A GRAPH THEORETIC MODEL OF DATA STRUCTURES

J. Heller and Ben Schneiderman
Department of Computer Science
State University of New York at Stony Brook

The widespread development of large shared data bases has proceeded mainly on empirical notions. Only in the past few years have there been attempts at formalizing the underlying concepts into a theory of data structures. The present models have been founded on the well-established mathematical theories of sets, relations and graphs, but few descriptions have been sufficiently developed to meet the needs of a complete theory of data structures. The set theoretic formalism offered by Childs[1] or Schwartz[2] or the relational model by Codd[3] are useful but fail to give a complete description of the complex structures found in modern data bases. The graph theoretic concepts found in papers by Harary and Hsiao[4] and Earley[5] are more appealing but need further elaboration. This paper is an attempt to develop a graph theoretic model into a useful formalism for describing and manipulating data structures.

Data existed long before computers were ever dreamed of and even the first librarian had some notion of the organization and structure of his data. Clearly, information has a logical structure independent of the physical implementation inside a computer. A theory of data structures should describe the logical structure separately from the implementation but should enable the system designer to evaluate the effectiveness a particular implementation. Furthermore, the theoretical formlation should be intuitively appealing and acceptable to data managers, such as librarians and executives, who may not have a sophisticated understanding of computer technology and techniques. The computer scientist must provide the data manager with a "magic genie" for manipulating data according to this logical view of the data.

To motivate the formal definitions of a graph theoretic model, we consider three simple examples: a one-way list with top and bottom pointers, a ring with one entry node, and a full balanced binary tree with one entry node.

Since one of the strong arguments in favor or the graph theoretic model are the visual cues taken from the picture of the graph, we will make frequent use of such pictures. Thus a one-way list with top and bottom pointers looks like:

Figure 1

The square nodes $E=\{e_1,e_2\}$, called entry nodes, provide access to the data nodes $D=\{d_1 \ldots d_n\}$. Informally, the entry nodes are always instantly accessible and all searches must begin at an entry node. The graph describes the logical structure of the data and the connections among the data nodes. The nodes can contain any amount of information and may have a complex substructure which might be another graph. One of the virtues of the graph describe data at the detailed level and the complex structure of entire files.

The logical structure given by the graph may be implemented in a multiplicity of ways. The data nodes might be maintained in the processor storage with implicit or explicit pointers. Alternatively, each node could be a region on a disk file with region addresses as the pointers.

The relationship among the nodes is described by a mapping

$$\Gamma:(D \cup E) \rightarrow D$$

where, in our example,

$$\Gamma e_1=\{d_1\} \quad \Gamma e_2=\{d_n\} \quad \Gamma d_i = \begin{cases} \{d_{i+1}\} & i<n \\ \phi & i=n \end{cases}$$

We can compound the mapping

$$\Gamma^2 d_i = \Gamma(\Gamma d_n) = \begin{cases} \{X_{i+2}\} & i<n-1 \\ \phi & i>n-1 \end{cases}$$

In general

$$\Gamma^n d_i = \Gamma^{n-1}(\Gamma di) = \Gamma(\Gamma^{n-1} di)$$

We further define the inverse map

$$\Gamma^{-1} d_i = \begin{cases} \{d_{i-1}\} & 1<i\leq n \\ \phi & i=1 \end{cases}$$

and an identity or zero map

$$\Gamma^0 d_i = \{d_i\}$$

In general the mapping is not single valued, but yields a set of nodes, that is

$$\Gamma d = A = \{d_{i_1}, \ldots d_{i_k}\}$$

where $\underline{k}$ is the out-degree of the node $\underline{d}$.

Thus

$$\Gamma^2 d = \Gamma A = \Gamma d_{i_1} \cup \Gamma d_{i_2} \cup \ldots \cup \Gamma d_{i_k}$$

or more succinctly,

$$\Gamma^2 d = \Gamma A = \bigcup_{\forall d \in A} \Gamma d$$

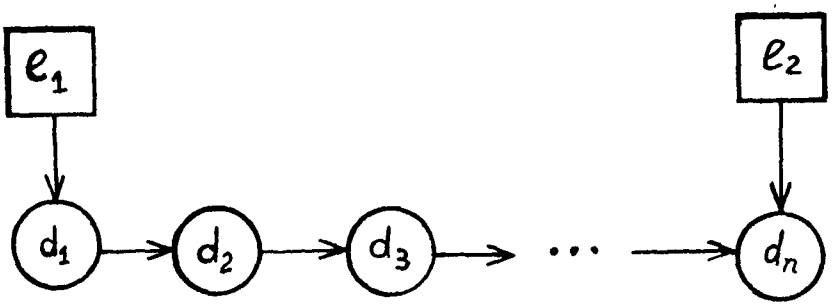

FIGURE 1

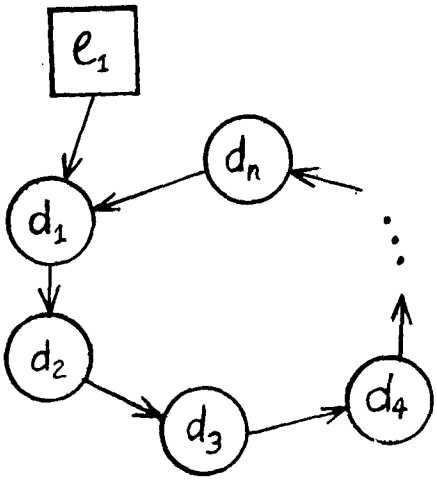

FIGURE 2

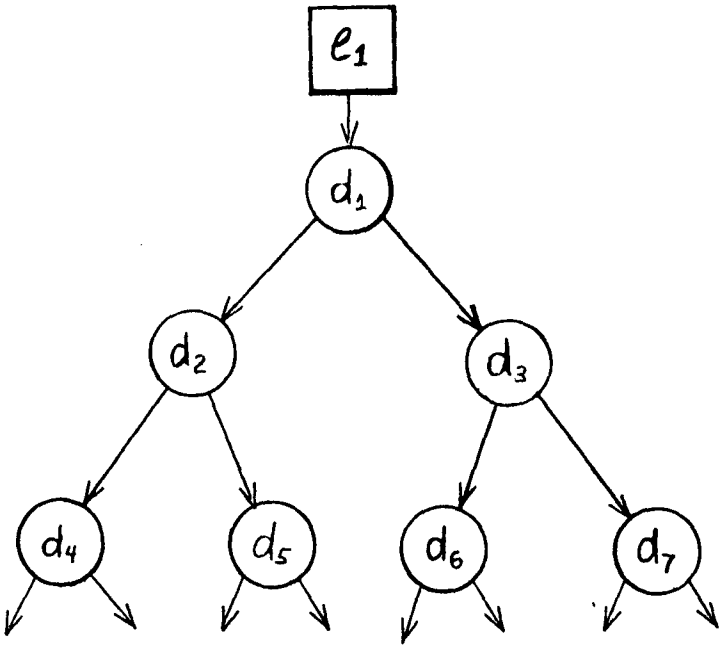

FIGURE 3

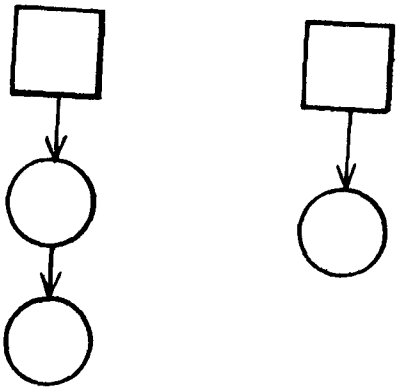

FIGURE 4

As our second example we consider a ring with one entry node

Figure 2

The graph consists of the entry node set $E=\{e_1\}$ and the data node set $D=\{d_1 \ldots d_n\}$ and the mapping

$$\Gamma e_1 = \{d_1\} \quad \Gamma d_i = \begin{cases} \{d_{i+1}\} & 1 \leq i < n \\ \{d_1\} & i = n \end{cases} .$$

Notice that in both our examples there are no nodes "floating" freely that all nodes are connected and any data node is reachable by following a path from one of the entry nodes.

As our third example we consider a fully binary tree with a single entry node $E=\{e_1\}$ and $D=\{d_1 \ldots d_n\}$ where n is 2^k-1 for some integer k, which represents the number of levels in the tree.

Figure 3

The mapping Γ for this structure is defined by

$$\Gamma e_1 = \{d_1\} \quad \Gamma d_i = \begin{cases} \{d_{2i}, d_{2i+1}\} & 1 < 2^{k-1}-1 \\ \phi & i \geq 2^{k-1} \end{cases} .$$

Clearly, all data nodes are reachable from the single entry node with a maximum number of steps equal to k, that is

$$\bigcup_{i=1}^{k} \Gamma^i e_1 = D .$$

Analysis of these three examples and the study of a wide variety of commonly used structures motivates the definition of a Well-Formed List Structure (WDLS). Definition: L=(D,E,Γ) is a Well-Formed List Structure if and only if (1) D$\cup$E is a connected set of nodes, that is, if we remove the direction on the direction on the edges, there is a path between every pair of nodes.

Formally if $A=D \cup E = \{a_i \mid i=1 \ldots n\}$

$$\forall (i,j)\, \exists\, a_i a_{i_1} a_{i_2} a_{i_3} \ldots a_{i_\ell} a_j \exists a_{i_{\ell+1}} \in \Gamma^k a_{i_\ell}, \quad k=\pm 1$$

(2) E is a non-empty set of nodes with each node having in-degree zero and D is a non-empty set of nodes with node having an in-degree greater than zero.

Formally $E \neq \phi \quad \forall e \in E \ \Gamma^{-1} e = \phi$
$D \neq \phi \quad \forall d \in D \ \Gamma^{-1} d \neq \phi$

(3) Any node in the set D is reachable from at least one of the nodes in the set E.

Formally, $\bigcup_{i=1}^{n^*} \Gamma^i E = D = \Gamma E \cup \Gamma^2 E \cup \ldots \cup \Gamma^{n^*} E$

where n^* is the smallest integer for which this true. Condition (1) is meant to exclude structures such as

Figure 4

which are two WFLSs which are not connected. Condition (2) is meant to exclude structures such as

Figure 5

which is a ring without an entry node, and

Figure 6

which is a degenerate one way list with top and bottom pointers, that is, a structure without data Notice that this structure also violates condition (1) concerning connectedness. Further violations of condition (2) are

Figure 7

since entry nodes may not point to each other and

Figure 8

since data nodes may not point to entry nodes. Finally condition (3) is meant to exclude structures such as

Figure 9

which contains a non-reachable ring.

The static description of a data structure is not sufficient; our model must include a notation for dynamically manipulating these structures. Consider the static description of a stack S=(D,E,Γ),

$$D=\{d_1 \ldots d_n\}, \ E=\{e_1\},$$
$$\Gamma e_1 = \{d_n\}, \Gamma d_i = \begin{cases} \{d_{i-1}\} & 1 < i \leq n \\ \phi & i = 1 \end{cases}$$

Figure 10

This could just as easily describe a one way list with a top pointer only. What distinguishes the above structure as a stack are the dynamic operations which are defined for a stack: PUSH or POP. For a stack we define POP (S) as a transformation which produces a new structures S'=POP(S) where S'=(D',E',Γ'), $D'=\{d_1 \ldots d_{n-1}\}$, $E'=\{e_1\}$,

$$\Gamma' e_1 = \{d_{n-1}\}, \Gamma' d_i = \begin{cases} \{d_{i-1}\} & 1 < i \leq n-1 \\ \phi & i = 1 \end{cases}$$

Figure 11

Similarly, we define S''=PUSH (S) where S''=(D",E",Γ"),

$$D''=\{d_1 \ldots d_{n+1}\}, \quad E''=\{e_1\},$$
$$\Gamma'' e_1 = \{d_{n+1}\}, \Gamma'' d_i = \begin{cases} \{d_{i-1}\} & 1 < i \leq n+1 \\ \phi & i = 1 \end{cases}$$

Figure 12

A two way list with top and bottom pointers

Figure 13

is identical with a queue or a di-que (deque) when examined for static structure. Only by observing the dynamic operations which are permitted can we distinguish between the structures.

Analogous operations of addition and deletion of nodes can be defined for other structures and can easily be described and implemented in terms of programming language subroutines. Such a discussion, while useful, would not contribute to the development of a more profound conceptualization or formalization of data structures. We seek a more general, abstract theory which would subsume each of these

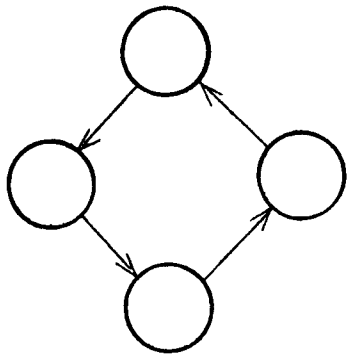

FIGURE 5

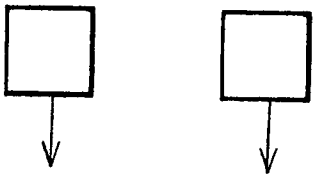

FIGURE 6

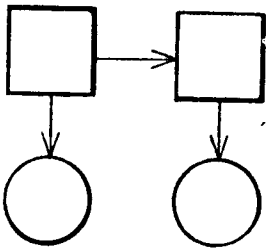

FIGURE 7

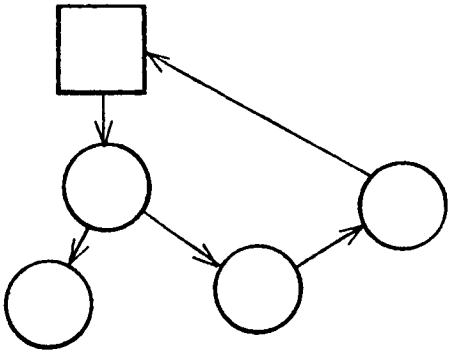

FIGURE 8

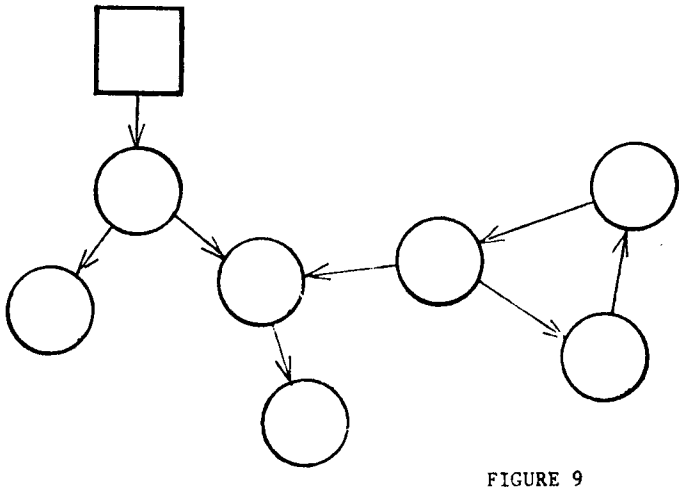

FIGURE 9

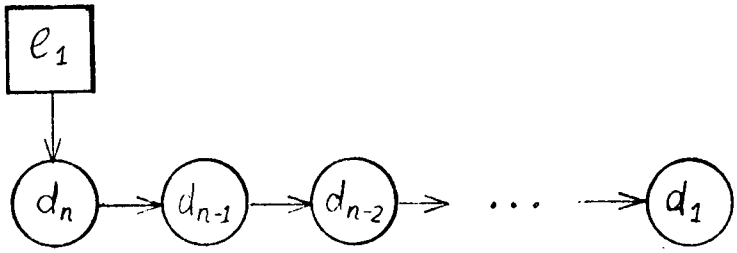

FIGURE 10

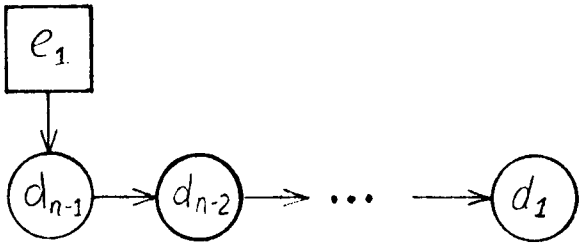

FIGURE 11

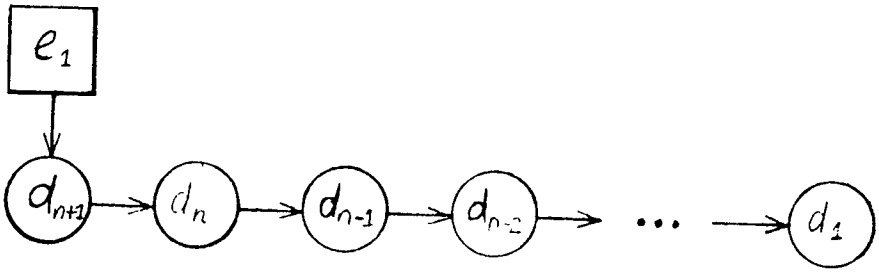

FIGURE 12

discussions as a special case. Thus, we are led to consider the primitives of graph theoretic operations.

Certainly we must be able to

1) Add a node and a branch
   Example: Add a new leaf to a tree
2) Insert a node and a branch and change at least one branch
   Example: Insert a node in a one-way list
3) Add a branch to connect two already existing nodes
   Example: Add a branch to convert a one-way list into a ring

and we must be able to perform the delete operations which are the inverse operations:

1) Delete a node and a branch pointing to that node
   Example: Delete a node of a tree and the branch pointing to that node
2) Delete an internal node with its outgoing and incoming branches
   Example: Delete a node in a one-way list
3) Delete a branch which leaves the structure well formed
   Example: Delete a bottom pointer

The key point in these simple operations is that each transformation must be defined so as to maintain a WFLS.

Entire structures can be added or deleted. We might be interested in adding a sub-tree at a leaf tree or in adding a one way list to a one way list. Similarly, we might delete large sections of a structure by creating subgraphs or partial graphs. A subgraph is created by deleting an edge and then deleting all unconnected nodes and branches. A partial graph is created by deleting a node and then eliminating unconnected nodes and branches.

In addition to transformations of a WFLS we need to be able to describe the rules for combining pairs of WFLSs. Two WFLSs can be combined to form a single WFLS in only two ways

Figure 14

In these two pictures we assume that the nodes D_1 and D_2 might represent complicated structures and that the symmetric combinations of W_2 pointing to W_1 are possible. Formally, if

$$W_1 = (D_1, E_1, \Gamma_1) \text{ and } W_2 = (D_2, E_2, \Gamma_2)$$

then

$$W_3 = (D_1 \cup D_2, E_1 \cup E_2, \Gamma_1 \cup \Gamma_2 \cup \Gamma_{12})$$

where

$$\Gamma_{12}: \quad (D_1 \cup E_1) \rightarrow D_2$$

Two structures which are frequently used for information retrieval systems, indexed sequential and a one-way list of one-way lists, can be easily represented in graph theoretic terms. An indexed sequential file has two entry nodes, one to the index and one to the file, and two types of data nodes; index nodes and record nodes. We have assumed that each index branch references the beginning of strings of equal length (in this case 3). The graph and the graph theoretic notation to describe it are:

Figure 15

As a more complex example, let us consider the GRIPHOS system; an information retrieval system which deals with sets of tagged strings. The data nodes are ordered pairs referred to as fields. Consider a tag

$$T = \{t_\sigma | \sigma = 1 \ldots \#T\}$$

and a string taken from the set:

$$S = \{s_\sigma | \sigma = 1 \ldots \#S\} .$$

We define a field as an ordered pair <t,s> of the set T X S. In a given data base the set of all fields will be:

$$F = \{f_\sigma | \sigma = 1 \ldots \#F\} \subseteq T \times S$$

A record in the GRIPHOS system is a set of fields:

$$r = \{f_\sigma | \sigma = 1 \ldots \#r\}$$

and the data base is viewed as a set of records.

The records are organized by the input program so that each record can be searched for the fields it contains or for the next record entered into the data base. When records are entered into that data base pointers are entered so that fields can be added to any record at a future time with a minimal amount of searching. If we designate the ith input record as r_i containing fields $f_{i\sigma}$ and a dummy record r_0 containing a top and bottom pointer, the organized records have the mapping Γ:

$$\Gamma r_0 = \{r_1, r_n\}$$

$$\Gamma r_i = \{r_{i+1}, f_{i1}, f_{i\#r_i}\} \qquad i<n$$

$$\Gamma r_n = \{f_{n1}, f_{n\#r_n}\}$$

$$\Gamma f_{i\sigma} = \{f_{i\sigma+1}\} \quad \sigma < \#r_i$$

$$\Gamma f_{i\#r_i} = \phi$$

Figure 16

It is possible to search out each field and tell which record contains a given field by sequentially searching for the next record and then sequentially searching for the next field. Since this mode of retrieval is very time consuming, the GRIPHOS system supports search strategies based on inverted files. The inverted files are based on a hashing function for all fields with a given tag.

We form a partition of the set of fields F by tag $t_i \varepsilon T$:

$$F_i = \{f_{i\sigma} | t_i \varepsilon f_{i\sigma}, \sigma = 1 \ldots \#F\} \quad i = 1 \ldots \#T$$

The hashing function h(f) further subdivides the set of fields F by partitioning the set F_1:

$$F_{ij} = f_{ij\sigma} | t_i \varepsilon f_{ij}\sigma, h(f_{ij\sigma}) = j, \sigma = 1 \ldots \#F_{ij\sigma} j = 1 \ldots \#h(f_{ij\sigma})$$

Associated with every field $f_{ij\sigma} \varepsilon F_{ij}$, there is a set of integers $\Pi_{ij} \subseteq \{1 \ldots \#r\}$ of which each ele-

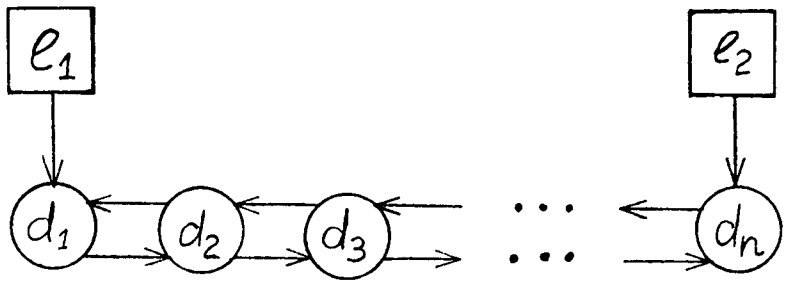

FIGURE 13

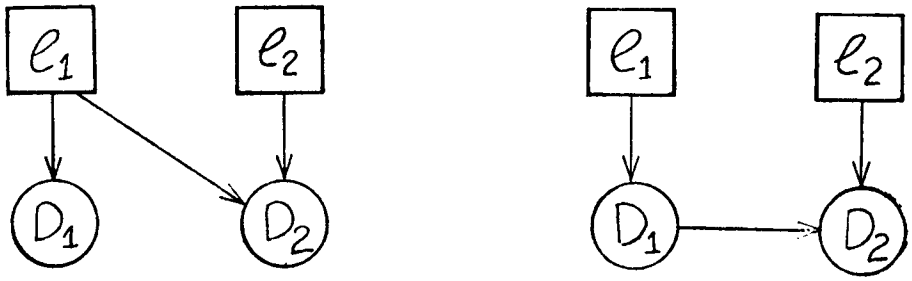

FIGURE 14

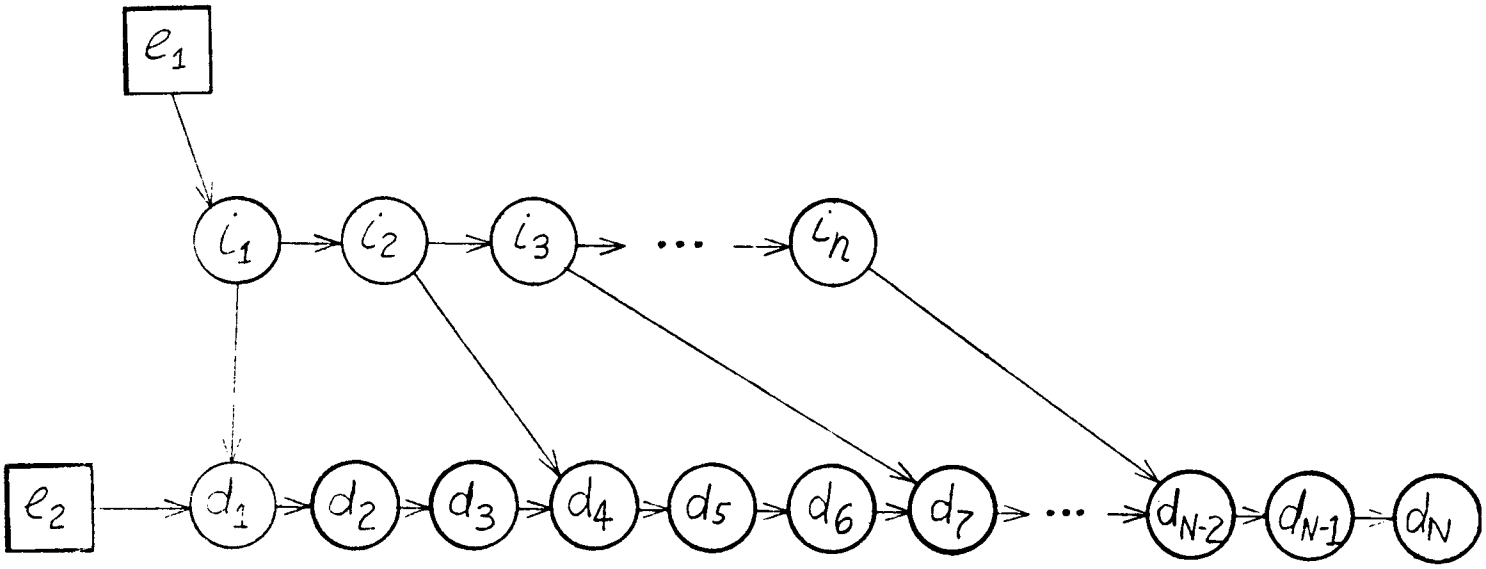

FIGURE 15a

$$\mathcal{G} = (D, E, \Gamma)$$

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

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

$$\Gamma e_1 = i_1$$

$$\Gamma e_2 = d_1$$

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

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

FIGURE 15b

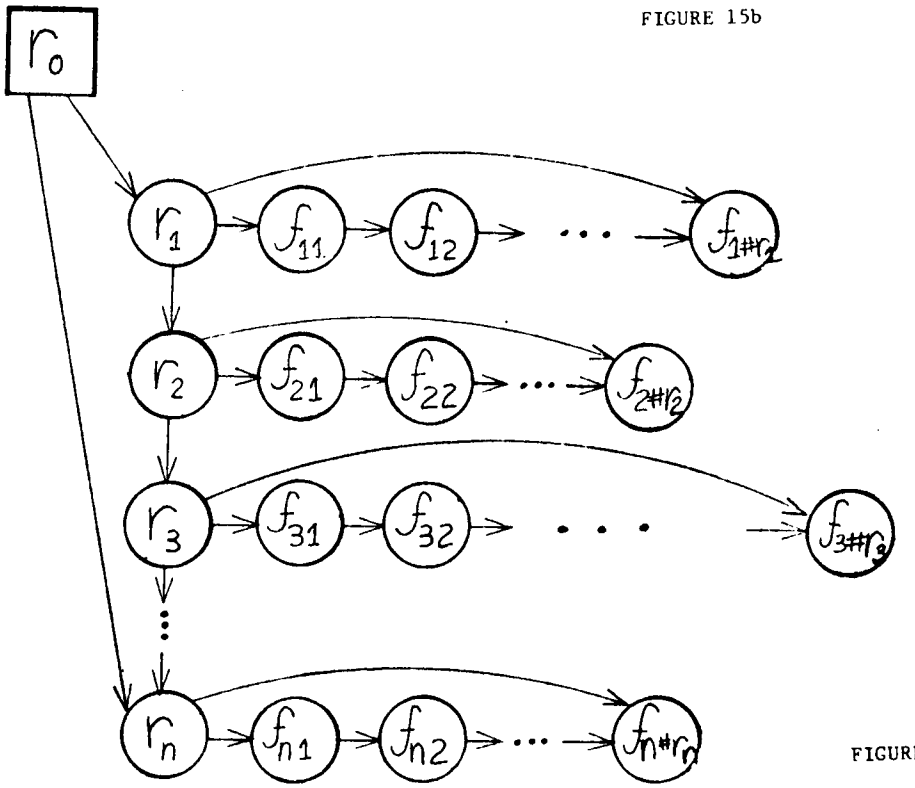

FIGURE 16

ment corresponds to a record containing a field with a tag t_i whose hash value is j:

$$\Pi_{ij} = \{\ell_\sigma | t_i \epsilon f_{ij\sigma}, h(f_{ij\sigma})=j,\ f \epsilon r_{\ell_e},$$
$$\sigma=1...\#F_{ij},\ \rho=1...\#\Pi_{ij}\}$$

The above partitioning of the fields is used by the GRIPHOS system to form a "tree searchable" index to the records $\{r_i\}$ of the data base. The entry nodes to the index portion of the data base is that set of tags $\{t_\sigma\} \subseteq T$ for which a user constructs an index. The mapping Γ which specifies the file organization is given by:

$$\Gamma t_i = \{f_{ij1} | t_i \epsilon f_{ij1},\ h(f_{ij1})=j\}$$
$$\Gamma f_{ij1} = \{f_{ij2}, f_{ij\#Fij}, \ell_1\}$$
$$\Gamma f_{ij\sigma} = \{f_{ij\sigma+1}, \ell_1 \qquad 1<\sigma<\#F_{ij}$$
$$\Gamma f_{ij\#Fij} = \{\ell_1\}$$
$$\Gamma \ell_1 = \{\ell_{\sigma+1}, r_{\ell_\sigma}\} \quad 1<\ell_\sigma<\#\Pi_{ij}$$
$$\Gamma \ell_{\#\Pi_{ij}} = \{r_{\ell\#\Pi_{ij}}\}$$

Figure 17

The GRIPHOS system has a data base organization which supports selective searching for records containing a given field or subset of fields and then a complete search of the record.

This paper is an introduction to the graph theoretic model of data structures. The following ideas will be more formally developed in upcoming reports:

1) By affixing a numerical value to each of the edges we can study the properties of particular implementations. The values represent the time necessary to traverse that branch. Thus the efficiency of various organizations and indexes can be compared.

2) The topology of a graph can be studies independent of any interpretation in the same way that flowchart schemata are studied. We are interested in considering reachability, connectivity and cyclicity. The length, diameter and other metrics are used to describe various graphs. Then an interpretation of the graph can be made by assuming that each node has information which influences our search techniques. For example, we can distinguish between the properties of a search on an ordered list and an unordered list.

3) The possibility of using "colored" branches further enhances the richness of structure. A single may contain several structures each of which can be visualized as having different colored branches. This further delineates the nature of structure and requires a redefinition of some of the properties of such structures.

4) By assigning values to each node, reflecting the probabilities of each node being accessed, we can more closely define optimum organizations.

5) Most interesting of all, are the properties of dynamic graphs, that is, graphs which change with time. We are interested in studying operations of addition, deletion and combination of WFLSs and how these operations of addition, deletion and combination of WFLSs and how these operations affect properties of the graph. There is very little work in graph theory on graph transformations, most of the effort has been devoted to describing the properties of an already existing graph. We would like to describe techniques for creating a graph so that it has certain properties, such as a maximum search length or freedom from cycles.

BIBLIOGRAPHY

1. Childs, D.C. Feasibility of a set-theoretic data structure. Proc. IFIP Congress 1968, Vol. 1, North Holland Pub. Co., Amsterdam, pp. 420-430.
2. Schwartz, J. Set theory as a language for program specification and programming. New York University, 1970.
3. Codd, E.F. A relational model of data for large shared data banks, Comm, ACM 13, 6 (June 1970), 377-387.
4. Hsiao, D. and Harary, F. A Formal System for Information Retrieval from Files. Comm. ACM 13, 2 (February 1970), pp. 67-73.
5. Earley, J. Toward an Understanding of Data Structures. Comm. ACM 14, 10 (October 1971), pp. 617-627.

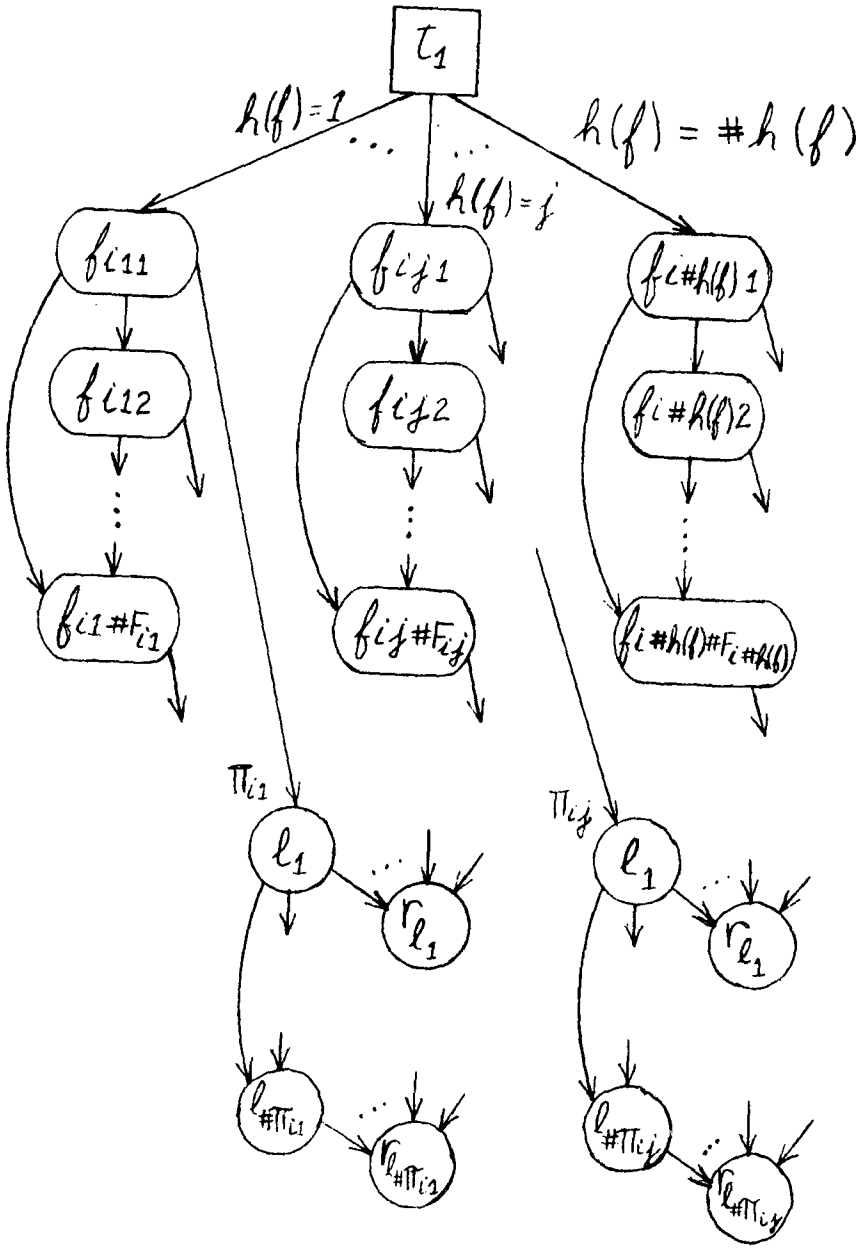
τ_1
$h(f)=1$
$h(f)=j$
$h(f)=\#h(f)$
f_{i11}
f_{i12}
$f_{i1\#F_{i1}}$
f_{ij1}
f_{ij2}
$f_{ij\#F_{ij}}$
$f_{i\#h(f)1}$
$f_{i\#h(f)2}$
$f_{i\#h(f)\#F_{i\#h(f)}}$
Π_{i1}
Π_{ij}
ℓ_1
r_{ℓ_1}
$\ell_{\#\Pi_{i1}}$
$r_{\ell_{\#\Pi_{i1}}}$
$\ell_{\#\Pi_{ij}}$
$r_{\ell_{\#\Pi_{ij}}}$

FIGURE 17