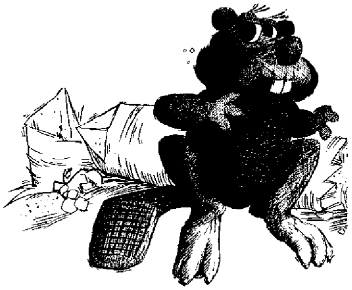

Further Analysis of External Hashing with Fixed-Length Separators

Per-Ake Larson

Technical Report
CS-83-18

July 1983

Further analysis of external hashing with fixed-length separators *

Per-Ake Larson

Data Structuring Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Technical Report CS-83-18

ABSTRACT

External hashing with fixed-length separators is a new file organization that guarantees retrieval of any record in one access. When inserting a new record into the file several pages may have to be modified. In this paper the probability distribution of the number of pages modified is studied. The average and variance can be computed fairly easily. A numerical procedure for computing the full distribution is given as well. As indicated by the numerical results the average number of pages modified is low (less than 2 pages) up to a load factor of 80-85%. If very short separators are used the variance will be quite high, however.

1. Introduction

External hashing with fixed-length separator is a new file organization that guarantees retrieval of any record in one access. It is based on hashing and makes use of a small in-core table to direct the search. The table contains enough information to uniquely determine on which page a record is stored. It is of fixed size and typically very small: one or less than one bit per record stored in the file. The method was introduced by Gonnet and Larson [2] in 1982. Larson and Kajla [5] discuss implementation issues and present results from a test implementation.

To achieve retrieval in one access, additional work is required when inserting a new record into the file. An insertion will occasionally involve some local reorganization of the file. The details will be discussed later, but in essence it amounts to relocating a few existing records to other pages in the file. The cost of an insertion is proportional to the number of pages that must be accessed and modified. In this paper we study the probability distribution of the number of pages modified during an insertion. The average and variance can be computed fairly easily. Furthermore, a procedure for numerically computing the full probability distribution is given.

* This work was supported by Natural Sciences and Engineering Research Council of Canada, Grant No A2460.

The rest of the paper is organized as follows. Section two contains a brief introduction to the basic ideas and algorithms of the new scheme. Section three summarizes the results from previous analyses. It also contains some new results that follow directly from the previous ones. Section four is the main section of this paper: here the probability distribution of the number of pages modified is derived. The final section contains numerical results and a discussion. The procedure for computing the full probability distribution is given in an Appendix.

2. Basic ideas

Assume that n records are to be stored in an external file consisting of m pages (buckets), where each page has a capacity of b records. The load on the file is measured by the *load factor* α, defined as $\alpha = n/(mb)$. In addition to the external file an internally stored *separator table* is needed. This table stores m *separators,* each of length d bits. Separator i, $0 \leq i \leq m-1$, in the table corresponds to page i in the external file.

The method is based on hashing with open addressing (no links or pointers are used). Any open addressing scheme is applicable, but in practice double hashing [4] would probably be used. We assume that given a key k, we can compute its *probe sequence* $H(K) = (h_1(K), h_2(K), \ldots, h_m(K))$. The probe sequence of a key defines the order in which the pages will be checked when inserting or retrieving the record with key K. It is uniquely determined by the key. Every probe sequence is a permutation of the set of page addresses $\{0,1, \cdots, m-1\}$.

For each record we will also need a *signature sequence* $S(K) = (s_1(K), s_2(K), \ldots, s_m(K))$, where each $s_i(K)$ is a d-bit integer. The signature sequence is also uniquely determined by the key. When the record with key K probes page $h_i(K)$ the signature $s_i(K)$ is used, $i = 1,2, \ldots, m$. An implementation of the hashing and signature functions has been presented in [5].

The separator table is used in the following way. Consider a page to which r, $r > b$, records hash. Because there is room for at most b records, at least $r - b$ of them must become overflow records, each one trying the next page in its probe sequence. We consider the r records to be sorted on their current signatures. Records with low signatures are stored on the page, while records with high signatures become overflow records. A signature value uniquely separating the two groups is stored in the separator table. The value stored is the lowest signature occurring among the overflow records. We may not be able to find a separator that gives exactly the partitioning $(b, r-b)$, because signatures and separators are limited to d bits. In that case we try the partitionings $(b-1, r-b+1)$, $(b-2, r-b+2), \ldots, (0, r)$ until we find the first partitioning where the highest signature in the first group is different from the lowest signature in the second group. This only means that a page having overflow records actually may contain fewer than b records.

Example: A page is probed by 5 records with signatures 0001, 0011, 0100, 0100 and 1000. If the page size is 4, we obtain a perfect partitioning; the first 4 records are stored on the page and the separator is 1000. If the page size is 3, a perfect partitioning cannot be obtained. The best one is (2,3): the 2 records with signatures 0001 and 0011 are stored on the page and the separator is 0100.

The separator table is easily updated during insertion. If a record probes a page having a separator that is greater than the signature of the record, the record "belongs" to that page. The record is inserted into the page. If the page was completely filled already, this will force out one or more records (those with the highest signatures) and the separator is updated accordingly.

Note that a record may be relocated after it has been inserted. A record forced out from a page must, of course, be reinserted into some other page, which in turn may force out other records that must be reinstated, etc. Hence insertion of a record may require modification of several pages. The problem studied in this paper is exactly that: how many pages will have to be accessed and modified when inserting a record?

Retrieval of a record requires only one disk access provided that the separator table is in main storage. The address computation algorithm is very simple: follow the probe sequence defined by the key comparing signatures and separators. As soon as a separator strictly greater than the corresponding signature of the key is found, the algorithm terminates and returns the address of the page. If the desired record exists in the file, it must be stored on that page. For a detailed algorithm, see [5].

3. Load and separator distribution

Using an asymptotic model Gonnet and Larson [2] were able to derive the load distribution resulting from external hashing with fixed-length separators. Let b denote the bucket size, α the load factor, d the separator length and set $m = 2^d$. Let $P_j(\lambda)$ denote the probability that a page that has not yet overflowed will contain j records and $Q_j(\lambda)$ the corresponding probability for a page that has overflowed. λ is a parameter whose value must be computed numerically. Gonnet and Larson showed that

$$P_j(\lambda) = e^{-\lambda} \lambda^j / j!$$
$$Q_j(\lambda) = \frac{\lambda^j}{j!}\left(1 - e^{-u} e_{b-j}(u)\right) \sum_{i=0}^{m-1} \left(\frac{i}{m}\right)^j e^{-iu}, \quad j = 0,1, \ldots, b$$

where $u = \lambda/m$ and $e_n(x) = \sum_{k=0}^{n} x^k/k!$. The value of the parameter λ can be found by numerically solving the equation

$$\alpha b = \sum_{j=1}^{b} j(P_j(\lambda) + Q_j(\lambda)) \tag{1}$$

Gonnet and Larson did not explicitly consider the separator distribution, but it follows immediately from their derivation of the load distribution. Let $S_i(\lambda)$ denote the probability that a page will have a separator equal to i, $i = 0,1, \ldots, m$. A page without overflow records is considered to have a separator equal to m. Then it follows that

$$S_i(\lambda) = e^{-iu} \sum_{j=0}^{b} \frac{(iu)^j}{j!}\left(1 - e^{-u} e_{b-j}(u)\right), \; i = 0,1, \ldots, m-1$$
$$S_m(\lambda) = e^{-\lambda} \sum_{j=0}^{b} \lambda^j / j!$$

where $u = \lambda / m$ and λ is defined by eq. (1). The expected separator value will be denoted by $S(\lambda)$, that is,

$$S(\lambda) = \sum_{i=0}^{m} i \; S_i \; (\lambda).$$

In fact, the *jth* term of the sums defining $S_i(\lambda)$ and $S_m(\lambda)$ is the probability that a page will contain j records and have a separator equal to i. In the subsequent analysis we will need the probability that a page is full (contains b records) and has a separator equal to i. Denoting this probability by $R_i(\lambda)$ we have

$$R_i(\lambda) = \left(1 - e^{-u}\right) \frac{(iu)^b}{b!} e^{-iu} \; , \; i = 0,1, \ldots, m - 1$$
$$R_m(\lambda) = e^{-\lambda} \lambda^b \; / \; b!.$$

4. Insertion costs

When inserting a new record into the file, it may hit a page that is already full. In that case it will force out at least one and possibly several records from the page. Each one of these records must be reinserted into some other page, and may in turn force out other records, etc. The separators of all the modified pages must be updated, of course. We are interested in the probability distribution of the total number of pages modified during insertion of a new record, because both the time and space complexity is proportional to the number of pages modified. Modifying a page requires two accesses: one read and one write. The insertion algorithm given in [5] also requires buffer space for each page modified.

Insertion of a new record can be modelled as a branching process, cf. [1, ch XII]. The records forced out by insertion (or reinsertion) of another record are considered to be direct descendants of the record inserted. Because we are using an asymptotic model (infinite file), the records of each "generation" act independently of each other. The original record and each record forced out, must be inserted into some page. Again because we are using an asymptotic model, the probability of two records hitting the same page is zero. Hence the number of modified pages equals the total progeny (including the ancestor) of the branching process. Our first task is to derive the probability generating function for the number of direct descendants of a record (records forced out). From this we can then compute the probability distribution of the total progeny, or, in our terminology, number of pages modified.

If a record is inserted into a non-full page it will have no descendants. If it hits a full page, it will have at least one descendant. How many it will have depends on the separator of the page and the signatures of the records.

Consider the situation when the record to be inserted hits a full page having a separator equal to i , $0 < i \leq m$. The b + 1 signatures (b old ones plus the new one) are all strictly less than i. The new separator will be less than i. Assume that the new separator will be equal to k , $0 \leq k < i$, and that j , $1 \leq j \leq b + 1$, records will be forced out. For this to occur there must be $b + 1 - j$ signatures less than k , j signatures equal to k and none greater than k. The probability of this event is $\binom{b+1}{j} \left(\frac{k}{i}\right)^{b+1-j} \left(\frac{1}{i}\right)^j$. Summation over k then gives us the probability that j records will be forced out:

$$P(j \text{ descendants} \mid i) = \binom{b+1}{j} \left(1/i\right)^{b+1} \sum_{k=0}^{i-1} k^{b+1-j} \; , \; j = 1,2, \ldots, b + 1$$

Let $F_i(s)$ denote the probability generating function for the number of records forced out from a full page (direct descendants), given that its separator equals i :

$$F_i(s) = \sum_{j=1}^{b+1} \binom{b+1}{j} (1/i)^{b+1} s^j \sum_{k=0}^{i-1} k^{b+1-j}$$
$$= (1/i)^{b+1} \sum_{k=0}^{i-1} \{(k + s)^{b+1} - k^{b+1}\}, \quad i = 1,2, \ldots, m$$

A page "attracts" records with a probability proportional to its separator, that is, when a record probes a page with separator i, the probability that it will have a signature less than i is i/m. The fraction of pages that are full and have a separator equal to i is $R_i(\lambda)$. Hence the probability of inserting a record into a full page with separator i is $iR_i(\lambda) \;/\; S(\lambda)$. The factor $S(\lambda)$ merely ensures that the probabilities add up to one.

Let $F(s) = p_0(\lambda) + p_1(\lambda)s + p_2(\lambda)s^2 + \cdots$ denote the generating function for the number of records forced out (direct descendants) when inserting a record. The term $p_0(\lambda)$ will be discussed later; $F(s)$ except this term is

$$F(s) - p_0(\lambda) = \sum_{i=1}^{m} F_i(s)\; i\; R_i(\lambda) \;/\; S(\lambda)$$
$$= \frac{1}{S(\lambda)} \Big\{ \sum_{i=1}^{m-1} i(1 - e^{-u}) \frac{(iu)^b}{b!} e^{-iu} (1/i)^{b+1} \sum_{k=0}^{i-1} \left((k + s)^{b+1} - k^{b+1}\right)$$
$$+ e^{-\lambda} \frac{\lambda^b}{b!} m(1/m)^{b+1} \sum_{k=0}^{m-1} \left((k + s)^{b+1} - k^{b+1}\right) \Big\},$$

which after some manipulation can be simplified to

$$F(s) - p_0(\lambda) = \frac{u^b \; e^{-u}}{b! \; S(\lambda)} \sum_{k=0}^{m-1} \left((k + s)^{b+1} - k^{b+1}\right) e^{-uk}$$

where $u = \lambda/m$. We can now easily compute the probabilities $p_j(\lambda)$ by differentiation: $p_j(\lambda) = F^{(j)}(0) \;/\; j!$. This gives us

$$p_j(\lambda) = \frac{u^b \; e^{-u}}{b! \; S(\lambda)} \binom{b+1}{j} \sum_{k=0}^{m-1} k^{b-j+1} \, e^{-uk}, \; j = 1,2,.., b + 1 \tag{2}$$

The missing term $p_0(\lambda)$ is the probability of inserting the record into a non-full page. It is most easily computed as

$$p_0(\lambda) = 1 - \sum_{j=1}^{b+1} p_j(\lambda). \tag{3}$$

As mentioned earlier the insertion process can be modelled as a branching process. We now have the distribution of number of direct descendants available and are ready to consider the total progeny, that is, in our case, the total number of modified pages. Let $G(z)$ denote the generating function for the total number of modified pages. This function is related to the function $F(s)$ by the equation, cf [1, ch XII],

$$G(z) = z\ F(G(z)) \tag{4}$$

From this equation we can easily compute the total expected number of modified pages, $E(\lambda)$, and the variance, Var (λ), by implicit differentiation:

$$E(\lambda) = G'(1) = 1/(1 - F'(1)), \tag{5}$$

$$\begin{aligned} \mathrm{Var}\ (\lambda) &= G''(1) + G'(1) - G'(1)^2 \\ &= \Big(\frac{F''(1)}{1 - F'(1)} + F'(1)\Big) \ / \ (1 - F'(1))^2 \end{aligned} \tag{6}$$

where $u = \lambda/m$, and

$$F'(1) = \frac{u^b(b+1)}{b!\ S(\lambda)} \sum_{k=1}^{m} k^b\ e^{-uk},$$
$$F''(1) = \frac{u^b(b+1)b}{b!\ S(\lambda)} \sum_{k=1}^{m} k^{b-1}\ e^{-uk}.$$

Making use of the fact that a probability generating function is a (formal) power series we can devise a procedure to numerically compute the full probability distribution of the total number of pages modified. Set $z = G^{-1}(s)$ and insert this into eq. (4). This gives us

$$s = G^{-1}(s)\ F(s)$$

or equivalently

$$G^{-1}(s) = s\ /\ F(s)$$

Because all the function involved are (formal) power series, the coefficients of the power series for $G(z)$ can be computed numerically by

(a) first performing the division $s\ /\ F(s)$, which gives the power series for $G^{-1}(s)$ and
(b) reversing the resulting power series, that is, solving the equation $z = G^{-1}(s)$.

An algorithm for the above computations is given in an Appendix. It outputs the probability distribution of the total number of pages modified. Numerical examples are given in the next section.

Let us finally summarize what computations are required. Given the input parameters (load factor α, page size b and separator length d) the value of the parameter λ has to be computed by numerically solving eq. (1). Then the expected value and the variance of the total number of pages modified can be computed from eq. (5) and (6). If we want the full probability distribution, the distribution of the number of direct descendants must first be computed using eq. (2) and (3), then the desired probabilities can be computed using the algorithm given in the Appendix.

5. Numerical results

No. of	$b = 10$		$b = 20$	
pages	$d = 4$	$d = 8$	$d = 4$	$d = 8$
1	0.7208	0.6695	0.8586	0.7990
2	0.1320	0.2159	0.0541	0.1535
3	0.0625	0.0732	0.0376	0.0350
4	0.0324	0.0259	0.0206	0.0090
5	0.0186	0.0095	0.0111	0.0025
6	0.0113	0.0036	0.0065	0.0007
7	0.0072	0.0014	0.0040	0.0002
8	0.0047	0.0006	0.0025	0.0001
9	0.0031	0.0002	0.0016	0.0000
10	0.0021	0.0001	0.0011	0.0000
11	0.0015	0.0000	0.0007	0.0000
12	0.0010	0.0000	0.0005	0.0000
13	0.0007	0.0000	0.0003	0.0000
14	0.0005	0.0000	0.0003	0.0000
15	0.0004	0.0000	0.0002	0.0000
16	0.0003	0.0000	0.0001	0.0000
17	0.0002	0.0001	0.0001	0.0000
18	0.0001	0.0000	0.0000	0.0000
19	0.0001	0.0000	0.0000	0.0000
20	0.0001	0.0000	0.0000	0.0000
Mean	1.6721	1.5119	1.3602	1.2660
Var.	1.602	0.912	1.193	0.612

Table 1: Probability distribution of the total number of pages modified during an insertion, $\alpha = 0.8$

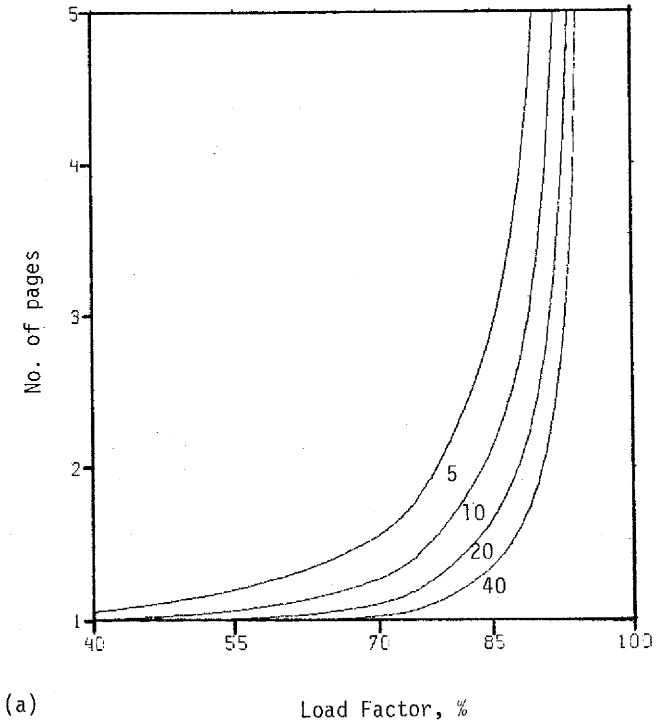

Figure 1(a): Expected number of pages modified during an insertion, $d = 4$, $b = 5,10,20,40$.

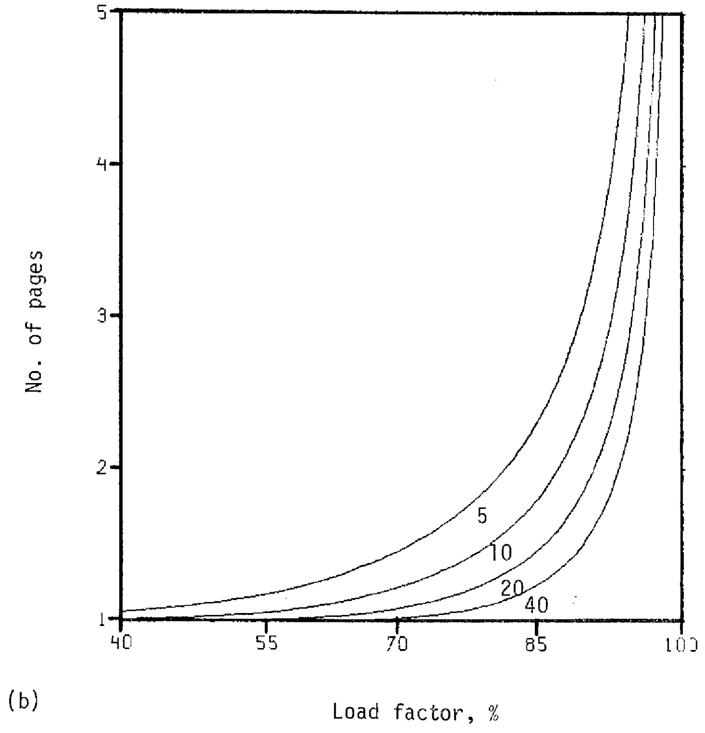

Figure 1(b): Expected number of pages modified during an insertion, $d = 8$, $b = 5,10,20,40$.

Table 1 shows the probability distribution of the number of pages modified during an insertion when the load factor is 0.8. Figure 1(a) and 1(b) show the average as a function of the load factor for four different page sizes. Note the size of the separator table; $\alpha = 0.8$, $b = 20$ and

$d = 4$ is equivalent to as little as 1/4 of a bit per record stored in the file.

The average number of pages modified is quite low, and with high probability an insertion will require modification of only one page. Increasing the page size and/or the separator length will lower the insertion costs as expected. The variance is quite large, however, especially for higher load factors. Increasing the separator length affects the variance more than increasing the page size. This is not unexpected; the main effect of increasing the page size is that there will be fewer full pages (increased probability of hitting a non-full page) while the main effect of longer separators is to reduce the number of records forced if a full page is hit.

There is clearly a trade-off between internal storage and external storage. A load factor of 0.8-0.85 while using one bit or less of internal storage per record seems a realistic goal of the page size is 10 records or more. It forced to use smaller pages, we may have to settle for a lower load factor and/or use longer separators. Pushing the load factor higher than 0.9 does not seem realistic unless very large pages can be used.

References

[1] Feller, W.: An Introduction to Probability Theory and its Applications, Vol. I (3rd ed), John Wiley and Sons, New York, N.Y. 1968.

[2] Gonnet, G.H. and Larson, P.-A.: External hashing with limited internal storage, Report CS-82-38, University of Waterloo, 1982.

[3] Knuth, D.E.: The Art of Computer Programming, Vol. 2 (2nd ed), Addison-Wesley, Reading, Mass., 1981.

[4] Knuth, D.E.: The Art of Computer Programming, Vol. 3, Addison-Wesley, Reading, Mass., 1973.

[5] Larson, P.-A. and Kajla A.: File organization: implementation of a method guaranteeing retrieval in one access, Report CS-83-4, University of Waterloo, 1983.

Appendix

The following algorithm computes the probability distribution of the total number of pages modified, given the probability distribution of the number of records forced out from a page when inserting a record. It is based on the algorithms for division and reversion of power series given in [3, section 4.7]

Input: b page size,
P_j , $j = 0,1, \ldots, b$ probability distribution of the number of records forced out,
ϵ stopping condition; the algorithm terminates when the remaining tail of the distribution is less than ϵ.

Output: Q_n , $n = 1,2, \ldots,$ maxn, probability distribution of the total number of pages modified

Auxiliary arrays: U_k , V_k , $k = 0,1, \ldots,$ maxn

```
$Q_1 := P_0$ ; $U_0 := 1$;
$V_1 := 1$ ; $s := Q_1$;
$n := 1$;

**repeat**
$n := n + 1$ ; $V_n := 0$;
**for** $k := 1$ **to** $n - 1$ **do** $V_n := V_n - V_k P_{n-k}$ **od** ;
$V_n := V_n / P_0$;

**for** $k := 1$ **to** $n - 2$ **do**
  **for** $j := 1$ to $k - 1$ **do** $U_k := U_k - U_j V_{k+1-j}$ **od** ;
  $U_k := U_k - V_{k+1}$;
**od** ;

$U_{n-1} := -n V_n$;
**for** $k := 2$ **to** $n - 1$ **do** $U_{n-1} := U_{n-1} - k U_{n-k} V_k$ **od** ;

$Q_n := U_{n-1} P_0^n / n$;
$s := s + Q_n$;

**until** ( $s \geq 1 - \epsilon$ or $n \geq$ maxn );
```

The algorithm seems to be numerically well-behaved. It was tested (VAX-11, double precision) by comparing the mean and variance computed from the distribution Q_i with those obtained from eq. (5) and (6). With $\epsilon = 10^{-8}$ the means agreed to 7 decimal places and the variances to 5 decimal places for the test cases considered.