International Journal of Computer and Information Sciences, Vol. 3, No. 1, 1974

A Model for Optimizing Indexed File Structures

Ben Shneiderman[1]

Received July 1972; revised November 1973

Multilevel indexes have long been used for accessing records in sorted files. Given the access cost at each level, the total cost of retrieving a record from the file can be substantially reduced by selecting the proper size of the index at each level. Organizations involving a variable number of levels are covered and binary searching is compared to sequential searching.

1. INTRODUCTION

The literature on the organization of file structures is largely qualitative, rather than quantitative, in nature. A number of books and survey articles give thorough discussions of possible indexed file organization strategies (e.g., Refs. 1–4). However, the development of techniques for comparing the efficiency of two structures or search strategies for a particular application is poorly covered. The goal of this paper is to promote a more mathematical approach to file organization analysis. Hopefully, the reader will profit by applying similar techniques to the analysis of other structures.

In the past few years several attempts have been made to develop a theory of data structures. The set-theoretic model of Childs,[5] the relational model of Codd,[6] or the grammar production model of Fleck[7] are concerned with the logical relationship among data items and exclude discussions of implementation efficiency. Rosenberg's algebraic analysis of data graphs[8–10] is more useful but limited by his consideration of only highly uniform structures. The graph-theoretic models of Hsaio and Harary[11] and Earley[12] are able to describe the access paths and are also useful

[1] Department of Computer Science, Indiana University, Bloomington, Indiana.

abstractions of implementations. Consequently, they provide useful models for analyzing efficiency.

2. GRAPH-THEORETIC MODEL

The use of graph theory to represent the access paths enables the designer to compare the efficiency of several possible structures. Heller and Shneiderman[13,14] developed the formal notions for describing data structures as directed graphs called the well-formed list structures (WFLS). For this paper we consider only the unlabeled WFLS's, which are specified as a triple $L = (D, E, G)$, where D is a set of data nodes, E is a nonnull set of entry nodes, and G is a mapping function (G: $D \cup E \rightarrow D$) which describes the access paths for searching the structures. The entry nodes (squares in the figures) represent immediately accessible data items. These might correspond to pointers that are available in the processor storage, file names, or addresses in the processor storage. All searches must begin at the entry nodes and data nodes may not point to entry nodes. The data nodes (circles in the figures) contain the information that is sought and may be physically implemented as words in the processor storage, tape blocks, or disk regions. The graph-theoretic model is a logical view of the data structure for which there may be several physical implementations. The data nodes must be reachable from the entry nodes, that is, there is at least one directed path from at least one of the entry nodes to each of the data nodes. The structure must also be connected, in the sense that if the edges were undirected, then there would be a path between any two nodes. The notions of reachability and connectedness are independent.

The WFLS model is a formalization of our intuitive notions of what data structures should be if they are to be useful in a computer environment. To evaluate a structure, the model must be enhanced by affixing a numerical value to each edge indicating the cost (in units of time or money) of traversing that edge. A further enhancement, not explored in this paper, would be to assign a probability of request for each node. We assume that the probability of request is the same for each node. For a given search strategy we can evaluate the average search cost and other measures such as the variance or worst-case search cost.

3. INDEXED STRUCTURES WITH FIXED NUMBER OF LEVELS

As a first example, we consider indexed sequential files. There are one or more indexes, at one or more levels, containing a variable number of fields each containing the address of a lower-level index or of a record. The

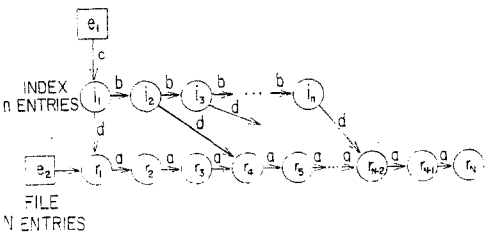

Fig. 1

records are arranged in ascending order by key value. Figure 1 depicts a one-level indexed sequential file. Entry node e_1 is the starting point for searches through the index and is used to access a record with a specified target key. Entry node e_2 is for a sequential search beginning with the first record. Figure 2 contains the graph-theoretic description of this structure. The index is a one-way list which is searched sequentially until the proper range of keys is found, then a search of a sublist of the records is made till the target key is matched. The cost of searching a single field of the index is b, while the cost of searching a single record of the file is a. The cost of accessing the index is c and the cost of entering the file is d. In this case we assume that the probability of request for each record in the file is equal and that the length of each sublist of records is equal. If there are N records in the file and if the length of the index is n, then the length of each sublist of records is N/n. If we are searching for a record with a specified target key, we must, on the average, access the index, search half of the index, enter the file, and search

$$I = (D, E, G)$$

$$D = \{i_j \mid j = 1 \ldots n\}, \{d_j \mid j = 1 \ldots N\}$$

$$E = \{e_1, e_2\}$$

$$Ge_1 = \{i_1\}$$

$$Ge_2 = \{d_1\}$$

$$Gi_j = \begin{cases} \{i_{j+1}, d_{(j-1)*(N/n)+1}\} & j < n \\ \{d_{(j-1)*(N/n)+1}\} & j = n \end{cases}$$

$$Gd_j = \begin{cases} \{d_{j+1}\} & j < N \\ \emptyset & j = N \end{cases}$$

Fig. 2

half of a particular sublist of the file. We assume, of course, that both the index and the records are sorted in ascending order by key values. Thus, the average search cost $\bar{S}$ is

$$\bar{S} = c + \tfrac{1}{2}nb + d + \tfrac{1}{2}(N/n)a$$

The size of the file N is given, but we can control the size of the index n, and thus the size of the sublist of records that is to be searched. The goal is to minimize the search cost by selecting the optimum value for n. We can accomplish this by taking the derivative of S and setting it equal to zero. We note that the conversion from the discrete to the continuous analysis is appropriate since the function is reasonably smooth:

$$d\bar{S}/dn = \tfrac{1}{2}b - \tfrac{1}{2}(N/n^2)a = 0$$

Solving for n, we get

$$n = [(a/b)N]^{1/2}$$

The noninteger result can be rounded up to the ceil(n), to account for possible additions to the file.

If we consider the case where the entire index and file are kept in the high-speed storage, then we can set $a = b = c = d = 1$ and we find that optimum index size is $n = \sqrt{N}$. If the index is kept in the high-speed storage but the file is stored on a disk, we might set $a = 100b$, in which case the optimum index size is $n = 10\sqrt{N}$.

Next, consider a two-level indexed sequential file with access times, as indicated in Fig. 3. We assume that the sublists at each level are of equal length, that there is an equal probability of request for any of the records, and

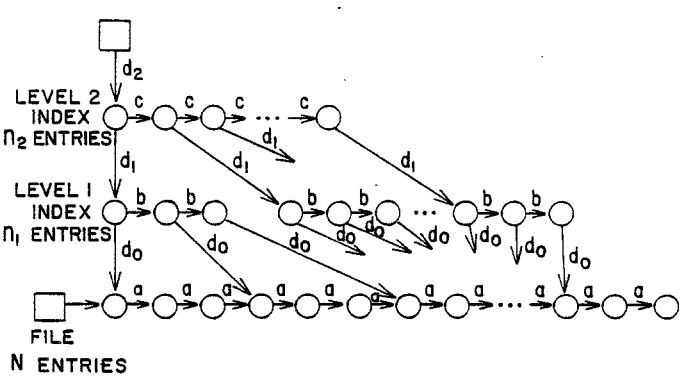

Fig. 3

that the records are sorted in ascending order by key value. The average search cost is

$$S = d_2 + \tfrac{1}{2}n_2c + d_1 + \tfrac{1}{2}(n_1/n_2)b + d_0 + \tfrac{1}{2}(Na/n_1)$$

Taking partial derivatives with respect to n_1 and n_2 and setting to zero, we get

$$\partial\bar{S}/\partial n_1 = \tfrac{1}{2}(b/n_2) - \tfrac{1}{2}(Na/n_1^2) = 0$$
$$\partial\bar{S}/\partial n_2 = \tfrac{1}{2}c - \tfrac{1}{2}(n_1/n_2^2)b = 0$$

Solving the equations, we get

$$n_1 = (N^2a^2/bc)^{1/3}, \qquad n_2 = (Nab/c^2)^{1/3}$$

If all the access times are equal (the case when the entire index and file are in the high-speed storage, for example), the results are

$$n_1 = N^{2/3}, \qquad n_2 = N^{1/3}$$

We can use these results to determine the size of the index and the average access time $\bar{S}$.

The technique can be generalized for a d-level indexed sequential file where c_i is the cost of an access within level i, n_i is the number of nodes at level i, and d_i is the cost of going from level i to level $i + 1$. The average search cost is

$$\bar{S} = \sum_{i=0}^{d} \left(\frac{1}{2}\frac{n_ic_i}{n_{i+1}} + d_i\right)$$

with $n_0 = N$ and $n_{d+1} = 1$. We are left with a system of nonlinear equations

$$[\partial\bar{S}/\partial n_i = 0], \qquad i = 1,\ldots, d$$

or

$$\left[\frac{1}{2}\frac{c_i}{n_{i+1}} - \frac{1}{2}\frac{n_{i-1}c_{i-1}}{n_i^2} = 0\right], \qquad i = 1,\ldots, d$$

The solution of this system of equations is

$$n_i = \left\{\left[\left(\prod_{k=0}^{i-1} c_k^{d-i+1}\right)\Big/\left(\prod_{k=i}^{d} c_k{}^i\right)\right] N^{d-i+1}\right\}^{1/(d+1)}, \qquad i = 1,\ldots, d$$

The foregoing analyses are general and can be modified to reflect constraints imposed by hardware considerations, such as fixed size disk sectors. The methodology can deal with extremely complex hierarchical file strategies. For instance, in a three-level indexed sequential organization the

level three index might be kept in the high-speed storage, the level two indexes on a drum, and the level one indexes on a disk or data cell with the file of records. For a given N the values of n_1, n_2, and n_3 could be determined to minimize the access cost.

4. VARIABLE NUMBER OF LEVELS

In these examples we have fixed the number of levels and allowed the size of the indexes to vary. Consider the more complicated case in which we fix the size of the indexes to be the same at each level and make the number of levels variable (see Fig. 4). This is a reasonable model of the case in which each index is considered as a "bucket." The problem is to optimize the size k (and therefore the number of levels) of the bucket based on the time to access a bucket, a. In the first case we assume a sequential search through each bucket and later compare this to the more complicated case where the buckets may be searched using the binary search technique.

If the index is to have v levels, then we have

$$N = k^v \tag{1}$$

and

$$v = (\ln N)/(\ln k) \tag{2}$$

If the time to access a record from the lowest level index is d, then the average search cost is

$$\bar{S} = d + vb + \tfrac{1}{2}vak$$

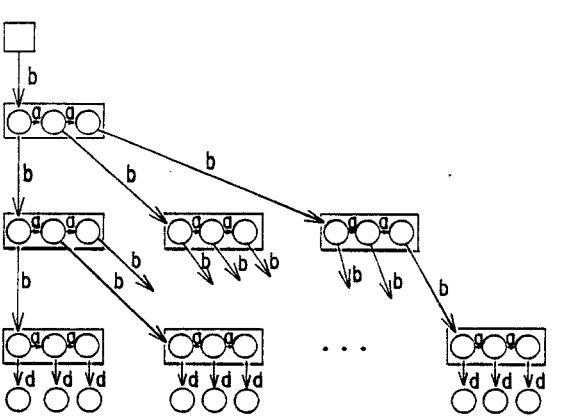

Fig. 4

Table I. Sequential Search of Indexes[a]

b/a	k	$v\ (N = 10^5)$	$v\ (N = 10^6)$	$v\ (N = 10^7)$
10	12.9	4.5	5.4	6.3
50	37.9	3.2	3.8	4.4
100	63.5	2.8	3.3	3.9
500	226.2	2.1	2.5	2.9
1000	400.6	1.9	2.3	2.7

[a] a is the cost of a sequential search step within the same level; b is the cost of going to the next level; k is the number of index entries per block; and v is the number of levels required for the given value of N.

Taking the derivative with respect to k, we find that

$$k(\ln k - 1) = 2b/a$$

It is interesting to note that the result is independent of the file size N. Brief consideration will reveal the validity of this situation; for a given N the number of levels necessary may be determined from (2). Table I gives the numerical results in some typical situations.

If each of the buckets is to be searched by a binary search then our graph model is as in Fig. 5, where each node is as in Fig. 6. We assume that each step of a binary search costs c and that the cost of going to the next level of buckets is b. For the analysis we assume that each bucket contains k fields, where k is one less than a power of two. If v is the number of levels and N the

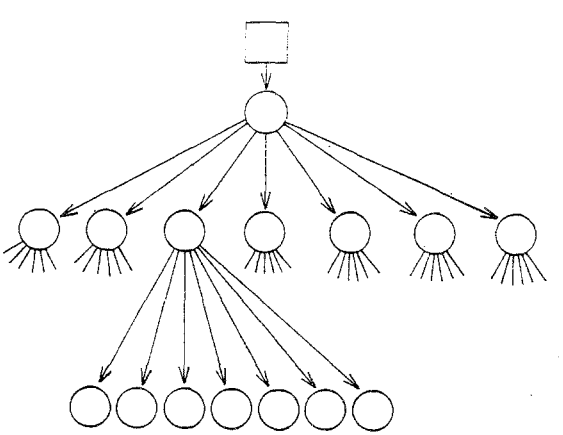

Fig. 5

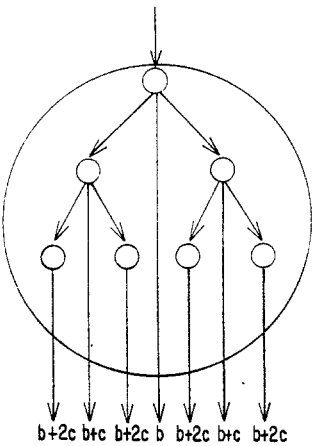

Fig. 6

number of records in the file, we again have (1) and (2). The average search time is

$$\bar{S} = d + v\left[\frac{b}{k} + \frac{2(b+c)}{k} + \frac{4(b+2c)}{k} + \frac{8(b+3c)}{k} + \cdots + \frac{2^r(b+rc)}{k}\right]$$

$$= d + \frac{v}{k}\sum_{i=0}^{r} 2^i(b+ic)$$

where

$$r = \log_2(k+1) - 1$$

It can be shown that

$$\sum_{i=1}^{r} i2^i = (r-1)\,2^{r+1} + 2$$

Applying the above formula and substituting for v, we get

$$\bar{S} = d + \frac{b \ln N}{\ln k} + \frac{c \ln N}{k \ln k}\left[\left(\frac{\ln(k+1)}{\ln 2} - 2\right)(k+1) + 2\right]$$

Taking the derivative with respect to k and setting the result equal to zero yields

$$0 = -bk - c(1 + \ln k)\left[\frac{(k+1)\ln(k+1)}{\ln 2} - 2k\right]$$

$$+ \frac{kc \ln k[1 + 2\ln 2 + \ln(k+1)]}{\ln 2}$$

Table II. Binary Search of Indexes[a]

b/c	k	$v\ (N = 10^5)$	$v\ (N = 10^6)$	$v\ (N = 10^7)$
5	3.3	9.6	11.6	13.5
10	10.2	5.0	6.0	7.0
20	99.2	2.5	3.0	3.5
30	1117.1	1.6	2.0	2.3
40	13396.0	1.2	1.4	1.7

[a] c is the cost of a binary search step within the same level; b is the cost of going to the next level; k is the number of index entries per block; and v is the number of levels required for the given value of N.

This unwieldy result can be solved numerically. Typical values are shown in Table II. Notice that the cost for one step of a binary search, c, is larger than the cost of one step of a linear search, a. The ratio of a to c must be kept in mind in deciding which technique to adopt. Comparing Tables I and II again demonstrates the overwhelming advantage of binary searching.

Assuming a system based on auxiliary storage devices, the model can be refined further by taking into account the higher cost associated with a larger bucket size. The time to go one level lower in the index is $b + ek$, where b is the seek cost (or latency) and e is transfer rate per field. Thus, larger values of k yield larger cost values.

For the sequential search we now get

$$k(\ln k - 1) = 2b/(2e + a)$$

Table III. Sequential Search of Indexes Including Transfer Rate Cost[a]

b/a	k	$v\ (N = 10^5)$	$v\ (N = 10^6)$	$v\ (N = 10^7)$
10	7.0	5.9	7.1	8.3
50	17.8	4.0	4.8	5.6
100	28.4	3.4	4.1	4.8
500	94.1	2.5	3.0	3.5
1000	162.9	2.3	2.7	3.2
5000	614.8	1.8	2.2	2.5

[a] Assume $a = e$ (single field search cost = single field transfer cost). a is the cost of a sequential search step within the same level; b is the cost of going to the next level; k is the number of index entries per block; and v is the number of levels required for the given value of N.

Table IV. Binary Search of Indexes Including Transfer Rate Cost[a]

b/c	k	$v\ (N = 10^5)$	$v\ (N = 10^6)$	$v\ (N = 10^7)$
10	6.3	6.3	7.5	8.8
50	44.2	3.0	3.6	4.3
100	89.5	2.6	3.1	3.6
500	396.4	1.9	2.3	2.7
1000	736.0	1.7	2.1	2.4
5000	3089.8	1.4	1.7	2.0

[a] Assume $e = c/5$ (single field transfer cost = 1/5 single field search cost). c is the cost of a binary search step within the same level; b is the cost of going to the next level; k is the number of index entries per block; and v is the number of levels required for the given value of N.

and for the binary search the new result is

$$0 = -bk + ek^2(\ln k - 1) - c(1 + \ln k)\left[\frac{(k+1)\ln(k+1)}{\ln 2} - 2k\right] + \frac{kc \ln k[1 + 2 \ln 2 + \ln(k+1)]}{\ln 2}$$

Tables III and IV give typical values for sequential search and binary search when the transfer rate is considered as a factor. Reasonable values for b, c, and e were chosen and then the optimal size k was computed.

5. CONCLUSION

Although the noninteger results must be rounded to obtain integer values, the method of analysis does produce reasonable results on which to base the implementation of an indexed file system. The costs of performing a sequential search or a binary search must be obtained through software measurement. The cost of performing the various accesses on differing hardware devices must also be determined. Having obtained these fixed parameters, it is possible to determine reasonably optimal estimates of the variable parameters, such as the number of index entries at each level or the bucket size.

Similar techniques can be adapted to study numerous indexed file organization problems, such as: the effect of additions or deletions on file access cost, the increase in efficiency obtained by batching requests in ascending order so that not all searches start from an entry node but use indexes accessed in the previous search, and the optimization of index structures and searches when the probability of request is not equal for all records.

ACKNOWLEDGMENTS

I would like to thank Prof. Jack Heller of the State University of New York at Stony Brook and the reviewer for their detailed comments, which greatly improved this paper.

REFERENCES

1. Ivan Flores, *Data Structure and Management* (Prentice-Hall, Englewood Cliffs, New Jersey, 1970).
2. David Lefkovitz, *File Structures for On-Line Systems* (Spartan Books, New York, 1969).
3. Gerard Salton, *Automatic Information Organization and Retrieval* (McGraw-Hill, New York, 1968).
4. George Dodd, "Elements of data management systems," *Computing Surveys* **1**(2): 177–133 (1969).
5. D. L. Childs, "Feasibility of a set-theoretic data structure," in *Proc. IFIP Congress 1968* (North-Holland, Amsterdam), Vol. 1, pp. 420–430.
6. E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM* **13**(6):377–387 (1970).
7. A. C. Fleck, "Towards a theory of data structures," *J. Computer System Sci.* **5**:475–488 1971).
8. Arnold Rosenberg, "Data graphs and addressing schemes," *J. Computer System Sci.* **5**:193–238 (1971).
9. Arnold Rosenberg, "Symmetries in data graphs," *SIAM J. Computing* **1**:40–65 (1972).
10. Arnold Rosenberg, "Addressable data graphs," *J. ACM* **19**:309–340 (1972).
11. D. Hsaio and F. Harary, "A formal system for information retrieval from files," *ACM* **13**(2):67–73 (1970).
12. Jay Earley, "Toward an understanding of data structures," *ACM* **14**(10):617–627 (1971).
13. Jack Heller and Ben Shneiderman, "A graph theoretic model of data structures," *SIGIR Forum*, Vol. 111, No. 4, Winter 1972.
14. Ben Shneiderman, "Data Structures: Description, Manipulation and Evaluation," Ph.D. Thesis, State University of New York at Stony Brook, 1973.

Printed in Belgium