Regular Paper

An Alternative Analysis of Linear Probing Hashing with Buckets

Tsuyoshi Itokawa† and Ryozo Nakamura†

In this paper, we propose a mathematical analysis of average search cost of linear probing hashing for external searching on secondary storage devices in consideration of both the bucket size and the frequency of access on each key. Then we propose the formulae to evaluate exactly the average and variance of the search cost and show some results of the numerical tests.

1. Introduction

Hashing method is an important and useful technique for implementing dictionaries, which requires constant time for inserting, searching and deleting a record on the average. The key, which uniquely identifies its record, is mapped into the hash table by using a hash function. In this case, there will probably be two or more keys hashed to one identical location of the table. Such an occurrence is called a collision. The techniques for handling the collision are chiefly classified into two categories: chaining technique and open addressing technique. Linear probing hashing is the simplest form of open addressing techniques.

Hashing is also widely used to provide fast access to information stored on external secondary storage devices as well as main memory. Thus separate chaining and linear probing hashing techniques lend themselves well to external searching on secondary storage devices such as disks and drums. When we consider the differences in access characteristics between main memory and external storage devices,it is the nature of secondary storage devices that the time to find a block and read it into main memory is very large compared with the time to process the data in that block. Therefore in the proposed analysis, the number of probes denotes only the number of accesses in the same way as the traditional analyses did [1),3)].

Here we review the history of analysis of the linear probing hashing algorithm. In 1962, Schay and Spruth first proposed an approximate analysis based on the assumption that the number of records stored in a bucket is 1 and all keys are uniformly accessed. In 1973, Knuth proposed the exact analysis based on the same assumption as Schay and Spruth did and also proposed the approximate analysis for external searching on secondary storage devices. In 1998, Viola and Poblete have proposed an alternative analysis for external searching in consideration of the bucket size [4)]. This analysis was based on the Robin Hood collision resolution in which the collided records are rearranged, but it had not been considered the frequency of access on each key.

By the way, the time required to solve a problem is one of the most important measures in evaluating an algorithm. The search cost is defined generally as the product of the number of probes and the frequency of access on a key. Provided that the frequency of access is uniform, the average search cost of linear probing hashing is independent of the order of inserting keys. However if the frequency of access on each key is not uniform, the inserting order plays a crucial role.

As mentioned above, the traditional analyses are unable to evaluate the search cost even if the probability of the frequency of access on a key is given. Taking account of the frequency of access on a key, it is necessary to clarify the relationship between the inserting order of a key and its locating position.

Therefore, in this paper we present an exact discrete analysis of the average search cost in successful search for linear probing hashing for external searching on secondary storage devices in consideration of both the bucket size and the frequency of access on each key. Then we propose the formulae to evaluate the average and variance of the search cost in a successful search and show some results of the numerical tests.

Before we get into analyzing the search algorithm of linear probing hashing, we will try to look more closely linear probing hashing model. On the generalized model each position of the hash table, called bucket, contains some records

† Department of Computer Science, Faculty of Engineering, Kumamoto University

which are accessed in unit time, and an each area in the bucket for storing each record is called a slot, the number of slots in a bucket is called bucket size.

For analyzing linear probing hashing algorithm, at first we assume the following three items.

(1) The hash table size is M and the bucket size is b, where the table with size M is indexed by positions 0 to$M-1$. The records are grouped into buckets, so that b records are accessed from the external memory each time. Here the maximum size of b is dependent of operating systems, but 4,096 bytes is typical.

(2) The keys are uniformly mapped into the hash table from position 0 to position $M-1$ by using a hash function $h(K)$ for a given key K. Thus assume that when n keys are inserted into the table by the hash function h, M^n hash address sequences are generated equally likely. Namely we assume that each of the M^n hash address sequences

$$a_1\ a_2\ \cdots\ a_i\ \cdots\ a_n,\quad (0 \le a_i < M)$$

is equally likely, where a_i denotes the initial hash address of the i-th key inserted into the table.

(3) Linear probing hashing uses the cyclic probe sequence as follows,

$$h(K), h(K)-1, h(K)-2, \ldots \ldots, 0, M-1, \ldots, h(K)+1.$$

The above probe sequence denotes a permutation of positions $\langle 0, 1, 2, \cdots, M-1\rangle$.

For example, inserting a record whose key value is K, at first position $h(K)$ in the hash table is decided, which ranges from 0 to $M-1$. If position $h(K)$ has one or more empty slots, then the record will be stored there. However, if there is no empty slot in position $h(K)$, then position $h(K)-1$ will be probed for the next candidate to put it. If there are one or more empty slots in the position, the record will be stored. But if there is no empty slot in position $h(K)-1$ then the same procedure is repeated until either finding empty slot or finding the table to be full.

Next we define two functions to analyze the algorithm under assumption that n keys are uniformly scattered to the hash table. First let $f_b(M,n)$ be the number of hash address sequences such that position 0 of the table will have at least one empty slot after n keys have been scattered, secondly let $g_b(M,n,h)$ be the number of hash address sequences such that position 0 of the table has at least one empty slot, positions 1 through h occupied and position $h+1$ has at least one empty slot after n keys have been scattered. The above mentions are argued, provided that the circular symmetry of linear probing implies that position 0 is empty just as often as any other position.

2. Traditional Analysis

In the traditional analysis [1] the bucket size b is fixed to 1, and $f_1(M,n)$ has been derived as follows,

$$f_1(M,n) = M^n\left(1-\frac{n}{M}\right),\quad (M \ge n > 0), \tag{1}$$

and from the relationship between the function f_1 and the function g_1, $g_1(M,n,h)$ has been given by the following formula

$$g_1(M,n,h) = \binom{n}{h} f_1(h+1,h) \times f_1(M-h-1, n-h),\quad (M \ge n \ge h \ge 0). \tag{2}$$

Here let $P_{n,k}$ be the probability that exactly $k+1$ probes will be needed when the $(n+1)$ st key is inserted. Using the function g_1, $P_{n,k}$ has been derived as follows,

$$P_{n,k} = \frac{1}{M^n}\big(g_1(M,n,k) + g_1(M,n,k+1) + \cdots + g_1(M,n,n)\big),\quad (M \ge n \ge k \ge 0). \tag{3}$$

In search algorithm, the probe sequence for searching key K traces the same probe sequence that the insertion algorithm examined when key K was inserted. It is also evident that the keys searched by k probes are ones inserted by just k probes among $(n-k+1)$ keys on and after the k-th key inserted.

In the analysis of considering the frequency of access on each key, we let ρ_i be the probability that the i-th key inserted will be retrieved, where $\sum_{i=1}^{n}\rho_i = 1$. Therefore the probability that exactly k probes will be needed for searching a key after all n keys are scattered in the table is given as follows [2],

$$\rho_k P_{k-1,k-1} + \rho_{k+1}P_{k,k-1} + \cdots + \rho_n P_{n-1,k-1},\quad (k = 1, 2, \ldots, n). \tag{4}$$

Finally the average and variance of the successful search cost can be expressed by the following formulae [2], where the search cost is de-

fined as the product of the number of probes and the frequency of access on a key.

$$S_n = \sum_{k=0}^{n-1}(k+1)\sum_{j=k}^{n-1}\rho_{j+1}P_{j,k} \tag{5}$$

$$V_n = \sum_{k=0}^{n-1}(k+1)^2\sum_{j=k}^{n-1}\rho_{j+1}P_{j,k} - {S_n}^2 \tag{6}$$

3. Proposed Analysis

In the traditional analysis, provided that the bucket size is one, both the function f_1 and the function g_1 have been simply derived, and the exact analysis of the algorithm has been proposed for internal searching on main memory. But under the consideration of the bucket size b as a parameter, neither function f_b nor g_b are derived completely.

In this section we will derive the functions f_b and g_b taking account of the bucket size b, and then propose the exact analysis of the average search cost of linear probing hashing for external searching on secondary storage devices in consideration of both the bucket size and the frequency of access of a key.

3.1 Function f_b

In this section, we will derive the function $f_b(M,n)$ mentioned previously. Now let k_i be the total number of keys scattered in positions 0 through i of the table under the condition that position 0 has at least one empty slot. Thus, the number of keys scattered in position 0 is represented by k_0, and the number of keys in positions 0 and 1 is k_1, and so on. Therefore $k_i - k_{i-1}$ means the number of keys stored in position i.

Here we consider the number of ways of scattering n keys to a hash table of size M with the bucket size b, where $Mb > n$. In the first, the number of ways of choosing k_0 keys from n keys is $\binom{n}{k_0}$, in this case k_0 takes the value from 0 to $b-1$, since position 0 of the table has at least one empty slot. Therefore the number of ways to distribute the keys in position 0 is $\sum_{k_0=0}^{b-1}\binom{n}{k_0}$. Next, $k_1 - k_0$ keys chosen from the remaining $n-k_0$ keys are distributed to position 1 of the table. If both the positions 0 and 1 of the table are occupied by the keys except only one empty slot in position 0, then k_1 becomes equal to $2b-1$. Therefore the number of ways to distribute the keys in position 1 of the table is $\sum_{k_1=k_0}^{2b-1}\binom{n-k_0}{k_1-k_0}$, and so on up to position $\lambda-1$, where $\lambda = \lceil n/b \rceil$ means $\min\{\lambda | \lambda \geq n/b, \text{integer}\lambda\}$.

Here when position 0 has just one empty slot, there are $b\lambda - 1 - k_{\lambda-1}$ empty slots in positions 0 through $\lambda-1$, since the total number of keys scattered from position 0 to position $\lambda-1$ in the table is $k_{\lambda-1}$.

Consequently, even if the remaining $n-k_{\lambda-1}$ keys are scattered at random to positions which follow position $\lambda-1$ of the table, the number of the overflow keys after position λ is at most $n - k_{\lambda-1} - b$, and it is less than or equal to $b\lambda - 1 - k_{\lambda-1}$. The above arguments prove that position 0 of the table has at least one empty slot. Therefore the number of ways in which $n-k_{\lambda-1}$ keys can be scattered arbitrarily after position $\lambda-1$ of the table is $(M-\lambda)^{n-k_{\lambda-1}}$.

From the above discussion, the predefined function $f_b(M,n)$ can be expressed by

$$\begin{aligned} f_b(M,n) = &\sum_{k_0=0}^{b-1}\sum_{k_1=k_0}^{2b-1}\cdots\sum_{k_{\lambda-1}=k_{\lambda-2}}^{\lambda b-1}\binom{n}{k_0} \\ &\times\binom{n-k_0}{k_1-k_0}\cdots\binom{n-k_{\lambda-2}}{k_{\lambda-1}-k_{\lambda-2}} \\ &\times(M-\lambda)^{n-k_{\lambda-1}} \\ = &\sum_{k_0=0}^{b-1}\sum_{k_1=k_0}^{2b-1}\cdots\sum_{k_{\lambda-1}=k_{\lambda-2}}^{\lambda b-1} n!(M-\lambda)^{n-k_{\lambda-1}} \\ &\Big/\big\{k_0!(k_1-k_0)!\cdots(k_{\lambda-1}-k_{\lambda-2})! \\ &\times(n-k_{\lambda-1})!\big\}. \end{aligned} \tag{7}$$

3.2 Function g_b

In the traditional analysis, assumed that the bucket size is 1, and the function g_1 has been expressed in a simple form as Eq. (2). In this section we shall derive the function $g_b(M,n,h)$ defined previously in consideration of the bucket size b. Therefore whenever any position of the table has empty, we need to consider all the cases from only one empty slot to b empty slots.

At first we consider the case that position 0 has an empty slot, and positions 1 through $h-1$ occupied and position h has an empty slot. Here let w denote the total number of keys be capable of storing in positions from 0 to $h-1$. Then w takes the value from $(h-1)b$ to $hb-1$. Furthermore, let i_k be the number of keys scattered from position k to position $h-1$.

Under the above assumption that an overflow from position h to position $h-1$ can not occur, i_{h-1} at position $h-1$ can take the value from b to w. If i_{h-1} will be b, no key overflows from position $h-1$ to position $h-2$, but when i_{h-1} will take w, the keys distributed in position $h-1$ fill up all positions from position 1 to position

$h-1$. Thus the number of ways to distribute the keys in position $h-1$ is $\sum_{i_{h-1}=b}^{w} \binom{w}{i_{h-1}}$.

Next at position $h-2$, $(i_{h-2}-i_{h-1})$ keys are chosen from $(w-i_{h-1})$ keys. At this time, since $i_{h-1}-b$ keys overflow from position $h-1$, the number of $(i_{h-2}-i_{h-1})$ keys which will be scattered in position $h-2$ will be the value from $b-(i_{h-1}-b)$ to $w-i_{h-1}$, as the result, i_{h-2} has the value from $2b$ to w. Thus the number of ways to distribute the keys to both position $h-2$ and position $h-1$ is

$$\sum_{i_{h-2}=2b}^{w} \sum_{i_{h-1}=b}^{w} \binom{w-i_{h-1}}{i_{h-2}-i_{h-1}} \binom{w}{i_{h-1}}.$$

Such process will be repeated till position 1.

Here we let $t_b(h,w)$ be the number of hash address sequences such that there are some empty slots in position 0, and positions 1 through $h-1$ occupied when w keys have been stored in positions from 0 to $h-1$, then $t_b(h,w)$ can be presented as follows,

$$\begin{aligned} t_b(h,w) &= \sum_{i_1=(h-1)b}^{w} \sum_{i_2=(h-2)b}^{w} \cdots \sum_{i_{h-1}=b}^{w} \binom{w-i_2}{i_1-i_2} \\ &\quad \times \binom{w-i_3}{i_2-i_3} \cdots \binom{w-i_{h-1}}{i_{h-2}-i_{h-1}} \binom{w}{i_{h-1}} \\ &= \sum_{i_1=(h-1)b}^{w} \sum_{i_2=(h-2)b}^{w} \cdots \sum_{i_{h-1}=b}^{w} w! \\ &\quad \Big/ \Big\{ (w-i_1)!(i_1-i_2)! \cdots \\ &\quad \cdots (i_{h-2}-i_{h-1})!\, i_{h-1}! \Big\} \\ &\quad (h>1, hb-1 \geq w \geq (h-1)b), \quad (8) \end{aligned}$$

provided that if $h=1$, $t_b(1,w)=1$ $(0 \leq w \leq b-1)$.

The function g_b already mentioned, which is the number of hash address sequences such that there are some empty slots in position 0 and positions 1 through h are occupied and there are some empty slots in position $h+1$ after n keys have been scattered, is given as follows,

$$\begin{aligned} g_b(M,n,h) &= \sum_{i=0}^{b-1} \binom{n}{hb+i} t_b(h+1, hb+i) \\ &\quad \times f_b(M-h-1, n-hb-i) \\ &\quad (Mb > n \geq hb \geq 0). \quad (9) \end{aligned}$$

For example, when $n=3$ keys will be inserted by order a_1, a_2, a_3 in the table of size $M=3$ and the bucket size $b=2$, the above arguments are shown concretely in **Table 1**. At first, $M^n(=3^3)$ hash address sequences are shown in second column from the left side. The column denoted Table shows the situation that 3 keys (a_1, a_2, a_3) are stored in each position of the hash table. The rightmost column shows the each case of the function g_2, which is calculated by t_2 and f_2 denoted in the second and third columns from the right side.

3.3 Average Search Cost

In consideration of the bucket size b, the probability $P_{n,k}$ of Eq. (3) is rewritten as follows,

$$\begin{aligned} P_{n,k} &= \frac{1}{M^n}(g_b(M,n,k) + g_b(M,n,k+1) + \cdots \\ &\quad \cdots + g_b(M,n,n)). \quad (10) \end{aligned}$$

For example, when inserting the fourth key into the hash table of size 3 with the bucket size 2, the probabilities $P_{3,0}$,$P_{3,1}$ and $P_{3,2}$ are calculated by the result of Table 1 as follows.

$$\begin{aligned} P_{3,0} &= \frac{1}{3^3}(g_2(3,3,0) + g_2(3,3,1) \\ &\quad + g_2(3,3,2) + g_2(3,3,3)) \\ &= \frac{20}{3^3} \\ P_{3,1} &= \frac{1}{3^3}(g_2(3,3,1) + g_2(3,3,2) + g_2(3,3,3)) \\ &= \frac{7}{3^3} \\ P_{3,2} &= \frac{1}{3^3}(g_2(3,3,2) + g_2(3,3,3)) = 0 \end{aligned}$$

Now we consider both the bucket size b and the frequency of access on each key $\rho_i, (i=1,2,\ldots,n)$, in this case the keys searched by k probes are ones inserted by just k probes on and after the $((k-1)b+1)$ st key inserted. Therefore Eq. (4) could be rewritten as follows,

$$\begin{aligned} &\rho_{(k-1)b+1} P_{(k-1)b,k-1} + \rho_{(k-1)b+2} P_{(k-1)b+1,k-1} \\ &\qquad + \cdots + \rho_n P_{n-1,k-1} \\ &\qquad (k=1,2,\cdots,\lceil n/b \rceil). \quad (11) \end{aligned}$$

Finally we consider the number of access (probes) as a random variable and its probability distribution in order to derive the evaluation formulae of the search cost. Then we can derive the average search cost S_n and the variance V_n as follows.

$$S_n = \sum_{k=0}^{\lambda-1} (k+1) \sum_{j=kb}^{n-1} \rho_{j+1} P_{j,k} \quad (12)$$

$$V_n = \sum_{k=0}^{\lambda-1} (k+1)^2 \sum_{j=kb}^{n-1} \rho_{j+1} P_{j,k} - {S_n}^2 \quad (13)$$

here $\lambda = \lceil n/b \rceil$.

4. Numerical Tests

With the proposed Formulae (12) and (13) we can evaluate the average search cost in accordance with any probability distribution of the frequency of access on a key.

Table 1 Feature that 3 keys of insertion order $a_1\ a_2\ a_3$ are mapped into the hash table ($M = 3, b = 2, n = 3$).

No.	Hash address sequence	Table 0	Table 1	Table 2	$t_2(h,w)$	$f_2(M,n)$	$g_2(M,n,h)$
1	0 0 0	a_1a_2		a_3			
2	1 1 1	a_3	a_1a_2		$t_2(2,3)$	$f_2(1,0)$	$g_2(3,3,1)$
3	2 2 2		a_3	a_1a_2	$t_2(1,0)$	$f_2(2,3)$	$g_2(3,3,0)$
4	0 0 1	a_1a_2	a_3				
5	0 0 2	a_1a_2		a_3			
6	0 1 0	a_1a_3	a_2				
7	0 2 0	a_1a_3		a_2			
8	1 0 0	a_2a_3	a_1				
9	2 0 0	a_2a_3		a_1			
10	1 1 0	a_3	a_1a_2		$t_2(2,3)$	$f_2(1,0)$	$g_2(3,3,1)$
11	1 1 2		a_1a_2	a_3	$t_2(2,2)$	$f_2(1,1)$	$g_2(3,3,1)$
12	1 0 1	a_2	a_1a_3		$t_2(2,3)$	$f_2(1,0)$	$g_2(3,3,1)$
13	1 2 1		a_1a_3	a_2	$t_2(2,2)$	$f_2(1,1)$	$g_2(3,3,1)$
14	0 1 1	a_1	a_2a_3		$t_2(2,3)$	$f_2(1,0)$	$g_2(3,3,1)$
15	2 1 1		a_2a_3	a_1	$t_2(2,2)$	$f_2(1,1)$	$g_2(3,3,1)$
16	2 2 0	a_3		a_1a_2	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
17	2 2 1		a_3	a_1a_2	$t_2(1,0)$	$f_2(2,3)$	$g_2(3,3,0)$
18	2 0 2	a_2		a_1a_3	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
19	2 1 2		a_2	a_1a_3	$t_2(1,0)$	$f_2(2,3)$	$g_2(3,3,0)$
20	0 2 2	a_1		a_2a_3	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
21	1 2 2		a_1	a_2a_3	$t_2(1,0)$	$f_2(2,3)$	$g_2(3,3,0)$
22	0 1 2	a_1	a_2	a_3	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
23	0 2 1	a_1	a_3	a_2	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
24	1 0 2	a_2	a_1	a_3	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
25	2 0 1	a_2	a_3	a_1	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
26	1 2 0	a_3	a_1	a_2	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$
27	2 1 0	a_3	a_2	a_1	$t_2(1,1)$	$f_2(2,2)$	$g_2(3,3,0)$

In the numerical tests let us assume the following two probability distributions of the frequency of access on a key.

(1) The probability of the frequency of access on each key is equally likely, called "*Uniform*", the probability ρ_i holds the relation $\rho_i = 1/n$ $(i = 1, 2, \ldots, n)$.

(2) The probability of the frequency of access on a key is reduced harmonically according to the order of inserting a key, typically called "*Zipf's law*", the probability ρ_i holds the relation $\rho_i = C_n/i$ $(i = 1, 2, \ldots, n)$, where $C_n = 1/H_n$ and H_n is a harmonic number.

Figures 1 and **2** show the average search costs obtained by the proposed Formula (12), with uniform and Zipf's law probability distributions for the frequency of access on a key, respectively. Two graphs in each figure are shown to account for the influence of bucket size. Thus the dotted line shows the case of the table size $M = 60$ and the bucket size $b = 5$, and the solid line shows the case of $M = 30$ and $b = 10$.

We can see the property of linear probing hashing that the average search cost is indeed satisfactory unless the table has gotten very full, but it becomes rapidly worse when the load factor α is close to 1. And we can also see that if the load factor α is fixed where $\alpha = n/Mb$, as the bucket size b becomes larger, the average search cost becomes smaller.

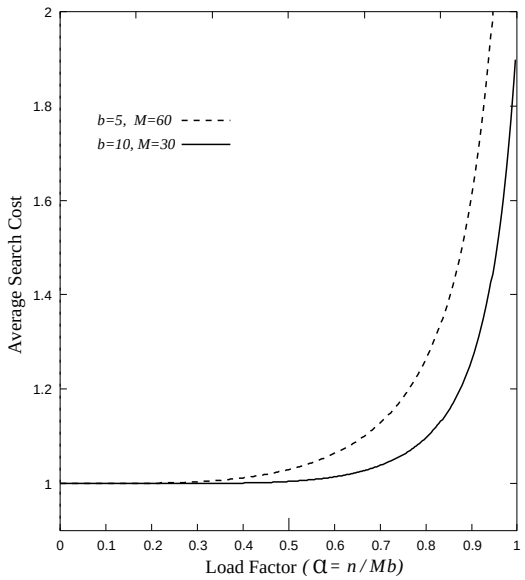

Fig. 1 The average search cost for uniform probing.

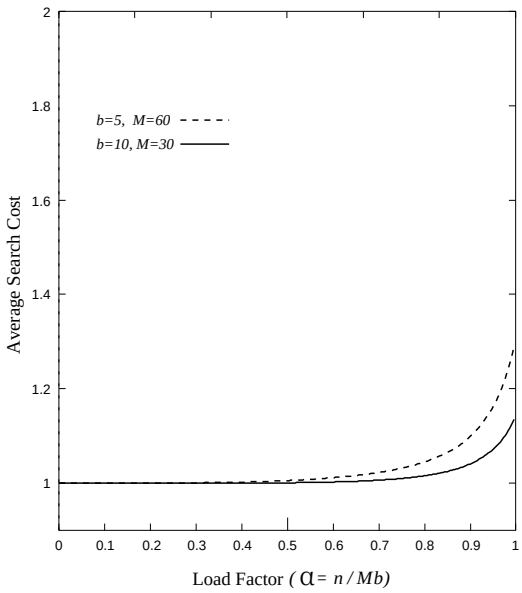

Fig. 2 The average search cost for Zipf's Law.

5. Conclusion

We have analyzed mathematically the average behavior of linear probing hashing for external searching in consideration of both bucket size and the frequency of access on a key.

In the proposed analysis at first when n keys have been scattered into the hash table with size M we have derived both the function $f_b(M, n)$ be the number of the hash address sequences such that position 0 of the table has at least one empty slot, and the function $g_b(M, n, h)$ be the number of the hash address sequences such that position 0 of the table has at least one empty slot, positions 1 through h occupied and position $h + 1$ has at least one empty slot, in consideration of the bucket size b.

Finally we have proposed the formulae to exactly evaluate the average search cost and its variance of linear probing hashing algorithm for external searching on secondary storage devices in consideration of both the bucket size and the frequency of access on a key, and have shown some numerical results from the proposed formulae.

Acknowledgments The authors would like to thank the anonymous referees for their valuable comments to improve the clarity of this paper.

References

1) Knuth, D.E.: *The Art of Computer Programming*, Vol.3, *Sorting and Searching*, Second Edition, pp.513–558, Addison-Wesley, Reading, Mass (1998).
2) Nakamura, R. and Matsuyama, K.: Considerations of the Number of Probes Deliberating the Probability Distribution for the Frequency of Probes, *J. IPS Japan*, Vol.24, No.4, pp.505–512 (1983).
3) Nakamura, R., Sun, N. and Nakashima, T.: A New Analysis of Hashing Algorithm for External Searching, *J. IPS Japan*, Vol.37, No.12, pp.2276–2283 (1996).
4) Viola, A. and Poblete, P.V.: The Analysis of Linear Probing Hashing with Buckets, *Algorithmica*, Vol.21, pp.37–71 (1998).

(Received October 16, 2000)
(Accepted July 2, 2001)

Tsuyoshi Itokawa received the B.E. degree from Kumamoto University in 1993. From 1993 to 1994 he joined Fujitsu Kyushu System Engineering LTD., and received the M.E. degree from Kumamoto University in 1997. He is presently a research associate in Department of Computer Science. His current research interests include the design and analysis of algorithms and data structures.

Ryozo Nakamura received the M.E. degree from Kumamoto University in 1968 and the D.E. degree in computer science from Kyushu University in 1985. From 1968 to 1974, he joined Chubu Electric Power Company. Since 1975 he has joined in Faculty of Engineering of Kumamoto University, and is presently a professor in Department of Computer Science. His current research interests include the design and analysis of algorithms and data structures.