P_{nk} \) on the point \( n=M \) and \( k=1 \) is the maximum of all of the values, it is about 37\% for an arbitrary \( M \). When \( n > M \), the larger the \( n \), the smaller the \( P_{nk} \). Furthermore, even though \( n \) keys are hashed uniformly to the hash table, for a given \( n \), with the increasing of \( M \), the number of elements in the set of possible \( k \) is decreasing a little, which is shown in Table 2.
The above conclusions can be proved by following formula,

\[
\frac{P_{nk}}{P_{nk-1}} = 1 + \frac{(n+1) \cdot \frac{1}{M} - k}{k \left(1 - \frac{1}{M}\right)}
\]  

(3)

When \( k < (n+1) \cdot \frac{1}{M} \) the value of \( \frac{P_{nk}}{P_{nk-1}} \) is increasing, when \( k > (n+1) \cdot \frac{1}{M} \) the value of \( \frac{P_{nk}}{P_{nk-1}} \) is decreasing. Hence, the maximum value of \( \frac{P_{nk}}{P_{nk-1}} \) is on the point between \( k = (n+1) \cdot \frac{1}{M} - 1 \) and \( k = (n+1) \cdot \frac{1}{M} \), which means for a given \( n \) and \( M \) the number of keys in a list is most possibly between \( (n+1) \cdot \frac{1}{M} - 1 \) and \( (n+1) \cdot \frac{1}{M} \). Fig.5 shows the results as well.

4 The Average Search Cost of the Open Hashing algorithm

We now concentrate on how the data structure of separate chain is involved in the performance of the open hashing algorithm. Suppose \( n \) keys are inserted into a list with the order \( a_1, a_2, \ldots, a_n \) and
these keys are inserted successively at the head of a list. We call this case Case 1. In a list with \(k\) keys of
Case 1 there are \(\binom{n-i}{j-1}\) possibilities to distribute the \((n-i)\) keys into the front \((j-1)\) positions, and
\(\binom{i-1}{k-j}\) ways to distribute the \((i-1)\) keys into the rear \((k-j)\) positions of the list. The probability that the
\(i\)-th key will be located in the \(j\)-th position from the head of a list with \(k\) keys can be expressed as follows,
\[
\binom{n-i}{j-1} \binom{i-1}{k-j} \binom{n-1}{k-1}
\]  
(4)

where \(\binom{n-1}{k-1} = \sum_{j=1}^{k} \binom{n-i}{j-1} \binom{i-1}{k-j}\)  
(5)

On the other hand, the case that a new key is inserted at the tail of a list is called as Case 2. In Case 2, the probability that the \(i\)-th key will be located in the \(j\)-th position from the head of a list with \(k\) keys becomes
\[
\binom{i-1}{j-1} \binom{n-i}{k-j} \binom{n-1}{k-1}
\]  
(6)

where \(\binom{n-1}{k-1} = \sum_{j=1}^{k} \binom{i-1}{j-1} \binom{n-i}{k-j}\)  
(7)

It is well-known that the time required to solve a problem is one of the most important measures in evaluating an algorithm. This time is also called the search cost of an algorithm. The search cost of an algorithm is the product of the number of probes and the frequency of access to each key. If the frequency is uniform, the average search cost of the open hashing is independent of the insertion order of keys. However, if the access frequency is not uniform, the insertion order of keys and the position where keys are located then play a significant role in the analysis of average search cost of algorithms.

In our analysis, \(\rho_i\) denotes the probability of frequency of the access to the \(i\)-th key and \(\gamma_{kj}\) denotes the probability that the \(j\)-th key from the head of a list with \(k\) keys will be probed.

For Case 1, from (4) and (5) the probability \(\gamma_{kj}\) can be expressed as following (8),
\[
\gamma_{kj} = \frac{\sum \binom{n-i}{j-1} \binom{i-1}{k-j} \rho_i}{\binom{n-1}{k-1}}
\]  
(8)

where \(1 \leq j \leq i \leq n, 1 \leq j \leq k \leq n\).

For Case 2, from (6) and (7) the probability \(\gamma_{kj}\) becomes
\[
\gamma_{kj} = \frac{\sum \binom{i-1}{j-1} \binom{n-i}{k-j} \rho_i}{\binom{n-1}{k-1}}
\]  
(9)

where \(1 \leq j \leq i \leq n, 1 \leq j \leq k \leq n\).

Both (8) and (9)
\[
\sum_{j=0}^{k} \gamma_{kj} = 1, \text{ for } \gamma_{k0} = \begin{cases} 1, k = 0 \\ 0, k > 0 \end{cases}.  
\]  
(10)

When the probability of frequency of access to each key is uniform, \(\rho_i = 1/n\) \(1 \leq i \leq n\), both Case 1 and Case 2, \(\gamma_{kj}\), the probability that the \(j\)-th key from the head of a list with \(k\) keys will be probed becomes \(1/k\). Whereas, the more different the probability of frequency of access to each individual key, the more different the probability \(\gamma_{kj}\).

In the internal hashing with separate chaining scheme the keys with the same hash address are linked one by one in the list, from above analysis, for a successful search the average and the variance of the search cost, \(S_n\) and \(V_n\), can be expressed as the following (11) and (12),
\[
S_n = \frac{\sum_{j=1}^{n} \sum_{k=0}^{n} \gamma_{kj} P_{nk}}{\sum_{k=0}^{n} P_{nk}}  
\]  
(11)

and
\[ V_n = \frac{\sum_{j=1}^{n} j^2 \sum_{k=1}^{n} \gamma_{kj} P_{nk}}{\sum_{k=1}^{n} P_{nk} - S_n^2} \]  

(12)

where \( \gamma_{kj} \) comes from (8) for Case 1 and from (9) for Case 2. Fig.6 shows some results of (11) for Case 1. Fig.7 shows ones of (11) for Case 2. In our experimental tests, \( M=50 \), three kinds of probability distribution of access frequency are used:

1. The probability of frequency of access to each key is uniformly, \( \rho_i = 1/n \).

2. Two probabilities \( \rho_i = 2^{-i} \) and \( \rho_i = \frac{c}{i} \), where
   \[ c = \frac{1}{H_n}, \quad H_n = \sum_{k=1}^{n} \frac{1}{k}. \]  
   They are reduced half and harmonically according to the insertion order of keys respectively.

3. Two exponential distribution of \( i \), \( \rho_i = e^{-\alpha i} \) and \( \rho_i = \alpha e^{-\alpha i} \), where \( \alpha = n/m \).

From our experimental results, when the frequency of access to each key is uniformly \( \gamma_{kj} \) of Case 1 is as same as \( \gamma_{kj} \) of Case 2, which equals \( 1/k \), for a successful search their averages of the search cost also are the same. However, if the \( \gamma_{kj} \) is variable according to the insertion order of key and the position where the inserted key is located, and the frequency of access to each key is not equally likely, the average of search cost of the two cases will be different, namely, different \( S_n \).

5 Conclusion

From our analysis to the probability that collision occurs we see that when keys are hashed equally likely to the hash table by a hash function, the increase of size of the hash table will not decrease the collision as much as to be desired. To analyze the performance of the open hashing algorithm we have clarified the relationship between the insertion order of keys and position that key is located. Taking account of the frequency of access to an individual key the evaluation formulæ of average and variance of the search cost have been proposed as well. Our experimental results say it is possible to exactly evaluate the average search cost of the open hashing algorithm by the proposed analysis.

References: