This paper describes an approach to data management that is based on a hierarchical organization of the data management control function and makes use of list processing concepts.

Discussed are the separation of the logical and physical control functions as well as the data-element and operating-system controls. This hierarchical approach establishes a common basis for the creation, maintenance, and retrieval of data in direct-access storage. Logical functions express the control and management of generalized physical data structures; the physical level typically includes strings for data retrieval and maintenance.

Undirected graphs, and matrices derived from them, illustrate the data management relationships within the physical level. The same type of analysis may be used to show relationships between the hierarchical levels.

Hierarchical structure for data management

by W. R. Henry

The concept of list processing, or chaining, has been used as a technique for the manipulation of logical data strings for many years and has been formalized as a language[1] for handling data in computer storage. List processing has also been utilized with limited success for the control of data in direct-access storage devices. In general, when list structures are used for external data control, only a subset of the possible data structures is implemented, and the logical and physical relationships are approached as a single entity. Thus, the many ventures into this area are highly individualized, resulting in duplication and incompatibility.

Consider various forms of list processing implicit within the SYSTEM/360 Operating System (OS/360)—access methods, compilers, and application packages. Such programs could be improved by a more formal discipline directed to the support of list data structures, particularly for data control where direct-access storage devices are used.

Data structures and control functions that are possible on direct-access storage devices must be defined prior to the development of a data base that supports a language or access-method package. Historically, the utilization of direct-access devices has been approached from three levels:

- Device or channel program
- Access-method package
- Composite data-management system

On today's sophisticated equipment, programming at the device or channel level usually cannot be economically justified.

Access-method packages and data management systems are widely used with considerable success where their capabilities match the data requirements of the user. However, inherent restrictions imposed on system design within this environment become evident when attempts are made to implement information systems that are based upon specific data organization or response-time requirements.

It is unlikely that a single access-method package or data management system could support either an information system based upon a hierarchical data structure or a communication-oriented system with definite response-time requirements. In order to implement a system at the simplest control level, where data is managed as a resource at the traditional logical record level, the user generally must provide his own management routines for the access methods.

This paper discusses data management as a hierarchical structure that incorporates concepts of list processing. First, some overall aspects of the data management environment are introduced as background material, stressing the distinction between the logical determination process and the actual "physical" retrieval of data. Next, a common physical data structure used for the retrieval from direct-access storage is developed. The logical organization and control of this physical data structure is discussed in the last section of the paper.

Data management environment

In the past few years, the design of system support programs has made it possible for application programs to become substantially independent of the access-method routines as well as the actual physical storage and retrieval of data. In such an environment,[2-4] application programs are generally written in a high-level language. Requests for data are directed through a common data-management interface module which, in turn, utilizes one or more access-method routines as its resource. The data-management interface validates the request and restructures the user's request into a format acceptable to the appropriate access-method routine. The routine operates on the data request and returns control to the data management interface, which posts the fact that the request has been completed.

traditional environment

The access-method routines receive a structured request. That is, the request is specific in terms of data-set name and key or address of the logical segment to be retrieved. The access-method routines translate the request and pass it to the input/output supervisor in the form of channel programs to be scheduled and executed.

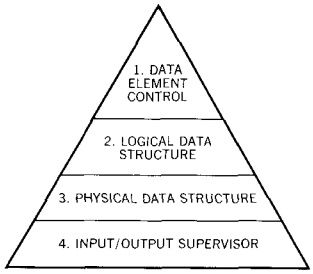
A distinction is made between the *logical* retrieval-determination process and the actual retrieval process at the *physical* level. The execution of the access-method routine merges two distinct functions into one: (1) the logical interpretation and

decision process of how the record is to be retrieved, and (2) the translation of the data into a series of "physical" requests to the I/O supervisor. The access method contains not only the logical retrieval-organization sequence, but also the characteristics of the physical data structure and the external storage medium. Today we see a trend wherein once data is placed under the control of a specific access-method routine, involving more than physical sequential retrieval, no other access-method routine can reference the data. Generally speaking, the lowest-level access-method routine is used to retrieve data.

Just as the current access methods are generally mutually incompatible, data-management systems are similarly irreconcilable. The more sophisticated the system, the more incompatible it is with other systems and access methods. Differences at the physical level may be minor or even nonexistent, but, as additional orders of logical interpretation are involved in translating a request to the physical level, differences grow.

A design criterion for an effective data management system is that it should be able to manage data as a resource at multiple levels in a dynamic environment. The user should be free to determine the level of data independence desired in any situation at execution time. It follows that the higher the degree of data independence, the greater the retrieval overhead. Although the overhead can be lessened by the choice of the physical data structure, each successive level of control must be built upon a common foundation in order not to preclude a lower interface when desired. Thus, not only may data be considered as a hierarchical structure, but also the data management is hierarchically structured.

Figure 1 Data management control hierarchy



The hierarchical structure of data management is represented by Figure 1, wherein the actual data at the element level is at the apex. Here, a *data element* may be defined as a field in the traditional sense. In some cases, the smallest element at the apex may be a quantum of data or a group item representing two or more physically contiguous data elements. At this level, the user is independent of both the logical and physical structure of the data, as exemplified by an inquiry system utilizing a natural language. A request for a data element is expressed without implied knowledge of how the data might be structured.

The second level of the pyramid represents the data hierarchy at the logical-string level, such as a line item in a purchase order. The user interface at this level implies knowledge of the logical structure of the data or the ability to determine its logical structure.

The third level represents the actual organization or physical list structure of the data. Extending the purchase-order example, an invoice line item might be in a physical list where its header is the invoice, the original purchase order, or a line item of the purchase order. The actual physical association of the data should be determined in accordance with the optimization of system resources. Restating this idea, factors such as frequency of use,

data reconstruction criteria, or response-time requirements should determine the number of hierarchical levels.

In terms of OS/360, the lowest level of control is through the Input/Output Supervisor (IOS), which handles the scheduling of device requests. The IOS does not include the logical or physical data structures. Thus, IOS must be supplied with the physical request in the form of channel programs. The physical level of control in the hierarchy (level 3 in Figure 1) formulates requests that reflect the user data requirements.

As soon as a higher control than the IOS is used, all commonality disappears, and the logical and physical structures of data are intertwined. For example, in order to reference a data set created by the Queued Sequential Access Method (QSAM) using another method such as the Basic Sequential Access Method (BSAM), Basic Direct Access Method (BDAM), or the system macroinstruction to execute a channel program (EXCP),[5] the user must know at least the logical record length, as opposed to the physical record length. This knowledge is not difficult where only physically sequential data sets are involved; but indexed data-set organizations, such as the Indexed Sequential Access Method (ISAM) or the Basic Partitioned Access Method (BPAM), present a significant problem.[5] Here, the logical and physical structures are merged, and access to those data structures using other than ISAM or BPAM is difficult.

An indexed file organization such as ISAM should itself be considered as a hierarchical data structure. In such a case, each index level is a list structure in which each entry points to a sublist until the actual data is reached. Within this context, all access methods are logically identical, and their differences appear only at the physical level, in terms of string relationships. However, if the logical versus physical distinction is maintained, then symbolic references to the physical data structure are independent of the actual organization. Thus, an access method for indexed-file organizations may be viewed as a general program for the creation of a hierarchical data structure that creates additional indices to existing data, since, at the physical level, all data management is string manipulation.

illustrative example

To illustrate the concepts of hierarchical access structures, the Bill of Material Processor[6] is chosen as a simple example. Although the name implies an application-limited system, we can look at it as a generalized file maintenance and retrieval program. At the physical level, the processor supports three specific forms of physical data structures. However, no other logical access method can be used to process those data structures.

The Bill of Material Processor supports the following physical data structures:

- Part-number and work-center master files, which are indexed master files that can be retrieved randomly or sequentially (called the Control Sequential Access Method—CSAM).

- A product structure file and a standard routing file, which are lists (or chained files) with variable numbers of repeating segments chained from a specific master record.
- The part-number and where-used files, which are imbedded list strings. They have their headers in individual master records, and they chain records or segments together in the product-structure and standard-routing files.

In terms of application requirements, there are many desirable logical sequences, all involving different physical sequences of string retrieval. In each case, the problem program can remain partially independent of the storage organization and retrieval by utilizing a string-retrieval macroinstruction to retrieve data in a variety of logical sequences.

The macroinstruction supplied with the Bill of Materials Processor for the string retrieval function is called CHA$E. Although this macroinstruction is highly structured in its informational requirements, it is powerful because the user may nest several levels of string retrieval. The routine is not recursive, but provides a similar facility. Thus, CHA$E provides a simple, but good, example of the list retrieval function within a physical data structure, particularly if CHA$E were expanded to operate at the symbolic level. In the present format, the user must have the master record or list header in main storage. The master record points to the string to be retrieved, and—in addition—the user must supply the macroinstruction with locations of all list pointers involved.

The symbolic elements of a request for CHA$E are simple to specify, and the elements exemplify the advantage of distinguishing between different levels of data organization. For example, if the user desires to know all the items in which a specified part is used and obtain the master record of each item, the general format of the request requires the following: names of the two lists to be retrieved, identification of the list header, and the location in which records should be placed. Expressed in a high-level language, such as COBOL, we might state:

```
CALL CHA$E USING STRING-1, STRING-2, LIST-HEADER,
WORK-1, WORK-2.
```

Thus it can be seen that the current implementation of the CHA$E string retrieval macroinstruction requires the user to be familiar with the physical and logical data structures plus the actual list organization. On the other hand, the example symbolic representation of the identical function allows the user to be free from the list structure itself but requires that he know the logical and physical data relationships. If the list structure of the where-used data were changed from an imbedded list to a sublist similar to the product-structure list, there would be no change to the user program. Although the list structure changes, the physical hierarchical structure does not.

If the physical hierarchy is changed, the problem program must be modified, since it has an interface at the physical and not at the logical level. For example, if the decision were made to store where-used information as repeating segments within each master record, the logical hierarchy would be unchanged, but the physical hierarchy would no longer recognize a physical string called "where used." Thus, the logical data hierarchy could be considered as a directory or index into the physical structure.

In spite of this change, the COBOL example could be identical except that the call would be to a logical CHA$E instead of a physical CHA$E function. If the logical and physical structures were identical (or parallel), the request might be directly transferred to the physical level. However, where the structure is not identical, the request would be remapped or translated into the appropriate symbolic physical requests. At this point, the user is working with logical strings of data, not with data elements (as discussed earlier) and the interfacing requests are highly structured even at the symbolic level.

Extending the example, assume that the requestor is not an application program but a person at a terminal who is conversing with the computer via an inquiry language. The previous retrieval request might be made as follows: "On which assemblies is part ABC used?" or "Is subsystem ABC used in system XYZ?" Now, an inquiry-language program must analyse the terminal input and attempt to correlate it with the logical-element data structure, which is an index or directory into the logical hierarchical data structure.

Within this environment, each level of data independence represents an entry into the total data management hierarchy. At the same time, each level is a list structure whether data, indices, directories, or catalogs are involved. It is necessary to address the physical characteristics of data as they are stored and retrieved on direct-access storage devices separately from the logical relationship of data as they are required by the ultimate user. This logical-versus-physical distinction is important: although their relationship may be identical (or parallel), this is not necessary and in many cases undesirable.

Physical data structure

In order to support a hierarchical data control function, a common physical data structure is now defined. Such a structure requires a different approach to data organization on direct-access storage.

A direct-access storage device is capable of supporting two physical data organizations, namely, sequential and direct. As an independent structure, the purely sequential organization has been reasonably well defined and implemented at the traditional access-method level. On the other hand, the direct organization has usually been categorized as being either a pure direct (i.e., using an actual address, a transformation of the address, or a

randomizing algorithm) or an indexed type organization, which provides both random and sequential retrieval capability.

The latter viewpoint toward direct and indexed data structures has tended to stratify the approach to the support of direct-access devices. Actually, the direct and indexing approaches merely define entry techniques to the prime data structure and have no inherent relationship to the actual prime data being stored on the device.

organizational structure

Any data structure for a direct-access device can be described in fundamental terms of external list organization and internal list structure, reflecting the organizational and retrieval interrelationships. The basic organization can be sequential or direct and the structure can be sequential, direct, or combined for efficient retrieval and maintenance. These relationships can be grouped into the five basic organization-structure retrieval specifications shown in Table 1.

Sequential organization and sequential structure (S/S) is the sequential organization that has a strictly sequential structure internally. Such a structure is a restatement of an ordinary sequential data set or list string. However, since the structure resides on a direct-access device, it has the following inherent facilities: update in place, start a sequential scan at a directly accessed location, and permit more than one scan on a concurrent basis within the same data structure. The structure may be considered as a list string with transparent pointers represented by physical continuity.

Sequential organization with a dual structure (S/SD) might be considered as a physically sequential string having the additional facility for logically inserting new segments into the list. Direct linkage is used to maintain logical ties between physically noncontiguous segments. Pointers may be specified at a definable control level, such as the segment, block, or track level.

Direct organization and direct structure (D/D) imply that the data string is discontinuous and that any sequential continuity is a coincidence. Thus, look-ahead physical-sequential buffering probably would be of no value. For string-retrieval purposes, the list pointers are individually checked for additional segment retrieval.

Table 1 Organization-structure retrieval specifications

		External list organization	*Internal list structure*
1	S/S	Sequential	Sequential
2	S/SD	Sequential	Sequential/Direct
3	D/D	Direct	Direct
4	D/S	Direct	Sequential
5	D/SD	Direct	Sequential/Direct

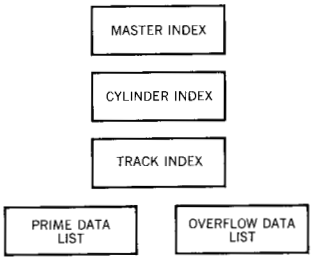
Direct organization with sequential list structures (D/S) is a list string or chain of data segments organized as a direct data structure. However, D/S has the characteristic that the segment groups are physically contiguous. The implications of this structure appear primarily in the areas of retrieval, where look-ahead is advantageous, and in segmenting an otherwise sequential organization.

Direct organization with a dual structure (D/SD) is an extension of the D/S structure with the facility to logically insert new segments into individual substrings of the direct organization. In general, the /SD type of internal list structure provides the ability to merge or add one string with another string without reorganizing both.

Each of these five organizational and structural retrieval techniques services a variety of possible list structures as well as the traditional sequential processing. There are many ways of creating list strings and manipulating list pointers. However, all the possible list structures can be categorized within the general organization and structural interrelationships presented at their creation.

The organization structures mentioned currently exist in various forms. They appear as subsets of access methods, data-management systems, and storage-paging systems. However, because the logical interpretation of the organization and the physical retrieval of the data are intermingled, the same basic data structures have become mutually incompatible. Figures 2 and 3 illustrate how two indexed access methods can be categorized according to their organization structure and retrieval interrelationships. Within the framework set forth in Table 1, the differences in physical implementation become minor and exist as a subset of the total data structure.

Figure 2 Indexed sequential access method



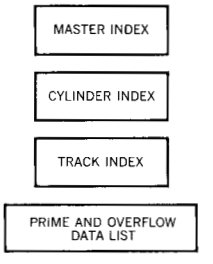
01	MASTER INDEX	S/S
02	CYLINDER INDEX	S/S
03	TRACK INDEX	D/S
04	PRIME DATA	D/S
04	OVERFLOW DATA	D/D

application areas

The S/S organization structure is analogous to traditional sequential organization, as mentioned earlier in this paper. The D/D form has been used for years as a method for handling synonyms in the random-data organization, which is an extension of the simple, direct list structure used to compensate for imperfect randomizing algorithms. The D/D structure is also used in the Bill of Material Processor for the maintenance and retrieval of imbedded lists (e.g., the where-used string).

The D/S form has been used in virtual-storage systems, data management systems, and in indexed access methods to support logical-record segmentation or repeating-field requirements. Thus, D/S provides a capability similar to the variable-length-record function and provides a useful data structure for supporting a hierarchical data base. The D/S form also appears as the imbedded track index of both ISAM and CSAM and as the prime data of ISAM. Virtual-storage systems[7] have used the same form as a continuous address-space image of storage sements, where the data are normally performatted into fixed-length blocks on the direct-access device.

Figure 3 Control sequential access method



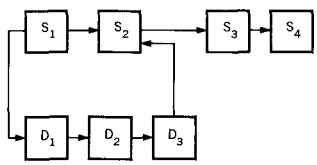
01	MASTER INDEX	S/S
02	CYLINDER INDEX	S/S
03	TRACK INDEX	D/S
04	DATA	D/SD

The S/SD and D/SD organization structures are extensions of S/S and D/S that reflect the ability to maintain logical strings using direct pointers for record insertion. D/SD represents the prime data structure of CSAM and is utilized for supporting chain files by the Bill of Material Processor.

Note that when physical data structures are examined within the framework illustrated by Table 1, the support of any device can be so categorized. The five generalized organization-structure relationships collectively represent a single level of list processing applicable to the physical storage and retrieval of data on direct-access storage devices, with the support of sequential devices as a logical subset. For example, if the S/S organization structure is supported for direct-access storage devices, the programming support for any sequential device can be incorporated by adding the device characteristics at the physical level. In addition, many of the functions could be incorporated into read-only storage or writable control store at the channel, control-unit, or device level.

Within this framework, it is possible for the user to specify the type of buffering he desires for string retrieval. The S/SD data-retrieval format might express buffer control in the form of S(n)D(n). Here, n is the number of buffers of appropriate length to be managed for each group. The S/SD format may also take the form (n)SD, where buffer utilization follows the logical string, as illustrated in Figure 4 and in the following:

Figure 4 Buffering specifications for retrieval formats



- For S(1) D(2) schedule S_1, D_1, D_2.
- For S(2) D(2) schedule S_1, S_2, D_1, D_2.
- For (5)SD schedule S_1, S_2, D_1, D_2, D_3.

Equivalent buffering specifications are also appropriate for the S/S, D/S, D/D, and D/SD formats. An additional topic within the scope of physical data structures is that of multiple indexing on attributes. Three basic approaches are possible, since common segments or logical attributes can be related to each other externally, internally, or both externally and internally combined.

A purely external attribute index is one wherein a secondary data structure is created as a cross reference to an internal attribute of an existing data string. Though the physical format of the index may vary, there is basically one data entry for each attribute occurrence. This structure is commonly termed an inverted list or inverted index.

Conversely, the actual data elements may be tied together, forming an imbedded list structure within another list-data string. This is the approach in the where-used chain of the Bill of Material Processor. Entry to the chain or list header is via a specific record in the parent list, i.e., the master file.

The third or combined approach occurs when the exact list header is unknown. In this case, an external index of the list headers is searched in order to locate the starting point of a particular imbedded list or chain.

Multiple attribute indexing is simplified at both the logical and physical levels because support facilities are inherent in the approach that distinguishes between the logical and physical retrieval of data.

Logical organization and control

The logical organization and control directs the creation of strings by supplying the physical data-management routines with the necessary information, which is assembled into list-data-structure control blocks. Information contained in the control block for any list structure assumes the characteristics of a list structure itself as shown by Table 2. The list header (01) contains status or control information about the list plus pointers to sublists (02) that represent the variable characteristics of the list. The sublists (03) may be repeating segments of the form D/SD, when each sublist represents such information as variable symbolic-key control data, list interrelationships, duplicating lists, aliases, generation changes, privacy, security, and data element descriptions.

An inherent characteristic of the proposed physical data management system is that it must support itself. Other characteristics relating to the management of the physical data structure can be generalized for any list structure:

- *Organization* implies the broad organization structure forms S/S, S/SD, D/S, D/D, and D/SD.
- *List-pointer format* provides pointers that may be classified as relative device-type addresses[5] (as used in OS/360), as symbolic keys which may be internal or external to the physical segment, as the actual device address, or as a transformation thereof. The pointers of a list should not be restricted to actual device addresses alone.
- *Maintenance control* involves the addition and deletion characteristics of a list, which can be categorized as either symbolic

Table 2 List-structure control block

```
01  LIST CONTROL BLOCK (FIXED)
    02    INTERNAL LIST CONTROL
          03  SYMBOLIC OR POSITIONAL CONTROL
          03  ALIASES
          03  DUPLICATING FUNCTIONS
    02    EXTERNAL LIST CONTROL
          03  PARENT LISTS
          03  SUBORDINATE LISTS
    02    LOGICAL LIST CONTROL
          03  GENERATION CHANGES
          03  PRIVACY & SECURITY
    02    DATA ELEMENT DESCRIPTORS
```

or positional. The symbolic list implies a collating sequence on specified control elements. A positional list assumes relative placement of such elements.

- *List interrelationships* involve an obvious aspect of the control of list structures that appears when logical records or segments are referenced by an actual or relative address. If such a list is nonsymbolically referenced only by itself, then its reorganization or movement is quite easily resolved at that time. However, if a direct or relative pointer is used by another list, the chains may be effectively severed until the external references have again been resolved. In this case, the information describing external linkages is a sublist from the specific list header. In essence, that sublist is an inverted list or a "by whom referenced" list.

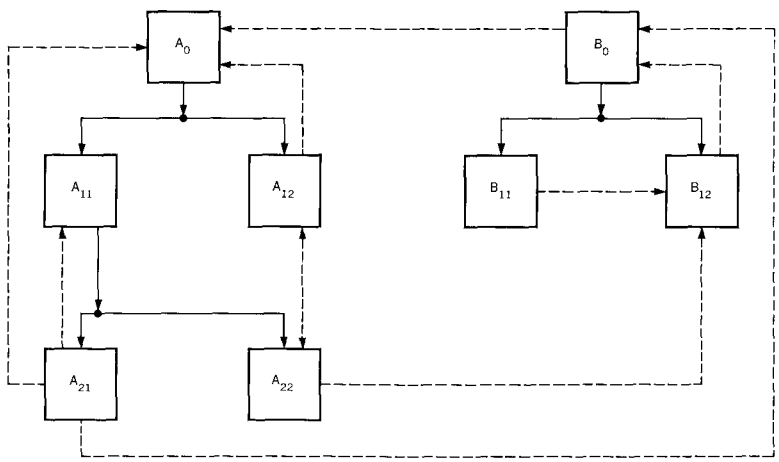
list interactions

The discussion of physical data-management control is now expanded to include a directory of list interactions and a hierarchical structure. A control directory in matrix form can be used as a compressed representation of specific sublists, as for example a multiple-level index to a prime data list, an index to determine alternate paths to specific lists in case the primary path fails, or a reflection of logical hierarchical modifications. A simple example is an access method involving a multilevel index, such as ISAM. The primary route to a logical record starts at the highest-level index and proceeds through each individual index level. If one index is unreachable for some reason, the prime data is still valid and can be reached by bracketing the index search at the next lower level. This procedure is slow, but preferable to obtaining no data.

undirected graph

A technique that appears to be quite usable as an index or directory to list structures is an undirected graph. Such a graph for use in list interactions is shown in Figure 5. The solid lines

Figure 5 Undirected graph for list interactions



represent primary hierarchical relationships and are the primary chain of control. The dotted lines indicate additional secondary linkages which are either unidirectional or bidirectional. For example, A_{22} is initially created from A_{11}; the prime parent (A_0) and attachments to other lists are secondary.

matrix representation

The undirected graph of Figure 5 may be written in matrix form as shown in Table 3; the matrix indicates the connections among the various lists. Table 3 is a one-step matrix showing all single-step relationships within the structure shown in Figure 5. The same type of undirected graph can be used as a logical-element association matrix when dealing with a hierarchical language structure. Thus, we can see at least one common approach to representing the logical data structure and the physical data structure. As previously mentioned, the two structures may be identical, although such a restriction is neither necessary nor desirable. The matrix in Table 3 also shows all the list interrelationships and thus reflects one of the variable sublists of the list-structure control block.

Continuing the discussion of the data structure in Figure 5, consider the question of alternate paths to the same list. If the one-step matrix in Table 3 is multiplied by itself, it becomes the two-step matrix shown in Table 4. This matrix shows the num-

Table 3 One-step matrix

	A_0	A_{11}	A_{12}	A_{21}	A_{22}	B_0	B_{11}	B_{12}
A_0	0	1	1	0	0	0	0	0
A_{11}	0	0	0	1	1	0	0	0
A_{12}	1	0	0	0	1	0	0	0
A_{21}	1	1	0	0	0	1	0	0
A_{22}	0	0	1	0	0	0	0	1
B_0	1	0	0	0	0	0	1	1
B_{11}	0	0	0	0	0	0	0	1
B_{12}	0	0	0	0	0	1	0	0

Table 4 Two-step matrix

	A_0	A_{11}	A_{12}	A_{21}	A_{22}	B_0	B_{11}	B_{12}
A_0	1	0	0	1	2	0	0	0
A_{11}	1	1	1	0	0	1	0	1
A_{12}	0	1	2	0	0	0	0	1
A_{21}	1	1	1	1	1	0	1	1
A_{22}	1	0	0	0	1	1	0	0
B_0	0	1	1	0	0	1	0	1
B_{11}	0	0	0	0	0	1	0	0
B_{12}	1	0	0	0	0	0	1	1

ber of different paths from one list to another list that involve exactly two steps. By the same process, if the one-step matrix is cubed, we have a three-step matrix and so on.

Comparing the two matrices and the undirected graph, we see that list A_{12} can be reached in one step from A_0 or A_{22}. The two-step matrix shows that A_{12} can be reached in two steps from itself via two separate paths, and from A_{11}, A_{21}, and B_0 by one two-step path each. Certainly, not every two-step path is usable, but the logical level of control should be able to analyze the physical data structure and determine if the logic of the original request can be redirected via an available alternate path. If the user were at a terminal, the inquiry system might respond by asking for a synonym related to one or more specific contexts, since it knows the alternate paths available to it.

data protection

An additional topic of interest involves data-set security and the protection of logical segments from dual updates in a multitask environment. The current approach to data-set security is based on the assumption that the logical and physical data sets are identical. When the logical and physical data relationships are separated, several additional levels of control are possible at the system level.

Referring to Figure 5, system-level control can be specified at the A_0 level either exclusively or inclusively. Exclusively, A_0 would control A_0 alone, and inclusively A_0 would control A_0, A_{11}, A_{12}, A_{21}, A_{22} and all their interlinkages. On the other hand, only the A_{21}-to-A_{11} list linkage might be specified. In that case, control is based on a physical relationship.

The same possibilities are available at the logical level. The example using Figure 5 may be extended to show control of the logical or physical list structures, in which case the control might be used on the data string from a specific header.

Protection may also be exercised at the physical-segment level, so as to prevent dual updating. Basically the procedure is this: if a single physical segment consists of one or more logical elements, the logical elements can be controlled using their list name, while another task has access to the same physical segment for the use of a different logical element. Thus, it is possible to protect discrete elements of a hierarchical data base at the system or IOS level, with only general logical identification supplied by the user.

Concluding remarks

Data management has been presented as a hierarchical structure incorporating the concepts of list processing. The ability to control the structure includes the facility to manage data at each of several levels within a compatible framework. Such a unified approach allows the proper definition of functions that might be incorporated in the actual circuit logic. The distinction between logical and physical data structures provides the user with a flex-

ible level of data independence that is compatible with each level. This integrated approach to the management of data structures may permit the investigation of other techniques that are applicable to data retrieval. Experimental approaches to problem-solving, utilizing heuristic programs and minimaxing,[8,9] have been tried and could be very useful in the area of information retrieval. The notation of sets might be a powerful technique in the specification of string or segment requirements.

CITED REFERENCES

1. J. McCarthy, LISP 1. *Programmer's Manual*, Computation Center and Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts (1960).
2. C. W. Bachman and S. B. Williams, "A general purpose programming system for random access memories," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **26**, 411–422, Sparten Books, Washington, D. C. (1964).
3. SYSTEM/360 *Generalized Information System, Application Description Manual*, H20–0521. International Business Machines Corporation, Data Processing Division, White Plains, New York.
4. Information Management, SYSTEM/360 *Application Description Manual*, H20–0524, International Business Machines Corporation, Data Processing Division, White Plains, New York.
5. W. A. Clark, "The functional structure of OS/360, Part III, Data management," *IBM Systems Journal* **5**, No. 1, 30–51 (1966).
6. SYSTEM/360 *Bill of Materials Processer, Application Description Manual*, H20–0197, International Business Machines Corporation, Data Processing Division, White Plains, New York.
7. IBM SYSTEM/360 *Time Sharing System, Concepts and Facilities*, C28–2003, International Business Machines Corporation, Data Processing Division, White Plains, New York.
8. A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development* **3**, No. 3, 211–229 (July 1959).
9. A. Newell and J. C. Shaw, "Programming the logic theory machine," *Proceedings of the Western Joint Computer Conference* **15**, 218–239 (1957).
10. D. G. Bobrow and B. Raphael, "A comparison of list-processing languages," *Communications of the ACM* **7**, No. 4, 231–240 (1964).
11. A. J. Perlis and C. Thornton, "Symbol manipulation by threaded lists," *Communications of the ACM* **3**, No. 4, 195–204 (1960).
12. C. T. Meadow, *The Analysis of Information Systems*, John Wiley and Sons, New York, New York (1967).