To the Graduate Council:

I am submitting herewith a thesis written by William B. Malthouse entitled "DSAM: An Enhanced File Access Method for Information Retrieval Applications." I recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Sara R. Jordan

Sara R. Jordan, Major Professor

We have read this thesis and recommend its acceptance:

Charles E. Hughes

Charles P. Pfleeger

Accepted for the Council:

Hilton A. Smith

Vice Chancellor
Graduate Studies and Research

Thesis
76
.M249
cop. 2

DSAM: AN ENHANCED FILE ACCESS METHOD FOR INFORMATION RETRIEVAL APPLICATIONS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

William B. Malthouse

December 1976

1305603

ACKNOWLEDGMENTS

The continued support, encouragement and advice of Dr. S. R. Jordan, Dr. C. E. Hughes, and Dr. C. P. Pfleeger, who formed the thesis committee for this research project, is gratefully acknowledged. Their advice in the preparation of this report was invaluable.

The work reported here was conducted at Oak Ridge National Laboratory, which is operated by the Union Carbide Corporation, on behalf of the United States Energy Research and Development Administration, as part of the ongoing RECON project. The author is thankful for the funding provided by ERDA, and for the time provided by the Computer Sciences Division, Computing Applications Department, for the actual implementation of DSAM. The contributions of the RECON project supervisor, S. L. Yount, and the other members of the RECON team are reflected in how well DSAM interfaces to the RECON system as a whole.

Special thanks must be given to the author's wife, Nancy Malthouse, for her suggestions, leg work in gathering materials and references, proofreading, and ever-present moral support during all phases of this work.

ABSTRACT

A software system and file structure termed DSAM, Directory Sequential Access Method, was developed for use with the index files of a large-scale bibliographic information retrieval system. DSAM was developed as a replacement for the vendor supplied ISAM access method which had proved to be too inefficient for the given application, and which lacked advanced capabilities desirable for the continued growth of the system.

DSAM was found to provide significant reductions in Direct Access Storage Device (DASD) space requirements for the index files, typically on the order of 30 to 50 percent. This reduction was primarily due to DSAM's support of variable length alphameric keys in the data records, which eliminated the storage of nonsignificant trailing blanks otherwise required with ISAM or competitive file organizations.

DSAM uses distributed free space within a file to handle insertion of new records, while always maintaining the logical records in collating sequence of their keys. Preservation of the collating sequence permits sequential access to the data, in both ascending and descending order, once logically positioned to a specified key via indexed access. Sequential access in descending order has only recently been added to competitive commercial software systems such as VSAM, the IBM Virtual Sequential Access Method, which could not be used for this application for a variety of reasons discussed in the report.

Both VSAM and DSAM may be characterized as access methods that use distributed free space within the file to accommodate subsequent insertion of new records, as opposed to an "overflow" strategy such as is used in the older ISAM technique. VSAM requires that this initial distribution consist of a uniform percentage of free space in each physical record and a (possibly different) uniform percentage of empty physical records within a defined group of physical records. In contrast, DSAM allows these percentages to be dynamically varied during file loading as determined by the alphabetic range of keys then being loaded.

This latter characteristic, nonuniform free space distribution, may provide a means of reducing overall file size requirements for the information retrieval system. Thus, a DSAM file used to hold an index of the surnames of authors of documents may be created such that there is more free space reserved for author names beginning with "S" than for those beginning with "X", for example. The distributions used can be taken from frequency analysis of existing author index files.

TABLE OF CONTENTS

CHAPTER		PAGE
INTRODUCTION		1
I.	THE APPLICATION ENVIRONMENT - INFORMATION RETRIEVAL	3
	Overview of Information Retrieval Systems	3
	Bibliographic retrieval systems	3
	Systems using inverted file structures	8
	The RECON System at Oak Ridge National Laboratory	13
	Command structure and services	16
	Existing file system structure	20
	Complications of file updating	23
	Revised file system design	30
II.	AREA OF RESEARCH - ADVANCED FILE SYSTEM DESIGN	32
	The Inverted Index Files	32
	Limitations of the existing access method	32
	Review of commercially available alternatives	34
	Design criteria for DSAM	35
	Research Objectives	36
	Reduction in on-line storage requirements	36
	Backward sequential access	37
	Fail-safe dynamic update	37
	Reduction in reorganization frequency	38
	Flexibility and extensibility	39
	Viability of nonuniform initial space distribution	40
III.	THE LOGICAL DESIGN OF DSAM	42
	DSAM Logical File Structures	42
	Basic terminology	42
	Logical structure of the directory	45
	DSAM Physical File Structures	53
	Definitions and terminology	53
	Directory area CINVs	56
	Data area CINVs	57
	Header/trailer CINVs	58
	Software System Characteristics	59
	System level functional specifications	59
	Level of implementation, a pseudo-access method	63
	Implementation choices, language, reentrancy	63
	DSAM Software Design	66
	DSAM software modularity	66
	Overview of software modules and functions	70

CHAPTER PAGE

Module interface specifications 70
Treatment of error and exceptional conditions 72
Functional specifications, by module 73

IV. PERFORMANCE TESTING AND EXPERIMENTS WITH DSAM 74

Critique of DSAM Performance 74
Storage space reduction 74
Estimated retrieval efficiency 84
File Updating 88
General considerations 88
Experimental design 89
Selection of test data 90
Experimental results 92
Evaluation of results 92
Observations concerning alphabetical distributions of index terms 96
A Note on the Mathematical Analysis of DSAM 102

V. SUMMARY AND CONCLUSIONS 104

Overview 104
DSAM System Design Review and Evaluation 105
Suggestions for Further Study 107

BIBLIOGRAPHY 109

VITA 115

LIST OF TABLES

TABLE		PAGE
I.	Estimated DSAM Space Reduction VS ISAM	76
II.	DSAM Space Reduction Examples	83
III.	ISAM VS DSAM: Performance Data - Retrieval Only	85
IV.	ISAM File Update: Timing Statistics	93
V.	DSAM File Update: Timing Statistics	94
VI.	Distribution of RECON Index Terms by First Character of Term	97
VII.	Distribution of RECON Author Index Terms by First Character of Surname	99

LIST OF FIGURES

FIGURE		PAGE
1.1	List of Available RECON Commands	15
1.2	Example Results of RECON Command: EXPAND IT=RADIATION EFFECTS	18
1.3	Example Results of RECON Command: EXPAND RELATED TERMS	19
1.4	Overview of RECON Command and File Structure Relationships	21
1.5	RECON File Update Procedure for a Single Collection	26
3.1	DSAM Directory Tree Structure	47
3.2	Example DSAM File Structure, Prior to Insertion of New Term: "DSAM"	49
3.3	Illustration of CINV Splitting	50
3.4	Final State of DSAM File After Insertion of New Term: "DSAM"	52
3.5	Functional Grouping of DSAM Modules at Level 1 and 2	67
3.6	Hierarchical Levels of DSAM Modules	68
4.1	Histogram of Index Term Length from EDB	79
4.2	Histogram of Index Term Length from NSA	80
4.3	Illustration of the Effects of Artificial Phrase Generation Within the ERDA/Energy Database File on RECON	82
4.4	Distribution of Index Terms by Initial Letter of Term	98
4.5	Distribution of Author Surnames by Initial Letter of Name	101

INTRODUCTION

The body of work reported in this thesis consists of the design, implementation, and initial testing of a file structure and associated software modules for use with an existing large scale bibliographic information retrieval system. This development must be viewed in the perspective of information systems in general and bibliographic retrieval systems in particular. This perspective is provided by the discussion in Chapter I.

Chapter II narrows the perspective down to the actual component of the overall retrieval system file structure which is of direct concern and establishes the basic research objectives. In particular, this chapter provides high-level definitions of the design goals for the new file structure, and compares and contrasts the limitations of the current organization with the expectations for the new.

In the third chapter, the reader is given the details of the design of DSAM (Directory Sequential Access Method), the name coined by the author to describe this new file organization and software system. The material presented includes an explanation of the logical structure of a DSAM file and its physical representation on direct access storage devices. Functional specifications for the overall DSAM system of program modules and their interrelationships are also presented.

The next major division of this thesis, Chapter IV, presents the results of experiments conducted with the embryonic DSAM software system. The general performance of DSAM was evaluated in terms of DASD

storage space reduction from that needed for the present ISAM files, and timing estimates were obtained.

The final chapter of the thesis, Chapter V, summarizes the results of the total design and implementation effort. The chapter reviews the major conclusions that can be drawn from the experimental data at a level whereby subsequent researchers may note the applicability of DSAM to more general information systems and may see extensions to the research effort presented here.

Throughout the thesis an organization of stepwise refinement of coverage area has been employed. Thus, the reader who is familiar with the basic concepts of information retrieval systems, and those familiar with an inverted file structure in particular, may skip the material presented in Chapter I and proceed immediately to the more detailed discussions in later chapters. The same comment applies to the material within a single chapter, and again the reader who already understands the early, broad-scale discussion, can skip to an appropriate point within the subsequent discussion.

CHAPTER I

THE APPLICATION ENVIRONMENT - INFORMATION RETRIEVAL

I. OVERVIEW OF INFORMATION RETRIEVAL SYSTEMS

Bibliographic Retrieval Systems

The general category of computerized information retrieval systems may be partitioned into a number of functional categories. Systems dealing with a corporate database containing fiscal and personnel data, sales and accounting information form one broad class. Another category of systems includes the generalized question answering systems and the artificial intelligence experimental systems that attempt to provide models of human "world knowledge" and behavior. A third category encompasses special purpose systems targeted specifically at a single well defined requirement, such as a commercial database/inquiry system concerned with motor transport routing, or a membership records system for an organization. The last broad category, and the area to which this thesis is confined, consists of the variety of computerized systems that are designed for the searching of bibliographic data, i.e., documents or citation and abstract document surrogates, taken from the scientific literature or perhaps from commercial publications and magazines.

These bibliographic retrieval systems may again be functionally divided into more narrow classifications as follows: (1) Current Awareness Systems, where a (static) user query or "profile" is

repeatedly run against only the most recent additions to the collection of documents; (2) Retrospective Search Systems, where the user's search profile or query is matched against the entire historical collection or group of collections maintained in the system. Even though a Current Awareness or SDI (Selective Dissemination of Information) system may occasionally be used for a retrospective search, simply by processing the query against the entire historical collection, a functional division exists between SDI and Retrospective Search systems in terms of optimization of the retrieval algorithms with respect to the (usual) collection size.

One further level of classification establishes the exact area of this research in terms of "Information Retrieval Systems." Three broad groupings are possible for the Retrospective Search systems. First, the term "Batch Search" refers to all systems that rely on a complete serial scan of the document collection comparing the user's query (or more often a large batch of such queries from several users) against every document. Examples of such systems are in the University of Georgia SDI-RETROSPECTIVE SEARCH programs or modified versions such as are in current use at ORNL (47). These systems typically process 10 to 500 or more search requests as a group, and involve mounting and reading numerous magnetic tapes. Thus, their operation is confined to slow response time situations. Furthermore, in such systems it is impossible for the user to modify a query while it is actually being run against the file. Thus, in these systems, the formulation of a precise and exhaustive query is required before searching. A number of

techniques have been devised for improving the user's observed results. These include sorting results by their degree of match to the query, and applying some cut-off value criteria to limit the output. Nevertheless, these systems are not "interactive" in any real sense of the word, and will not be discussed further in this report.

The second grouping of retrospective search systems is again not of primary interest in this report. This group contains the systems that rely upon a serial search of the (usually rather small) collection, using text-string pattern matching techniques to select relevant citations. Some such systems are in fact interactive, operating in an on-line manner, and may be typified by the ORLOOK (56) system at ORNL. The basic process used with ORLOOK is for the user to form a query that contains boolean combinations (AND, OR, NOT) of "terms" or, where applicable, numeric operators such as "less than" or "greater than." Except for the numeric terms, each term is a character string, which is sequentially matched against all of the text stored with a given document or the text within certain restricted fields such as "TITLE" or "AUTHOR." A term is found if the string exactly matches some subset of the text being scanned. Thus, a search term of "catas" would be located in the text if the words "catastrophe" or "catastrophic" occurred in the text. If, however, the search term had been "cata," the search would also match on "catapult," "uncataloged" or "catalepsy" which were presumably far from the user's desires in the example chosen. These systems, often called "text search" systems, suffer from the type of ambiguity shown, although the judicious use of surrounding blanks,

etc. can be made to reduce unexpected matches. Indeed, the Inverted File oriented systems discussed later in this report often permit (or simulate) this text search capability once the collection has been reduced to a reasonably small size by other techniques. The reason for the limitation on collection size (typically less than 8,000 documents) is simply the amount of time necessary to scan that amount of text, string by string. Recently, AHO (1) has shown that it is possible to scan in a time linearly proportional to the amount of text. Nevertheless, in terms of a user sitting at a terminal, the response time to even a single "term" query, given a large collection, would be unrealistic. Thus, text searching systems are simply not usable with the collections of 10,000 to 1,000,000 or more documents that are prevalent today.

Before leaving the group of text search systems, we must consider the SMART (49, 50) system for logical completeness. As implemented by Gerard Salton and his group, this system is an experimental vehicle for research, not an "operational system" for use with really large (more than 20,000 document) collections. Much of the software is devoted to allowing great flexibility in the algorithms used and the associated dictionary or thesaurus files, coupled with "instrumentation" to allow detailed comparisons of the results of the different experiments. It operates in an iterative batch mode, like the SDI and retrospective systems previously mentioned, but there the similarity quickly ends.

In SMART, both "documents" and "queries" are represented by n-dimensional numeric vectors of "term" weights. The quotations around these words are used because of the wide variance in their definitions from that common to other systems. Automated processes convert the queries and collections to this vector representation, using a variety of dictionary files, including word-stem dictionaries and artificial "phrase" generation algorithms.

Although for small collections the query vector could be directly compared with every document vector, (as was done in the original SMART experiments), most recent work has involved preprocessing of the collections to "cluster" documents into groups, based upon their similarity, and then use matching between the query and "centroid vectors" derived from the clusters.

This work in clustering has also yielded improved methods for dictionary construction. These improvements are achieved by using measures of document space density with and without a candidate term in order to rank potential terms by their discrimination value (8). Recently, this type of term selection process has been explored for use in inverted-file based system thesurus construction (39).

At this point, we arrive at the third category of bibliographic information retrieval systems, and the subject area of this research, namely, the Inverted File systems. These systems are called "Inverted File" because they rely upon extensive index files, where the entries in these files identify all documents containing a given term. Thus the file system has been "inverted" to provide information entries that

identify groups of documents having similar content, as opposed to entries which are themselves "documents" and contain lists of entries describing the contents of that document alone.

Systems Using Inverted File Structures

Several Inverted File retrieval systems are available on the commercial market. Others are available only to the academic community or the federal government. Among these are: The New York Times Databank; International Business Machine's STAIRS; Lockheed's DIALOG system; SPIRES (58) at the University of California; the NASA RECON system, and ERDA/RECON (5), as operated by Oak Ridge National Laboratory for the Energy Research and Development Administration of the federal government. The last of these examples, ERDA/RECON, will be used as an example in all that follows, in that it is both typical of inverted file systems in general, and the actual system with which this research was conducted. The Stanford University Comparative Study (42) and LANCASTER (32) are excellent sources for the reader interested in other systems.

Before proceeding to the discussion of RECON, it is necessary to characterize inverted file systems on a more general basis. Such systems possess the following characteristics:

1. They usually operate in an on-line interactive search mode. However, nothing about their design requires on-line execution. Indeed many of the systems permit their use in an overnight batch mode. In this latter mode, predeveloped or stored query sequences are supplied on

cards, and the card reader then assumes the role of the (only) on-line user. Any intermediary results are routed to the printer or suppressed. When used in a batch mode, these systems lose their primary effectiveness, in that the user may not modify the search strategy during the course of the actual search.

2. The systems typically serve multiple users simultaneously in a multi-tasking mode. Low user input rates (typing), file browsing, and evolutionary search strategy development (thought) characterize the systems. Thus the inefficiencies of nonreentrant software cannot be tolerated.

3. Extensive use is made of data communications facilities, including high-speed leased telephone lines, special-purpose terminals and dialup access over the conventional telephone network.

4. Collection size for a single group of documents begins at about 2,000 for the sample collections or special purpose subsets, and typically runs on the order of 50,000 to 2,000,000 documents for the largest files. An average size of 40,000 citations is realistic for the systems of today. Furthermore, most of these systems support multiple collections simultaneously, although often the collections must be searched separately by the user and the results combined only during final printing of the targeted material.

5. Documents may be retrieved only by use of the terms that actually occur in the (precreated) index and inverted files. Systems vary widely in how these "descriptors" are defined, which fields have been inverted during file loading and what control is exercised over

the indexing vocabularies. Thus, if every term in every field is made a descriptor, performance similar to full text searching can be achieved with these systems. Some systems extend this to provide a form of "context" or "associative" searching, as will be explained later, and still other systems may actually implement full text-string searching after the set of potentially relevant documents has been reduced to manageable size using the inverted files.

6. Depending upon the degree of inversion (number of document fields indexed) and the amount of information carried into the index files (only the citation numbers, or positional information as well), the index files and inverted file proper for a given collection may often exceed the storage requirements of the actual documents themselves.

7. Typically, the query logic permits boolean operators (AND, OR, NOT) on sets of documents derived from selected descriptors as the primary method of search structure. Several systems augment this with numerical and range operators (< = > ≠ etc.) or adjacency constraints. For example, "cat" in the same sentence as "dog," or "venetian° blind" where the ° implies that VENETIAN and BLIND are separate descriptors, but must occur in that order within the document, i.e., a reference to a "blind Venetian" would not satisfy the search criteria.

8. The systems provide response times on the order of one to ten seconds to all but the most complex queries. Certain operations such as the logical OR of two sets of documents, each over 20,000

citations in length, or requests to consult synonym dictionaries, may require up to a minute or more.

9. Search results, (the actual documents or document surrogates) may be displayed at the user's terminal, printed there, or printed at the central site overnight and mailed to the searcher.

10. Many systems allow the user to browse in the indexing vocabulary by presenting an ordered list of descriptors stored in the system. This is the usual mode of operation for the RECON-like systems.

11. With few exceptions, the collections are updated by an offline process which operates while the system is unavailable for queries. Due to their cost and dangers of file damage, updates are run as infrequently as the user community will tolerate. Typically, a given collection would be updated on a monthly or at most biweekly basis. Many systems including ERDA/RECON have reached the point where the number of distinct collections to be maintained requires the scheduling of one or more updates each week.

12. File updates consist almost exclusively of additions of new material. As vocabulary stabilization occurs, these document additions normally do not greatly increase the count of unique descriptors in the index files. The major exceptions to this appears to be indexes on the personal author(s) of the documents, where an average of one new author may be added for each two or three documents.

13. The update process typically involves sorting and merging terms with those existing in the file, adding new citation numbers to

the lists associated with existing terms, and adding new terms and their (short) citation lists as needed. In many cases, the merged results are used to completely replace part or all of the existing index files, rather than to update those files in-place.

14. Where updates follow the mode of a traditional merge-copy mode of operation, backup copies of the index files are critical. Often, there is simply not sufficient available DASD space to permit generation of the new version of an index file without deleting the current version first. Thus, such a process must occur when the system is unavailable for query use. Further, should the re-creation loading fail for some reason, there must be a backup copy of the old index that can be loaded very quickly and reliably.

15. The on-line environment places severe constraints upon the overall retrieval system software, and upon access methods in particular. The access methods must be efficient in terms of execution time, while simultaneously minimizing computer memory requirements due to the multi-user, multi-file demands placed upon them in a quick response time situation. Moreover, these same routines must be "fail safe" in the face of almost any conceivable calling program logic or hardware error condition, without themselves forcing an abnormal termination (ABEND) of the retrieval system.

16. Provision for file recovery in the case of a failing update run is generally handled by (multiple) tape copies of the existing files. These can be reloaded as needed after a crash. Unfortunately, due to the size of the files involved and DASD characteristics, it often

requires several hours of computer time to reload one or more of the larger index files. Recovery is also complicated by the fact that a large amount of time is needed to prepare the new tape copies following a successful update, and by the fact that the clerical task of keeping track of these "dump" tapes can become immense. Often this complexity itself leads to disastrous human errors.

Much of what has been stated above would seem to argue against the use of inverted file systems for bibliographic retrieval from large collections. Nevertheless, at the present time, systems such as this are the only practical method of searching the literature. The problems are often simply matters of scale and size, and given the growth rate of the literature, they seem to be almost unavoidable.

II. THE RECON SYSTEM AT OAK RIDGE NATIONAL LABORATORY

The ERDA/RECON (5) system at ORNL was chosen as the subject system for this research because of the author's long standing association with that system. In subsequent discussions the ERDA/RECON system will be referred to simply as RECON. The reader who is interested in the historical evolution of RECON at ORNL and in a high level overview of the system and file structure, is directed to the paper by Brooks et al. (5). Gillcrist (10) provides a complete but outdated user's manual. Full technical details of the retrieval system software may be found in Haeuslein (11). No corresponding document exists for the file updating and maintenance software, which has been in a continuing state of change since the origin of RECON at ORNL.

RECON currently supports over 40 concurrent users in a nationwide network. Each specialized communications terminal in this network operates over leased telephone lines. Up to 12 additional users may simultaneously access RECON through dial-up connections over the commercial telephone network, or FTS, the Federal Telephone System. These users may employ any of the so-called "teletype compatible" CRT or hard copy terminals, at communications speeds of 10, 30, or 120 characters per second. All of the system's files are stored at ORNL, and accessed through software that executes in large-core storage on the IBM 360/75 at ORNL. The retrieval system is in continuous operation from 09:00 to 20:00 each day. These hours permit RECON to handle time zone differences between terminals located on both the east and west coasts of the United States. Immediately after terminating the on-line service, a nightly batch program retrieves RECON user's requests for overnight printout of search results and generates both local printouts and a magnetic tape for routing to remote high speed printers over the government's SACNET communications network. These long hours of operation and the nightly print program leave no time during the week for any file updating operations, which, due to the nature of the present file system, cannot be run simultaneously with any retrieval operations. Thus, all file maintenance under the current system must be performed on weekends.

A complete list of the available RECON search commands is given in Figure 1.1, which is reproduced from one of the pages of explanatory and tutorial material that a user may obtain via the RECON HELP command.

```
?LIST   A BASIC LIST OF RECON COMMANDS:

COMMAND *          MEANING/USE.
======= = ============================
HELP    ? EXPLANATIONS OF COMMANDS ETC.
BEGIN   ! START A SEARCH, CHANGE FILES.
EXPAND  " SHOW DICTIONARY OF KEYWORDS.
SELECT  # MOVE TO WORK SET FOR LATER USE
COMBINE $ LOGICAL AND-OR-NOT, WORK SETS.
DISPLAY % SHOW CITATIONS THEMSELVES.
PRINT   & REQUEST REMOTE PRINT AND MAIL.
TYPE    / TYPE OUT ON YOUR TELEPRINTER.
KEEP    ( SINGLE ITEMS IN WORK SET 99.
LIMIT   ) BY YEAR OF PUBLICATION, ETC.
PAGE    0 NEXT PAGE OR CITATION, + OR -.
HISTORY @ SHOW CURRENT WORK SET HISTORY.
SEND    ] SEND MESSAGES TO OTHER USERS.
END     = RELEASE WORK SETS, END SEARCH.

* TYPE THIS, NOT COMMAND NAME.
FOR HELP WITH RECON COMMANDS, TYPE ? AND
THEN THE COMMAND NAME, E.G. ?BEG
ENTER:
```

Figure 1.1. List of Available RECON Commands. This figure is a copy of the display obtained on a RECON screen in response to the HELP command, ?LIST.

Each of these commands is input as a single special character code, corresponding to special labels on the terminal keyboard, but in this discussion the full name of the command will be used for clarity. Martin (41) comments extensively on the use of such abbreviated command dialogues, but the RECON command set has been found satisfactory in everyday use for several years, and there are no present plans to revise the command language to a longer format.

Of the commands listed in Figure 1.1, only the SELECT, COMBINE, EXPAND and EXPAND RELATED TERMS commands are pertinent to the aspect of the file structure discussed here. The other commands will not be elaborated upon in this report.

Command Structure and Services

As in the general inverted file systems discussed in the previous section, RECON searches are performed entirely within the inverted files and their associated indexes, without reference to the actual documents or document surrogates themselves, except for their display and printing as partial or complete search results. A RECON user may use the SELECT command to retrieve lists of documents, represented by their unique accession numbers, and then use the COMBINE command to form additional lists representing the logical intersection (AND), union (OR), and difference (NOT) of these lists. This process is iterated under user control with continuous feedback of the number of citations retrieved, until the user feels satisfied with the number and quality of documents retrieved. At any point in this process the user

may examine the retrieved documents via the DISPLAY command. Thus he may obtain guidance in the selection of additional index terms or logical operators to be applied to these partial results from the keywords displayed with each document.

One characteristic feature of the "RECON" systems is the EXPAND command. This command allows the user to display any portion of the indexing vocabularly in alphabetical sequence. The resulting screen of data, illustrated by Figure 1.2, will contain each term present in the inverted file together with a count of the number of documents that have been posted to the term (under the column headed CIT). In Figure 1.2 the numbers (E01-E10) in the left-hand column are a convenience for use in the SELECT command. These "E" numbers may be entered in a SELECT rather than requiring the user to reinput the entire alphabetic string shown under the "descriptor" heading of the screen image. The column at the extreme right of the data in Figure 1.2 (under the column headed RT) is the number (if any) of thesaurus-controlled "related terms" for each index term. These terms, which may have the usual thesaurus relations such as "broader term," "narrower term," "use for" or "see also," may be retrieved by use of the EXPAND RELATED TERMS command.

The result of EXPAND RELATED TERMS, an example of which is shown in Figure 1.3, is a display similar to the basic EXPAND command, but in this case, each term is related to the parent by one of the normal thesaurus relationships mentioned above. These relationships are indicated by a code in the extreme left column of Figure 1.3 under "T."

```
EXPAND  IT=RADIATION EFFECTS
 REF  DESCRIPTOR                CIT     RT
 E01  IT=RADIATION
       DOSIMETRY_________                1
-E02  IT=RADIATION
       EFFECTS___________      28839     25
 E03  IT=RADIATION EFFECT
      S REACTOR LOCKHEED_                1
 E04  IT=RADIATION
       EXPOSURE (DOSES)__                1
 E05  IT=RADIATION
       FACILITIES________         1
 E06  IT=RADIATION FLUX__        271      4
 E07  IT=RADIATION
       HARDENING_________        415      2
 E08  IT=RADIATION HARDEN
      ING (CHEMICAL)_____                2
 E09  IT=RADIATION
       HAZARDS___________       2193      7
 E10  IT=RADIATION
       HEATING___________        185      3

ENTER:
                                      -MORE-
```

Figure 1.2. Example Results of RECON Command: EXPAND IT= RADIATION EFFECTS.

```
REL. KEYS E02   IT=RADIATION EFFECTS
T R.NO. DESCRIPTOR                CIT    RT
  R0000 IT=RADIATION
          EFFECTS_____________    28839   25
2 R0001 IT=BIOLOGICAL
          RADIATION EFFECTS_     10562   20
2 R0002 IT=PHYSICAL
          RADIATION EFFECTS_      5548    3
2 R0003 IT=CHEMICAL
          RADIATION EFFECTS_      2373    6
3 R0004 IT=BIOPHYSICS______        99    9
3 R0005 IT=CRYSTAL DEFECTS_      1066   12
3 R0006 IT=RADIATIONS______      8028   18
3 R0007 IT=COMPARATIVE
          EVALUATIONS_______     3929   13
3 R0008 IT=RADIOBIOLOGY____       891    7
3 R0009 IT=RADIATION
          QUALITY____________     147    9
3 R0010 IT=DOSE RATES______      2285    5
3 R0011 IT=RADIATION DOSES_     20625   38
3 R0012 IT=IRRADIATION_____     19567   33
3 R0013 IT=ENERGY LOSSES___      3465   24
ENTER:
                                    -MORE-
```

Figure 1.3. Example Results of RECON Command: EXPAND RELATED TERMS. The PAGE command may be used to obtain the remaining 12 terms which did not fit on this screen. PAGE may be used whenever -MORE- appears at the lower right of a screen.

Note that again each such term is shown with its associated number of postings (under CIT) and that each term may itself have further "related terms" (under RT). Thus chaining through the thesaurus structure is possible.

Existing File System Structure

Support of the basic RECON commands described above (SELECT, COMBINE, EXPAND and EXPAND RELATED TERMS) is provided through the use of the following three files: (1) The inverted file (IF) proper; (2) The inverted index (IX) file; (3) The related terms (RT) file. Each of these three files is maintained separately for each unique collection of documents available under RECON. The contents and organization of each file will now be discussed in greater detail. Figure 1.4 may be used as a reference point in the following paragraphs, as it shows the file structure now in use.

The IF or inverted file proper. This file, the heart of RECON, contains the (sorted) lists of documents pertinent to each term in the total indexing vocabulary. Although accession numbers extracted from this list are presented to the RECON user in reverse chronological order (i.e., the highest accession number first), the current implementation stores them in ascending chronological order, necessitating the retrieval system's reading of the lists in reverse. Each list in the IF contains only the accession numbers, with no identifying text to indicate to which vocabulary term they are assigned. Instead, the location of the origin of each list and the count of the

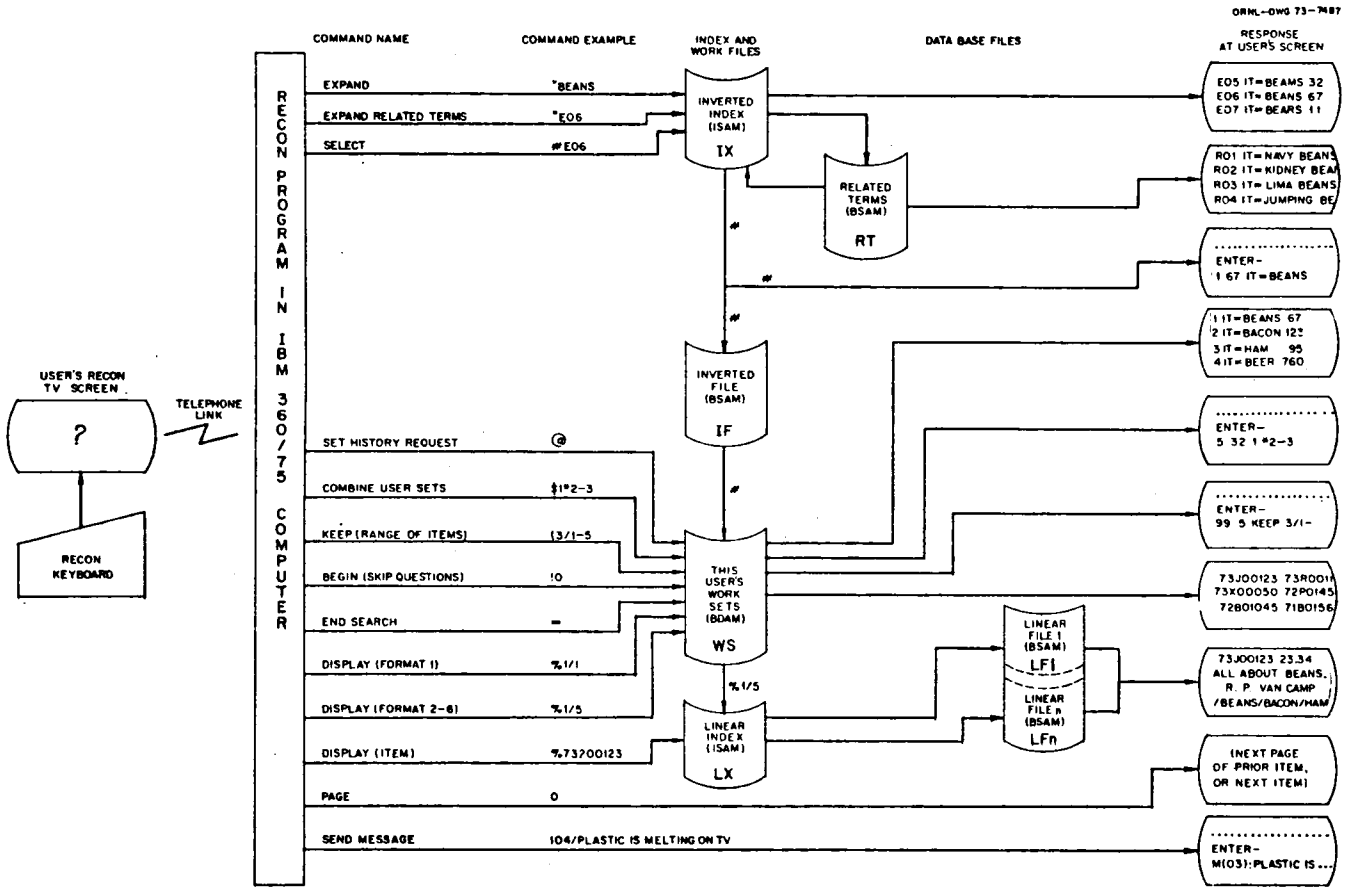


Figure 1.4. Overview of RECON Command and File Structure Relationships. Note that each of the illustrated files corresponds to a single document collection such as Nuclear Science Abstracts (NSA). Thus, there is another complete set of these files for every other literature collection such as Water Resources Abstracts (WRA) that the RECON user may search.

number of elements in that list are posted opposite the proper term in the inverted index (IX) file discussed below.

The IX or inverted index file. Every term used in a single collection's vocabulary has a corresponding entry in the inverted index (IX) file. The entries are stored in ascending alphabetical sequence and are presently maintained as an ISAM file. Each entry in the IX consists of a key field containing the index term itself and two pointer fields. The first pointer field contains the count and address of the corresponding list of accession numbers in the IF, while the second contains the count and address (if any) of related terms which are held in the RT file. Both the EXPAND and SELECT commands use this file as an index to locate the desired list of accession numbers in the IF. Furthermore, this file is used in a recursive manner by the EXPAND RELATED TERMS command as will be clarified below.

The RT or related terms file. This file is used exclusively with those RECON collections that have a controlled thesaurus for at least part of the indexing vocabulary, such as NSA (Nuclear Science Abstracts) and the ERDA Energy Data Base (EDB). Where such thesauri are not available in machine readable form, the RT file is not present for that collection, although such a file may be added at any time to an existing collection. RT consists of simple lists of (varying length) alphabetic terms, each preceded by a length byte and a byte giving its thesaurus relation code ("broader term," "narrower term"). A separate list is maintained for each term in the IX that has such a set of

"related terms." The lists are described only by their origin address and count information stored in the IX file. Note that the items in these related term lists do not contain pointers to the IF nor counts, but are simply lists of words or phrases with appended type codes. Thus, they cannot be used directly to generate the "EXPAND RELATED TERMS" display which includes that data. In operation, the display is formed by (1) retrieving from RT a single "related term" entry; (2) performing the internal equivalent of an EXPAND command using that term as argument, but retrieving only one record from the IX; (3) iterating steps 1 and 2 until all items in the related term list of the original index term have been retrieved, at which point the process terminates with the output of the display. In the actual implementation, the process is halted after the generation of a single screen of output, regardless of the total number of related terms, but may subsequently be resumed with the "next" logical item. This is done because lists of 100 to 500 "related terms" are occasionally encountered, and processing that many terms before returning to the user would degrade overall response time.

Complications of File Updating

As was mentioned in the introductory paragraphs of this chapter, the updating of the RECON file system is presently a background process that may only be run when the retrieval component is not in operation. While such a restriction was not of consequence in an earlier, smaller RECON, the situation has become intolerable today.

This then is the impetus behind the present research and the complete redesign of the file system and its update sequence.

A few statistics will establish the magnitude of the current RECON files, and indicate why the update problem has assumed such monstrous proportions. RECON currently supports some 14 independent document collections, ranging in size from the 560,000 or more items in Nuclear Science Abstracts (NSA), downward through several collections on the order of 50,000 to 100,000 items such as Water Resources Abstracts (WRA) and Nuclear Safety Information Center Database (NSC), to a number of smaller collections of 10,000 to 50,000 citations. Most of the larger collections are growing at the rate of from 10,000 to over 100,000 new citations per year. The user community accessing these files has come to expect that currency of information will be on the order of an update every two weeks for most collections, although the smaller files may have a longer period of stability.

The total collection, which in almost all cases does not even provide for the storage of full abstracts with each document surrogate due to storage limitations, presently exceeds 1.4×10^9 bytes. The linear files (LF) for these collections occupy over three and one-half IBM 2321 datacell drives, each with a capacity of 400,000,000 characters. At the same time, some nine full 2314 disk drives (29,000,000 characters each) are devoted to storage of the index files (IF, IX, RT and miscellaneous files) addressing this collection of data.

The rate of growth of the existing collections, and the constantly increasing demand for still more new collections has become such that RECON has (1) ordered approximately 400,000,000 bytes of new DASD disk storage (IBM 3330 type) for index files; (2) is negotiating the purchase of additional DASD capacity for the archival storage of the linear files of citations. Due to DASD space limitations and the basic update program philosophy inherited from the original system as furnished by Lockheed (5), the management of this system of files has become an almost intolerable burden on the operating staff and the available computational resources. In essence, the updating of any given collection as flow charted in Figure 1.5, may be capsuled as follows:

1. The new citations are appended to the end of the existing LF collection. Changes to existing citation records are not permitted, except by writing a copy of the complete citation as if it were a new record, and changing the linear index to point to the revised copy. The space occupied by the earlier version is not reclaimed.

2. In the process of appending the citations to the LF, two sequential files are generated. The first file contains the location of each new LF record paired with its accession number. The other file contains the text of all individual index terms paired with the source accession number.

3. The file containing the addresses and their paired accession numbers is sorted and merged with a tape corresponding to the current linear index (LX), which is then used to load a complete new linear

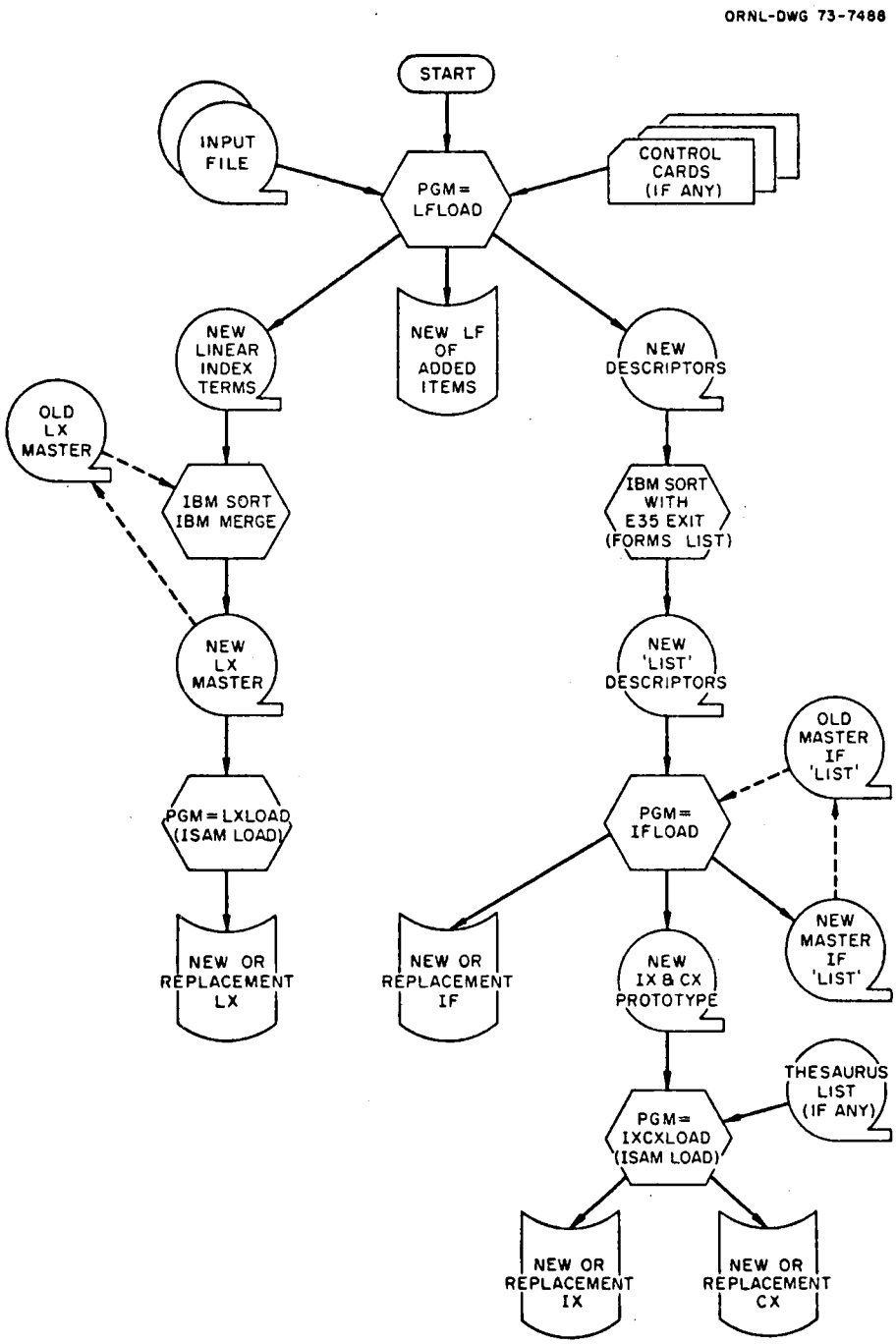


Figure 1.5. RECON File Update Procedure for a Single Collection. The file labeled "CX" is the author name index, and may be combined with the "IX" inverted index file for small collections.

index file for the updated collection. No attempt is made to use direct ISAM (Indexed Sequential Access Method) update of the existing file, even though the new material is often restricted only to the addition of new records with accession numbers higher than any already in the LX. The primary motivation for this is that should such an ISAM update exceed the total fixed original space allocation, an update ABEND (abnormal termination) would occur leaving the file totally unusable.

4. The file containing term-accession number pairs is sorted by term, then by accession number. After it is merged with the sequential master list to which it corresponds, it is used to create a new inverted file or IF.

5. As the new inverted file is created, the term corresponding to each list of accession numbers, paired with the address and count of the list itself, is output to another sequential file.

6. This last file is then merged with an index to the lists of terms in the related terms file (if any), and used to create a new IX (Inverted Index) file for the collection, thus completing the update. Again, the IX file is currently of the ISAM organization, and is never truly updated in place, but is totally re-created with each collection update cycle.

As should be clear, a RECON update is a background sequential process that is not a true "update" in the normal sense, but is a total re-creation of the file system for that collection, except for the linear file of citations. No use is made of any direct-access file

updating capabilities of the ISAM files, nor the BDAM (Basic Direct Access Method) organization of the inverted file.

Although such a sequential, replacement process was suitable for the original, very limited RECON collection at ORNL, its continued use in the present environment is completely unacceptable for the following reasons:

1. The size of the files to be re-created is now prohibitive in terms of both costs and time available. The inverted file proper for only the NSA collection now exceeds 1.5 2314 disk volumes, and the IX file has been split into two segments due to its size. The smaller of these segments, containing all terms other than the personal authors of each document, is stored on a 2314 disk and now exceeds some 1,000 tracks. The second and far larger segment contains some 360,000 index entries stored on a 2321 datacell drive, and occupies over 35 million characters of space.

2. Device characteristics, particularly those of the 2321 datacells, preclude simultaneous file loading and retrieval due to excessive degradation of response time if the same device is in use by both processes simultaneously.

3. Existing files may not be scratched (erased) until after the successful completion of a collection update. Otherwise, access of the collection by the on-line system would be lost for an excessive time period after a failed update. Therefore, DASD space for complete new copies of all of the index files must be available. Furthermore, due

to the costs and time required, this space should be so arranged that the new files need not be moved after re-creation.

4. The prior comment about not scratching existing files would seem avoidable by simply keeping magnetic tape copies, and reloading from these if the update failed. This alternative is precluded by the facts that (a) time available may be limited, and (b) the installation's current tape drives and volumes have been found to be highly unreliable for large amounts of data transfer, although they may well be satisfactory for smaller volumes of data.

5. For the reasons of operating security mentioned briefly above, each on-line volume, be it disk pack or datacell bin, is backed up by an interchangeable copy stored off-line, as well as by two complete magnetic tape dump copies in addition to the master tapes used in updating. Preparation of these copies, and maintenance of the tapes requires more than twice the time necessary to complete a "simple update" per se. (For example, three reels of 800 bpi density magnetic tape are required to dump a datacell bin, such as the author index of NSA, and the dump requires a minimum of 45 minutes of wall-clock time with present tape drives.)

6. Certain peculiarities of the IBM ISAM access method are such that a re-creation of an ISAM file may abnormally terminate leaving the existing part of the file unusable for retrieval or update. The situation is particularly dangerous during the re-creation of the large ISAM author index files, which are maintained on 2321 datacells and which require more than an hour to load.

7. The total time available for file updating on weekends has become too short for the frequency of updating demanded by the users.

Revised File System Design

The foregoing discussion has made it clear that the existing RECON update process cannot be allowed to continue for much longer as the normal mode of operation, if RECON is to continue to grow. The need for a completely revised update process, probably including total redesign of the file system, has indeed been apparent for several years.

The work presented here is the first major report of the redesign effort which has been in progress over the past two years. The development of an improved file organization and software system called DSAM (Directory Sequential Access Method) is a single phase of the overall design revision. The new update process will replace every one of the current tape-oriented batch re-creation update steps with in-place modification of existing data.

The major change to be made to the file structure is confined to three areas: (1) The ISAM IX files will be replaced by DSAM equivalents, (2) the basic linear files will be changed to the ORCHIS-ADSEP format for compatibility with the total ORCHIS (4, 6, 57) system at ORNL, (3) the IF and RT file organizations will be replaced by a chained BDAM scheme. The sorting order of the citation numbers in the IF will be changed to reflect the reverse chronological order in which they are presented to the RECON user.

Chapter I has established a framework within which the reader may now proceed to the specific topics of this thesis, as presented in subsequent chapters. The reader should now have some degree of familiarity with information retrieval systems in general, the inverted file oriented systems as a subclass, and ERDA/RECON as the specific software system environment within which the research was conducted. In Chapter II, the focus is narrowed to the problems associated with the inverted index or vocabulary files alone, and to DSAM as the proposed solution.

CHAPTER II

AREA OF RESEARCH - ADVANCED FILE SYSTEM DESIGN

I. THE INVERTED INDEX FILES

Limitations of the Existing Access Method

The RECON inverted index (IX) files are presently supported by the vendor-supplied ISAM (15, 18, 19, 25) access method. With the continued growth of the RECON file system, ISAM has been found to have the following basic limitations that preclude continued use for this purpose:

1. ISAM requires a fixed length key in each logical record of data. This leads to excessive DASD storage requirements for the RECON application.

2. ISAM files used in a true insertion update manner suffer severe performance degradation due to the chaining of "overflow" records.

3. ISAM error control and recovery options are inadequate for simultaneous update and retrieval operations in an on-line environment.

4. Creation of a new ISAM file requires precise knowledge of the total space requirement of the data before commencing file loading. If the given space is insufficient, no mechanism exists to recover the data already loaded, and the creation job must be run again from the start.

5. ISAM does not support reading records in descending order of their key fields subsequent to direct positioning to a given key.

The requirement for a fixed length key in each record is perhaps the most severe restriction upon the use of ISAM for the RECON IX file application. This restriction wastes a large amount of space in the files. First it is necessary to choose a rather long key length to minimize the number of records that "collide" because the keys have the first "n" characters in common. Second, a long key length is needed to preserve visually meaningful keys for use with the RECON EXPAND command, where English language key words are displayed in alphabetical order to the user. Thus, hash coding or any one of numerous key compression algorithms would require considerable processing overhead to re-create the actual keys.

The performance degradation observed in ISAM files after a moderate number of insertion updates is well known (3, 7, 49). This then requires rather frequent file reorganization efforts to minimize delays in retrieval performance for the on-line system. Such a reorganization also cannot be done in-place, but requires copying the file to another direct access storage device (DASD), or onto magnetic tape, followed by reloading. However, in the RECON environment there is insufficient DASD space available to accommodate frequent reorganization runs, and thus the files are simply created anew each time they are updated.

The third and fourth of the stated limitations of ISAM are severe operational problems for RECON. The system operates in an environment of severely limited DASD storage space for either maintenance of backup

copies or allocation of new file versions. This space limitation subsequently requires much shuffling of dataset locations to obtain sufficient, unfragmented space for a new file.

The fifth limitation is not very severe, in that it only causes difficulty in the EXPAND command when the user desires to scan backward in the alphabet without entering a new search string with a lower key. Nevertheless, it is annoying, and it would be desirable to have this capability available.

Review of Commercially Available Alternatives

In view of the previously discussed limitations of ISAM, several alternative systems were evaluated. Each of these systems alleviated some of the problems, but none met all the criteria in the DSAM design given below. The single system which provided the closest approximation to the RECON requirements was VSAM (Virtual Sequential Access Method) (21).

The IBM version of VSAM (20, 22, 23, 24) is not supported on the 360 hardware available for use with RECON, nor is it useable under our present operating system software, OS/360-MVT. However, ITEL has recently announced a 360 compatible version of VSAM, which would circumvent this problem if ISAM was otherwise suitable.

Preliminary study of the RECON files had shown that a 30 to 45 percent space reduction could be achieved for the IX files of RECON if the chosen access method supported variable length keys in the data records. Investigation of VSAM, including consultation with IBM

representatives concerning probable future enhancements to VSAM, unfortunately revealed no current or planned support for variable length keys, and revealed excessive software memory requirements. Thus, implementation of DSAM was begun, borrowing heavily from the concepts and terminology of VSAM.

Design Criteria for DSAM

The brief statements below constitute the original set of DSAM design criteria used in its development. Each point will be amplified in the discussion of "Research Objectives" in this chapter, and again in the form of the System Level Functional Specifications presented in Chapter III.

1. DSAM should provide all the capabilities available under ISAM, while reducing DASD storage requirements.
2. DSAM should support variable length keys in data records, as a means of achieving space reduction.
3. DSAM must provide increased reliability and data integrity protection over ISAM, and permit simultaneous update and retrieval with minimal possibility of a failure in the update causing an abnormal termination of RECON.
4. DSAM should minimize the need for file reorganization.
5. DSAM should extend the types of access supported by ISAM, including backward sequential access capability.
6. For ease of maintenance, DSAM should adhere to the common principles of good software design such as top-down

structured development, standard interfaces and restrictions on the scope of variables.

7. Minimization of both execution time and memory utilization is needed.

II. RESEARCH OBJECTIVES

Reduction in On-Line Storage Requirements

The ISAM version of a RECON inverted index (IX) file uses fixed length logical records of 66 bytes length for each term in the vocabulary. This is done regardless of whether the term is a descriptive phrase or an author's name. Of this, some 42 bytes are devoted to the fixed-length key portion, with the remaining 24 bytes of each record holding pointer and count information to be used in addressing the inverted file (IF) and/or related terms (RT) file. The content of the data portion of each record, and the fact that it too could be subjected to any one of several forms of compression, as should be done in the final version of the new file structure, is of no further concern at this point.

A quick estimate of the size of a RECON IX file follows directly from simply multiplying the constant 66 byte record length by the observed number of terms, and neglecting any overhead for ISAM index structures, recorded keys in front of blocks of logical records and interblock gaps. For example, the RECON Nuclear Science Abstracts (NSA) IX file has some 390,000 unique records, and thus needs an absolute minimum of over 25.7 million bytes of DASD storage. This

corresponds to almost an entire IBM 2314 disk (29 million bytes) for this single index file. Obviously, with 14 current file systems, and many more planned, anything done to significantly reduce this DASD space requirement is highly important. DASD storage space reduction is the primary motivation behind DSAM.

Backward Sequential Access

The RECON EXPAND command requires that the user be shown a display of terms in the alphabetical range near what he types as part of the command. With ISAM, it is trivial to retrieve the logical records with keys higher than the source key, but almost impossible to obtain those immediately preceding it in the collating sequence. Thus, a RECON user may "page forward" as much as desired (although not with the current retrieval software, which limits him to 40 terms), but cannot "page backward" beyond what he types as his key. It would be beneficial if this restriction could be lifted.

Fail-Safe Dynamic Update

ISAM files support updating simultaneous with retrieval, but are known to suffer from sudden crashes making the files unusable. Thus a simultaneous retrieval program may abort when it attempts to (re)open such files. DSAM must provide for improvements in this area, such that DSAM can be safely integrated into the RECON update sequence without fear of dynamic updates causing an abnormal termination in the executing version of RECON.

Reduction in Reorganization Frequency

When an ISAM file is subjected to insertion update, the technique used to handle the overflow soon results in long chains of such overflow records. So long as sufficiently large cylinder overflow areas are available, performance degrades only slightly, since there is no need to move the DASD access mechanism to follow these chains. However, additional delays due to waiting for DASD rotation are incurred since IBM's ISAM does not block logical records in the overflow chains.

Under ISAM, overflow records which cannot be accommodated in the cylinder overflow areas are placed in an independent overflow area. When this occurs, following overflow chains begins to require extensive movement of the access mechanism, and performance degrades severely.

The obvious cure for ISAM performance degradation is to reorganize the file by copying it to a new location. For RECON, and in particular the author index portion of the IX, reorganization would be needed after almost every update. In the present RECON update scheme, reorganization is combined with "update" by replacing the entire file from sequential input, rather than using ISAM update followed by this reorganization.

Clearly, DSAM must provide for rather heavy levels of insertion without performance degradation. This requirement quickly rules out the conventional overflow technique of handling such insertions. The alternative used in DSAM and VSAM is to distribute free space within the file to accommodate updates by insertion with automatic recovery of

the free space created by record deletions. Even here, care must be used in the treatment of overflow records. Otherwise, performance degradation will still occur due to access mechanism motion, as with ISAM.

Flexibility and Extensibility

The design of DSAM reflects what the author believes is a reasonable balance between software that can "do anything," and a system which is useful only to the present RECON version. The discussion below illustrates the type of design tradeoffs necessary in the development of a system such as DSAM.

Recall for a minute the fixed length (42 characters) key in every RECON IX record. Klingbiel (29) has studied bibliographic data bases with up to three million unique words in their vocabulary and has shown that the number of words exceeding 25 to 30 characters in length is near zero, even in scientific text. Before making a restrictive decision, however, remember that in RECON, phrases are often used as index terms. Additionally, pre-coordination of terms (main term/qualifying term) generates further, artificial phrases. Accordingly, a proposed maximum term length for DSAM would be longer than implied by the Klingbiel study of simple word lengths. The point here is not the numerical choice made in the particular case (254 bytes), but the fact that every stage of the design requires detailed analysis and tradeoff judgments.

For every choice made to achieve flexibility, there is always a penalty. The penalty may be in any one or more of the areas of

(a) program complexity; (b) memory requirements; (c) DASD storage requirements; (d) potential for errors in logic or unexpected hardware faults; and (e) requirements for excessive user intervention or parameter specification.

The choices made for DSAM are attuned to achieving software that is reliable, may be quickly implemented, and contains provisions for ease of maintenance and extension, as opposed to providing all of the potentially desirable features in the first version. What has been provided is a logical framework upon which to build, when it becomes necessary.

Viability of Nonuniform Initial Space Distribution

The author has a practical interest in the development of DSAM but he also has an academic or theoretical interest in the properties of vocabulary files and the systems that support them. After observing the RECON files over a period of years, and reading studies such as Klingbiel, the author developed the ideas that are stated below as a series of questions to be answered by research.

1. If DSAM uses distributed free space in a manner similar to VSAM to accommodate insertion updates, can we improve on the simplistic distribution of a uniform amount of such space throughout the file? In particular can we take advantage of knowledge obtained concerning the alphabetical distributions of terms within the file?

2. If such knowledge of an existing alphabetical distribution can be used successfully, how much does it reduce the need for file reorganization?
3. If we apply distribution data from one collection to a file built from a different collection, what degree of confidence do we have that the situation is then better than if we had used a simple, uniform free space distribution?

The attempt to answer the three questions posed above, at least to a limited extent, was an additional motivation for the development of DSAM.

At the close of this chapter, the reader should now have a much clearer understanding of the motivations behind the development of DSAM, the specific nature of the problems with the current ISAM implementation, and the goals to be achieved in the software design and experiments that are detailed in Chapters III and IV respectively.

CHAPTER III

THE LOGICAL DESIGN OF DSAM

I. DSAM LOGICAL FILE STRUCTURES

Basic Terminology

A DSAM file consists of two logical subsets of information, the Data Area and the Directory Area. The first of these houses the information-bearing logical records manipulated by the user of DSAM, while the second houses the internal indexing structure utilized only by DSAM itself to locate the user's data records in a random access mode.

These two areas are logical entities as opposed to physical areas on a DASD. The physical records used to house the data in both areas may in fact be either adjacent within a single dataset, but physically disjoint; intermingled within a single dataset (using linkage pointers to maintain their disjoint logical status), or stored as separate OS/360 datasets.

Much of the remaining DSAM terminology is adopted from that used for VSAM, as given by Keehn and Lacy (28). Both the Directory and Data Areas are further subdivided into a number of fixed length physical records, hereafter called Control Intervals (CINVs). Within either the Data Area or the Directory Area, all CINVs are of uniform fixed size, chosen under user control from a limited set of permissable sizes. When the two areas are stored as distinct OS/360 datasets, the CINV size used

in the Directory Area may be different from that used in the Data Area. Otherwise, all CINVs in the file must be of a single fixed length.

A Control Interval is further divided into three distinct fields: (1) Control Information Field (CIF); (2) Record Image Field (RIF); and (3) Free Space Field (FSF). Every CINV must have the fixed length CIF. Either of the RIF or FSF fields may occupy the balance of the CINV, although both of these fields are usually present.

The Control Information Field (CIF) contains DSAM status information for that CINV. For example, it contains an offset pointer to the start of the highest-keyed record in the CINV, an offset to the first byte of available free space (FSF), and DASD address pointers to the logically "prior" and "next" CINV in collating sequence. The CIF field also contains status flags indicating whether the CINV is in the Directory Area, Data Area, Empty, etc.

The Record Image Field (RIF) contains the logical records themselves, whether these are the user's data records or DSAM directory entries. The records are stored in ascending sequence of their keys within a CINV and are maintained in this order during any subsequent updating operations on that CINV.

Within each CINV, any bytes not used for either of the CIF or RIF fields are grouped together at the physical end of the CINV, and termed the Free Space Field (FSF). The FSF holds newly inserted records during updating or expansion of the RIF due to replacement of existing records with new versions that are longer than the original. Thus, the FSF grows and contracts during file updating as records are added,

deleted or replaced. When a CINV contains only the CIF and FSF fields, the CINV is termed EMPTY and an appropriate flag is set in the CIF field of that CINV.

One other term must be defined before discussing the DSAM logical directory structure, namely the Control Area (CAREA). A Control Area is defined as a fixed number of CINVs. The number of CINVs in a CAREA is specified by the user during initial loading of the file, and is thereafter unchanged. The CAREA is the basic unit of DSAM storage allocation of DASD. For example, if the size of a CINV is 1692 bytes, a CAREA size of 80 CINVs might be chosen. This would correspond to one cylinder on a 2314 disk. Thereafter, when DSAM acquired DASD space, it would be allocated in multiples of a CAREA, i.e., in units of cylinders for the example case. A DSAM file must contain at least two CAREAs, one to house the indexing information (Directory Area) and one to house the data records (Data Area). As will be shown in the discussion of the tree-structure of the DSAM index, a DSAM file's Directory Area should rarely exceed a single CAREA in size, whereas a larger number of CAREAs may be needed for the Data Area.

When a DSAM file is initially created, free space for updating may be provided in one of two ways: (1) a percentage of each CINV may be reserved for an FSF; or (2) a percentage of each CAREA may be reserved to house EMPTY CINVs. Either of these two methods of providing distributed free space may be employed, but normally a combination of both methods is specified. Note also that the percentages used may either be consant for the entire range of data loaded, or may vary from

CINV to CINV and CAREA to CAREA under user control. This latter technique permits tailoring the distribution of free space to conform to some expected distribution of update terms, which will hereafter be called variable free space distribution as opposed to the simpler fixed free space distribution, where the percentages are unchanged throughout the entire file.

Logical Structure of the Directory

To permit random access by key to any logical record, DSAM maintains a tree-structured index within the Directory Area. Each logical record consists of a key-pointer pair hereafter termed "index entry" which addresses some CINV in the Directory Area or in the Data Area. In every case, the "key" of the index entry pair reflects the highest key in the CINV addressed by that entry, while the "pointer" resolves to the DASD address of the CINV concerned. Thus in Martin's (40) terminology, the index is sparse or nondense, since the addressed entities are CINVs, not individual logical records.

The lowest level of this directory structure is termed the sequence set and consists of CINVs which contain the address of every nonempty CINV in the Data Area. The sequence set CINVs are identifiable by a flag value held in the CIF of every such CINV; this flag is not repeated in each index entry. Thus, within the sequence set, and indeed within the Directory as a whole, the index entries in a CINV must all occur at the same level in the tree structure.

For moderate to large files the sequence set will be too large to be held in memory. In addition, the sequence set cannot be searched with a binary search technique due to the varying length keys. Thus, a second and sometimes higher levels of index entries must be created which address the lower level entries.

In DSAM this tree-structure is built up until all index entries may be housed in a single CINV. The resulting tree-structure is illustrated in Figure 3.1.

The DSAM directory is clearly a variant of the B-Tree (2), whose basic properties are summarized by Knuth (30, 31), and recently reexamined by Held (13). The directory is a balanced tree (30, 31), since the same number of nodes must always be accessed to reach the data, starting from the root of the tree. The property of constant path length follows directly from the way the tree is built, and the requirement that all index entries in a given CINV occur at the same level in the tree structure.

Since the index is sparse, any insertions or deletions of Data Area logical records which do not change or delete the high key of a Data Area CINV will cause no change in the Directory tree.

When an update in the Data Area does result in a change in the high key in that CINV, or in the address of the CINV which contains the specified key, the directory changes will only have to occur in the sequence set entries, unless the change is propagated in the higher levels of the directory. Such propagation is, however, an exception rather than the norm.

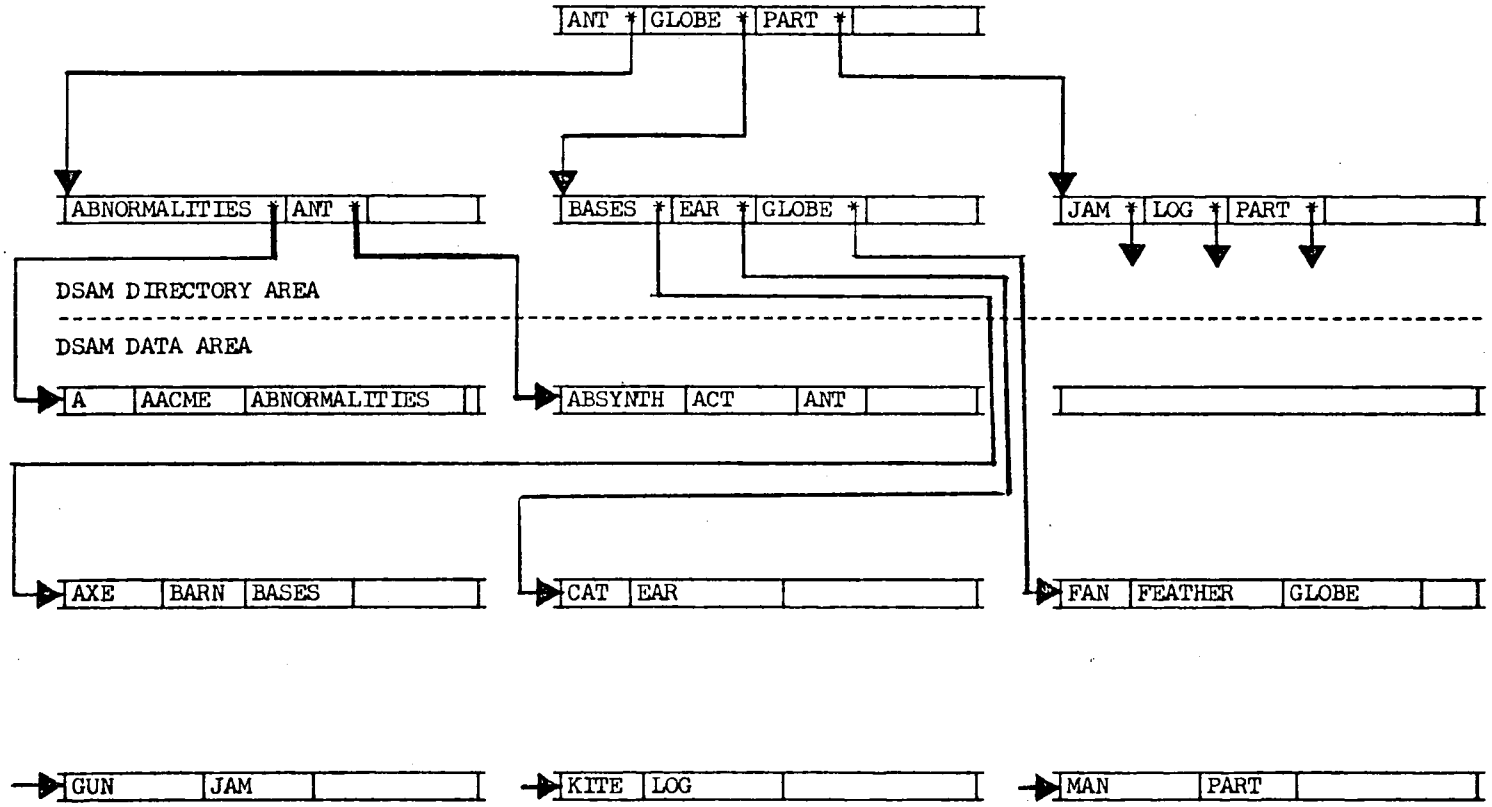


Figure 3.1. DSAM Directory Tree Structure. All pointers are Relative Block Addresses, (RBAs), of the CINV to which they point. Forward and backward pointers between all CINVs have been omitted for clarity in showing the tree structure.

Thus, it should be possible to support the bulk of routine updating activities for a DSAM file with minimal changes in the corresponding directory entries. However, occasionally an update will cause extensive changes in the directory, and the mechanisms used to handle that situation must therefore be addressed. In this discussion, we will use as an example the insertion of a new record into a Data Area CINV that cannot be accommodated in the remaining FSF bytes of that CINV. The basic method of handling the change for this case is easily related to the necessary actions in other cases.

Consider the example Directory and Data Area structure shown in Figure 3.2. If we now attempt to add a new data record with the key "DSAM," we find that it cannot be inserted in its proper sequence in the data area CINV which has "EAR" as its high key. DSAM resolves this problem as follows:

1. The Data Area CINV is split into two roughly equal halves, by using one of the EMPTY CINVs in the same CAREA, and relinking the chain of CINVs to insert the new CINV in its proper sequence. This stage is shown in Figure 3.3.

2. The pointer for the CINV containing "EAR" must now be changed to address the new CINV that now contains "EAR" as its high key.

3. The old CINV that now contains "DSAM" and ends with "EACH" must be up-posted in the directory, since "EACH" was not previously a key in the directory. Unfortunately, when this is attempted, the directory CINV which should contain "EACH" has insufficient free space to accommodate the new key, and itself must be split. When this has

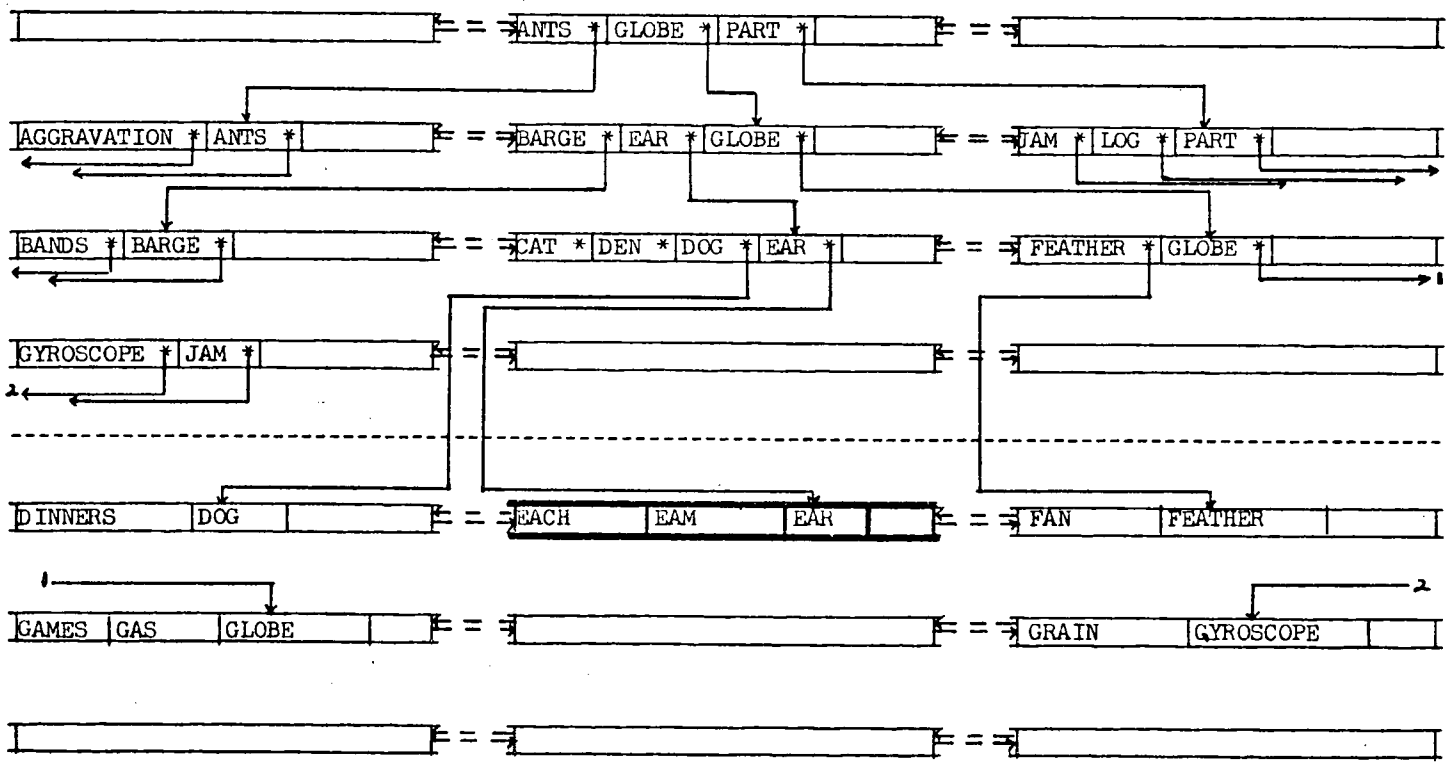


Figure 3.2. Example DSAM File Structure, Prior to Insertion of New Term: "DSAM." The CINV which should hold the new term, but cannot do so within its available free space is outlined. Note: Directory pointers are shown as pointing directly to a word within a CINV to highlight the relationship of the keys. In the actual file, these pointers address only the CINV, not any particular word within that CINV.

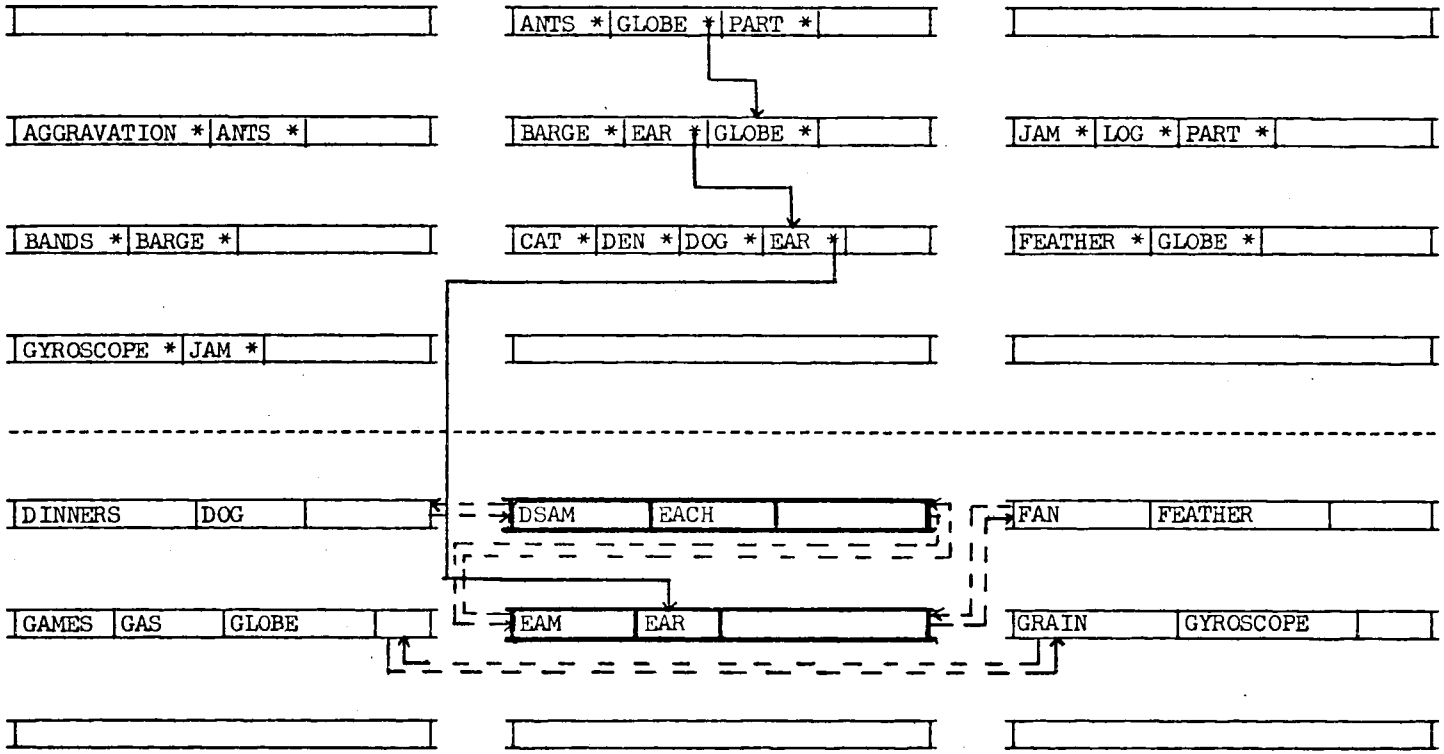


Figure 3.3. Illustration of CINV Splitting. This drawing represents an intermediate stage in the insertion of a new term: "DSAM" into the structure previously shown in Figure 3.2. All pointers not directly involved in the splitting process have been suppressed for clarity. Note the alteration in the linkage pointers between CINVs in the Data Area necessary to maintain proper collating sequence. The new high key "EACH" has not yet been inserted in the Directory.

been done, all that remains is to alter the pointer in the next higher level, and then insert "DEN" in that level of the index. The final structure is shown in Figure 3.4.

In the above discussion we ignored the situation when an EMPTY CINV could not be located to house the spill-over from the insertion. In DSAM, such a situation is handled by splitting the CAREA, in a manner directly analogous to the process of splitting a CINV. However, a CAREA split involves massive data rearrangement. Therefore, whenever CAREA splitting begins, it is a good indication that the file is in need of reorganization to recover additional free space within the CINVs. There is almost no likelihood that further additions to the CAREA which was split would result in further splitting. However, as shown by Keehn (28), it is quite likely that with random updates, some other CAREA in the file is also on the verge of splitting.

The design of DSAM defines the process of CAREA splitting, and includes provision for the output of warning messages whenever this process occurs. However, in the implementation completed at the time of the publication of this report, CAREA splitting is not supported. The current implementation simply rejects and outputs to an error file any transaction which would cause CAREA splitting. The update process is not terminated, and any subsequent transactions which do not require CAREA splitting are processed in the normal fashion. In the rare cases where some transactions are rejected, it is presently necessary to reorganize the DSAM file using a utility program. The rejected

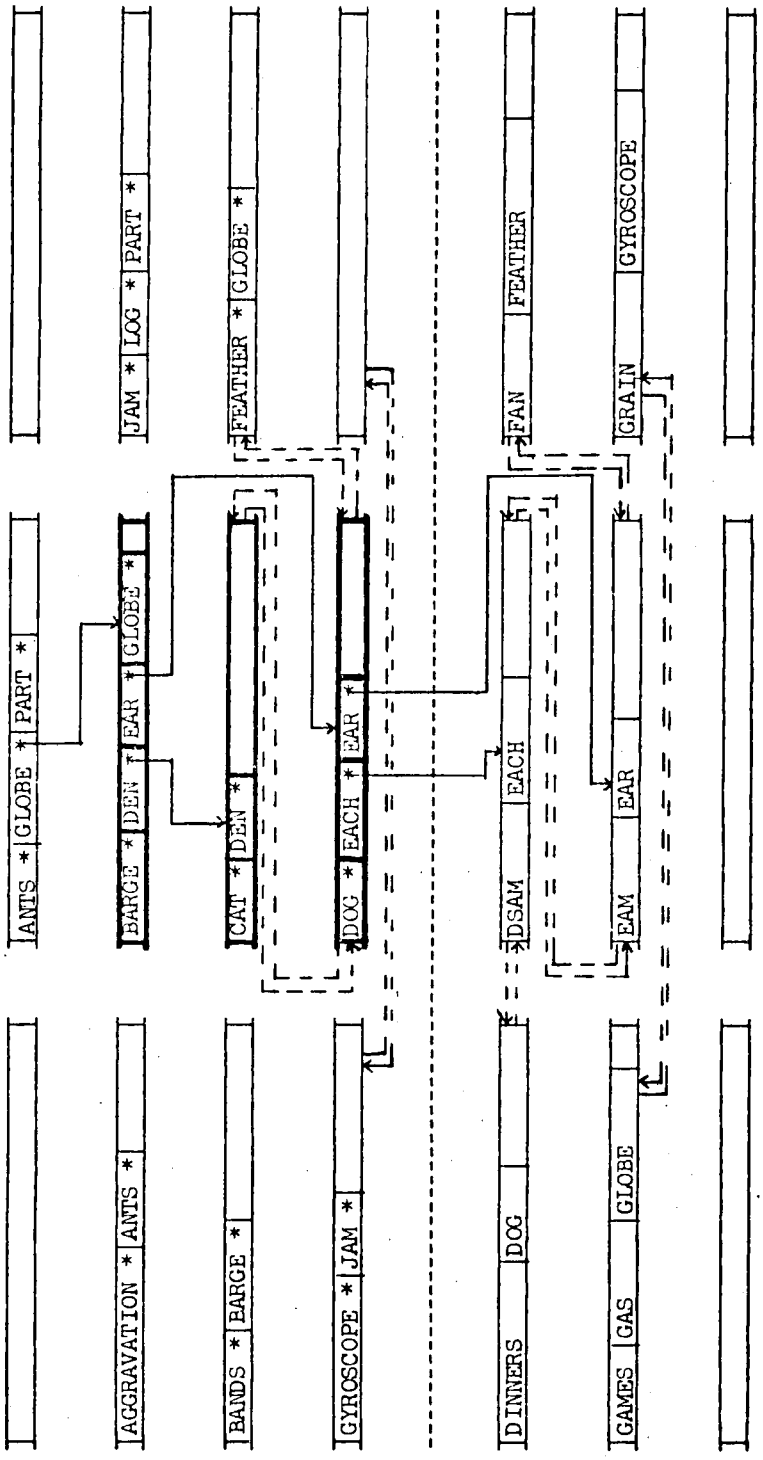


Figure 3.4. Final State of DSAM File After Insertion of New Term: "DSAM." Note that it was necessary to split a Directory Area CINV to accommodate the word "EACH" that was generated as a new key by the splitting of the Data Area CINV. After the split in the Directory CINV, it was necessary to insert "DEN" in the next higher level of the tree, as shown.

transactions would then be merged with the next normal update cycle or run as a separate update cycle.

II. DSAM PHYSICAL FILE STRUCTURES

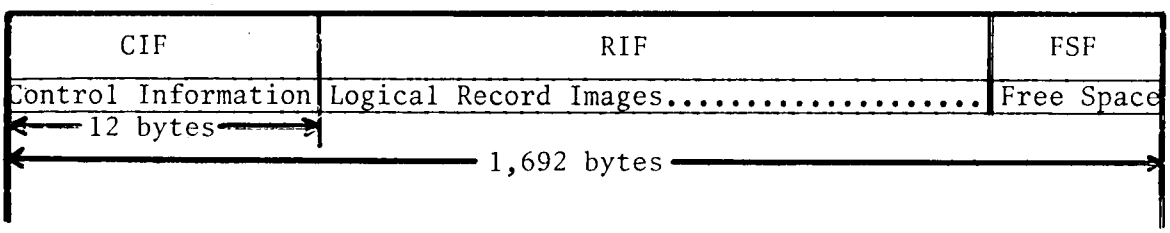
Definitions and Terminology

Up to this point in the report, a DSAM file has been defined only at the conceptual level, in terms of a Directory Area and a Data Area. Each of these areas is composed of control areas containing fixed length physical records or control intervals, which in turn house groups of logical records. The precise definitions of these concepts in terms of their realization under OS/360 as physical datasets on direct access storage devices (DASD) are presented below. In particular, the details of physical record format, control information fields and implementation restrictions are presented.

Logical record. A consecutive sequence of from two to 13,018 bytes of data. A logical record may be of either fixed or varying length. Varying logical records must be preceded by a two-byte long field giving the length of the record, exclusive of these two "length bytes."

Control interval (CINV). A fixed length physical record on DASD whose length is taken from a limited set of allowable values. All control intervals within the Data Area or within the Directory Area of a DSAM file must be of the same length, although the length may differ for the two areas if they are stored as separate OS/360 datasets. A uniform value of 1,692 bytes has been used in the experiments reported here.

CINV format. Every CINV in a DSAM file is of the following format:



Note that a CINV may be "empty," i.e., contain no logical records; "partially full," i.e., contain one to "N" logical records with unused free space grouped at the end of the CINV; or a CINV may be full, i.e., the free space, if any, is insufficient to accommodate a complete additional logical record.

Control information field (CIF). Each CINV begins with a twelve-byte long field reserved for DSAM control information for that CINV. This control information identifies the location of the logical record with the highest key in that CINV, if one exists. It also gives the location of the first byte of any available free space in the CINV. The CIF also contains two pointer fields which give the address of the "prior" and "next" CINV (if any) in terms of the collating sequence of the keys of the logical records they contain. A "prior" or "next" CINV may, of course, be (presently) empty (as indicated by hexadecimal FF in the first character of the CINV).

Record key. Each logical record in a DSAM file must contain an embedded "key," a sequence of from 1 to 254 bytes that uniquely identifies that logical record. The logical records in the directory area of the DSAM file form an index to the data records, using these key values to locate a desired data record by the specification of

that record's key. Keys may assume any value, ranging from X'00' through X'FF' in each byte position of the key.

Key position. The key of a fixed length record may start at any byte within the logical record, but must be a consecutive sequence of bytes. The key of a varying length record, whether it is itself of fixed or varying length, must begin at the first available byte in the logical record, i.e., immediately after the required two record length bytes. If the key is itself of varying length, it must begin with a one-byte length code, followed by the actual key bytes. The length code of the key excludes itself.

Note: Records and keys held in the Directory Area of a DSAM file are always coded in the varying length record, varying length key (VR-VK) format, regardless of how the records in the data area are coded.

Implementation restrictions. To simplify program development, a few restrictions have been placed on the logical records within a DSAM file as presently implemented:

1. A DSAM logical record may not "span" a CINV. Thus, if there is insufficient free space in the current CINV to accommodate the complete logical record, it must be placed in another CINV.
2. Only the VR-VK record format is supported.
3. Both the Directory and Data Areas of the DSAM file must reside in the same OS/360 dataset and have the same CINV size and record format.

4. Sequential extension of an existing DSAM file by the presentation of records with key values higher than the existing high key is supported, but not optimized.

This completes the enumeration of restrictions in the present implementation, and we may now resume the discussion of DSAM itself.

Directory Area CINVs

The CINVs within the Directory Area of a DSAM file are used in a manner somewhat different than those in the Data Area, in that their contents constitute an index to the logical records in the Data Area. Since this is the case, certain special conventions apply to the records in these CINVs, as well as to their relationships to each other and to the logical records in the Data Area:

1. All records in the Directory Area are coded in the VR-VK format.
2. The "data" portion of each record is always three bytes long, and is the relative block address (RBA) of (a) the Data Area CINV having this key as its highest key; or (b) another CINV in the Directory Area, at a lower level of indexing, which has this key as the highest key in that CINV.
3. All records within a given Directory Area CINV must be at the same indexing level, i.e., they must all address Data CINVs, or all address Directory CINVs at the same level in the index tree.

4. Under the restrictions given in (3), the DSAM Directory forms a constant path length index to the DSAM file, i.e., a balanced tree (30, 31). By this it is meant that any path through the directory in search of a given key must always access all levels of the tree.
5. A special "marker" key value is used to insure that no search key presented to the Directory Search Algorithm will ever be found to be "higher" than the highest key in the directory. This key has as its value 255 bytes of hexadecimal 'FF'.

Data Area CINVs

As is also true of the Directory Area CINVs, each CINV in the Data Area is an element of a doubly linked list of CINVs, formed by use of the PRBA ("Prior" Relative Block Address) and NRBA ("Next" Relative Block Address) pointers in each CINV. The list is formed during initial loading of the file, and includes, in ascending sequence of absolute RBA value, any empty CINVs required by the desired distribution of free space. Following updating of the original file, this list of CINVs may not continue to correspond to a monotonically increasing set of DASD record addresses. This is because the insertion of logical records may have resulted in a CINV split, whereby an existing CINV is fragmented into two parts, using an available empty CINV. In the splitting process, the order of CINVs in the doubly linked list is altered to reflect the new "logical" order of CINVs, rather than their "physical" order. Thus, any attempt to read the

Data Area in a purely "sequential" manner must utilize the list pointers, and not the increasing (or decreasing) sequence of RBA (Relative Block Address) values. Such an operation must also ignore any empty CINVs encountered during the scan, and detect the "header" or "trailer" CINVs that are described below.

Header/Trailer CINVs

Both the Directory and Data Areas begin with a special CINV called the "header" CINV. This CINV contains identifying information for the file and space for the storage of file updating and access statistics. Both areas (Directory and Data) are terminated by a trailer CINV, whose contents and format are identical to that of the header CINV, except for a flag bit used to differentiate header and trailer CINVs.

The information in all four of these special CINVs is altered during the successful closing of a DSAM file following an update, and any discrepancies between corresponding contents of header and trailer CINVs is an indication of an error during an update attempt. The sequence in which these CINVs are written during the close process is as follows:

1. Data Area Header CINV;
2. Directory Area Trailer CINV;
3. Data Area Trailer CINV; and
4. Directory Area Header CINV.

Thus, if discrepancies are noted during an attempt to OPEN the file for a subsequent update, it should be possible to determine the stage at which the prior update failed.

III. SOFTWARE SYSTEM CHARACTERISTICS

System Level Functional Specifications

The design, implementation and subsequent understanding of a complex software system such as DSAM is greatly simplified through the use of functional specifications. Such specifications define system capabilities, with minimal reference as to how those capabilities are to be supported. The statements below constitute the complete specifications for DSAM at the highest level of functional abstraction. These specifications are a refinement of the DSAM goals stated in Chapter II (page 35).

1. General. DSAM shall permit creating, updating, and maintaining collections of logical records, termed "files," on any suitable direct access storage device (DASD). Each record in these files has a unique (fixed or varying length) sequence of characters termed the "record key," or simply, "key," which differentiates that record from any other in the file.

2. Transferability. DSAM shall provide capabilities for the creation and use of an unloaded form of a DSAM file, stored on a sequential medium. This file may be used to recreate the original DSAM file on another DASD device, or used to support the transfer of a DSAM file from one computer system to another.

3. Device Independence. It shall be possible to alter the physical device type upon which a DSAM file is stored. The physical characteristics of the receiving device must be such that the largest

physical record stored on the source device may be accommodated on the receiving device. When this is true, a copying operation may be performed without altering the internal format of the data or invalidating any auxiliary indexing information which may have been created to assist in accessing the data stored in that file. This implies that all internal addressing information consist of relative block addresses (RBAs) as opposed to device dependent values such as relative track addresses or actual device (hardware) addresses.

4. Multiple Files. Under DSAM, a single calling program, termed the "user," may simultaneously process more than one DSAM file, without requiring multiple copies of the DSAM software system to be present in memory. This restriction need not be observed for any stand-alone DSAM utility programs such as those needed to create, unload or reload a DSAM file.

5. Simultaneous Use of a Single File. DSAM shall at all times, except during file (re)creation from a sequential source, permit the simultaneous use of the existing file by more than one program (or independent task if the program is multitasking). However, DSAM is not required to support more than one simultaneous program use for file updating. During an updating function, DSAM may temporarily prevent retrieval access to only those data records (and index records) actually being altered. It may not prevent (or unduly delay) retrieval of records not affected by the updating activity.

6. Data Integrity. Any error during an update (excluding hardware errors resulting in incorrect transmission of valid data)

shall not cause catastrophic error in a simultaneous or subsequent retrieval-only process. Furthermore, following such an error or abort in an update process: (a) data loss to ongoing retrieval programs shall be minimized; and (b) adequate methods shall exist to recover from the error without terminating ongoing retrieval operations.

7. Sequential Access. DSAM shall allow the user to retrieve all of the logical records in the file in ascending or descending collating sequence of their recorded keys. For this mode of access it shall not be necessary for the user to specify the key of each record to be retrieved.

8. Skip Sequential Access. DSAM shall allow sequential access (as defined in 7 above) to begin or to resume starting with any record in the file as well as with the lowest and highest keyed records. The origin of sequential retrieval is to be established or re-established by using "indexed access" to select the desired key at which to start. For skip sequential access, the specified key need not exactly match any recorded key, but may match a (high-order) fragment of a recorded key. In this case, retrieval is to commence with the first record having a key higher-than or equal to the specified key, regardless of whether ascending or descending order is specified for subsequent sequential access.

9. Indexed Access. DSAM shall maintain internal directory and indexing information within the file so as to permit retrieval, deletion, update (replacement) or insertion of new records in any desired order of specified key, without resort to reading all previous

or subsequent logical records. In this mode of access the following restrictions shall apply:

a. For deletion or updating by replacement of an existing record the user-specified key must exactly match the key recorded in the (desired) record or the transaction shall be rejected.

b. For insertion of a (supposedly) new logical record, the record shall not be inserted if it has a key which matches any existing key in the file.

c. For retrieval only, the first record in the file which has a key higher than or equal to the specified key shall be retrieved. DSAM shall indicate to the user whether the key matches exactly or not.

d. Record insertions and/or deletions shall not prevent sequential access in collating order, including retrieval of the inserted records. The file shall always appear to be in increasing order for all record keys.

10. Auxiliary Keys and Direct Access. During creation of or access to DSAM files, the user shall have the option of obtaining and recording the RBA at which a record was written or read. Subsequently, a user shall be permitted to retrieve a DSAM record by specifying this type of address rather than by use of the DSAM index structure.

Note: When using direct access, the user assumes total responsibility for the validity of the addressing information and its currency with respect to any DSAM updating activities that might have

occurred since the user's auxiliary index structure was last used. DSAM is not required to perform any checking on direct access requests. Direct access is to be supported for retrieval only; i.e., DSAM will not permit a user to write records using user-specified addresses other than through the index structure.

Level of Implementation, A Pseudo-Access Method

DSAM is not, in the terminology of OS/360, a true "Access Method" (15, 18, 25). Rather, DSAM consists of a series of PL/1 subroutines which perform services similar to those provided by the vendor-supplied access methods. These services are provided by means of PL/1 invocation of the BDAM (Basic Direct Access Method) routines supplied by the vendor (15, 18, 25). Thus DSAM is a pseudo-access method, in that it provides a logical interface to the user similar to any other access method, but achieves its results by means of invocation of a true access method, rather than by generating channel programs and interfacing to the OS/360 supervisor routines directly.

Implementation Choices, Language, Reentrancy

Once the basic decision to implement DSAM as a pseudo-access method was made, a number of other fundamental choices also had to be made before design development could progress. For the author, the next choice was the selection of PL/1 as an implementation language. DSAM was implemented using only the language supported under the IBM program Product, PL/1 Optimizing Compiler (26, 27) (which will be

denoted PL/1(X)). The choice of PL/1(X) was motivated by the author's familiarity with the language capabilities and its suitability for top-down program development.

Were DSAM to be given wide distribution for use in more general applications or for use where machine-level efficiency considerations were paramount, DSAM should be recoded at a level similar to the other true access methods, including the tailoring of channel programs and the optimization of the interface to the system routines. Coding of DSAM in a high level language has provided benefits such as speed of implementation, ease of debugging, and simplification of maintenance.

Use of the "Optimizing Compiler" was in turn dictated by the requirements for efficiency in execution and storage space management. In addition, several features of PL/1(X) were not supported under the earlier PL/1 compiler, or were too cumbersome and inefficient for extensive use. Among these PL/1(X) features, support of dynamically loaded subprograms, i.e., FETCH and RELEASE PL/1(X) statements, was most critical to the design of DSAM.

As in many large-scale software systems, conservation of memory space and maintenance considerations for DSAM suggested the use of the FETCH and RELEASE verbs to permit routines that are needed infrequently, such as OPEN and CLOSE functions for files, to be loaded into memory only when needed. PL/1(X) provides support for this concept. Nevertheless, there are still a number of implementation restrictions on the full use of FETCH and RELEASE as defined in the language reference manual (26). The most prominent of these

restrictions in terms of impact on DSAM design, is the restriction that a "FETCHed" subprogram may not itself "FETCH" any other subprograms.

In the design of DSAM, it would have been highly desirable to parallel the use of dynamically loaded (FETCHed) modules within OS/360 system software. Thus, a call to a high-level DSAM module would FETCH it into memory, then that routine would determine which other modules it required, and proceed to FETCH and execute them. Unfortunately, the current PL/1(X) compiler does not support this level of language. Thus, it was necessary to alter the design such that the highest level module of the DSAM routines, DSAM itself, must be linkage-edited with the caller's program and cannot be FETCHed. Furthermore, this one routine must itself handle all FETCH and RELEASE actions for any of the other modules which are themselves dynamically loaded into memory.

In view of the need to incorporate DSAM itself and any of its STATIC data areas into each calling program, every effort was made to minimize the storage requirements and code complexity of this module. In particular, almost all storage areas, control information, I/O buffers and the like are themselves dynamically allocated within some FETCHed module, and their addresses are then passed back to the calling routine, rather than defining these areas as STATIC variables within DSAM itself. Note, however, that once allocated and initialized, these areas are not released and subsequently recreated on each entry to DSAM, since this would have severe impact on execution efficiency. Rather, the control structures are allowed to remain in existence as long as they are needed, with their addresses being passed from module to module.

This handling of control information and buffers also simplifies generation of reentrant code for all the DSAM modules. The decision to maintain reentrancy was motivated by two major considerations: (1) update processes using DSAM are expected to operate simultaneously with retrieval-only access from RECON; and (2) eventual recoding in OS/360 assembler language as a true access method would require reentrancy.

IV. DSAM SOFTWARE DESIGN

DSAM Software Modularity

DSAM exhibits modular structure, with each module or group of modules performing specific, narrow tasks. These modules are organized into hierarchical groups as shown in Figure 3.5. Almost all the modules are dynamically loaded into memory only when needed and then discarded unless it is obvious that they will be needed again in the same execution. Thus, modules that open a DSAM file are loaded (FETCHed), executed, and discarded (RELEASEd), whereas modules which perform a READ, and are expected to have heavy use, are loaded, executed, and left in memory until the last possible moment they may be needed, i.e., until after a CLOSE of the file.

Overview of Software Modules and Functions

The numerous independent load modules of DSAM may be grouped into four functional levels as shown in Figure 3.6. These levels reflect the functional breakdown of DSAM as dictated by the constraints of the

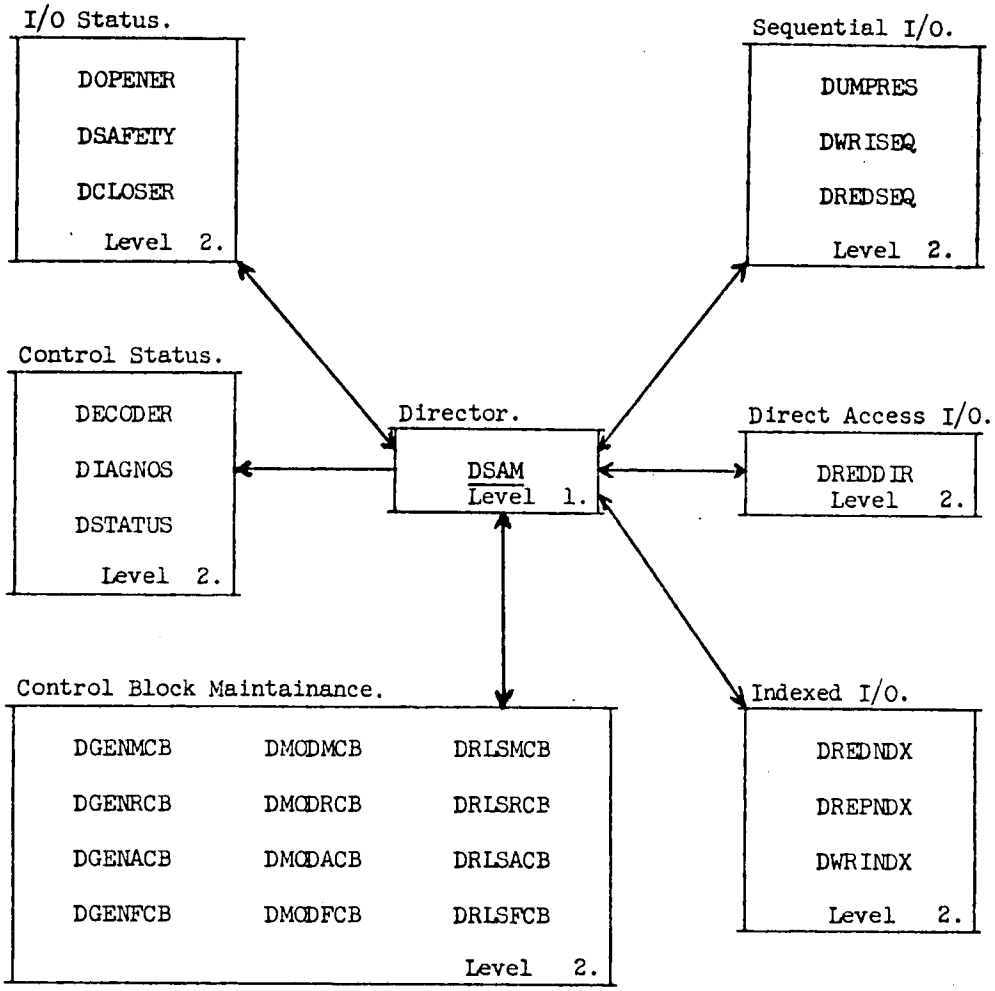


Figure 3.5. Functional Grouping of DSAM Modules at Level 1 and 2. There is no direct correspondence between the grouping indicated by the boxes in this illustration and physical load module structure.

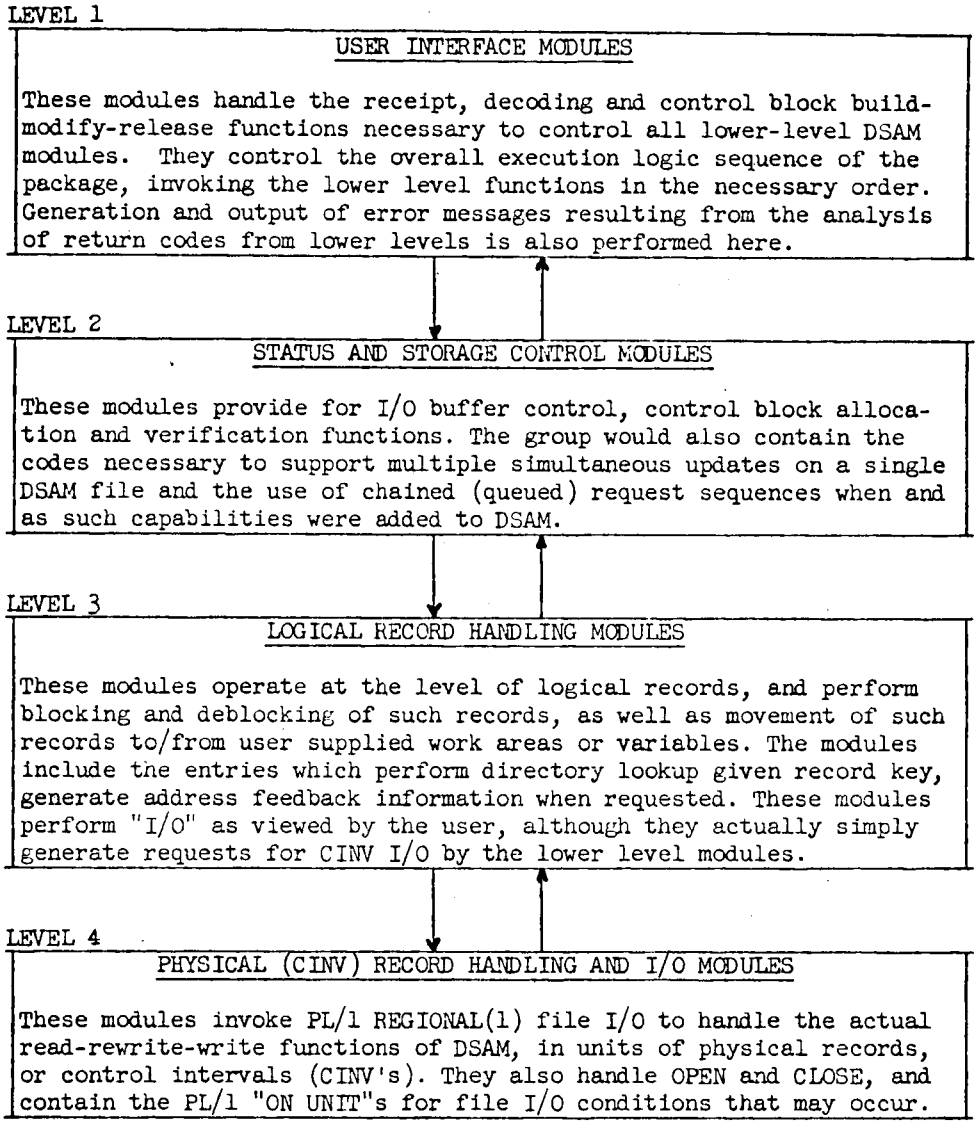


Figure 3.6. Hierarchical Levels of DSAM Modules. Module names for each level, and details of calling relationships are given in the DSAM software documentation itself.

language chosen for implementation and the logical hierarchy of microfunctions that must be performed to realize those ten requirements.

At level 1 the interface between DSAM and the user is defined and supported. At this level all requests for DSAM services are received, decoded and checked for validity. Legitimate requests are translated into appropriate values to be stored in the control blocks used by the lower levels. Invalid requests such as a request for "direct access write" would be rejected at this level. The code at this level is also responsible for overall control of the sequence of use of the other modules, and handles the FETCHing and RELEASE of the other modules.

Where creation of control blocks or buffers is needed before level 1 can complete its functions, level 2 modules are invoked to perform these functions. The modules in the level 1 group also insure that the files are properly closed in the event of disastrous errors that have not been handled by lower level routines.

The level 2 modules of DSAM control storage (computer memory) and verify the validity of the contents of control blocks. These modules also perform I/O (Input/Output) buffer management as needed for use with the lower level routines. After checking for appropriate status of control information, the modules at level 2 pass control on to the logical record handlers at level 3.

Level 3 modules perform all of the logical record handling activity as defined by the user, making use of information stored in control blocks by the higher levels. The modules at level 3 handle the

decoding (lookup) of keys in the directory and all blocking and deblocking of logical records within a physical block of DASD. If necessary, these modules move logical records between the I/O buffers and the user-supplied work areas or variables. These modules may, however, simplify return pointer information for the user to locate the record image within the existing buffer. Whenever handling record images requires the input or output of a physical block, the level 4 routines are called.

Level 4 modules perform the reading, writing or the rewriting of physical blocks (given the relative block address) and the opening or closing of DSAM files. In fact, since in PL/1 a file may be opened implicitly during the first READ or WRITE statement, any of these routines must carry along the PL/1 overhead code required for an OPEN, even though all DSAM files are always subjected to an explicit OPEN request and the default code is never executed. To minimize the potential overhead that is incurred if each module is stored separately, a single module (DOIOCIV) contains multiple entry points for the READ, (DREDCIV), REWRITE (DREPCIV) and WRITE (DWRICIV) functions, and thus only one copy of the overhead code is stored.

Module Interface Specifications

Because FETCHable DSAM load modules are separately compiled and stored in the DSAM run-time library, interaction between variables in separate modules requires the use of explicit calling sequence parameters. These modules cannot make use of "global" data that is

known to them through its mere declaration in a containing module, as the modules are not nested within one another.

STATIC EXTERNAL variables are likewise prohibited for FETCHable modules and hence the PL/1 equivalent of FORTRAN's COMMON is not used for transmission of data across the intermodule linkage.

Unfortunately, in a complex software structure such as DSAM, the number of unique variables or pieces of control data that must pass from routine to routine tends to be rather large. This would lead to long and error-prone calling lists, were it not for the fact that (pointer) based structures may be used to group such data into control blocks, and only the pointer upon which the structure is based (addressed) needs to be placed in the parameter list of a call. Thus, in DSAM intermodule interfaces, the calling parameter list contains a few simple variables and a list of pointers to the needed control blocks.

There is only one rigid requirement for calling sequences within DSAM. Every module must have a parameter list, the last element of which is a variable to house a return code from that invocation. The conventional OS/360 values of 00, 04, 08, 12 and 16 are used to indicate status as being (00) error free; (04) warning, execution successful but minor errors or exceptional conditions occurred; (08) errors occurred but execution may still have been successful; (12) severe errors occurred, extensive error recovery would be needed, or the function should be aborted; (16) terminal errors occurred, the

execution of DSAM should be terminated as soon as possible with all attempts to preserve file integrity being invoked.

Treatment of Error and Exceptional Conditions

In DSAM, an error that occurs in a lower level module is treated as follows: (1) If "fixup" action can be taken with reasonable expectation of success, then the fixup is handled at this point and the return code for the module set to 04 to indicate the occurrence of the error(s). (2) If the error is such that a "fixup" may or may not be successful, the module attempts fixup, but only after setting the return code to 08. It is assumed that the invoking module will check on the results of the fixup attempt and take further action. Where no reasonable fixup can be taken, the return code is set to 12 and the module returns to the caller, who is expected to "gracefully terminate" the process that was attempted. In no case does a higher level module attempt a fixup for a level 12 error. Instead, the calling module is expected to set the code up for a clean shutdown of DSAM and pass the 12 code back up to its caller in the chain.

If a return code of 16 is issued by an invoked module, there is no recourse but the immediate shutdown of DSAM. Thus, a calling module receiving a 16 return code does not even attempt to shut down gracefully, but immediately passes control back up to its caller, until the highest level module is reached. When the highest level module is reached, DSAM will attempt to close all files, and return to the user with code 16 also. If possible, DSAM will set internal flags before

returning that (a) indicate the nature of the fatal error, (b) prevent reinitialization of DSAM for this file, and (c) ensure that an error message has been output if at all possible.

DSAM treats "exceptional" conditions, such as end-of-file on a sequential read attempt, in a manner analogous to "errors." The parameter list of modules that may raise such conditions will have provision for status information to be returned to the caller in addition to the simple return code value, so that these conditions may be distinguished from true errors.

Functional Specifications, By Module

A detailed functional specification for each module within DSAM, and detailed notations concerning its actions in all normal and error conditions is beyond the scope of this report, but will be included with the detailed software documentation in a separate publication.

This chapter completes the presentation of the design of DSAM to the level pertinent to an overall understanding of the concepts involved. In Chapter IV, the performance of the DSAM software system in its embryonic stages of development is explored, and experimental results are presented.

CHAPTER IV

PERFORMANCE TESTING AND EXPERIMENTS WITH DSAM

I. CRITIQUE OF DSAM PERFORMANCE

Storage Space Reduction

The first major objective of the DSAM design was to reduce the DASD storage space requirements for the RECON inverted index vocabulary files through support of variable length keys in the data records. Although it was obvious that some gains might be made in this manner, it was necessary to obtain reliable numerical estimates of the potential reduction before beginning the design and implementation of DSAM.

A special purpose PL/1(X) program was therefore written to scan the existing ISAM files and obtain initial estimates of the DASD space reduction that could be achieved by DSAM. As high accuracy was not a requirement for this intial estimate, several simplifying assumptions were made in this program, all of which resulted in errors in the direction of underestimating the potential advantages of DSAM. The algorithm used by the space reduction estimating programs is shown below:

Let N be the number of logical records in the ISAM file.

L be the constant length of these ISAM logical records.

L_k be the length of the key field, including the prefix such as IT=, but excluding trailing blanks.

L_d be the constant length of the data portion of each record, whether ISAM or DSAM.

L_o be the constant length of the DSAM length code fields that are needed to specify record and key length.

Then:

$$\text{ISAM Space, } S_I \cong L * N$$

$$\text{DSAM Space, } S_D \cong \sum_{i=1}^{N} (L_o + L_{k_i} + L_d)$$

And:

$$\text{Percentage Reduction, } P \cong \frac{S_I - S_D}{S_I} * 100$$

The data presented in Table I were derived from the execution of the estimating program against most of the RECON inverted index files. The estimates are carried separately for the "author" (AU=) and nonauthor (IT=, SU=, etc.) portions of these files, since in RECON these parts of the inverted index are stored as separate datasets on different devices.

The estimates given in Table I are highly conservative, in that they neglect the following factors which favor an even greater actual space reduction for DSAM:

1. The ISAM Index and DSAM Directory Area storage requirements are assumed to be equal and very much smaller than the data storage requirements.

TABLE I

ESTIMATED DSAM SPACE REDUCTION VS ISAM

File	Nonauthor		Author	
	Total Terms	% Decrease	Total Terms	% Decrease
NSA	60,452	34.18	334,239	32.77
EIX	87,472	22.62	250,413	35.61
ESI	46,433	42.66	35,235	36.20
MEX	114,708	37.41	81,064	35.06
WRA	128,496	28.75	77,448	33.50
EDB	64,547	16.78	55,364	33.88
NSC	8,417	42.97	40,985	40.16
NSR	3,474	45.92	16,550	35.24
ERD	7,238	19.19	8,062	36.21
HEF	6,655	41.18	5,104	33.60
ENG	3,506	23.18	3,567	34.42
NRC	4,992	32.16	-----	-----

2. ISAM requires a redundant copy of the highest record key in each physical block of data to be recorded in front of that block on the DASD device as a separate physical record. No corresponding requirement exists for DSAM. The extra physical record in ISAM also causes increased DASD overhead due to the required gaps that must precede and follow this separate physical record.

3. The estimate makes no allowance for nonoptimal choice of physical record size. For some ISAM files that were subsequently loaded to DSAM format, this factor was significant. It was observed that a poor original choice of block length for the ISAM file, coupled with a much better DSAM block length, inflated the observed DSAM space reduction by as much as 8 percent.

In addition to the very encouraging amounts of space reduction shown in Table I, two further interesting observations were made.

First, it was observed that long words in the scientific vocabulary and the presence of phrase keywords results in the average term length being longer for the nonauthor part of each index than for the corresponding author entries. This gives rise to a relatively greater estimated DSAM space reduction for the author files than for the nonauthor files.

Second, the variation in term length within the nonauthor part of Table I is much greater than in the author part of Table I. In particular, the nonauthor files for ERD and EDB represent anomalous values that require comment. From an examination of typical ERD entries, using RECON, we found that a large percentage of the index

terms consist of fully spelled-out names of corporations, U. S. government agencies and university names, including departmental identification. Clearly for ERD, the lack of abbreviation in these entries accounts for the high average term length.

For the EDB file, however, the cause of the high average term length is quite different from that in the ERD file. In EDB, extensive use has been made of precoordination of index terms. This artificial phrase generation process has resulted in the highly skewed term length distribution shown in Figure 4.1. The EDB histogram shows a clearly bimodal distribution, which can be directly ascribed to the artificial phrases generated by precoordination. In contrast, Figure 4.2 shows the corresponding term length histogram for the NSA file, where no such precoordination exists, and a highly abbreviated vocabulary is in use.

Two very important conclusions for the designer of an information system may be drawn from the data on the EDB term length distribution. First, precoordination of index terms results not only in a proliferation of unique terms, with its consequent impact on file size, but also increases the average term length for the collection, with impact on any scheme of variable length keys. Second, although a fixed length field for text may be of adequate or even excessive length for single terms or most human generated phrases, automated term pairing processes give rise to phrases that frequently exceed reasonable field lengths. Truncating phrases at some point such as RECON's 39 character limit often destroys the understandability of the phrase. For example, a

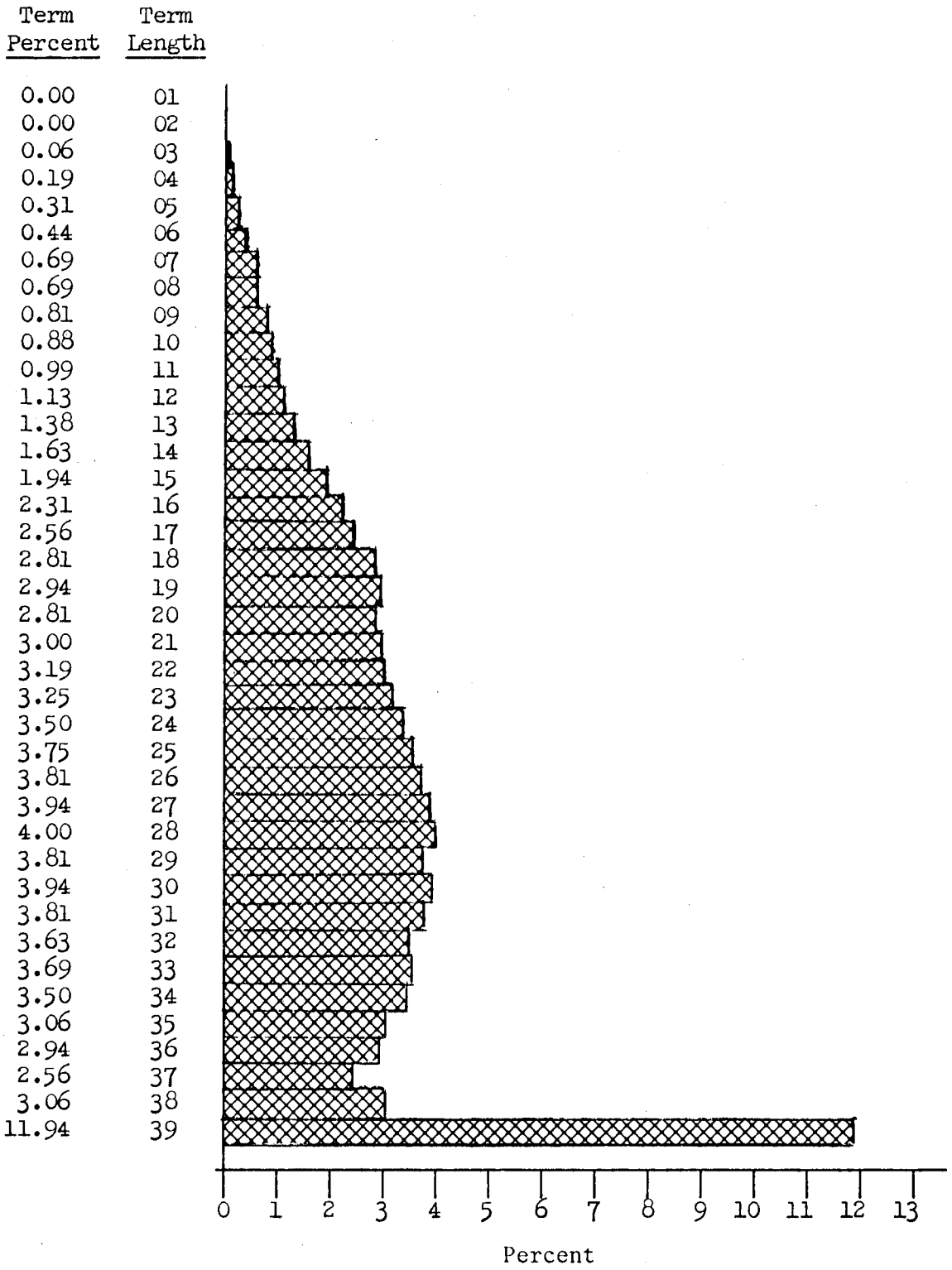


Figure 4.1. Histogram of Index Term Length from EDB. Note: Length excludes 3 character "IT=" prefix.

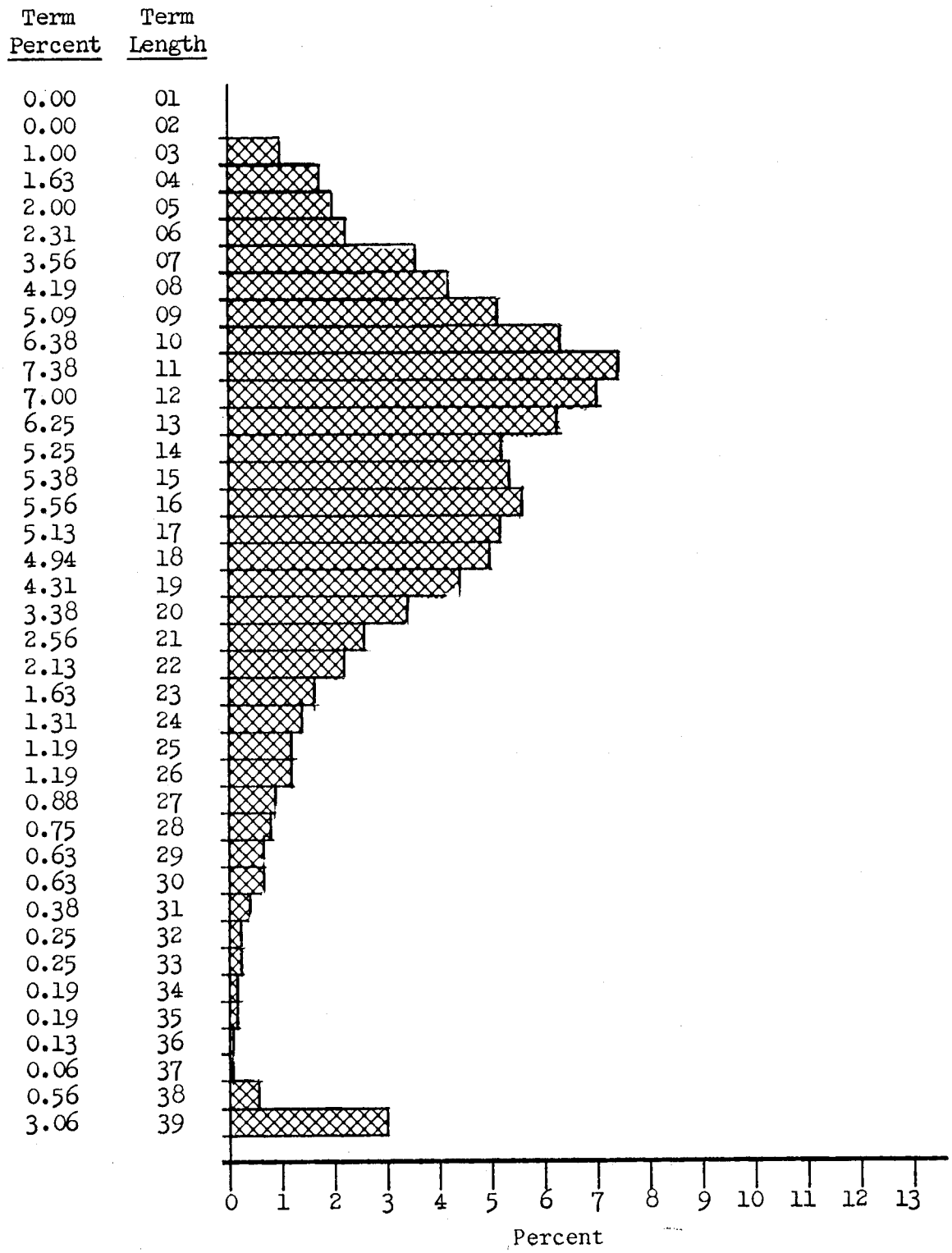


Figure 4.2. Histogram of Index Term Length from NSA. Note: Length excludes 3 character "IT=" prefix.

page from a RECON EXPAND command in the EDB file is shown in Figure 4.3. It is obvious that the user will have difficulty in determining the content of the truncated part of most of the phrases shown in this figure.

In contrast to the estimated space reductions shown in Table I, on page 76, the data in Table II reflects observed compression ratios obtained with actual DSAM files. In all cases, the DSAM file was created with approximately 30 percent free space reserved for insertion updating. An additional column in Table II labeled "No Free Space" has been provided showing the calculated compression factor if the DSAM file had been created without the indicated 30 percent free space allowance. In RECON, the ISAM versions of these files are presently loaded without any provision of space for updating. Thus, the data in Table II under the heading "No Free Space" should be compared to the estimates in Table I (p. 76).

As mentioned, the Table I (p. 76) estimates are smaller than the observed DSAM compression ratios due to the differences mentioned under the discussion of the estimating procedure. In particular, attention is called to the NSA file comparison, where the influence of a very poor choice of ISAM blocksize markedly inflated the estimated DSAM space reduction. The bias due to the blocksize differential was in the neighborhood of 8 percent in this case.

Clearly, the DSAM support of variable length keys has been shown to be a major achievement in terms of DASD storage space reduction for

```
EXPAND  IT=ENVIRONMENTAL IMPACT STATEM
 REF  DESCRIPTOR                CIT   RT
 E01  IT=ENVIRONMENTAL
       IMPACT STATEMENTS_       125    4
 E02  IT=ENVIRONMENTAL IM
      PACT STATEMENTS/BIB         1
 E03  IT=ENVIRONMENTAL IM
      PACT STATEMENTS/DAT         1
 E04  IT=ENVIRONMENTAL IM
      PACT STATEMENTS/LEG         1
 E05  IT=ENVIRONMENTAL IM
      PACT STATEMENTS/MEE         1
-E06  IT=ENVIRONMENTAL IM
      PACT STATEMENTS/MET
 E07  IT=ENVIRONMENTAL
        PROTECTION AGENCY_       142    2
 E08  IT=ENVIRONMENTAL PR
      OTECTION AGENCY/AGR         1
 E09  IT=ENVIRONMENTAL PR
      OTECTION AGENCY/BUD         1
 E10  IT=ENVIRONMENTAL PR
      OTECTION AGENCY/GOV         1
                                -MORE-
ENTER:
```

Figure 4.3. Illustration of the Effects of Artificial Phrase Generation Within the ERDA/Energy Database File on RECON. Note that it is difficult to determine the nature of the phrase which has been appended to the main terms due to truncation.

TABLE II

DSAM SPACE REDUCTION EXAMPLES

RECON[6] File System Name	Number of Index Terms	ISAM[4] Space Used (Trk.)	DSAM[5] Space Used (Trk.)	DSAM Space Reduction (Trk.)	DSAM Percent Reduction (As Loaded)	DSAM Calculated Percent Space Reduction (No Free Space)
NSA[1]	60,452	760	574	186	24.5	47.1
EDB[1]	64,547	880	880	0	0.0	30.0
HEF	6,655	100	75	25	25.0	47.0
ENG	3,506	80	78	2	2.5	31.2
NSR	3,474	60	47	13	21.0	45.8[2]
TEST[3]	11,991	160	122	38	23.7	46.2

Notes:

1. Both NSA and EDB ISAM files used nonoptimal blocksize values, whereas DSAM CINV size was near optimal. This inflates the DSAM space reduction figures by approximately 6 percent over what would have been observed if the ISAM blocksize had been optimal, i.e., one record per track.

2 This figure is smaller (by 0.1 percent) than the estimate given in Table I. The DSAM file reserved 80 CINV's for its directory, whereas it needed less than 10 CINV's. Thus there was a relatively large influence on the result, due to the small file size.

3 This file contains the same data as used in the file retrieval and updating experiments. For the purposes of this test only, the ISAM file was created without free space in a direct load, using all 11,991 terms, rather than being the result of ISAM update activities.

4 ISAM space used is rounded to the next highest cylinder, as ISAM files must be allocated in cylinder units. Files were compressed to minimal cylinder boundaries after initial loading to a larger area for safety.

5 DSAM space includes only actual data and directory tracks plus 8 tracks in last CAREA as free space, i.e., last CAREA was not treated as being full.

6 All files are the nonauthor index portion only.

the RECON inverted index vocabulary files. Thus, the design meets and exceeds the goals set for it in this important area.

Estimated Retrieval Efficiency

To be useful for the RECON application, DSAM must not only provide the levels of DASD storage space reduction reported here, but must also provide retrieval performance equal or superior to that of ISAM.

The results presented in Table III indicate that the retrieval-only performance of DSAM is indeed superior to ISAM, with the degree of its superiority dependent upon the amount of "clustering" of the input terms within the alphabet. The data are all the more interesting since the superiority was demonstrated with the PL/1(X) version of DSAM rather than with a carefully tuned 360 Assembler Language version such as will be incorporated into RECON itself.

The following procedure was used in performing the timing evaluations presented in Table III.

1. A master vocabulary tape, in ascending sorted order of its 11,991 unique terms, was used to create a DSAM and an ISAM file each containing some 8,040 of these terms selected in an essentially random manner from the 11,991 terms in the source. The origin and content of this test file is detailed later under "File Updating."

2. The master vocabulary tape was then subjected to two separate randomization procedures, consisting of sorting the terms on nonadjacent columns, excluding the first column of the file's keyword field. This process gave rise to two other files, each of which contained all 11,991 terms, but in "scrambled" sort order.

TABLE III

ISAM VS DSAM: PERFORMANCE DATA - RETRIEVAL ONLY[1]

Sorted Order of Transactions	Run	ISAM Seconds	ISAM Retrievals/Sec.	DSAM Seconds	DSAM Retrievals/Sec.
Ascending	1*	996	12.0	111	108.2
	2	720	16.6	32	374.7
	3	707	17.0	30	399.7
Randomized-1[a]	1*	1204	10.0	1416	8.5
	2	866	13.8	704	17.0
	3	868	13.8	660	18.2
Randomized-2[b]	1*	1010	11.9	1092	11.0
	2	845	14.2	662	18.1
	3	838	14.3	630	19.0

*Data from this run is included only to illustrate the difficulties to be encountered in attempting performance testing in a multiprogramming environment. Subsequent investigation revealed that the ISAM and DSAM runs had been run in parallel, using the same source files for an unknown period of time, after which the ISAM run was aborted by the operator and restarted at a later time. Since no printout had been produced by the aborted job, this fact could not have been noticed from the time and date stamps on the final output of the ISAM job, as these reflected the second execution of that run, which was not in any time overlap with the DSAM run.

[a]This file was formed from the Ascending file by sorting on nonadjacent multicharacter fields, excluding the first character of the term. The sorting alternated between ascending and descending sequence over four such fields.

[b]This file was also formed by sorting the Ascending file, but four nonadjacent single character fields were used. Again the sorting alternated between ascending and descending order over the four fields.

[1]The master file used contained 11,991 terms. A random sample of 8,040 of these were loaded into ISAM and DSAM files, and then processed using the entire master file (sorted in three ways) as the "retrieval" file.

3. The master sequential file, and each of the randomized copies of it, were then used as input to retrieval-only programs that operated on the DSAM or ISAM master files. These two programs were coded in PL/1, with one program simply using the PL/1 ISAM interface for retrieval, while the other invoked DSAM for the same services.

4. Both programs generated time and date stamps each time the files were opened and closed, and counted the total number of: (a) input records; (b) records located exactly by their keys; and (c) records found not to exist in the 8,040 term subset from the master ISAM (or DSAM) file.

5. The number of input records (11,991) was then divided by total elapsed time to give a "transactions per second" value.

As with any form of benchmark testing, the data presented in Table III must be viewed with caution. This is due both to the difficulty of relating the artificial conditions of the benchmark to the real-world environment, and also to the potential impact of the multiprogramming system within which the benchmarks were run. Nevertheless, the high degree of repeatability evidenced by run 2 and 3 of the tabular data lends credence to the benchmark's validity. In view of the almost disastrous multiprogramming interference evidenced by the situation that invalidated run 1, the author made certain that the second and third runs were made during a weekend slack period, under his personal supervision. Thus, the impact of other jobs was minimized during the execution of the timing comparisons for runs two and three.

In scanning the data in Table III (p. 85), one should note that DSAM's performance with the transaction file in perfect ascending order is more than twenty times faster than ISAM, whereas the DSAM results are only slightly better than ISAM for the two randomized input files. This tremendous difference in performance for the case where the input order exactly matches the DSAM file sequence is explained in the following paragraphs.

Although DSAM may be forced to utilize only a single memory buffer to process requests, the version tested here uses three buffers. The first buffer holds the highest level Directory CINV in memory at all times. The second buffer holds an image of the most recently used sequence set CINV, while the third buffer holds the Data CINV most recently read.

With this configuration of buffers, any new pseudorandom retrieval request is first checked against the current data buffer, then against the sequence set CINV buffer, and only if both of these tests fail does DSAM perform a full directory search, starting with the highest level index CINV buffer image. If the input transactions are highly clustered, or are in perfect ascending key order, DSAM will issue an absolute minimum number of physical record I/O requests. Most of the input requests will be processed directly within the buffer images rather than requiring actual I/O operations.

In contrast, ISAM normally maintains only the highest level index in a memory buffer, corresponding to the first buffer mentioned for DSAM. Therefore, every request for a new logical record will result in

physical I/O requests, even if the data item is already within the last buffer image read into memory. This difference in methodology accounts for the exceedingly high performance improvement for DSAM when all input terms are in strict ascending order.

The highly artificial case of perfect ascending order for input terms is not unrealistic because there are numerous cases within RECON where a large number of terms that are clustered in the alphabet are input to the term lookup algorithm in ascending order. Furthermore, in updating the RECON files, it is necessary to sort the new terms to collect all distinct document numbers that have the same descriptor, preparatory to forming the inverted file entries. Thereafter, these new terms, in ascending order, must be posted to the DSAM file. Neither ISAM nor DSAM allow insertion to be performed when a file is being processed in a "sequential access" mode. Thus, updating is performed in the "random access" mode even though the transaction file is sorted. The benefits of the DSAM buffering scheme for file updating then become immediately applicable.

II. FILE UPDATING

General Considerations

Although an update of a RECON IX file does not require the level of performance necessary for the on-line retrieval system, it may not be so poor as to require excessive personnel or computer utilization. If such a process requires excessive wall-clock time, then the probability of a system crash during the update is also increased.

Despite the fact that the input to an IX file update would already be in perfect sort order, the full "random access" capabilities of the access method must be employed, as pointed out previously. Accordingly, in the experiments reported here, "update" transaction streams were supplied in either strictly ascending or descending collating sequence of the keys.

As pointed out by Keehn and Lacy (28) in their report on VSAM, the VSAM or DSAM type of file organization is sensitive to the choices made for its basic parameters. Although the CINV size and CAREA size may have some impact, the primary area of sensitivity is in the percentages of free space reserved for insertion updates. In the experiments reported here, the applicable values were taken from the general recommendations given by Keehn and Lacy, as it was not possible to conduct a complete set of benchmark tests with a wide choice of these parameters within the time frame of this publication. The reader is therefore cautioned that these results should not be regarded as definitive, although they are encouraging. It is clear, however, that the choices made do not represent a pathological case.

Experimental Design

Since RECON has not in the past performed true ISAM updates on its IX files, there is no historical performance data available with which to compare DSAM performance in this task. Obviously, the author would have to create an artificial test file under both ISAM and DSAM, and then apply a well-defined (or random) set of update transactions to those files to obtain the needed comparison data.

In making this type of comparison, it would have been desirable to explore as wide a range of parameter choices as possible, varying the input sequence, amount of free space and distribution of free space to obtain a full characterization of DSAM. Neither time nor funding were available for such a study.

The experimental design then consisted simply of obtaining a suitable base file, creating ISAM and DSAM master files from this input, and subsequently applying a series of update transactions to those files, while measuring elapsed wall-clock time. Fortunately, a suitable test file and sequence of updates was already available as the byproduct of another researcher's efforts in an entirely different area (39).

Selection of Test Data

The origins of the master file used in these experiments are documented in the paper by Malthouse (39), but will be summarized here for clarity. Both the base files and the simulated update streams were taken from a file containing some 11,991 vocabulary terms. The terms themselves had been drawn from 1,000 document abstracts chosen at random from the 1973 and 1974 Water Resources Abstracts collection on RECON. Code values within the matrix records for each term could be used to partition the vocabulary into ten subsets, each containing the terms that occurred within that subset of 100 documents. Within each of the ten vocabulary subsets, approximately 3,000 terms also occurred in the total vocabulary, while 700 to 800 terms represented new additions to the cumulative vocabulary.

Since there was no reason to suspect that any one of these vocabulary samples was in any way nonrepresentative, the author then used the following procedure to perform the ISAM vs. DSAM update comparison desired.

1. An artificial ISAM and corresponding DSAM base file were created, containing only those terms which occurred in samples 1-5 of the master matrix file described above. This subset contained exactly 8,040 terms, out of the total of 11,991 terms in the vocabulary of all ten samples.

2. Each of the five remaining samples (6-10) in the master matrix was then used as a pseudotransaction update file to the base vocabulary files. The five updates were performed in serial fashion, such that terms inserted in the "i th" update were left in the ISAM and DSAM files during the next update cycle. Thus, upon completion of the five updates, the final files would contain all 11,991 terms present in the original master file matrix.

3. For any given update cycle, all 11,991 terms in the master matrix file were read, but only those terms whose coding indicated they had occurred in that sample were passed into the ISAM or DSAM update software. Terms which did not occur in that particular sample were simply read and ignored. The time required to simply read and decode these entries represents a constant, minimum processing time that is part of every quoted measurement for either ISAM or DSAM. The author has not attempted to subtract this baseline offset in the data

presented here, as it is regarded as being very much less than the time required to process the actual updates themselves.

Experimental Results

The behavior of the ISAM file under the artificial update sequence is presented in Table IV. Since it is well known (3) that an ISAM file's performance should improve when transactions are applied in descending rather than ascending order, duplicate runs are shown for both cases. As expected, the performance is uniformly better with the term sequence reversed. However, the performance improvement shown is not dramatic, and in general would seem only marginally worth the effort involved in processing the file "backwards."

The DSAM update data corresponding to the ISAM results in Table IV are presented in Table V. For the DSAM file there was no theoretical reason to expect any significant difference in performance between ascending and descending update sequences, and in fact none was observed.

Evaluation of Results

As was evident in the reported results for retrieval-only, where these same base files and transaction streams were applied, DSAM's handling of keys presented in strictly ascending sequence is greatly superior to that of ISAM. Furthermore, as the level of insertion updates for ISAM increased, the performance rapidly worsened, as expected. In contrast, DSAM's already high performance ratio decreased only slightly with the higher levels of insertion.

TABLE IV

ISAM FILE UPDATE: TIMING STATISTICS

Matrix File Sample Number	Total[3] Update Records Processed	Total Number of Records Replaced	Total Number of Records Inserted	Total[1] Elapsed Time in Seconds	Update[1] Rate per Second	Total[2] Elapsed Time in Seconds	Update[2] Rate per Second
6	2,894	2,069	825	1,152	2.48	1,102	2.63
7	2,914	2,061	853	1,373	2.12	1,174	2.48
8	2,938	2,187	751	1,567	1.87	1,342	2.19
9	2,966	2,243	723	1,820	1.63	1,514	1.96
10	3,005	2,268	737	2,088	1.44	1,801	1.67

Notes:

1 Transactions processed in ascending collating sequence of keys.

2 Transactions processed in descending collating sequence of keys.

3 This number corresponds to the total of records inserted and records replaced, and is determined by whether or not the term occurred in the given sample of the 11,991 term master file, as shown by codes in each master file record.

4 Updates were applied in a cumulative mode, as shown, starting with an ISAM base file containing 8,040 terms that had occurred in sample 1-5 of the master file matrix.

5 Due to a minor defect in the algorithm used to create the master matrix file, some 62 terms within the vocabulary were not coded as being present in any given subsample. Thus the total of the insertions, added to the 8,040 original terms, does not add up to 11,991, but to 11,929.

TABLE V

DSAM FILE UPDATE: TIMING STATISTICS

Matrix File Sample Number	Total[3] Update Records Processed	Total Number of Records Replaced	Total Number of Records Inserted	Total[1] Elapsed Time in Seconds	Update[1] Rate per Second	Total[2] Elapsed Time in Seconds	Update[2] Rate per Second
6	2,894	2,069	825	45	64.3	45	62.9
7	2,914	2,061	853	51	57.1	51	57.1
8	2,938	2,187	751	53	55.4	52	56.5
9	2,966	2,243	723	51	58.1	50	59.9
10	3,005	2,268	737	51	58.9	51	58.9

Notes:

1 Transactions processed in ascending collating sequence of keys.

2 Transactions processed in descending collating sequence of keys.

3 This number corresponds to the total of records inserted and records replaced, and is determined by whether or not the term occurred in the given sample of the 11,991 term master file, as shown by codes in each master file record.

4 Updates were applied in a cumulative mode, as shown, starting with a DSAM base file containing 8,040 terms that had occurred in sample 1-5 of the master file matrix.

5 Due to a minor defect in the algorithm used to create the master matrix file, some 62 terms within the vocabulary were not coded as being present in any given subsample. Thus the total of the insertions, added to the 8,040 original terms, does not add up to 11,991, but to 11,929.

TABLE V (continued)

6 No CINV splitting occurred in Sample Number 6. Thereafter, about 5 CINV splits occurred per update sample. No CINV was ever split twice, nor did any directory area splits occur.

7 The total percentage of terms inserted was 48 percent, distributed over the 5 updates. Note that this is a term count percentage, and does not account for varying term lengths. The DSAM file was created originally with 40 percent of each CINV left vacant, and 10 percent of the CINV's in each CAREA also left vacant, as the approximate total percentage of new terms was known in advance.

Given that simply reading 11,991 records may require from 5 to 30 seconds, DSAM's total elapsed time of 50 seconds for updating a file with some 2,800 to 3,000 transactions is well within reason for routine use in RECON updating. In contrast, ISAM required from 18 to 34 minutes to perform the same updates. Clearly, the latter rate of processing is totally unacceptable for RECON purposes, where an update may well involve as many as 30,000 transactions for a large file with many unique author index entries as well as a nonauthor vocabulary of 40,000 to 80,000 entries.

Observations Concerning Alphabetical Distributions of Index Terms

At the close of Chapter II, it was hypothesized that a knowledge of the alphabetical distributions of index terms might be utilized in the fine-tuning of a DSAM file's free space distribution to obtain some further space reductions, or to minimize the frequency of CINV splitting for a constant total amount of free space in the file.

Table VI presents the observed distribution of index terms in the RECON IX files, based upon their initial letter (exclusive of the "IT=" prefix). Figure 4.4 summarizes the same data in the form of a histogram, and includes a graphic representation of the variation from collection to collection by a +1 standard deviation extension of each bar. Table VII and Figure 4.5 present the corresponding data for the author index (AU=) entries of these same files. As can be seen from the histograms, the variation in the author index entries is much less than in the general index terms. This would indicate that

TABLE VI

DISTRIBUTION OF RECON INDEX TERMS
BY FIRST CHARACTER OF TERM
(Percentage of Total Terms)

First Character of Term	Bibliographic Collection Name						Average	Std. Dev.
	NSA	EIX	ESI	MEX	WRA	EDB		
A	6.88	6.94	8.56	8.38	7.00	5.00	7.13	1.29
B	5.00	3.56	4.44	3.62	3.75	4.38	4.13	0.57
C	8.69	10.69	8.00	11.38	8.62	11.00	9.73	1.45
D	3.81	2.38	3.56	1.88	5.56	3.00	3.37	1.29
E	4.12	7.12	7.44	3.19	4.56	5.00	5.24	1.69
F	3.94	3.50	4.38	4.75	5.94	3.50	4.34	0.93
G	3.56	3.69	2.44	2.94	2.88	3.69	3.20	0.52
H	4.00	2.31	2.81	2.69	3.31	4.38	3.25	0.80
I	3.62	4.62	3.12	2.44	4.00	3.31	3.52	0.75
J	0.50	0.12	0.50	0.06	0.38	0.44	0.33	0.19
K	1.50	0.19	0.50	1.56	0.25	0.56	0.76	0.61
L	3.75	3.12	3.25	2.38	3.56	3.31	3.23	0.47
M	5.75	6.81	6.69	4.69	5.25	6.19	5.93	0.87
N	4.06	3.44	4.81	5.50	2.38	4.19	4.06	1.08
O	2.25	1.88	2.81	0.81	2.81	2.94	2.25	0.81
P	8.31	11.06	7.19	4.88	7.62	8.69	7.96	2.02
Q	0.38	0.12	0.12	0.06	0.12	0.50	0.22	0.18
R	4.81	4.31	4.38	2.62	5.38	4.19	4.28	0.92
S	9.88	11.06	9.50	8.19	12.69	13.00	10.72	1.89
T	7.50	5.94	4.38	6.62	4.12	5.50	5.68	1.30
U	1.50	0.75	4.94	1.19	1.00	1.44	1.80	1.56
V	1.38	1.06	0.88	1.81	1.12	1.62	1.31	0.36
W	1.44	3.12	3.44	1.31	6.25	2.06	2.94	1.84
X	0.44	0.12	0.06	1.06	0.06	0.12	0.31	0.39
Y	0.56	0.19	0.19	0.56	0.12	0.25	0.31	0.20
Z	0.88	0.75	0.50	2.31	0.19	1.00	0.94	0.73
Total Terms	26,042	39,176	4,284	30,002	15,934	58,282		

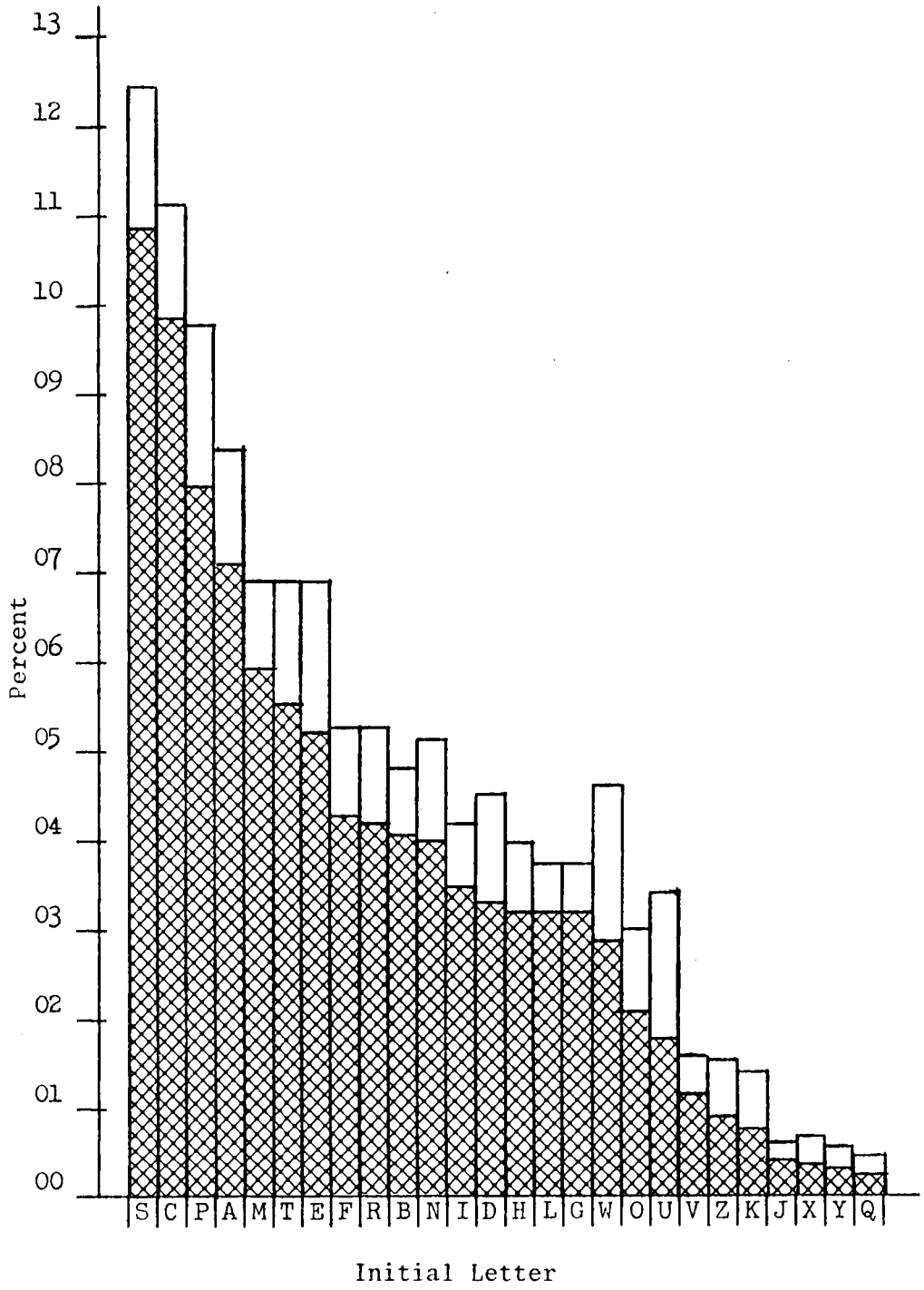


Figure 4.4. Distribution of Index Terms by Initial Letter of Term. The crosshatched bars give the average frequency over six document collections, while the outlined extensions show +1 standard deviation from that average.

TABLE VII

DISTRIBUTION OF RECON AUTHOR INDEX TERMS
BY FIRST CHARACTER OF SURNAME
(Percentage of Total Terms)

First Character of Surname	Bibliographic Collection Name						Average	Std. Dev.
	NSA	EIX	ESI	MEX	WRA	EDB		
A	3.62	4.44	3.25	3.69	3.44	3.44	3.65	0.42
B	8.38	8.56	8.88	8.00	8.69	8.38	8.48	0.30
C	5.38	5.12	6.56	4.19	6.00	5.50	5.46	0.80
D	4.68	4.50	4.38	4.38	4.44	4.44	4.47	0.11
E	1.88	1.88	2.12	1.75	2.00	2.06	1.95	0.14
F	3.31	3.25	3.62	2.88	3.38	3.38	3.30	0.24
G	5.31	5.31	4.94	5.38	5.00	5.38	5.22	0.20
H	5.25	5.56	7.81	3.88	6.69	6.00	5.87	1.33
I	1.06	1.06	0.56	1.44	0.81	1.00	0.99	0.29
J	1.62	1.56	2.19	1.25	2.12	1.69	1.74	0.36
K	7.56	7.75	5.00	9.69	6.00	7.25	7.21	1.61
L	4.69	4.69	4.81	4.56	5.06	4.88	4.78	0.18
M	8.00	8.00	8.69	8.00	8.56	8.19	8.24	0.31
N	2.50	2.44	2.00	2.81	2.31	2.38	2.41	0.26
O	1.69	1.69	1.69	1.75	1.62	1.62	1.68	0.05
P	5.25	5.25	4.88	5.69	4.88	4.81	5.13	0.34
Q	0.06	0.12	0.12	0.06	0.12	0.06	0.09	0.03
R	4.56	4.75	4.88	4.31	4.81	4.62	4.66	0.21
S	11.19	11.19	10.56	11.75	10.69	11.31	11.12	0.43
T	3.94	3.88	3.25	4.25	3.62	3.88	3.80	0.34
U	0.50	0.50	0.31	0.62	0.31	0.50	0.46	0.12
V	2.62	2.50	1.62	2.94	1.88	2.18	2.29	0.49

TABLE VII (continued)

First Character of Surname	Bibliographic Collection Name						Average	Std. Dev.
	NSA	EIX	ESI	MEX	WRA	EDB		
W	3.56	3.81	5.62	2.62	4.94	4.06	4.10	1.06
X	0.00	0.00	0.00	0.00	0.00	0.00	0	0
Y	1.00	1.00	0.62	1.25	0.88	1.06	0.97	0.21
Z	1.38	1.38	0.69	1.94	0.88	1.12	1.23	0.44
Total Terms	334,239	250,413	35,235	81,064	77,448	55,364		

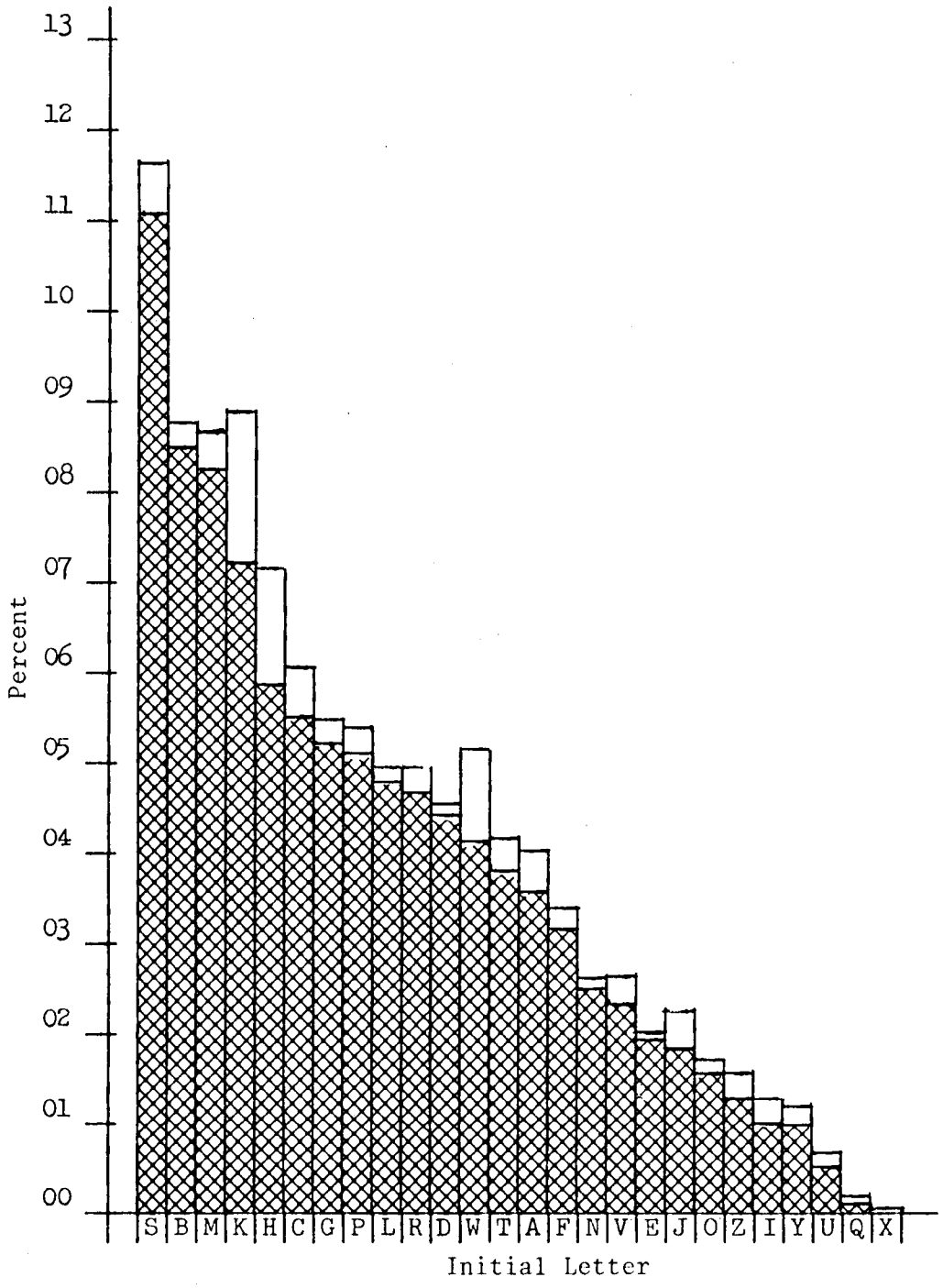


Figure 4.5. Distribution of Author Surnames by Initial Letter of Name. The crosshatched bars give the average frequency over six document collections, while the outlined extensions show +1 standard deviation from that average.

the greatest potential benefits of tailoring a DSAM free space distribution to match such a "native" pattern would be achieved in the author index files, precisely where it would prove most beneficial due to the large size and high insertion levels that characterize these subsets of the RECON indexes.

Thus, the distribution data in existing RECON IX files supports the author's hypothesis. The distribution data illustrates the stability of these patterns across bibliographic collection boundaries and the basic agreement with the results of the multimillion word vocabulary study of Klingbiel (29).

III. A NOTE ON THE MATHEMATICAL ANALYSIS OF DSAM

The design of DSAM so closely parallels that of IBM's VSAM that in many respects DSAM may be viewed as an OS/360 compatible subset of VSAM, with extensions to support variable length keys in the data records. Given the further fact that the author has chosen to adopt the same nomenclature (CINV, CAREA, etc.) as has been used in the VSAM literature, it is possible to avoid a rather lengthy restatement of some of the mathematical background pertinent to both access methods.

The reader who is interested in (a) formal algorithms for the calculation of VSAM (DSAM) direct access storage space requirements; (b) analytic modeling techniques for the probability of CINV (and CAREA) splitting; or (c) formulations for sequential and random access performance estimation; is directed to the excellent article by

Keehn and Lacy in the IBM Systems Journal (28). All of these formulations, of course, would need to be slightly modified to (a) remove parameters applicable only to VSAM, such as CINV control information field overheads, and replace these with corresponding DSAM values; and (b) extend the formulations to account for variable length records and variable key lengths within the data records.

The necessary modifications to the fundamental formulations presented by Keehn and Lacy are not trivial, particularly where the equations applicable to the probability of a record insertion causing a CINV or CAREA split are concerned. However, the reiteration of the basic formulas here would serve no useful purpose, and the extensions to those formulations are a suitable topic for further research.

CHAPTER V

SUMMARY AND CONCLUSIONS

I. OVERVIEW

This report began with an introduction to information retrieval systems, and the subset of such systems concerned with retrieval of bibliographic materials in particular. Emphasis was then placed upon on-line, real-time operation bibliographic systems, in contrast to the more traditional background sequential processes used in many retrospective search and selective dissemination of information programs. The large on-line retrieval systems were then characterized as most often using some form of inverted file structure. ERDA/RECON at ORNL (5, 10, 11) was then used as a typical example of such systems and the vehicle for all subsequent discussions. The report briefly explored the RECON command structure and its interaction with the general file structure. Attention was focused directly upon the inverted index (IX) vocabulary files of RECON. This discussion provided the motivation behind the author's development of DSAM, for use with these files as a replacement for the vendor-supplied ISAM presently in use.

A discussion of the RECON environment and the difficulties experienced with ISAM lead to the establishment of the basic set of design goals for DSAM:

1. Reduction in direct access storage space requirements;
2. Minimization of file reorganization frequency; and
3. Fail-safe support of simultaneous retrieval and in-place file updating activities.

The remainder of the report was devoted to the presentation of the DSAM design itself and to the results obtained from experiments using an initially implemented subset of DSAM.

II. DSAM SYSTEM DESIGN REVIEW AND EVALUATION

The DSAM design, and the subset which has presently been implemented, have met or exceeded the most important goals set forth for them in Chapters II and III. Much implementation work remains to be accomplished before DSAM software can provide all of the optional features indicated in the total design. Nevertheless, the DSAM software implemented has accomplished the following:

1. DSAM support for variable length keys in the data records has provided a mechanism to achieve a DASD space reduction of from 30 to 45 percent compared to ISAM.

2. DSAM supports both sequential and indexed-direct access to the user's data. Sequential retrieval commencing at any point within the file and proceeding in either ascending or descending collating sequence of the record key is also supported.

3. The retrieval-only performance of DSAM has been shown to be superior to ISAM for the same data collection, and very much superior

for those cases where input queries are presented in ascending or descending sequence of the record keys.

4. The random access insertion update performance of DSAM has been shown to be superior to ISAM due to the use of distributed free space techniques in contrast to the ISAM overflow technique.

5. The use of distributed free space, coupled with automatic reorganization through CINV splitting, permits DSAM to support relatively high levels of record insertion without severe performance degradation.

6. In DSAM, file reorganization is a dynamic process that is automatically performed during updating itself. Thus, the need for periodic off-line reorganization has been minimized. ISAM's need for reorganization was a major factor in RECON's use of total file re-creation rather than true update, and DSAM has all but eliminated the need for this procedure.

7. The choice of PL/1(X) for the initial implementation of DSAM proved to be sound. The expected benefits of developing software in a high level language were achieved, and are reflected in the speed of DSAM implementation, ease of software maintenance and debugging and clarity of the software logic itself. At the same time, PL/1(X) produced machine code that provided adequate performance to demonstrate superiority over ISAM for the same application.

8. Analysis of alphabetical term distribution data from existing RECON files showed a high probability for future DSAM performance improvement through variation of the free space distribution in a DSAM

file to match the expected distribution of terms added in insertion updating.

III. SUGGESTIONS FOR FURTHER STUDY

The DSAM directory structure is a variant of the B-Tree (2, 13, 30, 31) organization, tailored to variable length keys. As has been recently pointed out by Held (13), B-Trees are sensitive to the number of index entries held per node (fanout ratio), and may not be superior to a static directory organization such as ISAM in terms of performance. A theoretical investigation of the properties of the DSAM variant of B-Trees and extension of the work by Keehn and Lacy (28) to DSAM's variable length keys might provide insight into possible future performance improvements for DSAM.

Although the DSAM directory is small in relation to the size of the addressed data structure, it could itself be the subject of further compaction through the use of key compression techniques such as are employed in VSAM (20, 21, 22, 23, 24, 28, 59). The major benefit of using such a technique would not be the overall size reduction achieved, but rather would consist of an increase in the fanout ratio for each CINV in the directory. Moreover, if key compression were used to reduce the average length of a directory entry, many more key insertions could be accommodated before a directory CINV had to be split.

Held (13) has commented upon the possible difficulties that may occur when simultaneous retrieval and updating processes operate on the

same B-Tree structure. In particular, Held indicated that during the process of splitting a node (CINV in DSAM), the scanning pointers of a simultaneous retrieval process would need to be altered in a "nontrivial" manner. DSAM's use of internode linkages would seem to avoid at least part of this problem, although it is not clear that all potential cases of such conflict have been resolved.

As presented in Chapter IV, analysis of the alphabetical distribution of vocabulary terms has indicated that DSAM's support of a variable distribution of free space, under direct user control during file creation, might permit a reduction in total free space requirements. This would be achieved by tailoring the free space distribution to match the alphabetical distribution of the existing vocabulary. A series of experiments directed at establishing the viability of this concept, and an investigation of the transferability of a given distribution pattern across collection boundaries, should be conducted in the near future.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Aho, A. V. and M. J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, COMM. ACM, Vol. 18, No. 6, June 1975.

2. Bayer, R. and E. McReight, Organization and Maintenance of Large Ordered Indices, Proc. 1970, ACM-SIGFIDET Workshop on Data Description, Access and Control, Houston, Texas, November 1970.

3. Behymer, J. A., R. A. Ogilive and A. G. Merton, Analysis of Indexed Sequential and Direct Access File Organizations, Proceedings, SIGMOD 1974 Workshop (Rustin Ed.), May, 1974, pp. 389-417.

4. Brooks, A. A., ADSEP, An Automated Data Set Editing Program, CTC-34, Oak Ridge Gaseous Diffusion Plant, Oak Ridge, Tennessee, May 12, 1970.

5. Brooks, A. A., et al., The AEC/RECON Project at ORNL, ORNL-4913, Oak Ridge National Laboratory, Oak Ridge, Tennessee, November 1973.

6. Brooks, A. A., Oak Ridge Computerized Hierarchical Information System (ORCHIS) Status Report, ORNL-4929, Oak Ridge National Laboratory, Oak Ridge, Tennessee, July 1973.

7. Cardines, A. F., Analysis and Performance of Inverted Data Base Structures, COMM. ACM, Vol. 18, No. 5, May 1975.

8. Crawford, R. G., Automatic Dictionary Construction and Updating, PhD Thesis, Cornell University, September 1974.

9. Giering, R. H., Search Strategies and User Information, Presentation to the Conference on Large Data Bases, May 1974, Mead Technology Laboratories Research Park, Dayton, Ohio 45432.

10. Gillcrist, J. A., AEC/RECON User's Manual, ORNL-4943, Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1974.

11. Haeuslein, G. K. and A. H. Culkowski, The Oak Ridge Modular RECON Program, UCCND-CSD-9, Oak Ridge National Laboratory, Oak Ridge, Tennessee, April 1975.

12. Hairston, E. L., B. H. Stoutt and S. L. Yount, Console-Oriented Information Storage and Retrieval System for the Nuclear Safety Information Center, CTC-37, Rev. 1, Oak Ridge Gaseous Diffusion Plant, Oak Ridge, Tennessee, August 1973.

13. Held, Gerald and Michael Stonebraker, B-Trees Re-examined, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, California, July 1975.

14. Hsiao, D. and F. Harary, A Formal System for Information Retrieval From Files, COMM. ACM, Vol. 13, No. 2, February 1970.

15. IBM Corporation, IBM System/360 Operating System Supervisor and DATA MANAGEMENT MACRO INSTRUCTIONS, GC28-6647-5.

16. IBM Corporation, Information Management System/360, Version 2, General Information Manual GH20-0765-4.

17. IBM Corporation, Information Management System/360, Version 2, System/Application Design Guide, SH20-0910-3 (1972).

18. IBM Corporation, OS Data Management Macro Instructions, Release 21.7 GC26-3974-1.

19. IBM Corporation, OS ISAM Logic, Order No. BY28-6618.

20. IBM Corporation, OS/VS Virtual Storage Access Method (VSAM) Options for Advanced Applications, GC26-3819-2.

21. IBM Corporation, OS/VS Virtual Storage Access Method (VSAM) Planning Guide, GC26-3799-2.

22. IBM Corporation, OS/VS Virtual Storage Access Method (VSAM) Programmer's Guide, GC26-3838-0.

23. IBM Corporation, OS/VS Virtual Storage Access Method (VSAM) System Information, GC26-3835-0

24. IBM Corporation, OS/VS2 Virtual Storage Access Method (VSAM) Logic, Release 3.7, SY26-3825-1, File No. S370-30.

25. IBM Corporation, System 360 OS Data Management Services Guide, GC26-3746.

26. IBM Corporation, OS PL/1 Checkout and Optimizing Compilers: Language Reference Manual, SC 33-0009-2.

27. IBM Corporation, OS PL/1 Optimizing Compiler Programmer's Guide, SC 33-0006-2.

28. Keehn, D. G. and J. O. Lacy, VSAM Data Set Design Parameters, Vol. 13, No. 3, IBM Systems Journal, 1974, pp. 186-212.

29. Klingbiel, P. H., Multi-million Word Data Bases. A Preliminary Report, Vol. 1 and 2, Defense Documentation Center, DDC-TR-74-1.

30. Knuth, D. E., The Art of Computer Programming, Vol. 3, Sorting and Searching, Addison-Wesley Publishing Company, Reading, Massachusetts, 1973.

31. Knuth, D. E., Sorting and Searching - Errata and Addenda, STAN-CS-73-392, Computer Sciences Department, School of Humanities and Sciences, Stanford University, October 1973.

32. Lancaster, F. W. and E. G. Fayen, Information Retrieval On-Line, Melville Publishing Company, Los Angeles, California, 1973.

33. Lefkovitz, D., File Structures for On-Line Systems, Spartan Books, Hayden Book Company, Inc., Rochelle Park, New Jersey, 1969.

34. Lesk, M. E., Compressed Text Storage, Computing Science Technical Report No. 3, Bell Laboratories, Murray Hill, New Jersey, November 1970.

35. Lind, H., V. I. Lum and M. E. Senko, Calibration of the File Organization Evaluation Model (FOREM 1) in: Formatted File Organization Techniques, Final Report AF30602-69-C-0097, Information Science Department, IBM Research, San Jose, March 1970.

36. Liu, J. and J. M. Milner, Probabalistic Models of Inverted File Information Retrieval Systems, in "Proceedings of the International Symposium on Computer Performance Modeling Measurement and Evaluation," Department of Computer Science, University of Illinois, Urbana, Illinois 61801, March 29-31, 1976.

37. Lowe, T. C., The Influence of Data Base Characteristics and Usage on Direct Access File Organization, J. ACM, (A) Vol. 15, No. 4, October 1968.

38. Malcolm, M. A., PL360 (Revised) A Programming Language for the IBM 360, STAN-CS-71-215, Computer Science Department, School of Humanities and Sciences, Stanford University, May 1971.

39. Malthouse, N. S., INDEXGEN: Automatic Indexing for Large Bibliographic Document Collections, M.S. Thesis, University of Tennessee, June 1976.

40. Martin, J., Computer Data-Base Organization, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

41. Martin, J., Design of Man-Computer Dialogues, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1973.

42. Martin, T. H., A Feature Analysis of Interactive Retrieval Systems, Institute for Communications Research, Stanford University, SU-COMM-ICR-74-1, September 1974.

43. Milner, J. M. and J. W. Liu, Secondary Storage Access Scheduling for a Multiuser Inverted File Processing System, Presented at Special Interest Conference in the Theory and Applications of Scheduling at Orlando, Florida, February 1976.

44. Mullins, R. C., et al., File Organization Modeling System (FORMS) Volume I, AD-750 898, Planning Research Corporation, McLean, Virginia, September 1972.

45. Mullins, R. C., et al., File Organization Modeling System (FORMS) Volume II User's Manual, AD-750 899, Planning Research Corporation, McLean, Virginia, September 1972.

46. Nievergelt, J., Binary Search Trees and File Organization, Vol. 6, No. 3, Computing Surveys, September 1974, pp. 195-207.

47. Pomerance, H. S., A Short Guide to SDI Profiling at ORNL, ORNL/TM-5484, Oak Ridge National Laboratory, Oak Ridge, Tennessee, June 1976.

48. Reiter, A., Data Structures for Secondary Storage Representations, TSR-1554, Mathematics Research Center, University of Wisconsin, July 1975. (Also: Proceedings Very Large Data Bases Conference, Framingham, Massachusetts, September 1975).

49. Reiter, A., Some Experiments in Directory Organization - A Simulation Study, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin 53706, in: Proceedings of the International Symposium on Computer Performance Modeling, Measurement and Evaluation, March 29-31, 1976, Harvard University, Cambridge, Massachusetts.

50. Salton, G., Dynamic Information and Library Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

51. Salton, G., The SMART Retrieval System: Experiments in Automatic Document Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

52. Seastrom, D. E. and D. A. Thompson, Adaptive Document Retrieval Displays to Accommodate System Learning, ASIS, American Society for Information Science, 1972 Annual Meeting, October 1972.

53. Senko, M. E., et al., Semi-operational Evaluation of File Modeling Techniques, Final Report AF 30602-70-0114, Information Sciences Department, IBM Research, San Jose, February 1971.

54. Senko, M. E., V. Y. Lum and P. J. Owens, A File Organization Evaluation Model (FOREM), Proceedings of IFIP-1968, pp. C19-23.

55. Severance, D. G., Identifier Search Mechanisms: A Survey and Generalized Model, Computing Surveys, Cornell University.

56. Singletary, V. A., An On-Line Conversational Retrieval System for ORCHIS Text-Oriented Data Bases User's Manual, ORNL-4951, Rev. 1, Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1975.

57. Slusher, R., Computing Applications Department Cumulative Bibliography 1962-1975, K/CSD/INF-76/6 (Revision of UCCND/CSD/INF-36).

58. Spires User's Manual, Parts I-IV, 1st ed., Stanford Center for Information Processing (SCIP), Academic Computing Services, Stanford University, Stanford, California, October 1972.

59. Wagner, R. E., Indexing Design Considerations, IBM Systems Journal, Vol. 12, No. 4, 1973, pp. 351-67.

60. Yeh, J., Architectural Design of a File Management System, Technical Report TR-253 NGR-21-002-197 Supplement No. 3, University of Maryland, Computer Science Center, College Park, Maryland, September 1973.

VITA

William B. Malthouse is employed by Union Carbide Corporation as an Applications Analyst. He is a member of the Computer Science Division and Computing Applications Department of Oak Ridge National Laboratory, where he has been associated with the RECON project since 1971. Mr. Malthouse is a member of the Association for Computing Machinery, and a participant lecturer in the ORNL Traveling Lecturer Program. He has taught in-house training courses in advanced uses of PL/1 at ORNL.

The author joined Union Carbide as a chemist in 1966, where he became Head of the Analytical Chemistry Laboratory for the Greenville, South Carolina plant of the Electronics Division. In 1968 Mr. Malthouse moved to Oak Ridge area, accepting a position in the Materials Engineering Development Division of the Oak Ridge Y-12 Plant. His work at Y-12 in the development of improved carbon composites led to a U.S. Patent.

Mr. Malthouse graduated from Florida State University in Tallahassee, Florida in 1965, with a Bachelor of Science in Chemistry. As a sophomore at FSU, the author met and later married Nancy L. Sibbles of Mobile, Alabama, who as Mrs. Nancy S. Malthouse, recently completed her own M.S. degree in Computer Science from the University of Tennessee.

Mr. Malthouse was born in Detroit, Michigan on March 11, 1944. He moved to Orlando, Florida in 1956 with his family and graduated from Oak Ridge High School, Orlando, Florida, in 1962.