1976

A Hierarchical Access Model for Data Base Organizations

S. B. Yao

Report Number:
76-177

A HIERARCHICAL ACCESS MODEL FOR DATA BASE ORGANIZATIONS

by

S. B. Yao
Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

CSD-TR 177

May 1976

ABSTRACT

A generalized multi-dimensional model for physical data base organizations is presented. Existing data base organizations are shown to easily fit into the model as special cases. Generalized retrieval algorithms and cost equations associated with the model are developed and analyzed. The model provides a general design framework in which the distinguishing properties of data base organizations are made explicit and their performance can be compared.

Key Words and Phrases: data base organization, index organization, data base performance evaluation, data base model, inverted file, multilist, index sequential, B-tree.

CR Categories: 370, 3.72, 3.73, 3.74, 4.33

1. Introduction

Various data base systems are available for managing data. The type of (physical) data base organizations used in such systems has a very important effect on the overall system performance. Many data base organizations and access methods exist, and each has its advantages and disadvantages with respect to particular system objective. There is little common guideline as how the data base organization should be designed and implemented. It is the objective of this paper to introduce a model for data base organizations, the Hierarchical Access Model, that captures the parameters in the design alternatives. The model provides a general design framework in which the properties of data base organizations are made explicit and the performance of different data base organizations can be compared.

Traditionally, a physical data base organization is viewed as consisting of two parts: the directory and the file. The file is a set of records, subsets of which are accessed by users and the directory provides indices for locating the subsets. A list oriented file structure was introduced by Lefkovitz [7] and Martin [9]. A formal model was later introduced by Hsiao and Harary [5]. Figure 1 shows that each directory entry corresponds to a keyword and contains:

- K_i - value of the keyword indexed
- n_i - total number of records containing K_i
- h_i - the number of sublists for records containing K_i
- $a_{i1}, a_{i2}, \ldots, a_{ih_i}$ - set of head pointers to the sublists.

The only model parameter that can be controlled by a file designer is

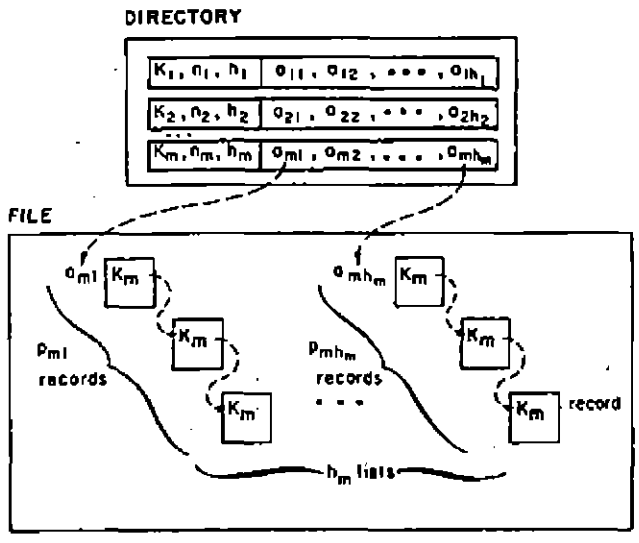
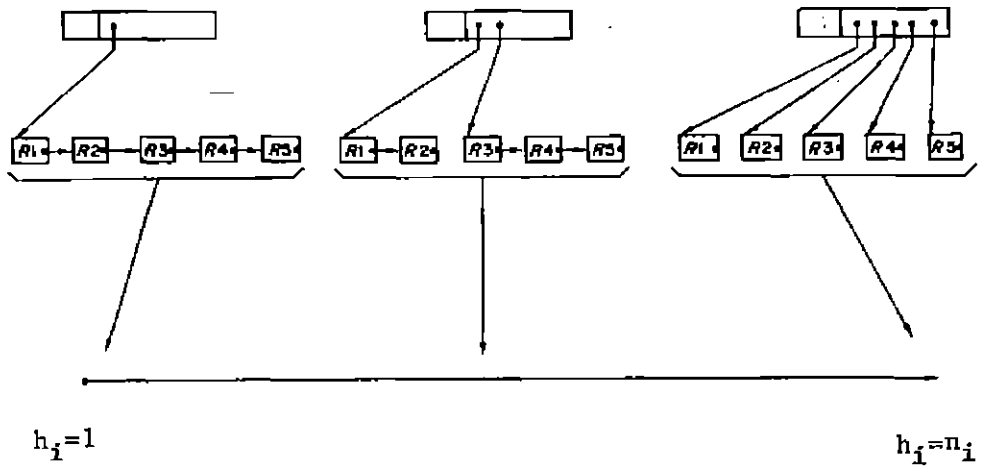


Figure 1. A formal model for list oriented file structures.



$h_i=1$ $h_i=n_i$

Figure 2. A one dimensional design space.

the number of sublists for each keyword K_i. Figure 2 shows that the design space is one dimensional from $h_i = 1$ (multilist) to $h_i = n_i$ (inverted).

It was observed by Severance that the model is unable to describe record sequencing which is not strictly list oriented, and a two-parameter model was introduced [11] for describing the file. The structure of each physical record may be (1) data direct - the record contains the desired data, or (2) data indirect - the record contains a pointer to the desired data. The inter record connection structure which defines a file may be (1) address sequential - the successor record is at the next sequential address, or (2) pointer sequential - the record contains a pointer to the successor record, Therefore, the two paramters of the model are:

P1 - the proportion of data indirect records

P2 - the proportion of pointer sequential record connections.

The two dimensional design space defined by P1 and P2 is shown in Figure 3.

We note that the model can only represent a one-level file index using the "data indirection". Large data bases usually require multiple-level indices. The model is incomplete in that, for example, the various cellular list organizations [6] can not be represented. The important concept of partitioning a list into sublists is ignored since there is no provision for "connection indirect" (i.e. connecting a set of indirectly stored data, as done in the first model). The representation for indexed sequential file is also imprecise. All these indicate that the model is still too simplistic. It appears that the two models may be combined to form a three-dimensional model. In this paper, fundamental properties of file search will be analyzed and a more accurate model with a multi-dimensional design space will be presented as an extension to the above two models.

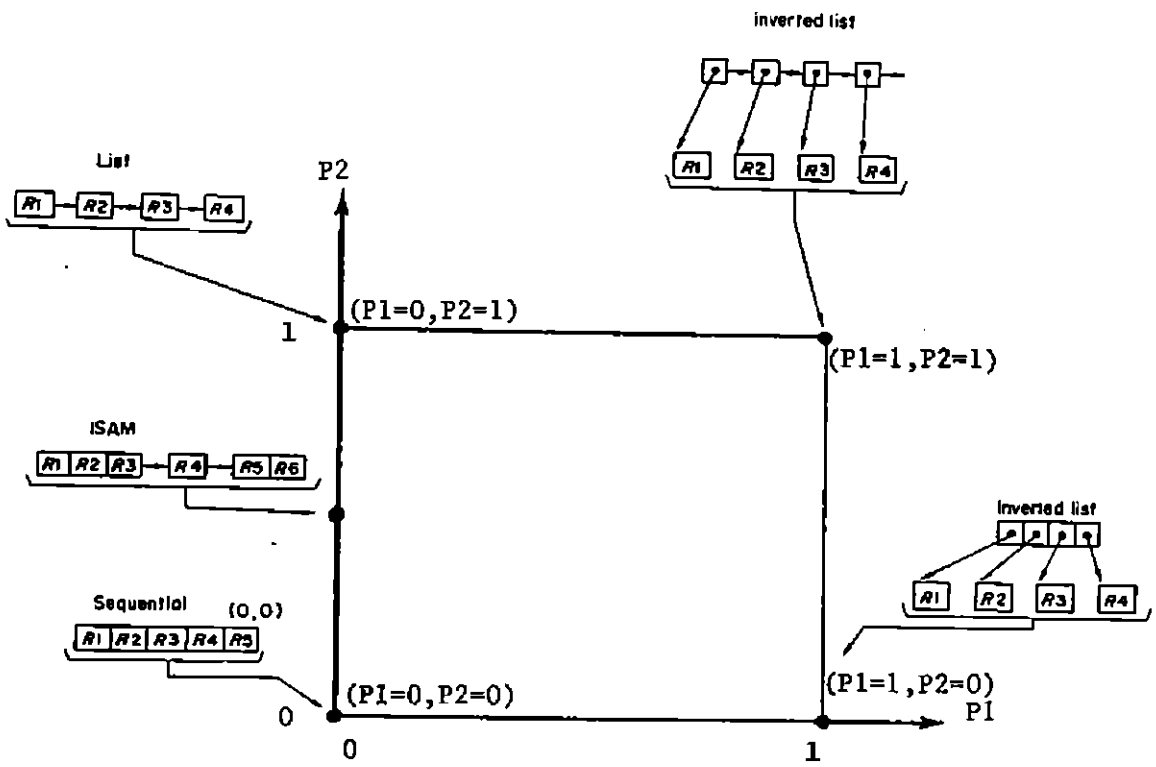


Figure 3. A two dimensional design space.

2. Basic Concepts

A file may be considered as containing symbolic descriptors which record a set of facts about some set of entities in the real or abstract world. Various techniques are available to facilitate these symbolic descriptors. In order to provide a common foundation for modeling the various data base organizations, let us consider the following formal model.

Definition 1. An attribute is a binary relation between the entity set and a value set which contains values or symbols. Each element in the attribute is called an item.

Definition 2. A record for an entity is the set of all items pertaining to that entity. A file is a set of records defined over the same set of attributes.

Example 1:

Consider an entity set containing two employees:

Entity set: $\{e_1, e_2\}$

Attribute name set: {jobcode, skillcode, salary}

Value sets: $V_{jobcode} = \{J1, \ldots, J9\}$

$V_{skillcode} = \{S1, \ldots, S9\}$

$V_{salary} = \{500, \ldots, 2000\}$

Attributes: JOBCODE = $\{(e_1,\text{jobcode},J6),(e_2,\text{jobcode},J5)\}$

SKILLCODE = $\{(e_1,\text{skillcode},S1),(e_1,\text{skillcode},S3)$
$(e_2,\text{skillcode},S3),(e_2,\text{skillcode},S5)\}$

SALARY = $\{(e_1,\text{salary},900),(e_2,\text{salary},1200)\}$

Records: $r_1 = \{(e_1,\text{jobcode},J6),(e_1,\text{skillcode},S1),$
$(e_1,\text{skillcode},S3),(e_1,\text{salary},900)\}$

$r_2 = \{(e_2,\text{jobcode},J5),(e_2,\text{skillcode},S3),$
$(e_2,\text{skillcode},S5),(e_2,\text{salary},1200)\}$

We note that items in attributes are denoted by (entity, attribute-name, value) triples. In what follows, when there is no possibility of ambiguity, we shall denote items in attributes by (entity, value) pairs and items in records by (attribute-name, value) pairs. Since attributes are defined as binary relations rather than functions as in [4], repeating items from an attribute are allowed in a record.

Definition 3. A keyword is an ordered pair of an attribute-name and a value. The set of entities (and hence records) that correspond to a keyword is a k-set. Similarly, when the value set is ordered, a range is a triple of an attribute-name, a low value and a high value. A range is equivalent to a set of keywords with values within the value range (i.e. $(a,v_1,v_2) = \{(a,v) | v_1 \leq v \leq v_2\}$).

Definition 4. A primary key is an attribute-name whose k-sets are singleton sets. A secondary key is an attribute-name which is not a primary key.

Definition 5. A query is a Boolean function of keywords or ranges using the operators ∧ and ∨, and is structured as a disjunctive normal form. The response set is the set of entities obtained by applying the set operations $\cap$ and $\cup$ respectively on the corresponding k-sets.

A query in disjunctive normal form can be described by the following parameters:

number of conjunctions in the query:	d
number of attribute-names specified in each conjunction:	$c_1, c_2, \ldots, c_d$
number of keywords specified for each attribute:	$q_{11}, q_{12}, \ldots, q_{1c_1}$
	$q_{21}, q_{22}, \ldots, q_{2c_2}$
	$\vdots$
	$q_{d1}, q_{d2}, \ldots, q_{dc_d}$
Total number of attributes specified in a query:	$p \leq \sum_{i=1}^{d} c_i$
average number of keywords specified for each attribute:	q
minimal number of keywords specified for an attribute:	$\hat{q} = \underset{i}{\text{avg}}\{\min_{j}\{q_{ij}\}\}$

Example 2.

Consider the file in Example 1:

Keywords:	(salary,900)
	(skillcode, S3)
Range:	(jobcode,J5,J6)
K-sets:	$S(\text{jobcode},J5,J6)=\{r_1,r_2\}$
	$S(\text{skillcode},S3)=\{r_1,r_2\}$
	$S(\text{salary},900)=\{r_1\}$
Query:	[(jobcode,J5,J6)∧(skillcode,S3)]∨(salary,900)
Response set:	$[S(\text{jobcode},J5,J6) \cap S(\text{skillcode},S3)] \cup S(\text{salary},900)$
	$= \{r_1,r_2\}$
	$d=2, c_1=2, c_2=1, q_{11}=2, q_{12}=1, q_{21}=1, p=3, \hat{q}=1.$

Definition 6. The query complexity is the average number of conjuncts in each query d, the average number of attributes in each conjunct c, and the average number of keywords specified for each attribute q.

3. Hierarchical Access Model

When a data base is accessed to retrieve a certain record, only part of the data base must be searched. This may involve table look-up, indexing, list traversing, etc. These procedures and structures are in fact partitioning the search space (records in the data base) into groups so that the search may be quickly narrowed down to smaller and smaller subsets until the appropriate record is located. Such a self-refining process can be modeled by an access tree; the branches at a particular access tree node represent the alternatives which partition the search space into groups.

The immediate identifiable file search stages are attribute, keyword, and record. Attributes and keywords are specified by the query and their representations in the file can be searched. The access paths required for this procedure constitute an access tree where the access paths are indicated by the hierarchy relationships.

Figure 4 shows the access tree of a file where the three stages are shown. Each triangular represents the set of access paths from the node at the top of the triangular to the nodes in the bottom. We note that from each attribute or keyword there are access paths leading to records

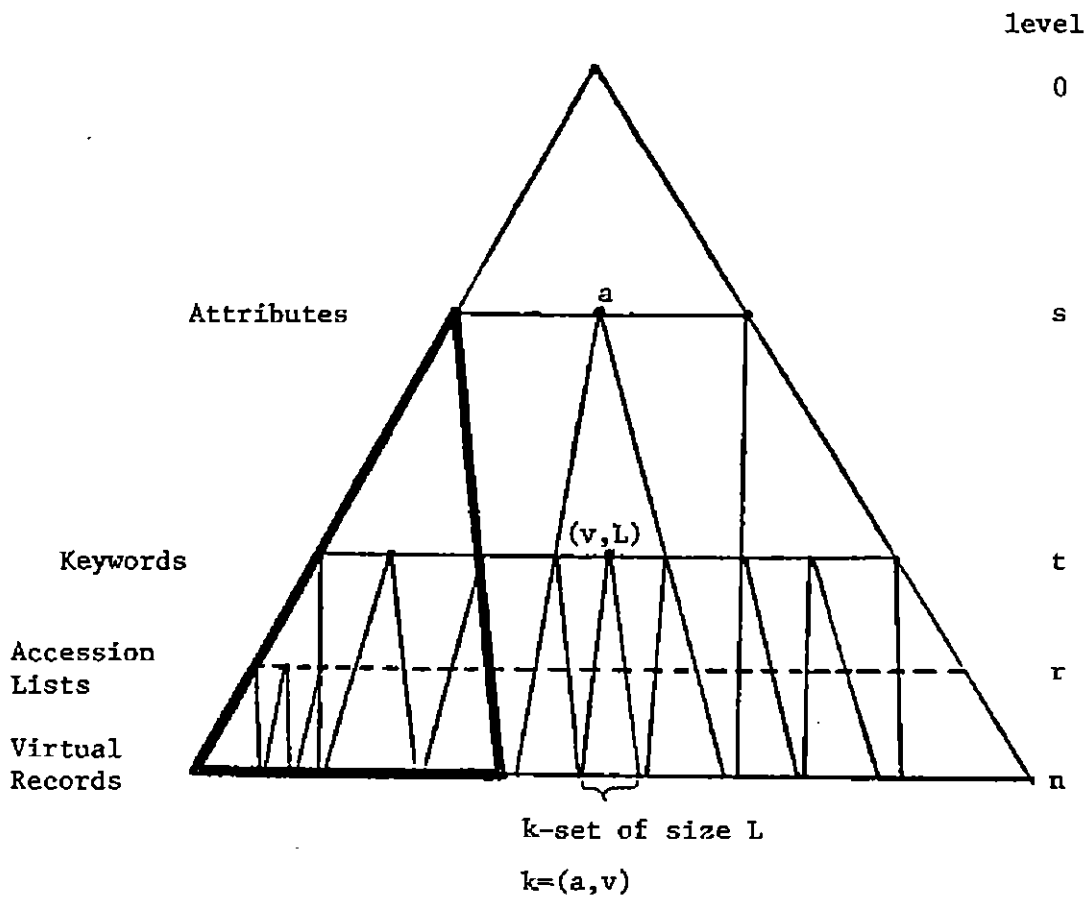


Figure 4. The access tree of data base organizations.

containing that attribute or keyword. In order to represent the access paths by a tree, a record will be shown by as many nodes on the record level as there are access paths to it. These nodes are called virtual records. Depending on the particular file design, there may be more levels between the three identified levels. These additional levels are called indices.

The portion of the access tree from the root to level t is called the directory which decodes a given keyword (a,v). The attribute-name a is decoded on level s and the value v is decoded on level t. Let c(x) denote the content (other than the access path information) of the node x and $\ell(x)$ denote the level of the access tree where x resides. Assume that attribute-names and values are ordered. We have

$$c(x) = \begin{cases} \text{highest a reachable from x} & \text{if } \ell(x) < s \\ a & \text{if } \ell(x) = s \\ \text{highest v reachable from x} & \text{if } s < \ell(x) < t \\ (v,L) & \text{if } \ell(x) = t \end{cases}$$

where L is the size of the k-set reachable from the keyword (a,v).

The portion of the access tree below level t is called the file. The file contains k-sets correspond to the keywords in the directory. Since the retrieval of a large k-set can be inefficient, it is sometimes desirable to partition the k-set into several k-subsets

and introduce an accession level r above the virtual record level. Corresponding to each k-set, there is an accession list which contains accession pointers on the accession level. Stored with each accession pointer is a length field

$$c(x) = L \quad , \quad \ell(x) = r.$$

where L is the size of the k-subset referenced by the accession pointer.

The k-sets reachable from an attribute are disjoint if the attribute is a function (i.e. single-valued binary relation). Otherwise, the attribute induces "repeating groups" in records. The subtree rooted at attribute level are not disjoint, however, since a record usually is in the domain of all attributes.

Example 3.

The access tree for the file in Example 1 is illustrated in Figure 5. The k-sets are disjoint for the attributes JOBCODE, and SALARY while the items in the attribute SKILLCODE form repeating groups in records.

In the case of a file organized with only a primary key, the access tree is reduced to the subtree rooted at one attribute. Since the keywords are identifiers of records, the k-sets become singleton sets. The lower levels of the tree degenerate and virtual records are attached to the keywords. Further examples of the access tree representation are provided in the next section.

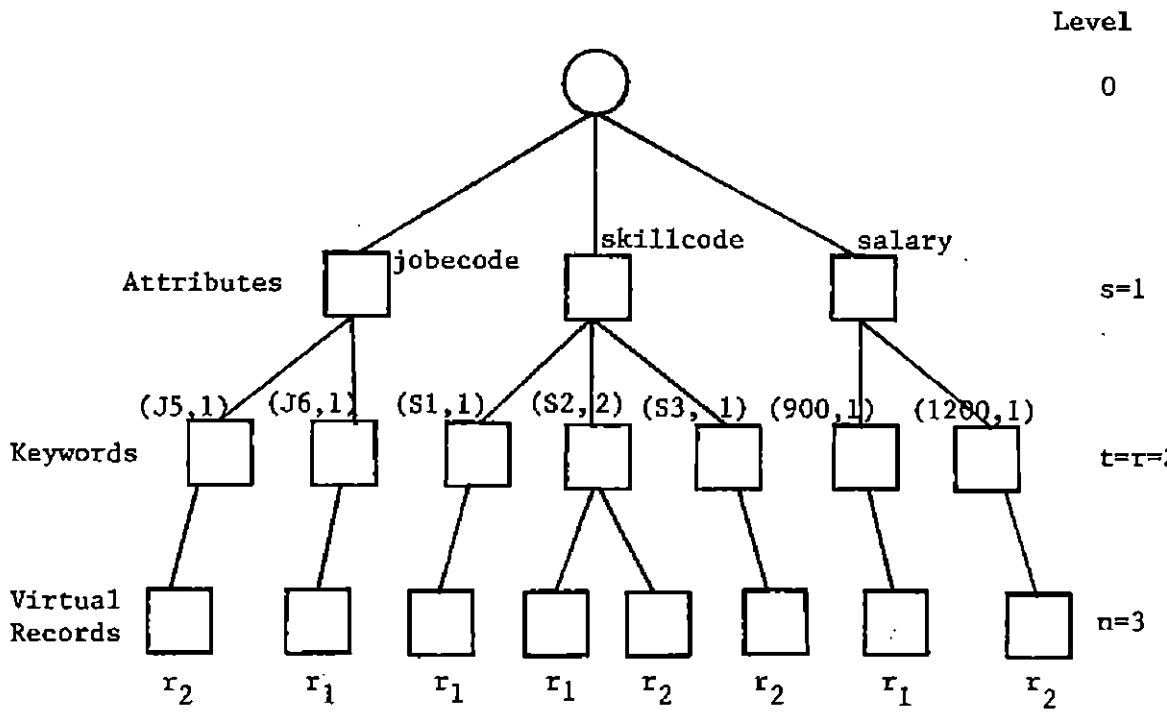


Figure 5. The access tree for example 1.

4. Access Path Representation

Consider the binary representation of the access tree (Figure 6a). In the binary representation, every node has two branches. The vertical branch indicates the superior relationship to the filial set (defined as a set of nodes with the same immediate parent node) and the horizontal branch indicates the successor on the same level. Branches in the access tree indicate the access paths, not their implementation. For the special case in which the branches are pointers, the binary representation of the tree is generally referred to as a doubly chained tree [14].

Any level of the access tree consists of the filial sets on that level. Let W_i denote the average size of the filial sets on the level i. The access path structure can be characterized by the set of variables

$$W_1, W_2, \ldots W_n$$

This sequence of variables determines the general configuration of the access tree. From the definition of a tree, the number of nodes on level i must satisfy the condition

$$S_i = \prod_{j=1}^{i} W_j$$

Assuming that attributes are on level s and that keywords are on level t, the following identities can be established:

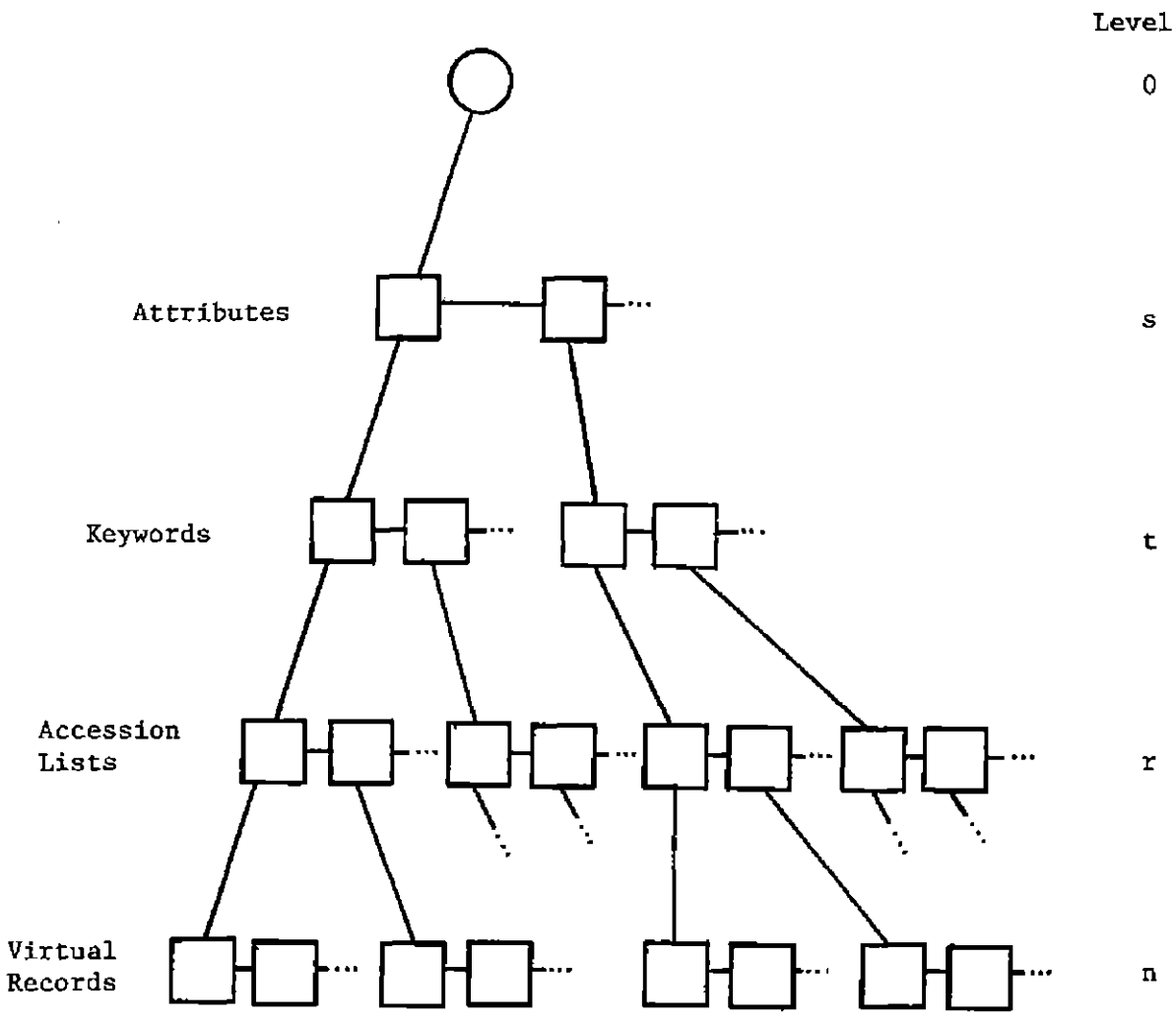


Figure 6a. Binary representation of the access tree.

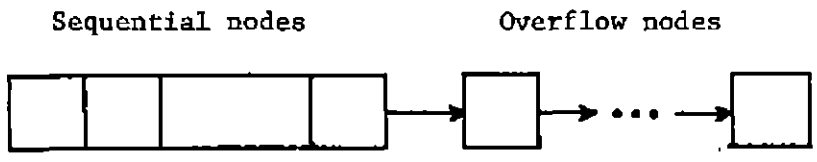


Figure 6b. Structure of a filial set.

number of attributes: $N_a = \prod_{j=1}^{s} W_j$

number of keywords per attribute: $N_v = \prod_{j=s+1}^{t} W_j$

total number of keywords: $N_k = \prod_{j=1}^{t} W_j$

average size of k-sets: $L = \prod_{j=t+1}^{n} W_j$

total number of virtual records: $R_v = \prod_{j=1}^{n} W_j$

The parameters on the left hand sides of the above equations are determined by the logical data base structure. The identities act as constraints on the access path parameters $W_1,\ldots,W_n$.

When the Hierarchical Access Model is used to represent data base organizations, it is necessary to consider realizations of the access paths in the model. As modeled by Severance, the successor connections between nodes may be:

(1) Address sequential-the successor of the current node is located at the next sequential address.

(2) Pointer sequential-the successor node of the current node is located by a pointer field in the current node.

These two types of successor connection can be mixed when an access path is constructed. The actual design is obtained when every branch in an access tree has been assigned one of the two types of connection. Graphically, the address sequential connection is denoted by $=$ and the pointer sequential connection is denoted by $\rightarrow$.

A sequential node is a node reached by an address sequential connection. An overflow node is a node reached by a pointer sequential connection. When a filial set contains both types of nodes, overflow nodes constitute an overflow chain from the blocks of sequential nodes. Figure 6b shows a filial set with both types of nodes.

If there are μ sequential nodes and ν overflow nodes in a filial set of W nodes, the proportions of sequential nodes and overflow nodes can be indicated by the overflow ratio P,

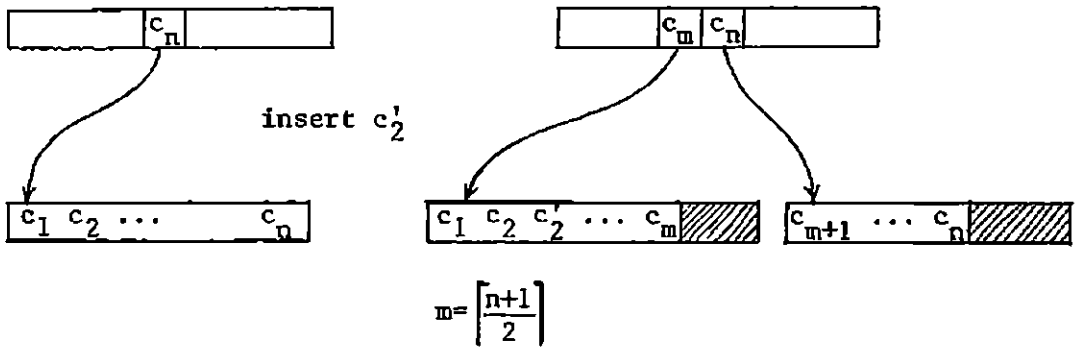
$$P = \frac{\nu}{\mu+\nu} = \frac{\nu}{W}$$

Representing the realization of the access paths on level i by the overflow ratio P_i, the realization of an access tree is described by $P_1, P_2, \ldots, P_n$.

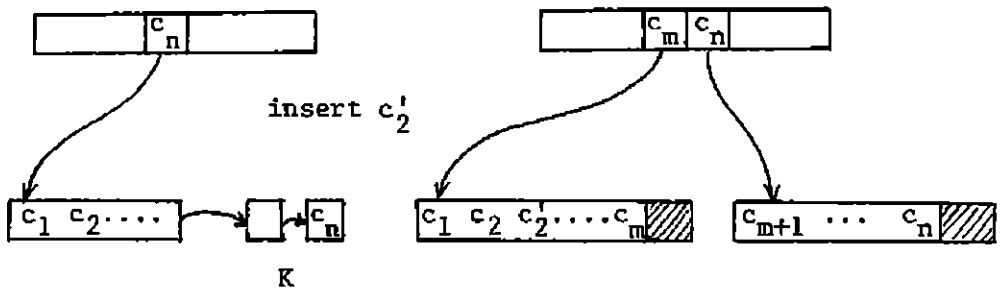
Insertion of new entries into the sequential portion of a filial set may cause it to overflow. In order to avoid or delay overflow, free space may be distributed in the sequential blocks. The amount of free space is measured by the loading factor

$$F = \frac{u}{u+w}$$

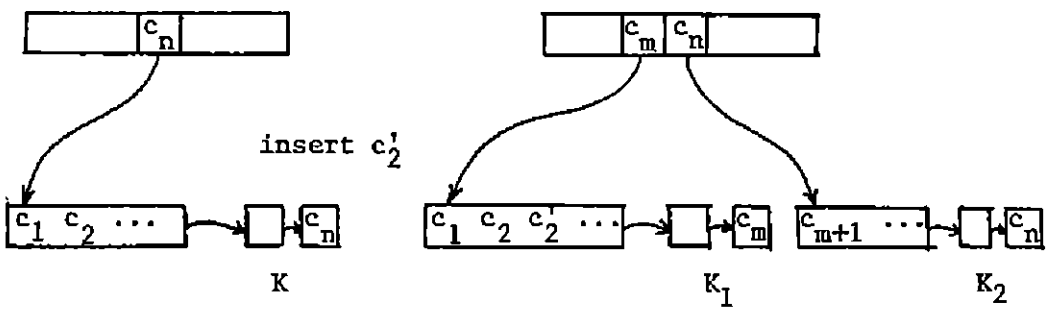
where u is the number of sequential nodes and w is the amount of free space. When free space in the sequential blocks is exhausted by subsequent insertions, overflow chaining can further be avoided by splitting the filial set [1]. Figure 7a shows that when an overflowed filial set splits, a new filial set is created and the nodes are averaged between the two filial sets. The middle node is inserted into the parent to establish an access path to the newly created filial set.



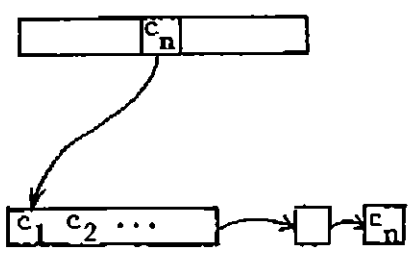
(a) Q = 0 Overflow by splitting.



(b) 0 < Q ≤ 0.5 Overflow by chaining and splitting.



(c) 0.5 < Q < 1 Overflow by chaining and splitting.



(d) Q = 1 Overflow by chaining.

Figure 7. Model of overflow handling methods.

In order to represent both types of overflow handling methods, consider Q as the maximum overflow ratio allowed in a filial set. Only when insertions would cause the overflow ratio to exceed Q, a filial set splitting occurs. This is illustrated in Figure 7. If Q=0, overflows will always be handled by splitting. While if Q=1, overflows are handled only by chaining.

In summary, the access path realizations are characterized by the sequence of $4n+4$ model parameters:

$$s,t,r,n,W_1,W_2,\ldots,W_n,P_1,P_2,\ldots P_n,F_1,F_2,\ldots,F_n,Q_1,Q_2,\ldots,Q_n.$$

The first $2n+4$ parameters describe the static structures of data base organizations and the last $2n$ parameters describe the dynamic structures for update and insertion. To illustrate the generality of the Hierarchical Access Model, some examples are presented to show that the most commonly used types of data base organizations are all special cases of the model. This model further provides a frame work within which new data base organizations may be generated. Examples 7 and 8 are primary key organizations that could be employed in the directory of multi-attribute organizations such as illustrated in examples 4,5 and 6.

Example 4. Inverted Data Base Organization

The inverted data base organization is structured on multiple attributes for query retrieval using secondary keys. The directory of the organization consists of two index levels, the attribute-name index and the keyword value index, as shown in Figure 8a. Each keyword value is associated with an accession list which points to records in a k-set. In most

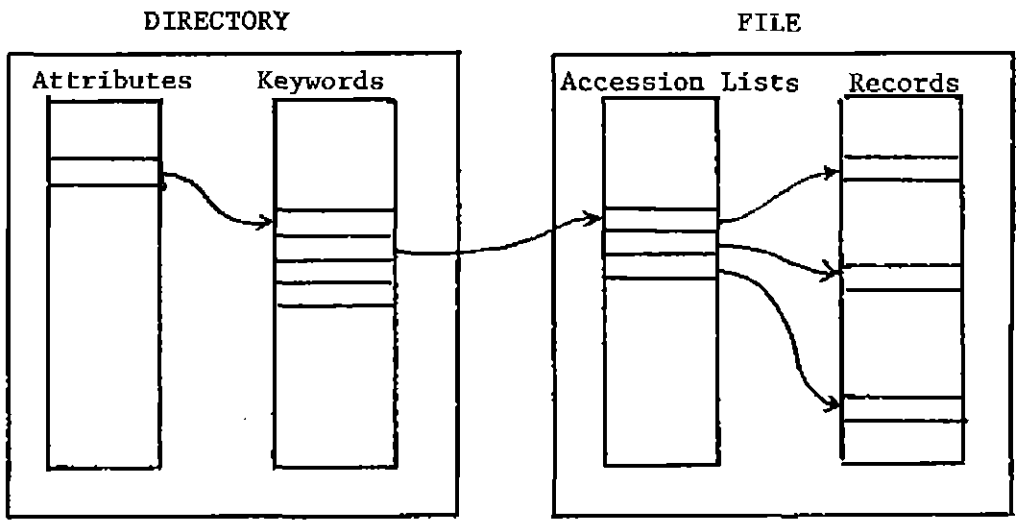


Figure 8a. Inverted data base organization.

Figure 8b. The access model representation of the inverted data base organization.

implementations, records are ordered by the primary key. The records are grouped into blocks and the blocks are grouped into buckets (cylinders).

The access tree representation of the inverted data base organization is shown in Figure 8b. The filial sets on each level are implemented sequentially. The access tree has four levels and the first three levels are indices. Assuming that the loading factor is f and the overflow is handled by splitting blocks, the design variables of the inverted data base organization have the following values:

$s=1, t=2, r=3, n=4$

$W_1=Na$ (number of attributes)

$W_2=N_v$ (number of keywords per attribute)

$W_3=L$ (average size of k-sets)

$W_4=1$

$P_i=0,\ i=1,2,3$

$F_i=f,\ i=1,2,3,4$

$Q_i=0,\ i=1,2,3$

$P_4=Q_4=1$

Example 5. Multilist Data Base Organization

The multilist data base organization has an identical directory structure as the inverted data base organization. As shown in Figure 9a, the accession list level in the file is eliminated and the accession pointers are embedded into the records. Each keyword value in the directory has a pointer that references the first record in its k-set. Successive records in the k-set are obtained by following a linked list sequentially. This may require to access records that do not satisfy the query. Multilist is generally inefficient for data bases with large k-sets.

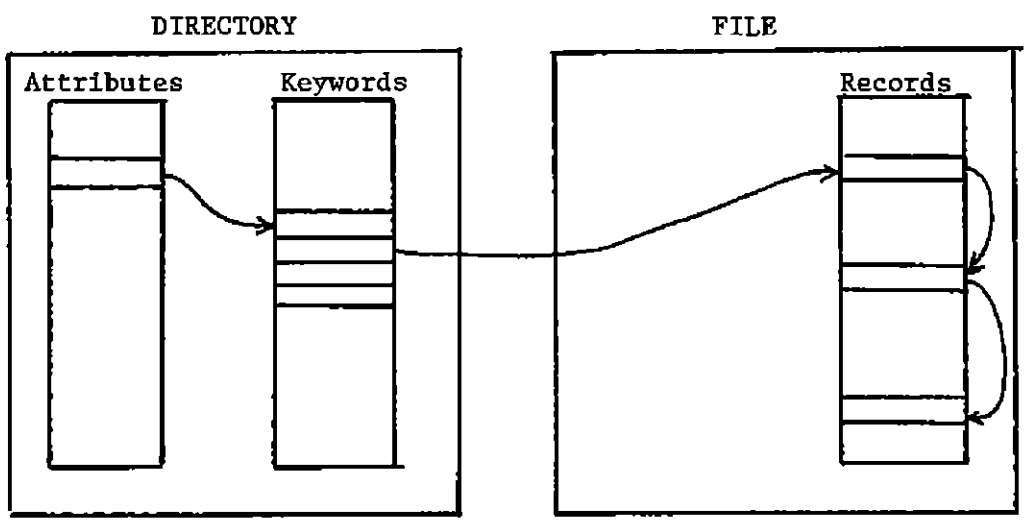


Figure 9a. Multilist data base organization.

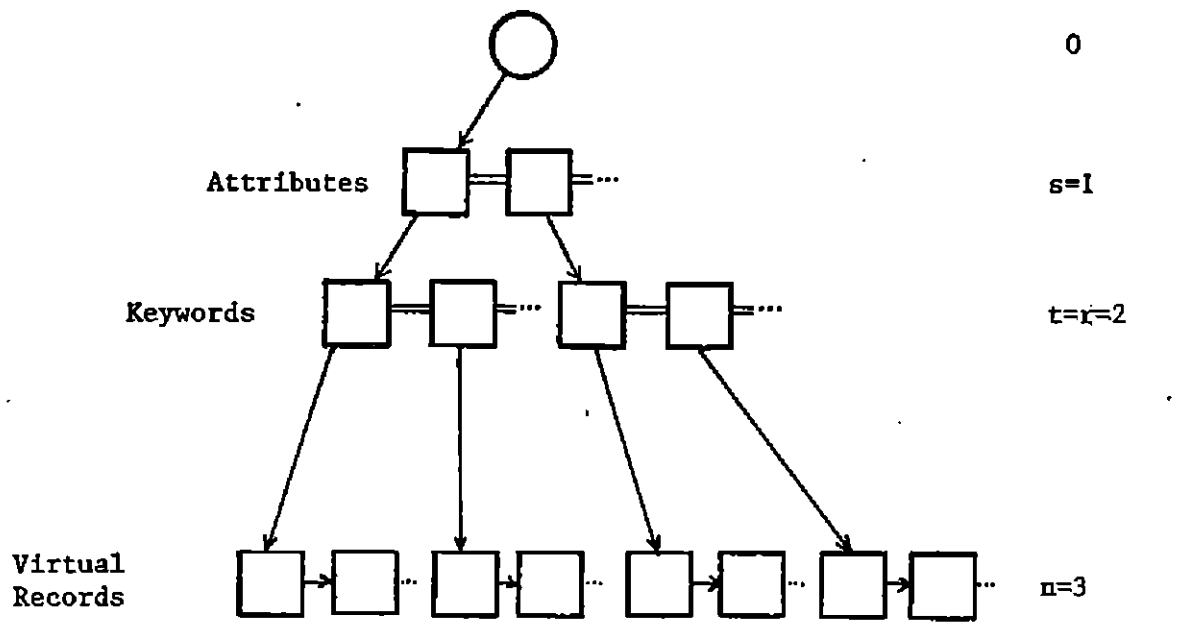


Figure 9b. The access tree representation of the multilist data base organization.

The access tree in Figure 9b has only 3 levels as a consequence of eliminating the accession list level. The filial sets in the directory are implemented sequentially while the filial sets in the file are implemented as linked lists. The model parameters have the following values:

$s=1, t=r=2, n=3$

$W_1=N_a$

$W_2=N_v$

$W_3=L$

$P_i=0,\ i=1,2$

$P_3=1$

$F_i=f,\ i=1,2,3$

$Q_i=0,\ i=1,2$

$Q_3=1$

Example 6. Cellular Data Base Organization

The Cellular data base organization can be considered as a compromise between the two extremes: inverted and multilist; the linked lists are shorter than those in multilist and the accession lists are shorter than those in inverted organization. For multilist the linked lists in the file have lengths equal to the sizes of k-sets (i.e. $W_3=L$) while for inverted organization the list length is always one (i.e. $W_4=1$). The linked lists in a cellular data base organization are restricted by the physical subdivisions of the storage device and are partitioned into sublists, as shown in Figure 10a. The accession list of a keyword value has only one pointer to each storage subdivision that contains a sublist.

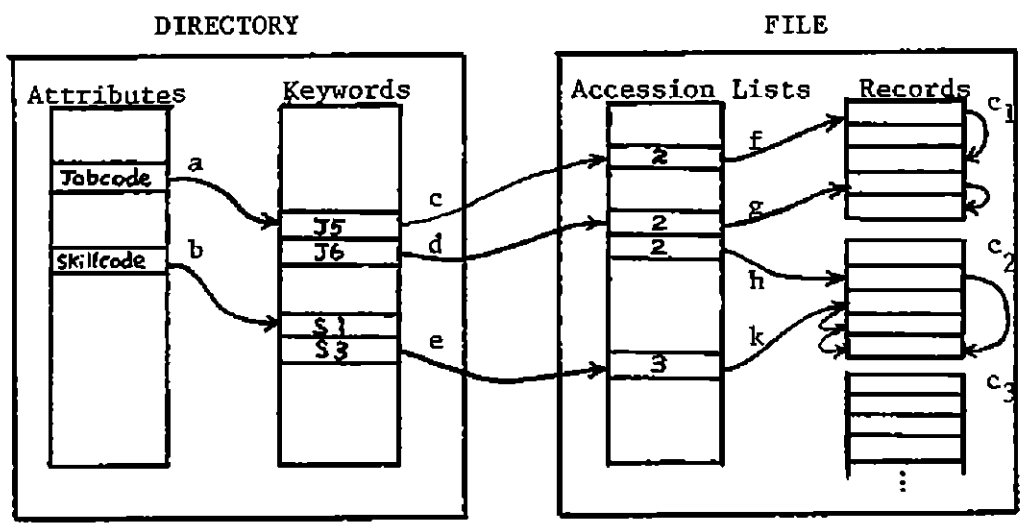


Figure 10a. Cellular data base organization.

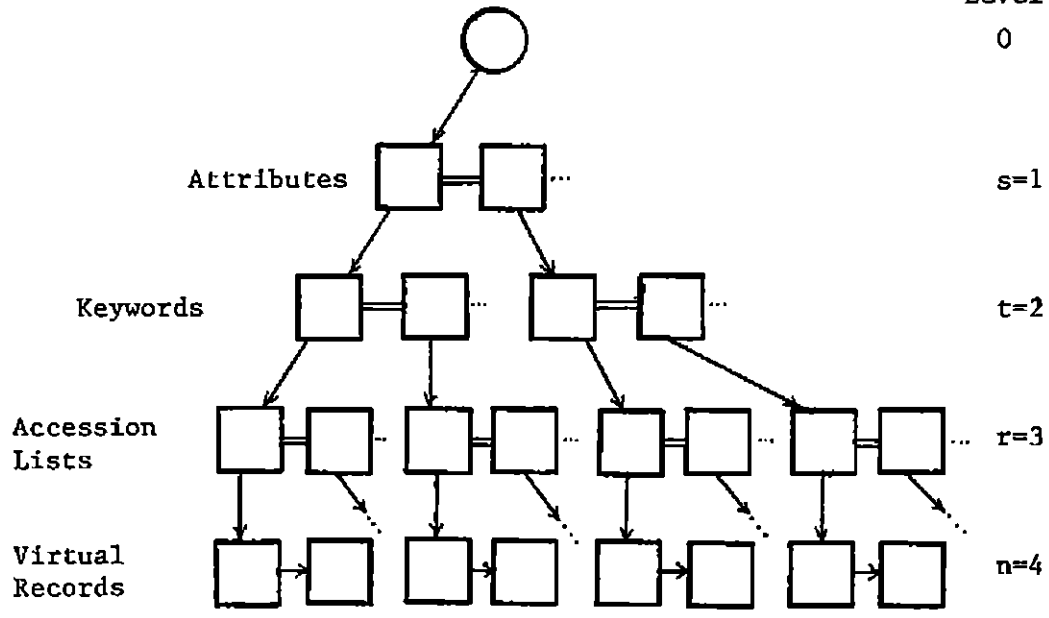


Figure 10b. The access model representation of the cellular data base organization.

Figure 10b shows the cellular organization as a combination of the two previous organizations. There are four levels in the access tree and the model parameters are:

$s=1, t=2, r=3, n=4$

$W_1=N_a$

$W_2=N_v$

$W_3=\lceil L/R_s \rceil = L_r$

$W_4=R_s$ (average list length in one storage subdivision)

$P_i=0,\ i=1,2,3$

$P_4=1,$

$F_i=f,\ i=1,2,3,4$

$Q_i=0,\ i=1,2,3$

$Q_4=1$

Example 7. Indexed Sequential Organization

The indexed sequential organization is organized with a single attribute - the primary key. It permits access to records of a file in either a sequential or direct access manner through the use of indices. When implemented on a disk device, records are ordered sequentially in "prime" tracks. As shown in Figure 11a, there is a track index and a cylinder overflow area in each cylinder. In addition, there is a cylinder index in each device. Master indexe may be constructed above the cylinder index level when the cylinder index is large. Entries in various index levels contain the highest reachable keyword values. Since no insertions occur above the keyword-value/record level, the index levels may be fully loaded. Overflows are handled by chaining the records in overflow areas.

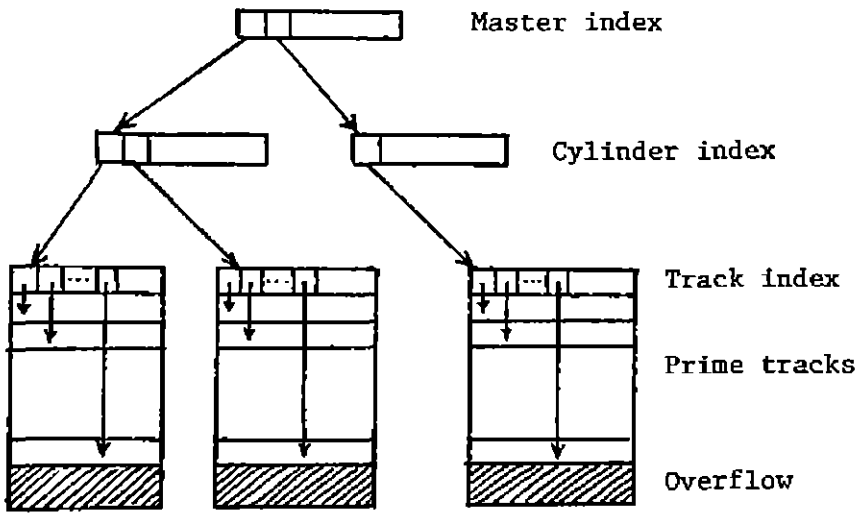


Figure 11a. Indexed sequential organization.

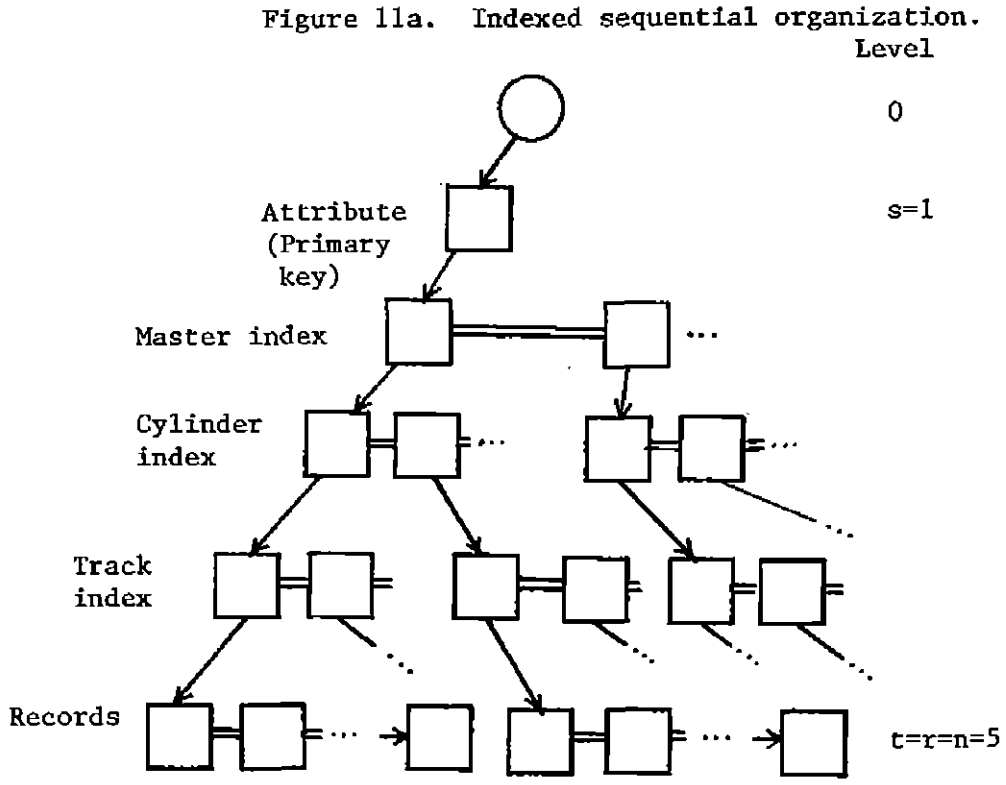


Figure 11b. The access tree representation of the indexed sequential organization.

As illustrated in Figure 11b, the access tree for this organization has only one attribute-name. Assume that there is one master index level. There are three index levels between the attribute-name level and the keyword value level. Since records are organized with primary keys, the keyword values are identifiers and each k-set contains a single record. This makes it possible to combine the keyword value level and the record level. The model parameters are:

$s=1, t=r=n=5$

$W_1=1$

$W_2=n_m$ (number of entries in the master index)

$W_3=n_c$ (number of entries in a cylinder index)

$W_4=n_t$ (number of entries in a track index)

$W_5=n_p$ (average number of records in a prime track and its overflow chain)

$P_i=0,\ F_i=1,\ Q_i=0,\ i=1,2,3,4$

$P_5=p$ (percent overflow)

$F_5=f$

$Q_5=1$

Example 8. B-tree Organization.

The B-tree organization is first introduced in [1] and is available in many variations for primary key searching [16]. The version illustrated in this example may be called the B^O-tree. In Figure 12a, the B^O-tree has a similar structure to the indexed sequential organization where records are on the terminal level and index entries contain highest keyword-values. The main difference is that the B^O-tree handles overflow

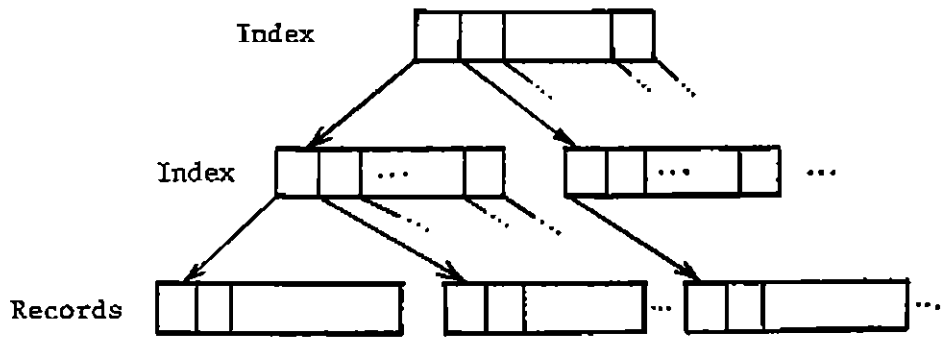


Figure 12a. B-tree Organization.

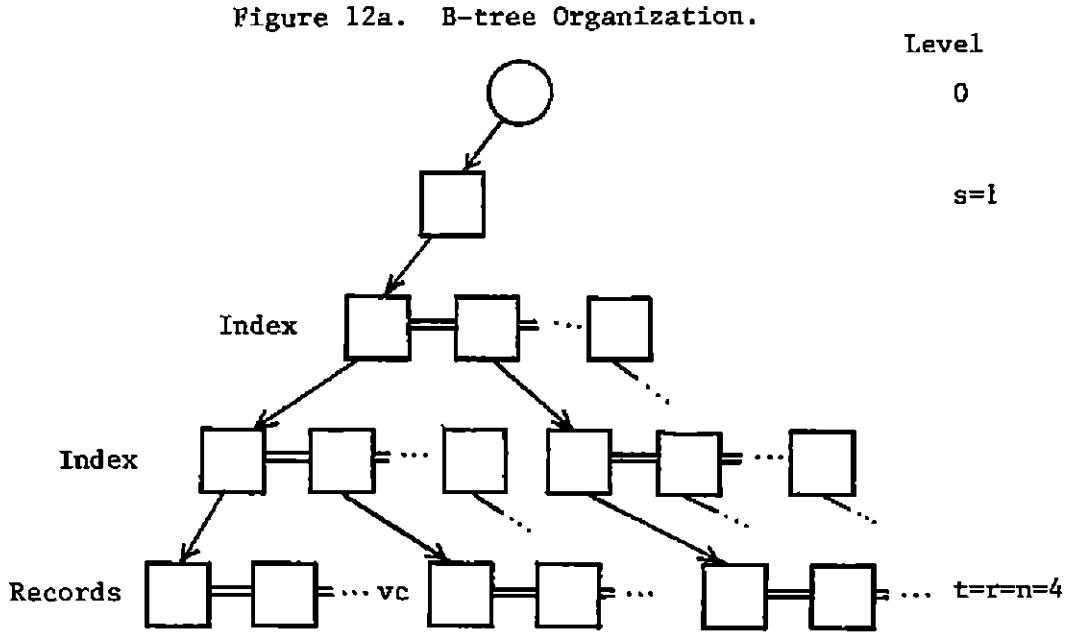


Figure 12b. The access tree representation of the B-tree organization.

by splitting blocks and avoids the costly overflow chain traversing. Insertion to higher level indices are only caused by overflow and splitting on the lower levels, thus a lower overall loading factor results ($.5 \leq f \leq 1$). In order to fit more entries into each index block, data compression techniques are often used. It is observed from the B-tree that the keyword-value in the last entry of each index may be eliminated since it is not needed in searching. In fact, this is true for all data base organizations covered by the model. In practice, however, this is not usually done since the saving is negligible for large indices.

The access tree representation of the B^O-tree is shown in Figure 12b. The model parameters are:

$s=1, t=r=n=4$ (assume that the B^O-tree has 3 levels)

$W_1=1$

$W_i=n_i$, $i=2,3$ (n_i is the average number of entries in a level i index block)

$W = n_p$ (average number of records in a record block)

$P_i=0, F_i=f \geq .5$, $Q_i=0$, $i=1,\ldots,4$

5. Generalized Retrieval Algorithms

In this section, it is demonstrated that the retrieval process can be modeled by a generalized parametric algorithm. Some search and retrieval primitive functions are defined. They are used to build the generalized directory retrieval and file retrieval algorithms.

It is convenient to define search primitives which indicate how to traverse the access paths of a data base organization. Each node in the access tree may have two successors: the sibling successor within the same filial set and the filial successor on the next level. Accordingly, we define the sibling successor function g and the filial successor function h. The domain and range of g and h are addresses denoted by x and y; both may be 0 to indicate the null address. We define

$$y = g(x)$$

$$y = h(x)$$

where

$$y = \begin{cases} x + \text{length of node } x & \text{if address sequential successor} \\ \text{sibling/filial pointer stored in node } x & \text{if pointer sequential successor} \\ 0 & \text{if no sibling/filial successor} \end{cases}$$

Using these primitive functions, we now describe an algorithm for searching the nodes in the access tree for a given set of search keys $R = \{\alpha_1, \alpha_2, \ldots \alpha_m\}$. The search begins at address x on the ℓ-th level and ends on the level j. The keys are parallel searched through the address space to reduce the access time. The response set R' contains the filial successor addresses of the nodes on level j which satisfy the search key.

Algorithm SEARCH (x,ℓ,j,R,R')

1. Form the set of triples:

   $S = \{(x,\ell,\alpha_1), (x,\ell,\alpha_2),\ldots,(x,\ell,\alpha_m)\}$

2. From S remove the triple (x_i,ℓ_i,α_i) with the smallest address $x_i \neq 0$ and $\ell_i \leq j$. If no such triple is available, the algorithm terminates and the response set is R'.

3. Retrieve the node at x_i if it has not previously been retrieved.

   If $\alpha_i > c(x_i)$ then the sibling is to be further searched.

   Insert $(g(x_i),\ell_i,\alpha_i)$ in S.

   If $\alpha_i \leq c(x_i)$ and $\ell_i < j$ then the filial is to be further searched.

   Insert $(h(x_i), \ell_i + 1, \alpha_i)$ in S

   If $\alpha_i = c(x_i)$ and $\ell_i = j$ Insert $(c(x_i),h(x_i))$ in R'

4. The algorithm is repeated at step 2.

We further describe an algorithm for retrieving all nodes on level j of the access tree reachable from a given set of starting addresses $X = \{x_1,x_2\ldots x_m\}$ on level ℓ. Similar to the SEARCH algorithm, the search is performed parallel through the address space. The response set R' contains the contents of nodes retrieved on level j.

Algorithm RETRIEVE (X,ℓ,j,R')

1. Form the set of ordered pairs

   $S = \{(x_1,\ell), (x_2,\ell),\ldots, (x_m,\ell)\}$

2. Remove from S the pair (x_i,ℓ_i) with the smallest $x_i \neq 0$. The algorithm terminates when $S = \phi$.

3. Retrieve the node at x_i if it has not previously been retrieved.

   If $\ell_i = j$ then $R' \leftarrow R' \cup \{(c(x_i), h(x_i))\}$

   If $g(x_i) \neq 0$ then $S \leftarrow S \cup \{(g(x_i), \ell_i)\}$

   If $h(x_i) \neq 0$ and $\ell_i < j$ then $S \leftarrow S \cup \{(h(x_i), \ell_i+1)\}$

4. The algorithm is repeated at step 2.

It will be shown that more complex search algorithms are all based on the above primitive algorithms. Using the above algorithms, generalized query retrieval is described next. We recall that a query is expressed in the disjunctive normal form.

$$(t_1 \wedge t_2 \wedge \ldots \wedge t_{c_1}) \vee (\quad) \vee \ldots \vee (\quad)$$

where t_i is a term corresponds to the i-th attribute in a conjunct,

$t_i = k_{i1} \vee k_{i2} \vee \ldots \vee k_{iq_{ji}}$ for the j-th conjunct.

Consider the following sets related to the query.

$A = \{a_i \mid i=1,\ldots,p;\ a_i$ is an attribute specified in the query$\}$

$K_i = \{k_{ij} \mid k_{ij}$ is a keyword value of the attribute a_i in the query$\}$

The set A is the collection of all attribute-names in the query. The set K_i contains the keywords in which the attribute-name a_i is specified.

The generalized query retrieval is performed in two phases: *directory retrieval and file retrieval.*

Directory Retrieval Algorithm

1. Search for attributes. Starting from the filial pointer r_o (address of the first level 1 node) of the root, search for the specified attributes on level s.

   SEARCH(r_o,1,s,A,A')

2. Search for keywords. For each pointer address a_i' in A', search for the set of keywords K_i on level t.

   SEARCH(a_i',s+1,t,K_i,K_i')

The response set of the directory retrieval is the set $K' = \bigcup K_i'$ of k-set (value,size,pointer) triples for the keywords in the query. These k-set pointers are the starting address for further searching in the file. The File Retrieval Algorithm retrieves and examines records in the k-sets to obtain the query response set.

File Retrieval Algorithm

1. Retrieve accession pointers for the k-sets.

   If t<r then RETRIEVE($\pi_h(K')$,t+1,r,G)

   else $G = \pi_{s,h}(K')$

   Noted that $\pi_{s,h}(K')$ is the projection of K' on the s (size) and the h (pointer) domains and each element g in G contains a k-subset size and an accession pointer, (i.e., g = (L,h)).

2. Pre-search the file and identify the "essential" accession pointers for each conjunct. Let G_i^j denote the subset of G for the i-th attribute in the j-th conjunct. Find the "intersection" G^j of G_1^j, G_2^j, ..., G_m^j for the j-th conjunct. That is,

   $G^j = \{(g_1^j, g_2^j, \ldots, g_m^j) \mid g_i^j \in G_i^j$ and g_i^j, i=1,...,m reference the same subdivision$\}$

Each tuple in G^j contains accession pointers for every attributes in the j-th conjunct and references one storage subdivision. The subdivisions corresponding to the tuples in G^j are called the essential subdivisions, which contain all the records satisfying the conjunct.

3. Find the prime accession pointers. For each conjunct and each essential subdivision, select the prime accession pointer h which has the minimal k-subset size. i.e., from each tuple $(g_1^j,\ldots,g_m^j) \in G^j$, find $\hat{g} = g_i^j = (L,h)$ which has the minimal L. The prime accession pointers will be used to retrieve the k-subsets in the essential subdivisions. Let X be the set of all prime accession pointers.

4. Search the file.

   RETRIEVE (X,r+1,n,R')

   Examine the records in R'. The query response set R consists of records in R' that satisfy the query description.

In order to see how the generalized retrieval algorithm accesses the specific data base organizations, consider the following example on the cellular list organization which is a combination of inverted and multilist organizations.

Example 9

The cellular list organization is illustrated in Example 6 and Figure 7. Consider the following query to the data base:

(jobcode,J5,J6)^(skillcode,S3)

where the query parameters are:

d=1

c_1=2

q_{11}=2 , q_{12}=1

A= {jobcode,skillcode}

K_1= {J5,J6}

K_2= {S3}

The example data base in Figure 7a is searched to answer the query. Using the generalized retrieval algorithm, the steps of searching are listed as follows. The symbols a,b,...,k indicate pointers, and c_1,c_2,... indicate the storage subdivisions (e.g. cylinders).

Search step	result
Directory Retrieval	
SEARCH(r_0,1,1,A,A')	A'={(Jobcode,a),(skillcode,b)}
SEARCH(a,2,2,K_1,K_1')	K_1'={(J5,2,c),(J6,4,d)}
SEARCH(b,2,2,K_2,K_2')	K_2'={(S3,3,e)}
	K'=K_1'$\cup K_2$'={(J5,2,c),(J6,4,d),(S3,3,e)}
File Retrieval	
RETRIEVE({c,d,e},3,3,G)	G={(2,f),(2,q),(2,h),(3,k)} (Jobcode; skillcode)
pre-search	G^1={((2,h),(3,k))}
	essential subdivision = c_2
find the prime accession pointers	X = {h}
RETRIEVE(X,4,4,R')	R' = {r_6,r_7,r_8}
examine the records	R = {r_8}

For inverted data base searching, the step to find the prime accession pointer is unnecessary since all the essential accession pointer have k-subset size 1. A storage subdivision is a block. Similarly, for multilist data base searching, the pre-search is unnecessary since the entire device is one storage subdivision and every accession pointer is essential.

It is clear that the searching for single attribute organizations (e.g. indexed sequential and B-tree) can be performed by applying only the directory retrieval algorithm. In fact, the directories in data base organizations actually consist of single-attribute organizations.

6. Generalized Performance Equations

It is sometimes desirable to estimate the performance of a proposed or an existing data base organization. This is typically done by analyzing the particular data base organization in detail, which can be a difficult and time consuming task. The Hierarchical Access Model presented in the previous sections enables us to develop a set of generalized cost equations for all data base organizations covered by the model. As will be shown later in this paper, previous cost analyses on specific data base organizations are special cases of the generalized cost equations.

The cost equations consist of two components. The storage costs include the storage for indices and the storage for records. The time costs include the secondary storage access time and the main memory search time. The search time is usually insignificant compared with the access time and is not included in the cost equations. In this paper, generalized cost equations for the retrieval access time are presented. Other cost equations, including the storage requirement and update costs, can be found in [18].

The storage system which contains the data base is modeled by a hierarchy of storage "buckets." The storage system is divided into buckets, each bucket is further divided into smaller buckets, and so on. The smallest buckets are known as blocks. Information is accessed and transfered between the storage system and the main memory (i.e., stored and retrieved) on the block level. Each bucket in the storage system is associated with two parameters: random access time and sequential access time. When a block is to be accessed randomly, all buckets containing it must also be accessed randomly. To access a sequential set of blocks, the blocks and all buckets containing them are accessed sequentially, except within each bucket the first sub-bucket is accessed randomly. In the scope of this paper, the hierarchy is considered to be a three-level structure as shown in Figure 13. Using the disk storage as an example, the level 1 buckets correspond to the cylinders and the level 2 buckets correspond to the blocks within cylinders. To access blocks w,x,y and z, for example, it is necessary to randomly access a, w and y, and sequentially access b, x and z. Assuming that the consecutive blocks within a cylinder can be accessed without delay, the storage system related parameters are:

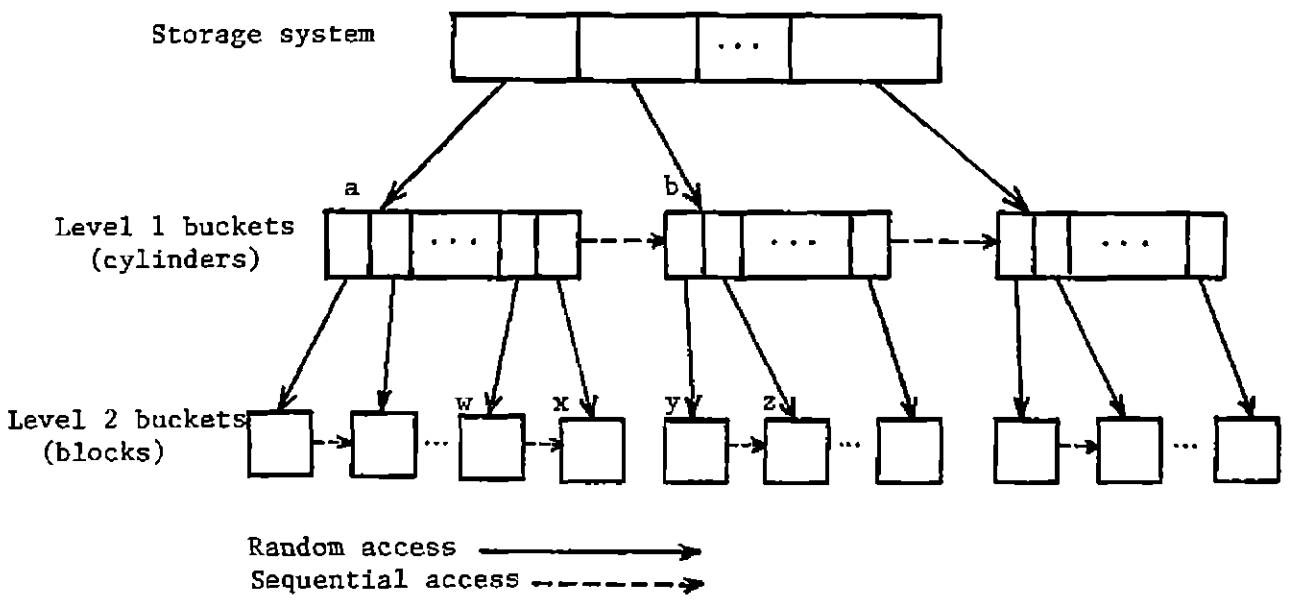


Figure 13. A hierarchy of storage subdivisions.

t_f = transfer time per unit of storage (e.g. character)

b = block size

$M_b = \lfloor \frac{b}{r} \rfloor$ blocking factor, r is the size of record stored in the block.

t_r = block random access time (1/2 revolution)

$t_r' = 0$ block sequential access time

c = cylinder size

$M_c = \lfloor \frac{c}{b} \rfloor$ number of blocks in a cylinder

t_s = cylinder random access time (average seek time)

t_s' = cylinder sequential access time (minimal seek time)

Suppose that there are N records to be stored sequentially in the storage system. The number of blocks required is $B = \lceil \frac{N}{M_b} \rceil$; each of the first $B' = \lfloor \frac{N}{M_b} \rfloor$ blocks contains M_b records while the last block, if not full, contains only $r_b = N(\text{mod } M_b)$ records. If all N records are equally likely to be searched, the expected number of blocks accessed in searching for one record is

$$E_b(N) = \frac{M_b}{N}(1+2+\ldots+B') + \frac{r_b}{N}(B'+1)$$

$$= \frac{B'+1}{2N}(M_b \cdot B' + 2r_b) = \frac{B'+1}{2N}(N+r_b)$$

$$= \frac{B'+1}{2}\left(1+\frac{r_b}{N}\right) \qquad (1)$$

Similarly, the number of cylinder required is $C = \lceil \frac{N}{M_b M_c} \rceil$ and the expected number of cylinders accessed in searching for one record is

$$E_c(N) = \frac{C'+1}{2}\left(1+\frac{r_c}{N}\right) \qquad (2)$$

where $C' = \lfloor \frac{N}{M_b M_c} \rfloor$ and $r_c = N(\text{mod } M_b M_c)$.

The expected access time for searching sequential and chaining records is computed next. The expected access time to search for all the N sequential records is given by

$$S(N) = t_s+t_r+ (C-1)(t_s'+t_r) + B\ b\ t_f \qquad (3)$$

The expected access time to search for one record from the N records is given by

$$S'(N) = t_s+t_r+ (E_c(N)-1)(t_s'+t_r)+ E_b(N)\ b\ t_f \qquad (4)$$

To analyze the expected access time for searching linked records, consider that the linked list is distributed over a storage subdivision that can hold X_r records in X_b blocks, using X_c cylinders. Assuming that the N records are randomly assigned to the X_r locations, the analysis in [17] approximates the expected number of blocks required (i.e., blocks that contain at least one record in the linked list).

$$G(X_b,N) = X_b(1-(1-1/X_b)^N)$$

The approximated number of cylinders required is similarly computed by.

$$G(X_c,N) = X_c(1-(1-1/X_c)^N)$$

The nature of the linked list searching requires the blocks and cylinders to be accessed randomly, and the expected access time for searching all the N records in a linked list is given by:

$$V(N) = G(X_c,N)t_s + G(X_b,N)(t_r + b\ t_f) \qquad (5)$$

To search for one record, it is on the average required to access only half of the linked list, and the access time is computed by:

$$V'(N) = G(X_c,\tfrac{N}{2})t_s + G(X_b,\tfrac{N}{2})(t_r + b\ t_f) \qquad (6)$$

Using eqs. (3),(4),(5), and (6), the access times for the algorithms SEARCH and RETRIEVAL are computed as follows

Algorithm SEARCH(x,ℓ,j,R,R')

Recall that at any stage of the searching, the filial set accessed may include a sequential portion of average size $(1- P_i)W_i$ and a chaining portion of average size P_iW_i. When m keys are batched in searching from level ℓ to level j, the number of filial sets accessed on level i, $\ell \leq i \leq j$, is $G(\prod_{k=\ell}^{i-1} W_k, m)$, since there are $\prod_{k=\ell}^{i-1} W_k$ filial sets reachable from a level ℓ node and the m search keys are distributed among them randomly.*

The sequential portions of the level i filial sets are searched independently with the access time

$$G(\prod_{k=\ell}^{i-1} W_k, m)S'((1-P_i)W_i)$$

However, the chaining portions share common overflow areas and their access time is computed by

$$V'(G(\prod_{k=\ell}^{i-1} W_k, m)P_iW_i)$$

Therefore, the access time for searching one record in a level i filial set is

$$\mathcal{S}_i(\ell,m)=(1-P_i)G(\prod_{k=\ell}^{i-1} W_k, m)S'((1-P_i)W_i)+P_iV'(G(\prod_{k=\ell}^{i-1} W_k, m)P_iW_i) \qquad (7)$$

The total access time for the algorithm SEARCH from level ℓ to level j, given m search keys, is thus

$$\mathcal{S}(\ell,j,m) = \sum_{i=\ell}^{j} \mathcal{S}_i(\ell,m) \qquad (8)$$

---

*Notation. $\prod_{k=\ell}^{j} W_k = 1$ for all $j < \ell$.

Algorithm RETRIEVE(X,ℓ,j,R')

There are $\prod_{k=\ell}^{i-1} W_k$ level i filial sets reachable from each of the m starting nodes on level ℓ, and every reachable filial set is accessed. Similar to the analysis in SEARCH, the level i access time is

$$\mathscr{R}_i(\ell,m)=(1-P_i)(\prod_{k=\ell}^{i-1} W_k)mS((1-P_i)W_i)+P_iV((\prod_{k=\ell}^{i-1} W_k)mP_iW_i) \qquad (9)$$

The total access time for the algorithm RETRIEVE from level ℓ to level j, given m starting addresses, is thus

$$\mathscr{R}(\ell,j,m) = \sum_{i=\ell}^{j} \mathscr{R}_i(\ell,m) \qquad (10)$$

Directory Retrieval Algorithm

Recall that the queries are described by the following parameters:

d = average number of conjuncts in a query

p = total number of attributes specified in a query

q = average number of keyword values specified for each attribute

q̂ = minimal number of keyword values specified for an attribute

The access time required to search the p attributes on the attribute level s is $\mathscr{S}(1,s,p)$. For each attribute, the access time for further searching the q keyword values on level t is $\mathscr{S}(s+1,t,q)$. The total access time for directory retrieval is therefore

$$\mathscr{D}= \mathscr{S}(1,s,p) + p\mathscr{S}(s+1,t,q) \qquad (11)$$

File Retrieval Algorithm

There are pq keywords specified in the query. The access time required to retrieve their accession lists is $\mathscr{R}(t+1,r,pq)$.

From the definition of a query, the i-th conjunct has c_i attribute-names specified and the number of keywords specified for the attributes are $q_{i1},q_{i2},\ldots,q_{ic_i}$. The average length of the accession list for each keyword is $y_r = \prod_{k=t+1}^{r} W_k$. The accession list lengths for the attributes are $q_{i1}y_r,q_{i2}y_r,\ldots,q_{ic_i}y_r$. After the pre-search step to find the "intersection" of the accession lists, the number of essential subdivisions for the ith conjunction is smaller than the length of the shortest accession list $\min_j\{q_{ij}\}y_r$. Averaging over the conjunctions, the number of essential subdivisions for each conjunction is bounded by $\operatorname{avg}_i\{\min_j\{q_{ij}\}y_r\} = \bar{q}\, y_r$. Assuming that the keyword values are distributed uniformly, the keywords for each attribute which does not have the minimal accession list further restrict the shortest accession list by $\frac{q}{N_v}$ which is the ratio of specified/actual keywords per attribute. Therefore, the number of essential subdivisions is approximately

$$\mathscr{S} = \left\lceil \bar{q} y_r \left(\frac{q}{N_v}\right)^{c-1} \right\rceil d$$

where c is the average number of attributes per conjunction.

The retrieval time for records in the essential subdivisions is $\mathscr{R}(r+1,n,\mathscr{S})$ and the total access time for the file retrieval algorithm is

$$\mathscr{T} = \mathscr{R}(t+1,r,pq) + \mathscr{R}(r+1,n,\mathscr{S}) \qquad (12)$$

Using eqs. (11) and (12), the total access time for a data base organization can be obtained. To validate the generalized access cost equation, it is sufficient to show that previous analyses on data base organization can be derived as special cases of this model. The following examples show the synthesis of cost equations for the inverted, multilist and cellular data base organizations.

Example 10

The inverted data base organization was analyzed by Cardenas in some details and the cost equations for query retrieval was derived and used in a simulation system to obtain performance evaluation results [3]. This example shows that Cardenas' cost equations can be synthesized as a special case of the generalized cost equation given in this section.

From equations (11) and (12), we have the query cost equation

$$\mathscr{C} = \mathscr{S}(1,s,p)+p\mathscr{S}(s+1,t,q)+\mathscr{R}(t+1,r,pq)+\mathscr{R}(r+1,n,\mathscr{B}) \qquad (13)$$

In the simple case when we have a one-level index in each stage of searching (i.e. s=1,t=2,r=3,n=4), the query cost equation is simplified. Using eqs. (7)-(10), we have

$$\begin{aligned}\mathscr{C} &= \mathscr{S}_1(1,p)+p\,\mathscr{S}_2(2,q)+\mathscr{R}_3(3,pq)+\mathscr{R}_4(4,\mathscr{B}) \\ &= (1-P_1)S'(\mu_1)+P_1V'(\nu_1)+p(1-P_2)S'(\mu_2)+pP_2V'(\nu_2) \\ &\quad + (1-P_3)pqS(\mu_3)+P_3V(pq\nu_3)+(1-P_4)\,\mathscr{B}\,S(\mu_4)+P_4V(\mathscr{B}\nu_4) \qquad (14)\end{aligned}$$

where $\mu_i=(1-P_i)W_i$ and $\nu_i = P_iW_i$.

Under Cardenas' assumption, for an inverted data base organization, we have

$E_c(N_a)=E_b(N_a)=1$ (all attribute-names are stored in one block)

$E_c(N_v)=E_b(N_v)=1$ (keywords of each attribute-name are stored in one block)

C=1 on level r=3 (each accession list requires no more than one cylinder)

C=B=1 on level n=4 (each record requires no more than one block)

From example 4, the other relevant parameters are

$W_1=N_a, W_2=N_v, W_3=L, W_4=1, P_i=0, i=1,2,3$, and $P_4=1$. Thus eq. (14) is further simplified.

$$\begin{aligned}\mathscr{E} &= S'(N_a)+pS'(N_v)+pqS(L)+V(\mathscr{S}) \\ &= t_s+t_r+bt_f+p(T_s+t_r+bt_f)+pq(t_s+t_r+Bbt_f) \\ &\quad + G(X_c,\mathscr{S})t_s+G(X_b,\mathscr{S})(t_r+bt_f) \\ &= T_T+pT_T+pq\left(t_s+t_r+\left\lceil\frac{L}{M_b}\right\rceil bt_f\right)+G(X_c,\mathscr{S})t_s+G(X_b,\mathscr{S})(t_r+bt_f) \qquad (15)\end{aligned}$$

where $T_T = t_s+t_r+bt_f$ denotes the access times for one block and

$\mathscr{S}=\left\lceil qL\left(\frac{q}{N_r}\right)^{c-1}\right\rceil d$ is the expected size of accession lists intersection.

Comparing eq. (15) to the access time equations of Cardenas (eqs. (23) and (24) in [3]), the following differences are observed:

1. Instead of $\left\lceil\frac{L}{M_b}\right\rceil T_T$, eq. (15) uses $t_s+t_r+\left\lceil\frac{L}{M_b}\right\rceil bt_f$, since we assume that the sequential access of $\left\lceil\frac{L}{M_b}\right\rceil$ blocks can be performed without delay.
2. Cardenas approximated the intersection size of accession lists by the shortest accession list (i.e. $\mathscr{S}' \doteq qLd$), and therefore ignored the effect of pre-search.
3. For the last two terms of eq. (15), Cardenas used $G(X_b,\mathscr{S}')T_T$ which is less accurate since not every block accessed requires a random cylinder seek.

Example 11

From Example 5, the multilist data base organization has parameters $s=1, t=r=2, n=3, W_1=N_a, W_2=N_v, W_3=L, P_1=P_2=0$, and $P_3=1$. From the query cost equation (13), we have

$$\mathscr{C} = \mathscr{S}_1(1,p)+p\mathscr{S}_2(2,q)+\mathscr{R}_3(3,\mathscr{E})$$

$$= S'(N_a)+pS'(N_v)+V(\mathscr{E}L)$$

$$= T_T+pT_T+V\left(\left\lceil \hat{q}\left(\frac{q}{N_r}\right)^{c-1}\right\rceil dL\right)$$

For small q and $\hat{q}$, we have

$$\mathscr{C} \cong T_T+pT_T+V(dL)$$

$$= T_T+pT_T+G(X_c,dL)+G(X_b,dL)(t_r+bt_f) \qquad (16)$$

The cellular data base organization in Example 6 has parameters $s=1, t=2, r=3, n=4, W_1=N_a, W_2=N_v, W_3=L_r, W_4=R_s$, $P_i=0, i=1,2,3$, and $P_4=1$. From eq. (14), we have

$$\mathscr{C} = S'(N_a)+pS'(N_v)+pqS(L_r)+V(\mathscr{E}R_s)$$

$$= T_T+pT_T+pq\left(t_s+t_r+\left\lceil\frac{L_r}{M_b}\right\rceil bt_f\right)+G(X_c,\mathscr{E}R_s)t_s+G(X_b,\mathscr{E}R_s)(t_r+bt_f) \qquad (17)$$

where $\mathscr{E} = \left\lceil \hat{q}L_r\left(\frac{q}{N_r}\right)^{c-1}\right\rceil d$ is the expected size of the accession lists intersection.

The preceding analysis and synthesis shows that the generalized retrieval algorithms and cost equations can be applied to common data base organizations. The model and cost equations cover, however, many other data base organizations. For example, cases in which the large number of records and keyword values requires the introduction of additional index levels are included in the model. The generality of the model allows the setting of the parameters to study other possible refinements of data base organizations.

7. Applications

One of the important problems in the design and administration of data base systems is the performance evaluation of physical data base organizations. Previous analytical approaches to performance evaluation requires separate models for different data base organizations. Examples of single-attribute organization analysis include the VSAM model [6] and the FOREM simulation model [10]. Previous models for multi-attribute data base organizations include the analytic models developed by Lowe [8] and Martin [9], and the simulation models reported by Cardenas [2] and Siler [13]. In each case, a given structure is analyzed and an analytic or simulation evaluation is performed. Since different assumptions and techniques are used in different analyses, it is difficult to compare results from individual models.

The model and cost equations presented in this paper are used to implement a File Design Analyzer which evaluates and ranks the most well known data base organizations in terms of the access time and storage requirements [18,19]. In order to evaluate the generalized cost equations, the Analyzer accepts as inputs, parameters which describe the storage characteristics, the logical data base structures, and accessing workload in terms of simple retrieval, query retrieval, insertion, deletion and modification of records. The Analyzer was implemented in ANSI/FORTRAN and is operational on the Honeywell 635 and IBM 370/168. An extended version is available on the CDC 6500. Preliminary experiments of using the File Design Analyzer to investigate the sensitivity of design parameters for specific real-life data base organizations are reported in [15].

It is evident from example 10 that the generalized cost equation can be more accurate than some specific analysis. This is because the generalized Hierarchical Access Model can be adjusted and validated by comparing it to many independent analyses, thus unifies the advantages of individual models. It should be noted, however, that the generalized cost equation is less accurate than some detailed models for analyzing the "microscopic" performance of specific data base organizations. An analysis accounting for the low level details of access methods and data management of a particular operating system and I/O subsystem would be exceedingly complex. In contrast, the parameters required for the generalized cost equations are simple and readily obtainable. In such cases, the Hierarchical Access Model and its cost equations may be used as references to validate the initial low level model design.

The generalized performance evaluation is useful for the system designer to tune an existing data base organization by varying certain parameters and observing the effect on system performance. It can also be used to estimate the performance of proposed data base organizations prior to implementation. A simulation system for the selection of inverted, multilist, and doubly chained tree data base organizations was reported by Cardenas [2]. Severance also reported a system which generates design alternatives and compares their performance [12]. The wide range of performance evaluation using the Hierarchical Access Model provides a sound basis for extending the scope of data base organization selection. A physical data base optimization system which automatically selects data base organizations was developed and some preliminary test cases were

reported [20]. The data base system designer selects a subset of model parameters as design variables. The optimal values of design variables which minimize the generalized cost function are numerically determined by the optimization system using mathematical programming techniques. The data base organization corresponds to these optimal variable values, which may or may not be a well known design, is presented to the system designer as an initial design. Low level design and implementation details will be determined and "tuned" by the designer to satisfy a particular application.

8. Conclusions

The Hierarchical Access Model provides a general framework for data base organizations. It is shown that most commonly used index and data base organizations are all special cases derivable from the model. The distinctions and relationships among the various index and data base structures are made explicit. It is clear that the model also enables the characterization of new structures or combinations which may have promising applications.

A set of cost equations associated with the model are presented. Special cases of the cost equations are compared with previous data base analysis and are demonstrated to have adequate precision. The model and cost equations are general in that a wide variety of data base organizations with arbitrary complex index structures can be characterized. It should be emphasized that the Hierarchical Access Model presented in this paper provides a systematic approach to analyze and evaluate data base organizations, and its advantages over more specific models are obvious.

REFERENCES

1. Bayer, R. and McCreight, E.M. "Organization and maintenance of large ordered indexes" Acta Informatica 1, 1972, 173-189.

2. Cardenas, A. F. "Evaluation and selection of file organization - a model and system", Comm. ACM 16,9 (Sept. 1973), 540-548.

3. Cardenas, A.F., "Analysis and Performance of Inverted Data Base Structures." Comm. ACM 18,5 (May 1975), 253-263.

4. Chen, P.P.S. "The entity-relationship model - Toward a unified view of data" ACM Trans. on Database Systems 1, 1 (March 1976), 9-36.

5. Hsiao, D. and Harary, F. "A Formal System for Information Retrieval from Files", Comm. ACM 13,2 (Feb. 1970), 67-73; "Correction" Ibid., 13,4 (April 1970) p. 266.

6. Keehn, D.G. and Lacy, J.O. "VSAM data set design parameters." IBM Sys. J. 13,3 1974, 186-212.

7. Lefkovitz, D. File Structures for On-Line Systems. Spartan Books, New York, 1969.

8. Lowe, T.C. "The Influence of Data Base Characteristics and Usage on Direct Access File Organization" J.ACM 15,4 (Oct. 1968), 535-548.

9. Martin, L.D. "A Model for File Structure Determination for Large On-Line Data Files", Proc. FILE 68 International Seminar on File Organization, 1968, 793-834.

10. Senko, M.E., Lum, V.Y. and Owens, P.J. "A File Organization Evaluation Model (FOREM)", Proc. IFIP 1968, 1968, C19-C23.

11. Severance, D.G. "A parametric model of alternative file structures", Information Systems 1,2 (1975), 51-55.

12. Severance, D.G., Some Generalized Modeling Structures for Use in Design of File Organizations, Ph.D. Dissertation, University of Michigan, 1972.

13. Siler, K.F. "A Stochastic Evaluation Model for Database Organizations in Data Retrieval Systems" Comm. ACM 19,2 (Feb. 1976), 84-95.

14. Sussenguth, E.H. "Use of tree structures for processing files" Comm. ACM (May 1963) 272-279.

15. Teorey, T.J. and Das, K.S. "Application of an Analytical Model to Evaluate Storage Structures," Proc. ACM-SIGMOD Conf. (June 1976).

16. Wagner, R.E. "Index design considerations" IBM Sys. J. 12,4 1973, 351-367.

17. Yao, S.B. "Approximating the Number of Blocks Access in Data Base Organizations" Tech. Report CSD-TR 184, Computer Sciences, Purdue University, (April 1976).

18. Yao, S.B., Evaluation and Optimization of File Organizations through Analytic Modeling, Ph.D. Dissertation, U. of Michigan, 1974.

19. Yao, S.B., Das, K.S. and Teorey, T.J. "A dynamic database reorganization algorithm". to appear ACM Trans. on Database Systems.

20. Yao, S.B. and Merten, A. "Selection of file organizations through analytic modeling" Proc. Very Large Data Bases (Sept. 1975).