DICTIONARY STRUCTURES FOR A DOCUMENT RETRIEVAL SYSTEM.

Franklin Jose Castillo M.

A Thesis

in

The Department

of

Computer Science.

Presented in Partial Fulfillment of the Requirements

for the degree of Master of Computer Science at

Concordia University.

Montreal, Quebec, Canada.

May, 1979.

ABSTRACT.

Dictionary Structures for a Document Retrieval System.

Franklin Jose Castillo M.

This investigation proposes three feasible structures for the dictionaries of an inverted file organisation, commonly used in on-line document retrieval systems for library applications. The study is dependent on a number of a given properties and assumptions regarding the nature of a bibliographic document data base. The conclusions reveal that the relative efficiencies of the different schemes are dependent on the particular attribute field used.

ACKNOWLEDGEMENTS.

The writer gratefully acknowledges the guidance and assistance of Professor H.S. Heaps, who enabled the completion of this thesis.

He wishes to extend this acknowledgement to the Government of Venezuela which through the Foundation Gran Mariscal de Ayacucho sponsored his studies, and also to his wife Ana Maria and his son Franklin Jr. for their moral encouragement.

TABLE OF CONTENTS.

CHAPTER ONE.

1. GENERAL VIEW AND BASIS OF THE PROPOSED SYSTEM.

1.1. INTRODUCTION.

The volume of scientific and technical information has followed an exponential growth since the middle of the 18th century, and has now reached a level of about 100,000 published journals. The need for efficient procedures for searching and accessing the body of recorded knowledge has increased accordingly [7].

The traditional place for storage of publications is the library, and the traditional access tools are the library catalogs. The function of library systems is to store and locate specific documents and make them available to users.

The field of Document Retrieval deals with storage, maintenance, indexing, and retrieval of documents. To retrieve documents relevant to any user query or search request by means of a computerized search it is usually necessary to replace the full text by some other representation in order to reduce storage space. The substitute to be used in place of the text has been the subject of much research.

As described by J. Minker [1], in a document retrieval system the input data is usually represented in the form of natural language text. When text systems are mechanized it is necessary to provide some abbreviated description of the text in order to facilitate its retrieval. To retrieve text without any such description, or index, usually takes a prohibitive length of time. A major problem is the determination of how to describe texts, and one of the many possible descriptions is through replacement of the text by an abstract, by keywords or phrases, or by terms selected according to some particular classification scheme. Once it is decided what means are to be used, it is necessary to maintain appropriate dictionaries, thesauri, and other auxiliary fields in order to aid in the retrieval process.

In response to a question a document retrieval system usually compares terms or phrases in the query with terms or phrases in the document substitute, and then orders the retrieved documents according to some measure of their closest match to the question. The output of a document retrieval system may be a list of addresses or names of documents, some substitutes such as abstracts, or the entire text.

Thus document retrieval systems may consist of:

- an input source language,
- some means to transform queries and documents to a representation for storage in the machine,
- a search strategy that matches the queries against the document substitutes,
- an output language,
- files and data management.

Data structures play a central role in affecting retrieval efficiency [13]. However, the best choice of data structure is very dependent on the amount of data input, data modifications, the frequency of these operations, and the type of query. For a given type of retrieval system there may be many data structures that are applicable. However, the physical device on which the data is to be stored and accessed greatly influences the selection of the most suitable one.

1.2. OBJECTIVES.

The present thesis attempts to identify some of the important parameters that affect retrospective search time, which is one of the critical cost factors in an information retrieval system [2]. The term retrospective is used to denote retrieval from accumulated document collections, in contrast to current awareness mechanisms that are concerned with only the more recently published documents. K.L. Montgomery [11] reported that among the principal parameters that affect search time are:

- file structure,
- the retrieval computer program,
- the available computer system.

The cataloguing scheme also directly affects the retrieval process, but it will not be taken into account in this study.

Formatted data bases are usually smaller, in terms of numbers of bits, than the collection of documents, and also they allow faster retrieval of information. A representation of a data base structure may be kept available in either core storage or in units of auxiliary memory, such as disk, that are brought into core unit by unit (unit of the data base) as needed.

The analysis of data structures has become an important topic of study in computer science, since it has a direct affect on any retrieval process. As mentioned by L.B. Doyle [2] the method of retrieval depends on the method of storage, and in turn the method of storage depends on what are conceived to be the major processing tasks, and these tasks will be subject to the constraints of the particular available computer system.

A comparative analysis of three possible schemes to process the storage, maintenance, and retrieval of documents by means of an inverted data base structure is undertaken in the present thesis. Also, an evaluation in terms of efficiency is presented.

The main criteria used in the evaluation are the time required to process a query and the space required to store the data base. Lefkovitz [11] notes that the question processing is a function of the file organisation technique, the characteristics of the direct access storage device, and the number of records accessed from the file. The CPU execution time and the elapsed time of the user's program are not considered since they are dependent on programming details or are subject to interruptions by the computer operating system.

1.3. ASSUMED CONTENT OF THE DOCUMENT COLLECTION.

1.3.1. Type of information selected to describe texts.

To allow meaningful comparisons of different techniques it is assumed that the data base may contain the following information related to documents: title terms, keywords, author names, journal codens, date of publication, language of publication, pages, and volume numbers.

In the subsequent discussion the estimates of the sizes of the different amounts of information are chosen according to what is believed to be typical of a large class of practical document retrieval systems. It is believed that the conclusions are not critically dependent on the assumptions, and hence are not valid only in special, and artificially restricted, instances.

1.3.2. Organisation of the data base.

Recorded information exists in great quantity and diversity, and if it is to be utilized in a document retrieval system it must be given structure. Perhaps the most important factor that affects the overall performance of operations on secondary file structures is the manner in which the indexes and associated lists are physically organized and managed in storage. A model of the inverted file must be proposed and then analyzed in order to obtain estimations of average access time (for retrieval) and total storage performance. The schemes derived take into account the storage structure, method of search of the indexes, statistics of the contents of the data base, and the logical complexity of queries. The incorporation of all such factors, whose influence cannot be disregarded in a real data base system, is a main feature of the present investigation.

The organisation of the data base is chosen to contain the following components:

1- Four dictionaries and their corresponding inverted lists for title words, keywords, author names, and subject headings.

2- One document directory file in which are stored:

- pointers to link the location of every document item in the coded sequential data base,
- journal codens,
- publication date.

These two last fields can be included in the queries in order to specify more precision in user questions if desired.

- language of publication,
- number of pages, and type of retrieval reference,
- volume number.

3- Coded sequential data base in which are stored all the document records. Within each record the sequence of terms is stored in coded form. This file is accessed in order to print out the text of those document records found to be relevant to queries.

1.3.3. Considerations concerning addition of documents to the data base.

The number of documents in a collection at a time t depends on such factors as the budget allotted by the institution to their information system, the specific goals of the information system, and also on the policies with regard to acquisition of new documents [9]. These factors may have considerable influence on the growth of the data base and therefore on the vocabulary of attributes. Nevertheless, repetition of many terms always occurs, and helps to prevent the drastic increase in the size of the dictionaries that would certainly occur if the rate of growth of the dictionaries was the same as that of the total text length [3]. However any document data base may be expected to grow in size and complexity over a period of time, and consequently a performance degradation is highly probable as a result of updates. Therefore a costly reorganisation of the data base may be needed quite frequently. It is important that the model of the data base allows a growth consistent with that observed in existing data bases used in document retrieval systems.

More details of vocabulary growth will be discussed in a later section, since a knowledge of certain typical behaviour proves to be very useful in the prediction of the expansion of document data bases in general.

1.4. SEARCH STRATEGY ADOPTED.

1.4.1. Description of search.

A search may be described as the process of gaining access to records that are matched with queries. The search strategy is based on comparison of the query with the stored document records, and the documents retrieved may be more or less relevant to the request [10]. The method used to search a data base system is a function of the file set organisation and of the logic permitted in the query [11]. The method proposed is to remain fixed during analysis of the overall retrieval system.

The principal elements of file organisation that determine the method to be used in the search are:

- file sequence,
- storage medium,
- file content (the set of record types and fields contained in the file).

Also, a major factor, independent of file organisation, is the logic allowed in the queries.

A source input language is a means of communication between the user and the system [1]. There are several different source input language types: command, data description, file maintenance, procedural, and query. The command language is used to communicate with the system in order to specify various commands and to initiate jobs. The system must be able to distinguish between the different commands such as: QUERY, INSERT, CHANGE, and DELETE.

The data description language is used to describe the logical structure of data in the system. A file maintenance language is used to delete, change, or add data to the system; it provides the user with the means of control of the system.

A query language is the main means by which any user may express his request to the data base, and it may range from prescribed input format, in which the user inserts parameters, to natural language input. A procedural language may be utilized to implement programs that perform any of the above functions.

In the context of a document retrieval system a query may consist of a list of terms or phrases with, or without, weights to designate the importance of the term or phrase as an indicator of the relevance of a document to the interest of the user [1].

1.4.2. Boolean search.

Throughout the present investigation the specific search method to be applied in response to the query language is the Boolean Search, whose strategy consists of retrieving those documents that are "true" for the query [10]. The formulation of queries is expressed in terms of index terms (or keywords) and the means of combining them by the usual logical connectives "AND", "OR", and "NOT". For example, if the query is

Q = (K1 AND K2) OR (K3 AND (NOT K4))

then the Boolean search will retrieve all the documents indexed by K1 and K2, as well as all documents indexed by K3 but not indexed by K4.

A way to implement the Boolean search is through the use of inverted files. For each keyword the inverted file contains a list of the numbers of the documents that contain that particular keyword. To determine the document numbers of documents that satisfy the request a set of operations corresponding to logical connectives on the KI-lists, is performed. The intent and strategy of the search may be rather general or very specific. For example, a five-term question can often eliminate the whole file, while a two-term question may give a response of several hundred document numbers. The general effect of logical connectives may be summarised in the form:

1- "AND" and "NOT": decreases responses and narrows specificity.

2- "OR": increases response and broadens specificity.

3- ">=", "<=", "=", ">", "<": applied to dates, decreases response and has minor effect on specificity.

4- "≠": decreases response.

Boolean algebra may be applied to process statements of search criteria that contain logical connectives. In order to limit the length of questions and the types of logic used by queries in order to fit the best overall objectives of a system it is essential to adopt a standard format for the statement of requests. The input request is then transformed by a computer program into a sequence of operations to be applied to lists of the inverted file.

1.4.3. Weights of terms used in the search.

Sometimes it is desirable to allow the option of assigning different weights to the terms in a search query. Some terms may be more significant than others, in which case it is worthwhile to have the option in the query language. A document is then classed as relevant if both the question logic is satisfied and also the accumulated weight of terms in the query is greater than a threshold value. These threshold values may be specified to follow the logic operators and may be composed of two-digit numbers [3].

1.4.4. Truncation.

Ther are several types of truncation mode that are desirable for specification of attributes. For example, right truncation may be of two different types: the first type is through use of the character * to represent unlimited truncation to specify a search for a stem followed by any combination of characters. Another type is through use of the character + to represent limited truncation in which a stem is to be followed by no more than a given number of characters. However truncation of question terms is not permitted in the present approach in order to allow a hash storage scheme to be used in the term dictionary and in the inverted files. It is believed that in many systems the advantage of rapid access to the dictionary is more important than the provision for the use of truncation in specification of question terms.

1.4.5. Structure of system query language.

The present approach requires that inputs and queries be specified in a standard form so that the computer is not required to perform any semantic analysis of the question. This is in the interest of efficiency and economy since present day computers are well able to perform tasks such as matching, counting, and looking up words in a list. While such operations may be performed with precision and speed, a computer analysis of the meaning of free text is much more costly and less reliable [2].

In order to facilitate the data manipulation all queries are represented by commands (English keywords) based on a consistent keyword-oriented syntax. The command language must allow users to express their requests in a suitable form for processing by the information storage and retrieval system. It should provide for specification of search descriptors, output limiters, verbs and some user aids, as follows:

1- Search descriptors. Users may enter single descriptors combined by logical search expressions. To initiate a search at least one descriptor must be entered.

2- Output limiters. They are used to limit the date of the retrieved references, the form of the retrieved reference (i.e., book, journal article, proceeding, report, thesis), the journal in which the reference was published, and the type of the retrieved reference (i.e., application, criticism, method, review, theory.)

3- Verbs. They allow users to express an action for the program to execute and provide an appropriate response. Verbs can be utilized to merge two or more previously entered request sets into a logical search expression.

The logical connectives AND, NOT, OR can be used to form the expression, to nullify one or more previously entered retrieval sets in order to free the working storage area, or to terminate a search query formulation in order to begin output of the retrieved references.

4- User Aids. They provide user assistance and information to aid in search query formulation. For example they may be used to explain the use of one or more commands and to supply simple representative examples. They may also help the user by providing instructions and examples of options available at major decision points of the question formulation process.

The structure of each question statement is composed of lines, and each separate line may consist of one of the following:

1- One search term. Such a term may be either:

- A term to be searched for in the author name field. This term must be preceeded by "A/" to allow its identification.
- A term to be searched for in the keyword field; it must be preceeded by "K/".
- A term to be searched for in the subject heading field, it must be preceeded by "S/".
- A term to be searched for in the title term field. It may be preceeded by "T/", or by no character in which case it is regarded by default to be a title term.

2- Sequence of specified search terms, and/or previous line numbers, each followed by a comma used as a separator, and/or followed by one of the logic operators: AND, OR, NOT.

3- "SEARCH" command, which requests the start of execution of a search. It may be formed by a sequence consisting of

- a line number, if it is required, followed by :
    - -Description of a desired date and its comparison sign,
    - -specific journal codens with sign "+".

One of the main advantages of conducting searches on-line is that it allows the possibility of interaction with the data base during formulation of the search question. Other advantages are the possibility of increased involvement of the end-user in the search formulation, and the reduced waiting time before the results of the search are available. As computer processing becomes cheaper it is becoming more economical to do searches of single questions on retrospective files, and as computers become faster it is becoming more attractive to perform such searches on-line [2].

A processing function of an information storage and retrieval system is the transformation of the query language statements into an intermediate language that contains the basic information for the processing of the data base.

1.4.6. Backus Normal Form description of input format.

```
    <INPUT> ::= <SEARCH>,<LINE #>.  /
         <SEARCH>,<LINE #>,<comparison> <month><year>,‡<journal>.
         / <statement>.
<comparison> ::= = / < / > / <= / >=
<month> ::= <string> / b
<year> ::= <digit><digit> / 18<digit><digit> / 19<digit><digit>
<journal> ::= <digit><string> / <string>
<string> ::= <letter> / <string><letter>
<statement> ::= <line #>.<sentence>
<line #> ::= <digit><digit>
<sentence> ::= <keyword sentence> / <title sentence> /
         <subject sentence> / <author sentence> /
         <line sentence>
<author sentence> ::= <A/><author line><extent>
<author line>::= <author> / <author>,<author line>
<author> ::= <string>b<weight> / <initial>b<string><weight>
<initial> ::= <letter> / <letter> b<letter> /<letter><letter>
<weight> ::= <digit><digit> / b
<logic> ::= <weight>b<symbol><threshold> / b
<threshold> ::= <digit><digit> / b
<keyword sentence> ::= <K/><keyword line> <extent>
<keyword line> ::= <keyword> / <keyword>,<keyword line>
<keyword> ::= <string>b<weight>
<title sentence> ::= <T/><title line><extent> /
         <title line><extent>
<title line> ::= <title> / <title>,<title line>
<title> ::= <string>b<weight>
```

<line sentence> ::= <numbers>b<weight><extent>

<numbers> ::= <line #> / <numbers>,<line #>

<letter>::= a/b/c/ ... /z

<digit>::= 0/1/2/ ... /9

<symbol>::= AND / OR / NOT / ADJ / PRE

<extent> ::= ,<logic> / b.

1.4.7. Examples of search questions.

Example 1.

```
1.T/VOCABULARY,GROWTH,AND
2.K/DOCUMENT,RETRIEVAL,AND
3.1,2,AND
4.SEARCH,3,>1975,≠JASIS.
```

This is a request for items that contain VOCABULARY(TIT) and GROWTH(TIT) and DOCUMENT(KEYW) and RETRIEVAL(KEYW), where VOCABULARY(TIT) denotes the term VOCABULARY in the title field. The retrieved items must also have a date of publication greater than 1975 and the journal coden of the publication must be different from JASIS.

Example 2.

```
1.A/A CARDENAS
2.K/GDBMS,MIS,GFMS,OR
3.1 02,2,OR03
4.SEARCH,3.
```

This question requests a search for A. Cardenas(AUT) combined with any of the terms GDBMS(KEYW), MIS(KEYW), GFMS(KEYW); or else all three of the terms GDBMS(KEYW), MIS(KEYW), and GFMS(KEYW).

Example 3.

1.K/ARRAYS,TREES,ALGORITHM,OR

2.K/DATA,STRUCTURE,AND

3.1,2,AND

4.RECURSIVE,3,AND

5.SEARCH,4.

This question requests a search for the word RECURSIVE(TIT) combined with any of the terms ARRAYS(KEYW), TREES(KEYW), ALGORITHM(KEYW) or else with all the three terms, and also such a combination must appear together with DATA(KEYW) and STRUCTURE(KEYW).

1.5. ASSUMPTIONS.

1.5.1. Distribution of title terms according to Zipf's Law.

The frequency with which every different term in each searched attribute field of the data base occurs within the entire data base determines the length of the corresponding list in the appropriate inverted file.

The known vocabulary frequencies of typical document data bases may be used for prediction of the size of the lists in the inverted files. In this connection G.K. Zipf observed that the Nth most common word in natural English language text tends to occur with a frequency inversely proportional to N [3].

Zipf's first law states that if words of any given text are arranged in order, such that the most frequently used word is placed first and has rank one, the next most frequent is placed second and has rank two, and so on, then

$$R * F = C$$

where R is the rank, F is the number of times that the word is used in the text, and C is a constant for a given text. According to previous studies, it appears that this law holds true only for words of high frequency of occurrence in the text.

The frequency of occurrence of a descriptor of rank R is

$$N * Pr,$$

where N is the total number of occurrences of all descriptors in the data base, and Pr is the probability that a term chosen randomly from the text of the appropriate field is a given one of rank R.

For words of low frequency of occurrence Zipf also proposed a second law which has been revised and generalized by Booth [3] in the form

$$I1/In = n(n+1)/2,$$

where In is the number of different words that occur n times in the text, and I1 is the total number of different words that occur only once in the text.

Both of the above generalized expressions have shown excellent agreement between predicted values and observed data. The two different rules may therefore be used to predict the two extremities of word distributions in any given text [3].

It has been hypothesized by M.L. Pao [8] that the most content-bearing words for a given text occur in the critical region between high-frequency and low-frequency occurrence.

1.5.2. Distribution of other search attributes.

1.5.2.1. Keywords and Subject Headings.

Examination of other studies suggests that it may not be too far from reality to assume that the distributions of subject headings and keywords are approximately according to Zipf's law. However the frequencies estimated for the most frequent terms or phrases will not be so high as for title terms since some frequent words, such as stop words, may have been eliminated so that only significant words remain.

One may infer that in comparison to title words there will be fewer terms in both the high frequency and very low frequency areas. In conformity with this inference, Zipf's distribution may be modified so that the probability becomes of the form

$$Pr = A / (R(0) + R),$$

where R(0) signifies the number of ranks to be skipped in order to reach a good approximation value, R is the term rank, and A is a constant given by

$$A = 1 / (\log_e D + Y)$$

Zipf's law implies that a term that occurs exactly n times has rank $R(n) = A*N / n$. However, some terms may occur the same number of times, and within any group of such terms the ordering for the purpose of rank assignment is arbitrary. If it is supposed that the rank given by $R(n) = A*N / n$, applies to the last of the descriptors that occur at least n times, then there are $R(n+1)$ descriptors that occur more than $n+1$ times [3]. Thus the number of distinct terms that has exactly n occurrences is:

$$I(n) = [R(0) + R(n)] - [R(0) + R(n+1)] = R(n) - R(n+1)$$
$$= (AN / n) - [AN / n(n+1)].$$

The highest ranking term has rank (R0+D), and if there is at least one term that occurs only once then

$$R(0)+D = AN / 1.$$

Substitution of AN from the relation $R(0)+D = AN / 1$ then leads to

$$I(n) / [R(0)+D] = 1 / [n(n+1)].$$

1.5.2.2. Author names.

Lotka suggested an empirical law to measure the scientific productivity of authors. According to him, if x authors contribute exactly one publication each then An, the number of authors contributing to n publications, will be given by

$$An = Kx / n ** L, \text{ for } n = 1,2,...$$

where K and L are constants.

Lotka's law, in its generalized form [4], appears to be applicable to the publications of authors within a single periodical. However, when applied to all the publications of authors within various journals, the observed values deviate considerably from the predictions of Lotka's law.

Since there is not any other well-established law for prediction of scientific productivity of authors and it is known that the law does not fit observed results with high accuracy it has been decided not to assume the validity of Lotka's law.

1.6. INVERTED FILE SYSTEM.

1.6.1. Characteristics.

An inverted data base structure consists of a set of files in which, for a given keyword Ki in any searchable attribute, there is an entry in an attribute dictionary that points to the storage location of a list of numbers of all documents that contain the keyword Ki. Thus the logical structure of an inverted index allows access to a data file via secondary access paths, and this generally allows efficient retrieval [14]. Although the concept of the inverted data base has been known for a long time there still exist questions concerning optimal implementation, particularly with regard to procedures for updating [15]. The present approach uses storage cost and access time as a basis for evaluation and design of three inverted data base structure schemes. One of the main factors that affects in the discussion is the average length of the inverted file lists.

The definition of an inverted file does not require the content of each list to be arranged in any particular order. However, to facilitate operations such as conjunction, AND, and disjunction, OR, on any two inverted lists the values of the document numbers are normally arranged in order of magnitude so that AND and OR operations may be performed with one pass through both lists. The penalty paid for such ordering is that the inverted file becomes slower to update.

The specific performance of the inverted index is obviously dependent on the distribution of list lengths and the frequency of access to, and modification of, each list. Also, a physical parameter whose choice affects performance is the bucket length.

1.6.2. Advantages.

Creation of an inverted file and its associated dictionary needs a very simple programming effort, and can result in a very fast response to queries formulated in terms of Boolean expressions. Unfortunately, performance degradation is highly probable as a result of updates, and as a consequence costly reorganisation of the data base frequently becomes necessary [14,16].

1.7. EVALUATION CONSIDERATIONS.

1.7.1. General conception.

Regardless of the goal of an information retrieval system, its ultimate sucess or failure is dependent on the degree of reconciliation of the following two opposing criteria [18]:

- maximum EFFICIENCY in terms of speed and economy of operation, and

- maximun EFFECTIVENESS in terms of its compatibility with the user requirements and preferences.

Much effort and research has been devoted to solving the problem of evaluation of information storage and retrieval systems. Nevertheless it is true to say that most people active in the field of information storage and retrieval still feel that the problem has not yet been solved [10]. However the first consideration in evaluation of a fully automatic and interactive retrieval system should be to measure the benefits, or disadvantages, realisable from it and to estimate the possible cost of its use. In fact, Cleverdon [10] gave the following as some of the measurable quantities concerned with automatic document retrieval systems:

1- the coverage of the data base,

2- the average interval between the time a search request is submitted and the time an answer is received,

3- the form of presentation of output,

4- the effort required of the user in order to obtain an answer to his search request,

5- PRECISION: the proportion of retrieved documents that is relevant,

6- RECALL: the proportion of relevant documents retrieved in answer to a search request.

Precision and recall are two of the most commonly used measures of the effectiveness of a retrieval system.

1.7.2. Efficiency of a system.

In the present investigation an analysis of efficiency as a function of various parameters is undertaken for each of the schemes proposed. The most important parameters considered are:

1- Storage required to store the data base.

2- Number of file accesses needed to locate a specific term.

3- Alterations required when inserting, updating, or deleting terms in the data base.

4- Question logic processing.

Since the data base is envisaged as growing day by day the manner of handling overflow areas, and the size predicted for these overflow areas are among the most important considerations in evaluation of each possible scheme.

CHAPTER TWO.

2. GENERAL DESCRIPTION OF THE PROPOSED DOCUMENT RETRIEVAL SYSTEM.

2.1. FUNCTIONAL DESCRIPTION.

The present thesis is concerned specifically with the structure of the dictionaries of an inverted file organisation as commonly used in on-line document retrieval systems for library applications. As described by J.J. Dimsdale, and H.S. Heaps [5], an integrated sytem for on-line library automation requires a number of computer accessible files, which it proves convenient to divide into the following three principal groups:

- on-line catalog subsystem,
- acquisition subsystem,
- circulation subsystem.

The files for an on-line catalog subsystem should contain all the bibliographic details normally presented in a manual catalog, and the file should be organized to allow searches to be made with respect to title terms, author names, keywords, subject headings, codens, etc. The file organisation should be designed to support economic searching with respect to questions in which terms are connected by the logic operators, AND, OR, and NOT. It should also allow terms to be assigned weights and to be connected by ADJACENCY and PRECEDENCE operators. It may be desirable for the file organisation to include a thesaurus that may be used either directly by the user or by the search program to narrow, or broaden, the domain of the initial query. However, incorporation of a thesaurus will not be taken into consideration in the present treatment.

The file organisation and search strategy should be chosen to ensure that the user of the on-line catalog system receives an acceptable response time to his query, although it must share computer time with circulation transactions, other tasks originating from separated terminals, or from batch input. Such tasks might be generated by acquisition operations or by update and maintenance of the on-line catalog.

Since an on-line catalog is a large file, and hence expensive to store in computer accessible form, it is desirable to store it in as compact a form as possible. Among the problems associated with modern information retrieval systems is the lack of any systematic approach and lack of suitable tools for system development [20]. For large systems the amount of programming and maintenance required to produce an effective system is considerable. Attempts have been made to develop generalised information retrieval systems; however, they have a very limited capability and a somewhat inflexible file organisation.

The approach of the present investigation is based on the following model of an information retrieval system:

1- Whenever any query appears, an evaluation of its grammatical structure is performed. If its construction is correct then processing starts; if not, a message is sent from the system to the user and no operation is performed.

2- The process starts once the command "SEARCH" is found. As a query is input, each entry is added succesively to the top of a stack. Every end of a logical level is followed by a return to a higher level of logic in the question statement. It is marked by placing a flag in the logic stack.

3- Terms are read from the top of the stack and placed in a temporary stack until a logic element is reached. Descriptors are searched for in the dictionaries, and inverted files are accessed in order to obtain their document lists.

4- A set of four stacks is required to process the logic applied to the document lists. The logic processes two document lists at a time, and all lists contain document numbers in ascending order. The result of the operation is a list located in any of the free stacks, and deletion of the two initial lists. After a term has been located in its dictionary, and the inverted file has been accessed to obtain the corresponding document list, logic processing is applied again to the new term and the list at the top of the stack. The same iterative processing continues until the end of the query statement is reached.

5- The stack that contains the final result contains a set of temporary document numbers. They are regarded as temporary since a further search for dates and/or codens may be required.

6- The last step is to access the coded sequential file to print the output in textual form. Codes are of restricted variable length form in order to combine some of the benefits of the Huffman codes with some of the benefits of a fixed length code

[3]. The codes also represent the corresponding addresses of the term positions in a dictionary. Thus to decode the coded terms it is necessary to access the dictionary.

For a computer with a 6 bit representation of characters the coding process may be described as follows: Suppose the most frequent 2**5 (=32) terms in the dictionary are assigned different codes, each of six bits ~~of~~ which the first bit is always set equal to 1. The next most frequent 2**10 (=1,024) terms in the dictionary are assigned different codes of length 12 bits in which the first bit is always 0 and the seventh bit is always 1. The next most frequent 2**15 (=32,768) terms in the dictionary are assigned different codes of length 18 bits in which the first and the seventh are always 0 and the thirteenth is always 1. The rest of the terms in the dictionary are assigned 24 bit codes in which the first, the seventh, the thirteenth are always 0 and the nineteenth is always 1; this implies that a maximun of 1,048,576 codes may be assigned to the remaining dictionary terms [5].

2.2. CONCEPTUAL LEVEL DEFINITION OF THE ARCHITECTURE OF THE DATA BASE.

The stored data is shared among several different files in an attempt to minimize the redundancy of the information and the storage space.

Dictionary files for each attribute are formed as follows: Descriptors are grouped as logically ordered sets, and each descriptor is stored with a pointer that points to the list of documents in which it appears. These sets are stored sequentially as blocks.

> Definition: a block is the amount of data physically transferred between secondary storage and main memory as a result of a single I/O operation, and its size is usually given in bytes or characters.

For each attribute there is an inverted file that contains lists of document numbers for each descriptor in that attribute. Every list has a field that indicates its length, and the lists are usually stored sequentially in a chain of blocks.

A document directory file is needed in order to access the coded sequential file to allow production of output in textual form and also to store, and allow search of, other information (date of publication and journal code) in order to increase the specificity of questions. Records of this file are stored in physical order according to the values of their respective identifiers (documents numbers).

A coded sequential file is used to store the textual-code form of each document of the collection. It is arranged in a sequential organisation wherein records are filled in order of arrival.

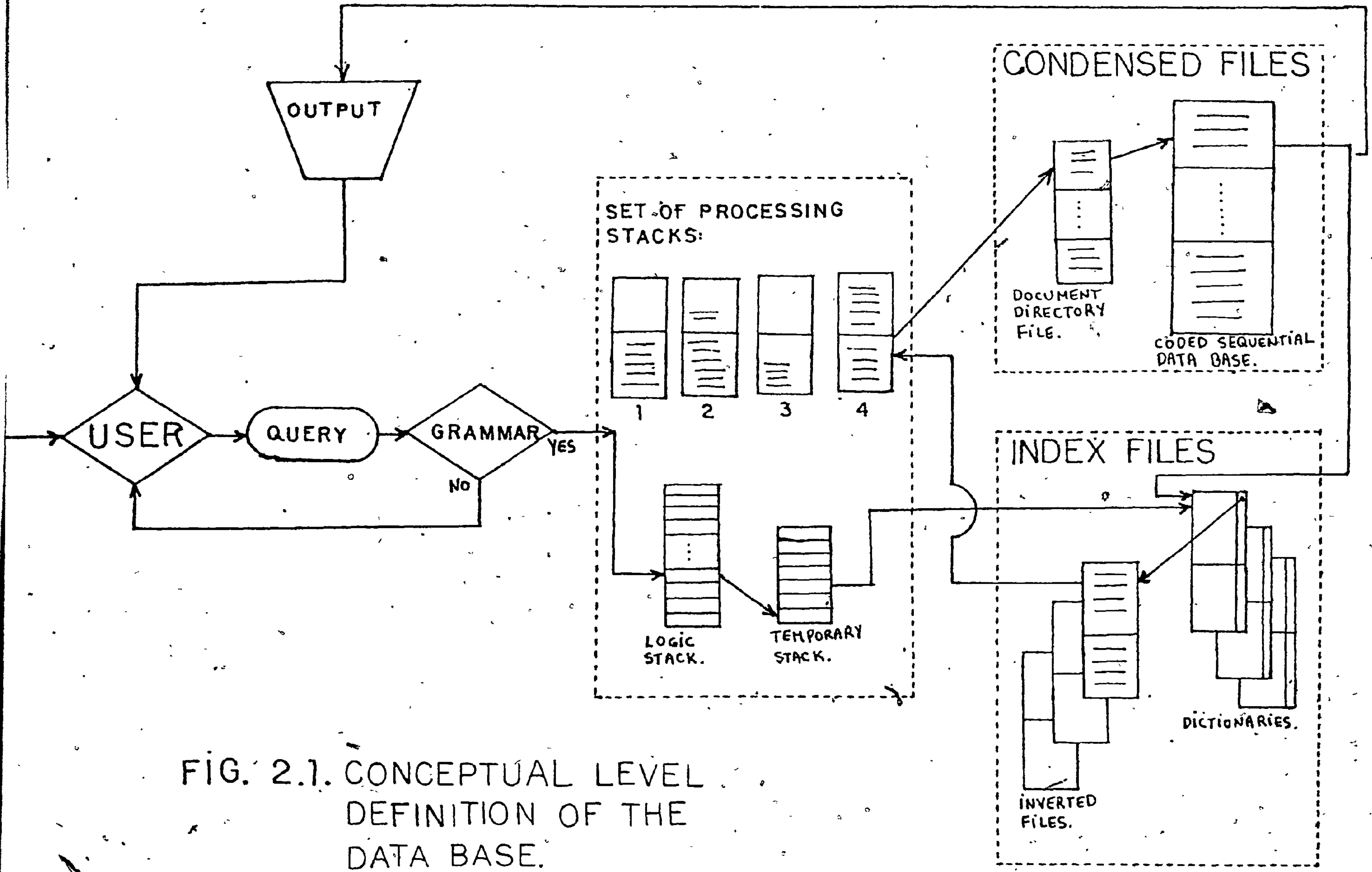


FIG. 2.1. CONCEPTUAL LEVEL DEFINITION OF THE DATA BASE.

2.3. CONSIDERATIONS OF GROWTH OF THE DOCUMENT COLLECTION.

2.3.1. Significant factors that affect performance.

A static document data base is very unlikely in the real world, since libraries continue to grow. The need to add new documents is considered to be an important factor that impairs the performance of the file structure used in any system. As additions are made to a data base, the search cost is liable to increase unless the data base is reorganised. Nevertheless, since the cost of performing a reorganisation is relatively high, the frequency of reorganisation should be kept low. Reorganisation may be performed either at fixed time intervals or when the average search cost has deteriorated to a certain level. A comprehensive study of reorganisation is not contemplated in the present investigation, but some significant decisions must be made including the following:

1- selection of a suitable algorithm to handle overflow file areas;

2- selection of a time interval or critical level at which to perform reorganisation.

The present estimation of the rate of expansion of the data base is based on some statistics adopted from existing document retrieval systems actually in service. Typical increases range among the following values : 100,000 new articles per year in a large library; 50,000 new articles per year in a medium one; and 25,000 new articles per year in a small one. The reason for quoting such values is to allow illustration of how critical situations will be handled with the schemes proposed using typical values.

2.3.2. Estimated values.

The length of the sample data base may be estimated by considering the typical vocabulary characteristics of a document retrieval system. It may be predicted that, although the individual terms occur with different frequencies in different data bases, the overall statistics are similar [3]. In order to allow useful estimates of various factors, a data base of 1,000,000 title terms (N) is selected and it is supposed that of this one million there are approximately 40,000 different title terms (D).

According to observation of document descriptions it may be predicted that the following are averages for general English text: 8.4 title terms/document, 1.3 authors/document, and from 2 to 10 keywords and subject headings per document.

Dividing 1,000,000 title terms by 8.4, gives an approximate number of documents as 119,047. In summary the initial size of the assumed data base is as follows:

	Initial total number	Different words
Title terms	1,000,000	40,000
Author names	154,761.9	13,768.9
S.headings Keywords	(238,095.2; 1,190,476)	(17,078.2; 38,188.1)

As new document articles are added to a data base the vocabulary of the attributes may expand in size but, because of repetitions of terms, the rate of growth is likely to be less than that of the total text length. As mentioned by H.S. Heaps [3], for general English text of up to at least 20,000 words it is found that as the text length increases the number D of different words is related to the total number N of words by an equation of the form

D = k*N**b,

where k and b are constants that depend on the particular text [6]. For estimation purposes the values have been chosen as k=35 and b=0.5.

Approximate predictions of yearly attribute vocabulary behaviour may be obtained by applying the above formula and considering a rate of growth in the order of 100,000 new documents every year with values of k=35 and b=0.5. Then the following table is applicable:

	FIRST YEAR		SECOND YEAR	
	N	D	N	D
Title	1,840,000	47,476	2,680,000	57,297.4
Author(*)	284,762	18,677	414,762	22,540.7
Keywords & S.headings	(438,095; 2,190,476)	(23,166; 51,801)	(638,095; 3,190,476)	(27,958; 62,517)

The number of document numbers in the inverted files increases linearly as new document items are added. The length of each list of document numbers will also increase linearly.

However, the number of very infrequent terms will increase according to the growth of D, and therefore more slowly than the growth of N [3]. Considering the Zipf's law it may be predicted that 50% of the different title terms occur in only one document and 16.7% occur in only two documents. Thus, in the first year with D=40,000 there will be approximately 26,680 very infrequent terms (40,000*.5=20,000 + 40,000*.167=6,680). The second year, with D=47,476, there will be 31,666 very infrequent terms. Subtracting 26,680 from 31,666 gives 4,986 as the number of additional infrequent terms encountered during the first year. From these estimated values it may be inferred that approximately 66.69% of the new title terms will be very infrequent ones. Such an inference will be considered later during the analysis of the behaviour of the schemes proposed.

2.4. CREATION OF THE DOCUMENT DATA BASE.

2.4.1. Source of the information.

The processing required to implement an inverted data base system depends on the prime source of the information. However our main objective is not to choose any specific application, but to present a feasible and generalized algorithm that specifies linkages among files and a suitable structure during the creation phase.

2.4.2. Algorithm.

For a set of document records that contain searchable attributes the following steps must be performed:

1- Since it is likely that the source data will be obtained from more than one source, the first step is to merge all the existing sources and to generate only one, called the new source file, with the document records in ascending order of document number.

2- To create each dictionary and its corresponding inverted file for each searchable attribute according to any particular file structure scheme, different subsets of the source file must be generated, and each subset must contain the document number and value of the appropriate attribute.

3- Each subset is sorted with respect to the value of the attribute.

4- Examine the records of each subset and perform the following twofold task: Create the inverted file and, whenever the attribute value changes, create an intermediate dictionary with the following information: attribute value, number of times this value appears in the attribute text, and the address where its corresponding inverted list is stored.

5- To segment dictionaries according to any particular proposed scheme it is necessary to sort the intermediate dictionaries with respect to the number of occurrences of each attribute value in descending order, so that the most frequent attribute value is stored at the first position of the dictionary, the second most frequent is stored at the second one, and so forth.

6- To generate the document directory file and the coded sequential file the new source file must be accessed again, since the code of a term is dependent on the rank of the term in its dictionary.

Description of overflow area structures, and their appropriate handling, will be presented in a later section.

2.4.3. Code generation.

Space to store the sequential file is saved by use of the coding scheme presented in section 2.1. The values of codes are generated according to the corresponding addresses of term locations in a dictionary whose organisation is governed by the rank that each term occupies in a specific attribute dictionary. Such rank is dependent on the number of occurrences of the term in a specific field.

2.5. QUESTION LOGIC PROCESSING. (Q.L.P.)

2.5.1. Basis for support of Q.L.P.

As shown in section 1.4.6 the description of the user's question is expressed in terms of logic operations applied to pairs that consist of terms preceded by an attribute descriptor that specifies the field within which the term is to be searched. Processing a user's question and searching through the document data base normally require sets of terms, logic operators to be placed in temporary lists that are implemented as pushdown stores (stack), and delimiters. Only the last element of a list may be added or removed as the processing proceeds. Only one item, located at the "top" of the store, is accessible at any given time; new items are placed on the top of the items already stored, the latter being "pushed down" in the process. A pointer identifies the top of the store [13].

2.5.2. Implementation method.

Analysis of a given question sentence proceeds from left to right and location of entries of the query in the logic stack starts after the command "SEARCH" is found. As a question is input, entries are added succesively to the top of a logic stack. Each end of a logic level that is followed by a return to a higher level of logic in the question statement causes a flag, denoted by "?", to be placed in the logic stack. Terms are read from the top of the logic stack and placed in a temporary stack (tempstk) until a logic element is reached.

e.g. For a question in the form

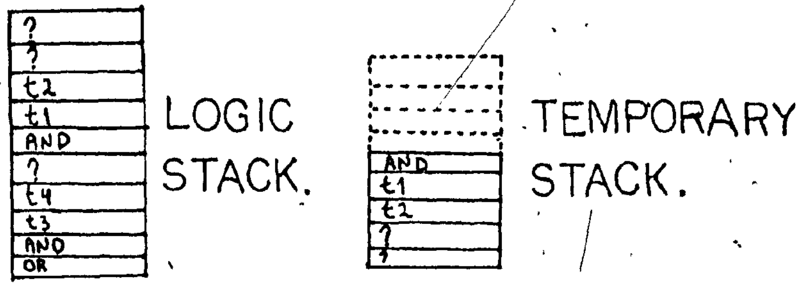
1.t3,t4,AND

2.t1,t2,AND

3.1,2,OR

4.SEARCH,3.

the logic stack and temporary stack are initially in the form:



Similarly for the question

OR AND T3 T4 ? AND T1 T2 ? ?

After a term has been located in its dictionary, its inverted file must be accessed to obtain the appropriate document list. This list must be arranged in ascending order before it is processed further. The document lists are processed logically by pairs and therefore partial results appear and are placed on the temporary stack (tempstk) in order to continue the logic processing if required. Again terms are read from the top of the logic stack and placed in the temporary stack (tempstk) until another logic element is reached. The processing continues until the final result is found. A set of four stacks is used to process the document lists.

It is difficult to determine how much space should be allocated for the storage of the stacks. However, this difficulty may be resolved by reserving a relatively small amount of storage in core and creating a backup stack that is stored on disc. Each stack will be partitioned between core storage and backup stack. In order to avoid frequent transfers to and from the backup store the stacks will be organised in the following manner [3]. Suppose that every stack in core has a capacity to store 1,000 numbers; when the stack length becomes larger than 1,000 the lower half L1,L2 of the stack (stack1) is transferred to its backup stack; thus the stack becomes of form (stack2) and L1=501 and L2=1000, also L3=1 and L4=500. Whenever it is required to add further document numbers to the stack the portion from L1=501 to L2=1,000 is first transferred to the backup stack so that the stack then has the same form as during its initial phase. Initially a pointer (topstack) to the top of the stack is set to "0", a counter (backup) is set to "0" to indicate that no portions have yet been transferred to the backup stack on disc, and a value of a field, called (free), is set to "1" to indicate that the in-core stack is empty.

Processing method:

1- A list is read from the inverted file, and its document numbers are placed in ascending order in stack-one.

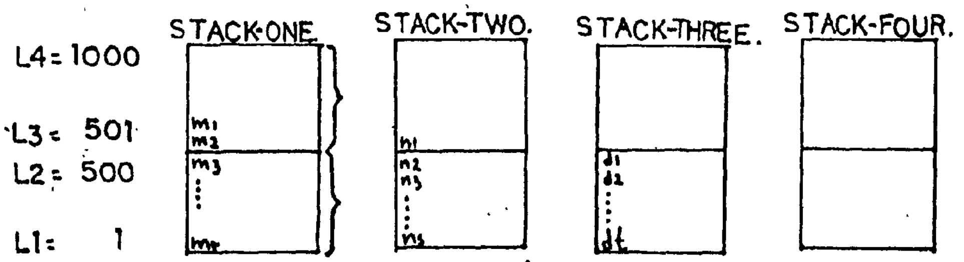
2- The same process is applied to the second list to place its document numbers in stack-two.

3- The logic operation is executed on the two lists and the resulting list is stored in stack-three.

4- At this point, the spaces used to store stack-one and stack-two are released in order to allow their use for other logic operations.

5- The values of stack-three are stored or combined with other lists as needed.

6- Finally, the stack that contains the result of merging all lists by appropriate logic operations will contain the document numbers relevant to the specific question.



2.5.3. Handling of term weights.

Some terms may have weights which must be added to form a sum that is compared with a threshold value, placed to the left of its immediate logic operator, to determine if it is as large as the threshold weight of the question. Calculations to check weights specified in questions are included in the process of applying question logic at each level.

2.5.4. The operators PRECEDENCE and ADJACENCY.

These operators may be treated as a single "AND" operator during the processing of lists and during the decoding and output phase of the relevant items. PRECEDENCE and ADJACENCY operators may then be applied to the retrieved records in order to reject documents that do not satisfy the required PRECEDENCE or ADJACENCY logic.

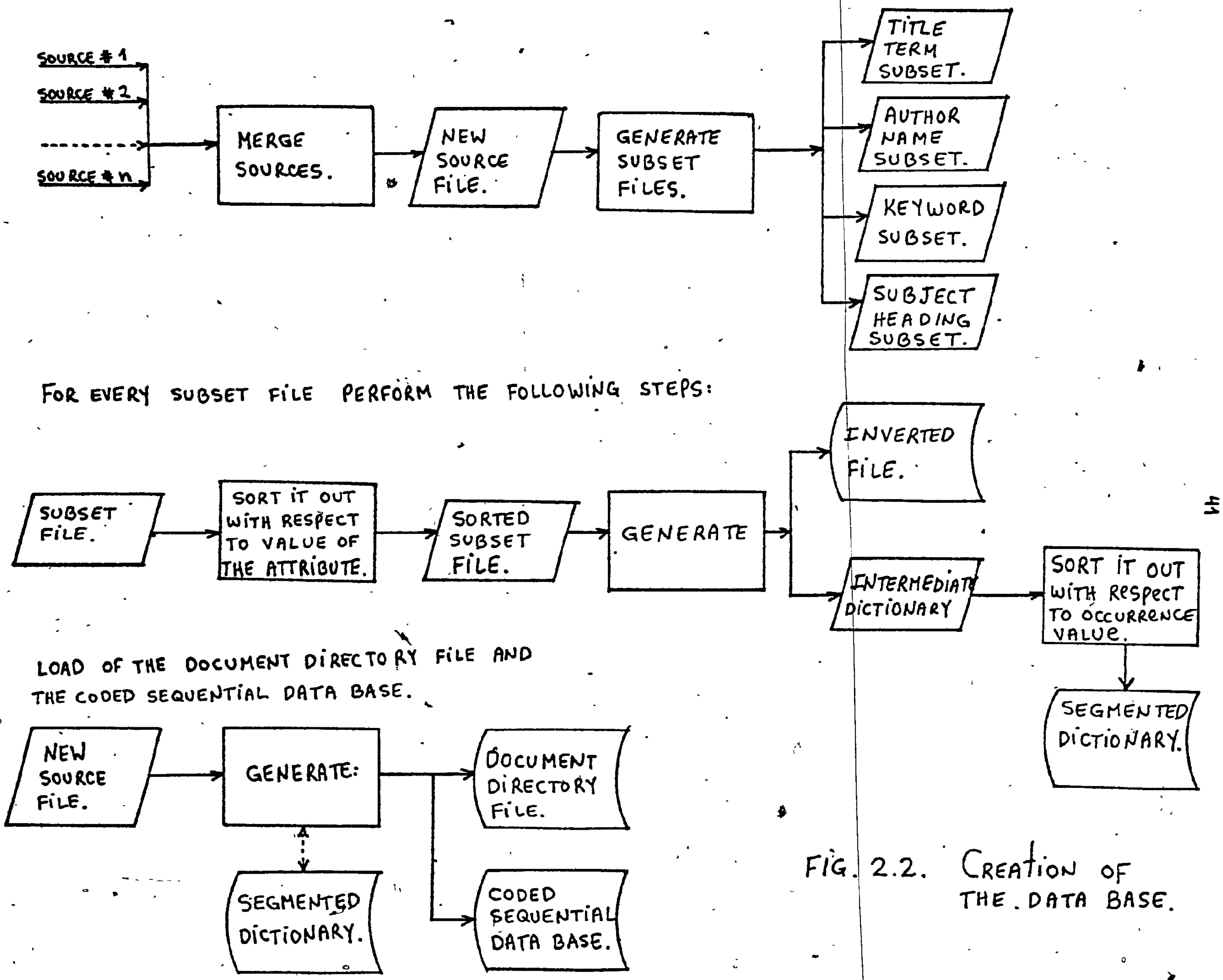


FIG. 2.2. CREATION OF THE DATA BASE.

CHAPTER THREE.

3. ILLUSTRATION AND EVALUATION OF THREE DIFFERENT SCHEMES.

In the present chapter a description is presented of the search strategies and storage schemes for the three retrieval procedures proposed in this thesis. In describing the modelling of the search strategies it is of particular interest to describe the number of blocks transported (located and transferred) from secondary storage to main memory in the course of an index search. The number of blocks transported may be represented by a random variable denoted by J. The distribution of J is, of course, a function of the file parameters as well as the search strategy. Once J has been defined, it remains only to model the secondary storage devices. For the purpose of this investigation it is supposed that secondary storage is accessed through a moving arm device such as a disc unit. The retrieval time is a function of the file parameters, the characteristics of the storage media, and the overflow strategy employed.

Throughout the analysis it is assumed that there are no delays caused by I/O queues. Organisation of dictionaries is accomplished by ordering according to the number of occurrences of each descriptor in any specific attribute, so that the most frequent descriptor is stored at the first place in the dictionary, the second most frequent one is stored at the second place in the dictionary, and so forth. However, it is evident that it may not be possible to store 40,000 different descriptors in main memory, and so segmentation of dictionaries is implemented in order to store some part of the dictionary in core and the rest in secondary storage. The segmentation is performed after consideration of the frequencies of occurrence of descriptors in the relevant field, so that some of the most frequent descriptors are stored in core and the rest are stored in secondary storage that is shared with other segments.

The three proposed schemes present three different organisations of dictionaries, but the inverted files, document directory file and coded sequential file remain unchanged. However, a short description of proposed updating procedures is presented. Also, a brief view of how updating of these files may affect the overall performance of the three schemes is discussed.

A general survey of the proposed retrieval process for each scheme is presented and described under the following plan:

3.1. File structure of dictionaries.

3.1.1. Segmentation of dictionary.

3.1.2. Space required to store segments. Bucket capacity.

3.1.3. Search logic and handling of overflow buckets.

3.1.4. File accesses and required comparisons in core.

3.1.5. Statistical distributions of descriptors in queries.

3.2. Dictionary updating.

3.2.1. Updating algorithm and procedures to handle overflow areas.

3.2.2. Retrieval performance. (Reorganisation)

3.3. Evaluation. Advantages and disadvantages.

SCHEME NUMBER ONE.

3.1. File structure of dictionaries.

As mentioned in chapter two, it is assumed that the four dictionaries are organised according to the rank of the frequencies of occurrence of descriptors in each attribute.

3.1.1. Segmentation of dictionaries.

1- Segment one.

Since it is assumed that the 10 most frequent title terms (often used as stop words) provide very little information about document content it has been decided that they will not be allowed in any search request, therefore no inverted lists are stored for them. They will be stored in core in order to allow rapid decoding of such terms during the output phase. They may be stored in core in alphabetic order in order to allow rapid execution of a binary search whenever it is required.

It is realised that the assumption of negligible information content may not be true for the most frequent words of the other attributes, since stop words may not exist. Thus for the other attributes the inverted lists for the 10 most frequent descriptors are created and may be used in searches. The words are stored in core as described above.

2- Segment two.

The next 500 most frequent descriptors in each attribute are stored in core in alphabetic order.

3- Segment three.

Zipf's distribution is assumed to apply to the following attributes: subject headings, title terms, and keywords. Thus the number of different terms that occur exactly n times may be predicted as follows. The Zipf's law implies that a term that occurs exactly n times has rank

$R(n) = (A*N)/n$.

However, several descriptors may occur the same number of times, and within any group of such descriptors the ordering for the purpose of rank assignment is arbitrary. If it is supposed that the rank given by $R(n) = (A*N)/n$, applies to the last of the descriptors that occur at least n times then there are $R(n)$ descriptors that occur n times, and there are $R(n+1)$ descriptors that occur more than n+1 times. The number of different descriptors that occur exactly n times is given by [3]

$I(n)=R(n)-R(n+1)=(A*N)/(n(n+1))$

If D is considered to be the rank of the highest ranking term, and if there is at least one term that occurs only once, then

$D=(A*N)/1$.

Substitution of D for (A*N) shows that $In/D = 1/(n(n+1))$. Thus

for $n=1$, $In/D= 1/(1(1+1)) = .50$

for $n=2$, $In/D= 1/(2(2+1)) = .167$

According to this result it may be supposed that approximately 50% of the different title terms occur only once in general English text, and 16.7% of the different terms occur only twice, so the number of terms stored in the segment three may be estimated by calculating the numbers of terms that occur once and twice in the text, adding 510 terms from segments one and two, and subtracting this total value from the initial number of different terms (D).

As mentioned in section 1.5.2.2, it is known that Lotka's law may not be used to estimate values with high accuracy in the author name attribute, and since there is not any other well-established law for prediction of the scientific production of authors, it has been decided to make the same assumption as above for all the other attributes. Furthermore, it is believed that the assumption may not greatly affect the performance since most of the authors probably appear relatively few times in the data base. The following table shows the approximate number of terms in sements three and four:

	INITIAL D	50% OF D	16.7% OF D	SEGM. 4	SEGM. 3
title	40,000	20,000	6,666	26,666	12,812
keyword & s.heading	(23,166; 51,801)	(11,583; 29,900)	(3,868; 8,650)	(15,451; 38,550)	(7,515; 13,251)
author	18,677	9,338	3,119	12,457	6,620

Segment three of the dictionary contains terms arranged in alphabetic order and stored in secondary storage according to the following criteria: An index table is stored in core and contains 26 elements, each associated with one of the letters "A" to "Z". Each element of the index table stores a pointer that points to a block that contain terms beginning with the corresponding letter.

4- Segment four.

As mentioned by R. Morris [21], the fundamental concept of scatter storage is that the key associated with the desired term is used to locate the term in storage. Some transformation is performed on the key to produce an address in a table that stores the key and the term associated with the key. A good transformation is one that spreads the calculated addresses uniformly across the set of available addresses. If the calculated address is already filled with some other key, because two keys happened to be transformed into the same calculated address, then a method is needed for resolving the collision of keys. For purposes of analysis it is assumed that the addresses generated by the hash functions for the L "randomly" chosen terms are independently and uniformly distributed on the set.

The terms that appear only once or twice in the text of a particular attribute are stored in secondary storage. These infrequent terms are clustered in buckets in order to take advantage of collisions that result in calculation of the hash function.

3.1.2. Space required to store segments. Bucket capacity.

1- Segment one:

10 terms * 15bytes/term is equal to 150 bytes in core.

2- Segment two:

500 terms* (15bytes/term +3bytes/pointer to inv.file) = 9,000 bytes in core.

3- Segment three.

Three different block sizes 3,000, 4,000, 5095 have been simulated with 3 different fixed term lengths, 12, 15, and 18 respectively, in order to determine how the average number of file accesses may be affected. Since terms will be grouped according to the first letter in a term, it may be predicted that there are different numbers of records for each letter. For the arrangement of segment three in a computer accessible form it may be useful to know how many terms begin with a given initial letter. Several studies have been performed on general full text in the English language. Considering the proportions of different words that begin with a given letter, as given by the study of Dewey [3], it may be estimated how many terms may contain each letter group. Thus an estimate of a functional record size and block size may be made in order to produce a prediction of the average number of file accesses needed to find a term in the segment.

A block size of 4,000 bytes, a fixed term length of 15 bytes, and 3 bytes for each pointer to the inverted list are selected as reasonable values.

Variable length records may be chosen with a maximum length of 4,000 bytes in order to save some space. Since a binary search is applied to blocks that contain terms beginning with a given letter, then in order to reduce the number of file accesses during the search for a term it is essential to store an in-core index table that contains the estimates of the number of blocks in each letter group required.

The space required to store such a segment for ~~the~~ four dictionaries is shown in the following table:

	NUMBER TERMS	BYTES NEEDED	NUMBER BLOCKS
title	12,812	230,616	72
keyword & s.heading	(7,515; 13,251)	(135,270; 238,518)	(49; 74)
author	6,620	119,160	45

4- Segment four.

Choosing an initial block size of 4,000 bytes, a fixed term length of 15 bytes, and a 3 byte pointer, allows space estimates to be calculated as follows:

	NUMBER TERMS	BYTES NEEDED	NUMBER BLOCKS
title	26,666	479,988	120
keyword & s.heading	(15,451; 38,550)	(278,118; 693,900)	(69.5; 173.4)
author	12,457	224,226	56

Dividing 4,000 bytes by 18 bytes/term, indicates that 222.2 slots are contained in each block. Terms may be clustered within blocks by taking advantage of the fact that collisions result by application of a hashing function, called hash-one. Once the appropriate block is accessed, another hashing function, called hash-two, is applied so that the new transformed addresses may be uniformely distributed among the s slots available in the block. Nevertheless, the inclusion of an overflow area for each block is essential to handle collisions. The size of a feasible overflow area may be calculated by taking into consideration the following [22]: Let Ti be the random variable that represents the calculated address of the ith term to be stored. It is assumed that (Ti) i = 1,2, ... is a sequence of independent, identically distributed random variables with

Pr(Ti=q) = 1 / Q for all i and q,

where Q is the number of buckets. It is assumed that the total number of records to be stored is m, and hence the load factor is L = m / Q . Let Vk be the random variable that represents "the number of records whose calculated address is k". Then Vk has a binomial distribution. As m and Q tend to infinity, and L remains constant, then Vk tends to the limit of a Poisson distribution. In other words, the probability that a real address corresponds to just k keys is the following as given by Murray [5]:

Pr(corresponding to just k keys)= $e^{-L} * L^{K} / K!$

The probability that a collision occurs when the function hash-two is applied is

$$\Pr(\text{collision}) = \sum_{K=2}^{m} \Pr(k) = 1 - (\Pr(0) + \Pr(1))$$
$$= 1 - ((e^{-L} * L^{0} / 0!) + (e^{-L} * L^{1} / 1!))$$
$$= 1 - e^{-L} * (1 + L)$$

If the load factor (L) is equal to 1 then

Pr(collision) = 1 - (.3678 * 2) = 1 - .7356 = .2642

Using the above formulas the probability of the size of the overflow area may be estimated as

222.2 slots/block * .2642 = 58.70. This value (58.7) is the

expected number of slots that may need to be added to the initial block size in order to store the term values of the expected collisions in the overflow slots of the appropriate block. Thus, the number of slots in each block is 281.
(222.2+58.7 = 280.9 = 281)

Since the new number of slots per block is 281 then the new block size becomes 5,058 bytes. Every slot in each record, except those of the overflow area, also stores one bit that indicates whether or not there is any collision and hence whether a search in the overflow area must be performed. If 120 different groups are employed to cluster terms by use of the hashing function hash-one, the number of overflow bits is (222.2 slots/block * 120) = 26,666 bits for the title term dictionary, from 15,451 to 38,550 bits for the keyword and subject heading dictionaries, and 12,457 bits for the author dictionary.

3.1.3. Search logic and handling of overflow buckets.

Assuming that question term distribution is according to Zipf's law, it may be predicted that there will be a tendency for search terms to be located in the area between very infrequent terms and very frequent ones [9]. To search for a term in these segmented dictionaries it is necessary to search in all four segments only in the worst case. The strategy used to perform a search is therefore as described as follows:

1- Segment one.

Whenever a term is to be searched for, a binary search is first applied to the 10 terms of the segment. If the term is not found it is necessary to search for it in segment two. The maximum number of comparisons needed to find a term present in segment one is therefore

cell $\log_2$ (10) = cell (3.3219) = 4

2- Segment two.

The 500 terms of the segment two are also stored in core in alphabetic order, and to locate a term in this segment a binary search is also applied. The average maximum number of comparisons is given by

$\log_2 (500) = \log_e 500 / \log_e 2 = 8.96$

3- Segment three.

When a term has not been found in segment one or segment two then the in-core index table must be accessed to find the pointer that provides a link to the block that contains terms beginning with the particular letter. As it is supposed that the number of records needed to store all the terms beginning with a particular letter may vary among different letter groups, a binary search is applied to the number of blocks needed for a specific letter stored in the index-table. Once a block is brought into core a comparison is performed between the term being searched and the highest ranking term in the block in order to determine if the search term is present. If the highest term of the block is found to be of higher rank than the searched term then another higher ranked block must be retrieved by the binary search. Otherwise, if the result of the comparison is "less than" another comparison must be performed with the lowest ranked term in the block. If the result is also "less than" it is required to access a lower ranked block. On the other hand if the result is "more than" then the binary search must be applied to the terms of this block.

In estimation of the average maximum number of both file accesses and comparisons in core needed to locate a term in this segment the block size is a significant factor, particularly with regard to transfer time. Examination of the previous section 3.1.2, showed that a block size of 4,000 bytes, and a fixed term length of 15 bytes plus 3 bytes for the pointer to the inverted file, provides an average maximum number of file accesses of 1.65. On the other hand, the average maximum number of comparisons in core for any attribute is approximately (1.5 comparisons * 1.65 accesses) + $\log_2$ (40,000/18) = 2.47 + 7.79 = 10.26

4- Segment four.

Any key may be operated on by a hashing function, hash-one, which transforms it into a pointer to an entry in an index-table in core. This index-table contains pointers to blocks that contain terms with the same key address. Thus advantage is taken of the collisions generated by hash-one in order to cluster infrequent terms. Once a block is brought into core, it is necessary to hash again, but this time hash_two is used in order to locate the term being searched for in the block. If the collision bit is found to be "1" in the corresponding address generated by hash-two, and the term is not the same, then the overflow area must be examined slot by slot. If the collision bit is not "1" the term may not be retrieved without the need for further examination.

The proposed hash coding scheme uses the DIVISION method [23], which is a well-known, and frequently used, technique in which the key is divided by a positive integer that is often chosen as a prime number. In the division method the remainder obtained from the division becomes the address for the key. The divisor is therefore chosen to be approximately equal to the number of available addresses. The technique adopted to accommodate overflow terms is to store them in a separate overflow area. A combination of letter values in the search term are added, and for hash-one the resulting value is divided by the number of available blocks. For hash-two it is divided by the number of slots in an available block. The remainder of the first division (hash-one) determines the block to be accessed, while the remainder of the second division (hash-two) determines the location of the search term within the block. For use of the hash-one function it is supposed that groups of terms map to the same block. However, for the hash-two function it is supposed that collisions may be avoided, and so the choice of the number of slots per block is a critical factor. According to several research studies performed in order to obtain a desirable overall performance, it is advisable to choose the largest prime number that is close to, but less than, the size of the address space. This has been suggested by Buchholz [23], who has indicated that uniformity in the distribution of addresses is not synonymous with the mapping of a key into addresses with equal probability. Consequently an efficient transformation method should attempt to preserve whatever uniformity exists in the keys.

It may be concluded that in order to locate a term in segment four it is necessary to perform only one file access, two operations in core, and a few comparisons in core whenever a collision is found.

The storage required to store the dictionaries of the data base, given in bytes of 6 bits, is shown in the following table:

	seg. 1		seg. 2		seg. 3		seg. 4	
	core	disk	core	disk	core	disk	core	disk
title	150		9,000		156	230,616	369	479,988
keyword & s.heading	150		9,000		156	(135,270; 238,518)	(210; 522)	(278,188; 693,900)
author	150		9,000		156	119,160	168	224,226

3.1.4. File accesses and required comparisons in core.

	seg. 1		seg.2		seg. 3.		seg. 4	
	f.a.	c.c.	f.a.	c.c.	f.a.	c.c.	f.a.	c.c.
title		3.32		8.98	1.65	10.26	1	(1;max. 61)
keyword & s.heading		3.32		8.98	(1.27; 1.69)	(9.69; 10.32)	1	(1;max. 61)
author		3.32		8.98	1.23	9.63	1	(1;max. 61)

f.a.=file accesses and c.c.=comparisons in core.

3.1.5. Statistical distributions of descriptors in queries.

Supposing that question terms are distributed according to Zipf's law, the following relations may be assumed:

$$\Pr(\text{find a term in core}) = A * \sum_{i=1}^{510} (1/Ri)$$

$$= A * (\log_e 510 + .5772) = .6096$$

where A=.08949 for D=40,000. This value of D corresponds to N=1,000,000.

$$\Pr(\text{find a term in segm.3}) = \Pr(\text{find it in 13,324 most freq.}) - \Pr(\text{find it in 510 most freq.})$$

$$\Pr(\text{find a term in 13,324 most freq.}) = A * \sum_{i=1}^{13324} (1/Ri) = A * (\log_e 13{,}324 + .5772) = .9016$$

$$\Pr(\text{find a term segm.3}) = .9016 - .6096 = .2920$$

Thus, to summarise the behaviour of the four dictionaries:

	prob(find a term)
SEGMENT ONE	.2577
SEGMENT TWO	.3519
SEGMENT THREE	.2920
SEGMENT FOUR	.0985

3.2. Dictionary updating.

3.2.1. Updating algorithm and procedures to handle overflow areas.

It is assumed that no deletion of any term will take place under any circumstance, and also that most of the new terms to be included in the dictionaries are infrequent terms. Therefore, in order to avoid a frequent reorganisations of segment three it has been decided to include all new terms in segment four so that the other segments remain unchanged.

There exist obvious considerations to be examined before choosing an update algorithm. For any particular installation it may be important to either keep the cost of storage low without regard to response time, or to try to obtain the minimum response time without regard to the storage cost. The updating algorithm described here is based on the assumption that the later consideration is the most important.

As shown in section 3.1.2, each block of segment four was enlarged in size by 26.42% because of the expected number of collisions. The operation of the update algorithm may be explained as follows:

1- Whenever a new term is to be added to the dictionary then both functions hash-one and hash-two are applied in order to determine if there is any free slot available in the initial block area. If no such space is available, either at this specific address or at the overflow area of the initial block, then the following procedure must be applied.

2- Another alphabetic index of 26 elements is stored in core. Each element contains the address of the corresponding block, or group of blocks, needed to store the new terms that begin with a specific letter. The new terms are stored in these overflow blocks in order of arrival, and therefore every time an access to a block, or a group of blocks, is performed a sequential search must be applied in order to search for the term. Each initial block of segment four stores a tag to specify whether a new overflow area has been allocated for new terms, and also whether a further search must be performed.

3.2.2. Retrieval performance. (Reorganisation)

As new terms are included in the new overflow blocks the retrieval performance degrades because of the increase in both the number of file accesses and in-core comparisons required to search for a term. However, since question terms are supposed distributed according to the Zipf's law, it may be predicted that the probability of access to records of segment four may be kept low unless the new terms have a high probability of occurrence in question terms. It was inferred in section 2.3.2, that 66.69% of the new terms may be very infrequent. On the other hand, it is certain that the dictionaries will continue to grow until it is considered necessary to reorganise the data base because of degradation of the response time and the overall performance of the retrieval system. If a block size of 5,058 bytes is chosen, it may be concluded that, in the worst case, for each newly filled block it is required to make a maximum of 281 comparisons in core whenever a search is performed in one of the new overflow blocks.

3.3. Evaluation. Advantages and disadvantages.

The main advantages of the proposed overall dictionary structure are as follows:

1- Since the question search terms are assumed to be distributed according to the Zipf law it may be inferred that segment two and segment three are the most frequently accessed because of the assumed significance of their terms as question terms. Then, as presented previously in section 3.1.4, the average value of the maximum number of file accesses is 1.65 for titles terms, from 1.27 to 1.69 for keywords and subject headings, and 1.23 for author names. All these averages remain unchanged during the operation of the retrieval system.

2- Use of segment four allows the dictionaries to grow freely for as long as desired. Consequently, deterioration of the retrieval process increases in proportion to the number of new terms. Another possible disadvantage is that if there is a group of new terms that subsequently become very significant as question terms, the fact that they are stored in the new overflow areas causes degradation of the retrieval performance. As proposed previously, the new terms are stored in the new overflow area in sequential order of arrival and with regard to the initial letter of the term. If it is assumed, as in section 3.2.1, that the number of terms that begin with a given letter may be estimated then a prediction may be made of the number of new blocks to be added to segment four during the course of one year. A calculation may be made to determine the average of the maximum number of both file accesses and comparisons in core. In section 2.3.2 it was estimated that for the title terms there will be an increment of 7,476 new terms during the first year of operation. It is supposed that not all the new terms are to be stored in the new overflow blocks, since some of them may be stored in the free space found in one of the initial blocks of segment four. In order to obtain an estimate of the average of the maximum number of file accesses to segment four it may be noted that, according to the study of Dewey [3], in full text the highest proportion of different words that have a given initial letter occurs for words beginning with the letter "T". For the previously assumed number (7,476) of expected new title terms in the first year in the letter "T" group the maximum number of new blocks is 5; this implies that for title terms the average maximum number of file accesses is 6, and the average maximum number of comparisons in core is 1,464. It is important to note that these values are only estimates made with a view to permitting an evaluation of the scheme proposed.

3- In reorganisation of the dictionaries it is necessary only to reorganise segment four. The other three segments may remain unchanged if desired.

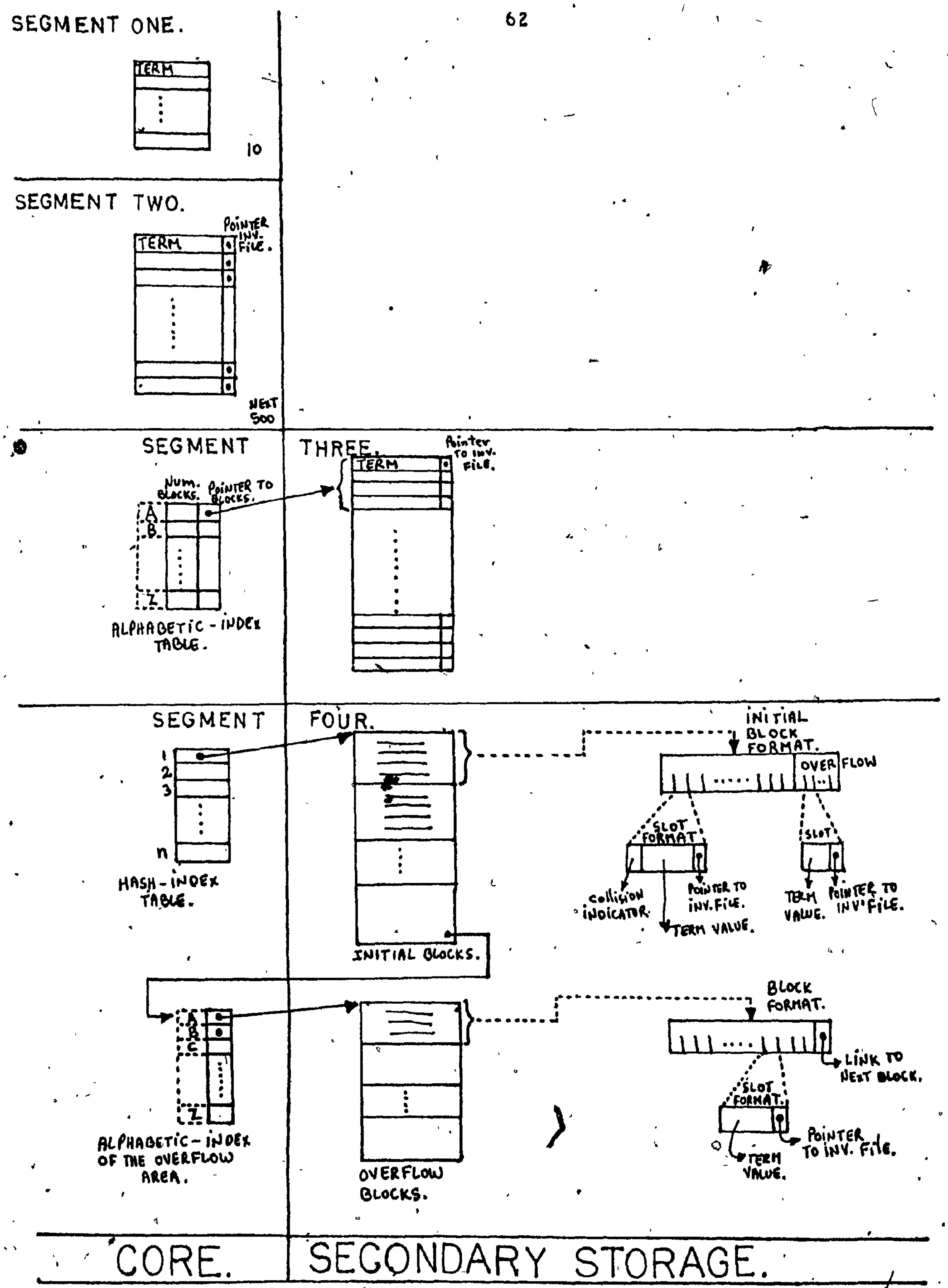


FIG. 3.1. SCHEME NUMBER ONE. (DICTIONARY)

SCHEME NUMBER TWO.

3.1. File structure of dictionaries.

3.1.1. Segmentation of dictionaries.

The segmentation of dictionaries is performed after the dictionary terms are organised according to their frequencies of occurrence in ascending order.

1- Segment one.

The 16 most frequent terms of each dictionary attribute are stored in core in alphabetic order. If is assumed that these 16 most frequent terms in the title attribute have very little significance as question terms and so it is not necessary to store pointers to the inverted file since no corresponding inverted lists are stored. Assuming Zipf's distribution, it may be estimated that this allows a reduction of 29.97% in the inverted files space since

$[A*N*(\log_e 16 + .5772)] = .2997 * N$,

where the constant $A=.08949$ is calculated by assuming that $D=40,000$ and $N=1,000,000$.

These terms are stored since they are needed during the process of decoding the sequential file.

2- Segment two.

The next 2,048 most frequent terms are stored in core in alphabetic order. Each term in this segment stores a pointer that points to the inverted file block that contains the document numbers of the documents in which it appears.

3- Segment three.

Application of a hash function to each term allows determination of the address corresponding to the location of the block in the inverted file that contains the document numbers in which the specific term appears. Such an address is obtained by computing some arithmetic function. Consequently the initially assigned terms of dictionary segment three, and all the new terms that appear from day to day, will not be stored anywhere. Therefore no disc accesess to segment three of any dictionary is required during the process of searching for any given term. Consequently no update algorithm is needed since both segments, one and two, will remain static during the operation of the system.

In practice there is a tendency for identifiers not to be completely independent of each other; many identifiers have a common suffix or prefix, or are simple permutations of other identifiers. Hence it is expected that, in practice, different hash functions will result in different hash table performances.

In examination of the performance figures for various hash functions it is apparent that the division method is generally superior to other types of hash functions, and for a general application it appears advisable to use the division method. However, it may be remarked that the performance of a hash function depends only on the method used to handle overflows and is independent of the hash function so long as a uniform hash function is being used. An update algorithm for the inverted files is discussed in a later section of this chapter.

3.1.2. Space required to store segments. Bucket capacity.

It is assumed that the terms of the dictionaries are stored in a fixed length of 15 bytes, and that the corresponding pointers that contain the addresses to the inverted file blocks of their corresponding terms are stored in a fixed length of 3 bytes.

1- Segment one.

16 most freq. terms * 15 bytes = 240 bytes.

2- Segment two.

For each dictionary the next 2,048 most frequent terms are stored in core.

2,048 terms*(15bytes/term + 3bytes/pointer) = 36,864 bytes.

3- Segment three.

Not used. One of the most important benefits that this scheme presents is that no storage is needed to store the remaining terms.

3.1.3. Search logic and handling of overflow buckets.

Since it may be unreal to assume that all question terms have the same probability of appearing in any search question, it is assumed that question terms are distributed according to the Zipf's law [9]. Then, as supposed in the description of the previous scheme, there may be a tendency to search for terms that are located in the region placed between the very high frequency terms and the very low frequency terms.

1- Segment one.

Whenever a term is to be searched for, it is first looked for in segment one by application of a binary search. The average number of maximum comparisons in core is equal to
$\log_2 (16) = 4.$

In order to save some storage in the sequential file, a code for these frequent terms is chosen in order to allow relatively fast decoding of the relevant documents found as a result of a search.

2- Segment two.

Binary search is also applied to segment two, and the average maximum number of comparisons in core is given by:
$\log_2 (2,048) = 11.$

3- Segment three.

Every time a term has not been found in the previous segments a hash function H1(x) is applied to determine the location of the block in the inverted file in which the given term may appear. It is supposed that the hash function H1(x) distributes addresses uniformly among the large number of different available blocks. The transformed value of a term, as given by the hash function, provides the address of its corresponding block so that there will be the same number of blocks as there are initial terms for each segment three in each dictionary at the creation time of the data base.

Each block contains a collision bit, an overflow bit, and the space to store document numbers. The blocks are of variable lengths, and have a minimum capacity of 7 document numbers per block. Collisions may be found in some blocks, and new document numbers are to be added during updates of the data base. Therefore, the time needed to process the question logic may increase since there will be some blocks that will store more document numbers than initially expected and allocated. Therefore, whenever the collision bit of the retrieved block is found to be "1" a sequential search is applied to the semi-coded sequential file in order to find the documents relevant to any specific query. Only the 2,064 most frequent terms will be stored in coded form in the sequential file, and the remaining terms are stored in the original form without any code.

Whenever there exists either a collision or an overflow, or both, as indicated by the respective bits, it is necessary to apply the question logic to the lists of all corresponding document numbers since it is impossible to know which document number belongs to which term, since no terms are stored.

It is important to remark that, because of collisions found in some of the blocks, there exists the possibility of avoiding the repetition of some document numbers, and thus the inverted file may be reduced in size.

An estimation of the average of collisions generated must be calculated in order to evaluate the overall performance of this scheme, which is affected by the expected increase in time for processing of question logic because of the expected increase of document numbers in the inverted lists. Suppose that the hashing function, H1(x), is chosen so that the transformed keys are uniformly distributed among the R blocks available for hash adrresses. If there are M keys then L=M/R is called the load factor. The probability that any given hash address corresponds to k keys is given by Murray as

$$Pk = e^{-L} * L^{k} / k!$$

Hence, for any given address the probability of a collision occurring is

$$C = \sum_{K=2}^{M} Pk = 1 - P0 - P1 = 1 - (1 + L) * e^{-L}$$

The structure of the available storage is arranged in such a manner as to theoretically avoid excessive processing of question logic and to reduce the sequential search as much as possible. It is believed that allowing a minimum space for the document numbers does not result in any drastic increase in the number of file accesses during the future updating of both the dictionary and its corresponding inverted file. As supposed in the description of the scheme one, if it is assumed that the terms within the dictionaries are distributed according to the Zipf's law, then a range of frequencies of the terms of this segment may be estimated as follows: According to the Zipf's law the frequency of a term whose rank is 2,065 in the initial dictionary of title terms is given by

Fr=A*N/r,

F(2,065)=A*N/2,065=43.3

assuming D=40,000, N=1,000,000, and hence A=.08949.

On the other hand, it is predicted that 50% of the title terms occur only once in the text, and 16.7% of the title terms occur only twice in the text. Therefore a large number of the terms of segment three have very small inverted lists, and in fact the maximum number of document numbers in any inverted list is approximately 43.3 for title terms. The same organisation is applied to the other attributes, even though it is believed that the frequencies may be different.

Initially the entire storage available for the inverted file is structured in such a way that it has the capacity to store all the document numbers associated with each term in a different variable length block. As an indication of how the performance of such a scheme is affected by the amount, and the structure, of the storage available, the following theoretical calculation is presented. The entire available storage is organised initially in the following manner. At creation time there are 37,936 title terms belonging to segment three; thus, as mentioned above if the load factor is equal to 1 the probability of a collision occurring for any given address during application of H1(x) is calculated with the following formula:

$C(H1) = 1 - (1 + 1) * e^{-1} = .2642.$

Therefore, supposing that the dictionary remains static, the probability of needing a further sequential search is also .2642.

Obviously the use of such a hash function H1(x) causes an increase in the amount of storage allocated, but fortunately it may be expected that there is a proportional improvement in terms of file accesses and required in-core comparisons since most of the additional blocks may have very short length. Therefore the required storage may not increase drastically.

Once the hash function H1(x) has been applied and there is an overflow then a new hash function H2(x) is applied, using as available storage an area equivalent to 25% of the initial blocks used for H1(x). Thus, for example, for the title term attribute there will be 9,484 new blocks, each of them containing an overflow bit and a fixed space to store 7 document numbers. This space is also allocated in order to maintain some availability of storage for the new document numbers that appear as a result of updating of the data base. Such an allocation tends to reduce degradation of the hashing scheme.

The storage, in bytes, required to store the dictionaries of the data base is shown in the following table.

	segm. 1		segm. 2		segm. 3	
	core	disk	core	disk	core	disk
title	240		36,864			
keyword	288		36,864			
s.heading	288		36,864			
author	288		36,864			
	1,104		147,456			

3.1.4. File accesses and required comparisons in core.

As shown in the table below, no file accesses are needed to search for any term in the dictionaries. However, a relatively large amount of main memory is required for storage of the most frequent terms. Also, a sequential search may be required during access of relevant items in the semi-coded sequential file. Nevertheless, it has been estimated that the probability of needing such a sequential search may be kept low if the dictionary stays static, and it depends on the number of blocks available for storage of terms that correspond to segment three in the inverted file.

	segm. 1		segm. 2		segm. 3		
	f.a.	c.c.	f.a.	c.c.	f.a.	c.c.	a.o.
title		4		11			2
keyword &		4		11			2
s.heading		4		11			2
author		4		11			2

f.a.=file accesses, c.c.=comparisons in core, and a.o.=arithmetic operations in core.

3.1.5. Statistical distributions of descriptors in queries.

It is believed that it is more realistic, in practice, to assume that question terms are distributed according to the Zipf's law, and thus it may be assumed that the probability of finding a term in core may be estimated as follows:

$$Pr(\text{find a term in core}) = A * \sum_{i=1}^{2064} (1/Ri)$$

$$= A * (\log_e 2{,}064 + .5772)$$
$$= .7346$$

Consequently, the probability of needing to apply the hash function may be .2654. These estimations, and the constant A, have been calculated assuming that N=1,000,000 and D=40,000 in the title term attribute, and that the constant A is estimated as $A=1+1/2+1/3+...+1/D \approx 1 / (\log_e D + .5772)$

3.2. Dictionary updating.

3.2.1. Updating algorithm and procedures to handle overflow areas.

Since in this scheme segments one and two are to remain unchanged, and no storage is needed to store terms of segment three, then no dictionary updating is required. Instead, an efficient update algorithm is needed for the inverted file in order to decrease the probable extra processing time required for execution of question logic and for the sequential search.

The following steps must be performed in creation of the appropriate inverted files:

1- Classify the terms in ascending order of frequencies of occurence.

2- The variable length blocks of the inverted files corresponding to H1(x) have a minimum capacity length of 7 document numbers. The number of different blocks required for H1(x) is equal to the expected number of terms for segment three for each different attribute, and the structure of the inverted file must be according to the specifications given in section 3.1.3. The number of blocks allocated for H2(x) is equal to 25% of the number of initial blocks allocated for H1(x), and each block is of fixed length with a capacity to store 7 document numbers.

3- Apply the hash function H1(x) to each term of segment three of the dictionaries, beginning with the most frequent term of segment three and using the division method in which the remainder becomes the address of the block. Store the document numbers corresponding to each term in the appropriate block in the inverted file.

4- The document numbers assigned to any particular block number are classified in ascending order of document number in order to improve the processing time of questions terms by allowing only one pass through any two inverted lists during their processing with respect to any logic operation.

3.2.2. Retrieval performance. (Reorganisation)

The retrieval performance clearly depends on the distribution of terms in the available storage generated by the hash functions. It also depends on the storage structure adopted for the inverted file lists, and on the update algorithm used to handle the new document numbers and the structure of the overflow areas in those inverted files. As new terms are added to any collection, the hash function H1(x) is applied to each of them. Also their corresponding document numbers are stored at the address generated by the hash function. In the case that the overflow bit of the block is found to be "1" then H2(x) is applied to the term, and its document number is added to the block in the corresponding storage area. It is supposed that most of the inverted lists are of short length, and such inverted lists are stored in a fixed length block per term. Thus the need for an external overflow area is reduced if each of these fixed length blocks is given the minimum length required in order to have the capacity to store the expected new document numbers for any term. Consequently, increases in the estimated initial storage requirements are required in order to maintain the free space for a period of time.

In the other hand, for an inverted file organisation it is recommended that the document numbers belonging to each term be arranged in ascending order so that any pair of inverted lists may be processed in a single pass.

As the variable length inverted file blocks are filled, an external overflow area is required and each block must store a pointer to chain it to the block that contains the new document numbers. The organisation of the external overflow area must allow a capacity to store 125,000 new document numbers, and the space is divided into 500 fixed blocks of length 1,000 bytes each (bytes of 6 bits). Assuming that a document number may be coded in four bytes there is thus a capacity to store 250 document numbers in each overflow block. These overflow blocks are organised in such a way that for title terms for every 19 (9,484/500) adjacent address locations generated by the hash function H2(x), beginning with the address "0", there exists one overflow block in the external overflow area. In such an overflow block are stored all the expected new document numbers that correspond to any of the 19 contiguous address locations used by H2(x). Therefore, every time that one of these overflow blocks is accessed the question logic is processed with respect to all the document numbers stored in it, and consequently a further sequential search must be applied.

Once one of the overflow blocks is filled, a reorganisation must be performed since it is to be expected that by this time the deterioration of the overall performance will have become serious.

The storage required for the hash function H2(x) and the corresponding external overflow area for the remaining attribute dictionaries is calculated using the same approach as presented above.

3.3. Evaluation. Advantages and disadvantages.

The design of the scheme presented is based on an attempt to obtain the minimum response time without regard to storage cost. Thus, the principal advantages and disadvantages of the proposed overall dictionary structure are:

1- Only the 2,064 most frequent terms of each entire dictionary are stored physically. The storage required is selected to be in core, and as a consequence it is necessary to have a computer system with the required relative [illegible] amount of available storage.

2- The dictionaries are permitted to grow freely. Unfortunately, deterioration of the retrieval performance increases in proportion to the number of new terms added from day to day.

3- No file accesses are needed to search for a term in any dictionary. This results in an increase in the processing time of question logic applied to the inverted lists and also in a sequential search of the semi-coded sequential file. However it is believed that in use of this scheme the need to perform such a sequential search may be kept low by employment of an adequate structure of storage for the inverted lists. In the worst cases 3 file accesses are needed to access the inverted files.

4- The sequential file is not fully coded because the terms of segment three do not have any code since they are not stored anywhere. Therefore it is necessary to allocate a large amount of secondary storage.

5- The increase of storage for the inverted files and the sequential file may be partly compensated by the fact that no storage need to be allocated for segment three of the dictionaries and, of course, there is no need for file accesses to these segments.

6- Since collisions may be found in the storage accessed by H1(x) and H2(x), the repetition of document numbers is avoided since transformed keys map into the same block.

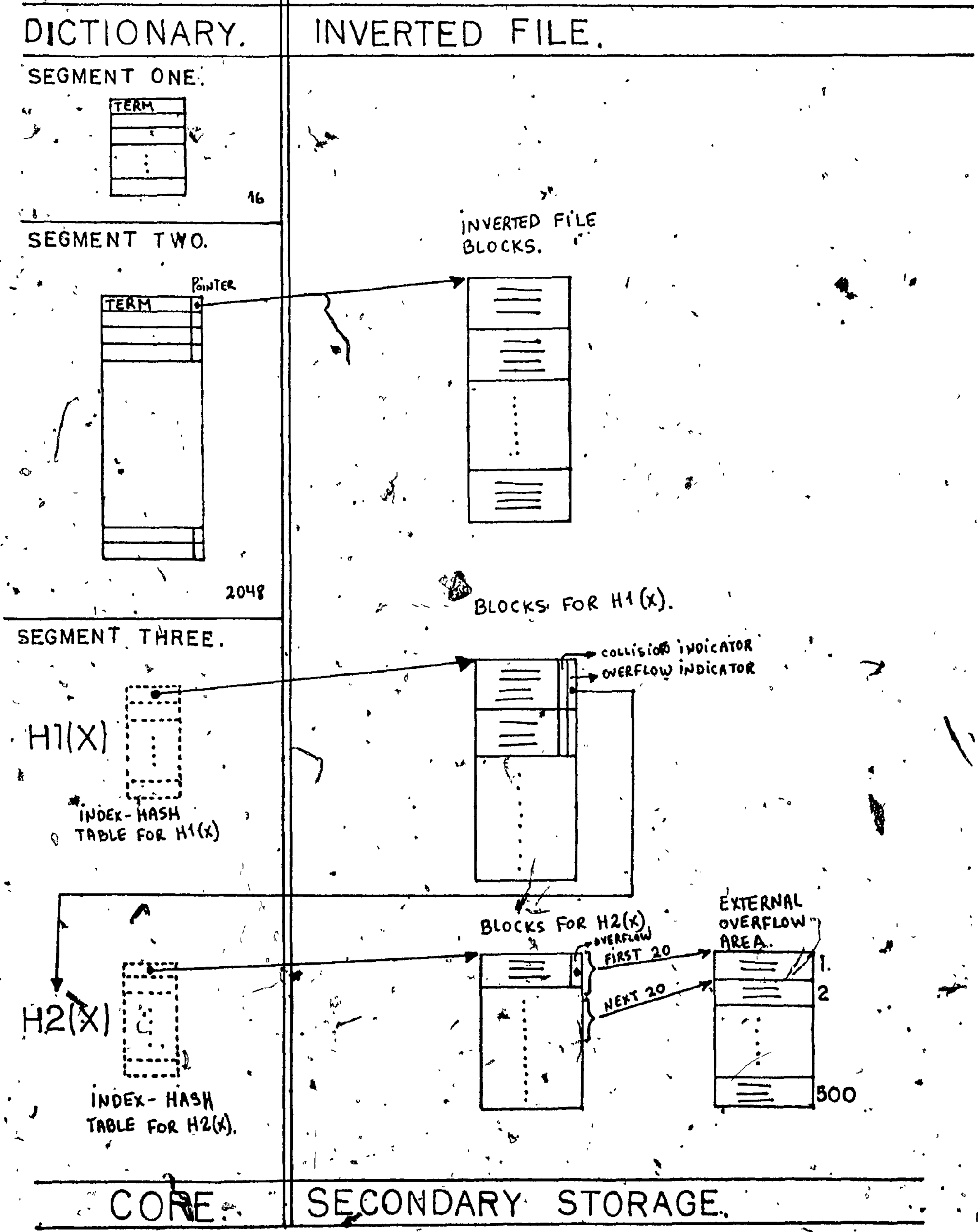


FIG. 3.2. SCHEME NUMBER TWO.

SCHEME NUMBER THREE.

The third scheme presented is based on a combination of what is believed to be the significant conceptions in each of the previous schemes proposed. However it is considered important to mention that the same assumptions are made about the distribution of different terms in the attributes. Also, it is assumed that the four dictionaries are organised according to the rank of the frequencies of occurrence of descriptors in each attribute.

3.1. File structure of dictionaries.

3.1.1. Segmentation of dictionaries.

1- Segment one.

The 16 most frequent terms of each dictionary are stored in core arranged in alphabetic order, and each descriptor of each dictionary, except those of the title term attribute, is stored with a pointer that points to the corresponding inverted file block that contains the appropriate document numbers. The 16 most frequent title terms are often used as stop words because of the very little information they provide about the document content. Therefore all the information associated with these 16 most frequent terms is neglected.

2- Segment two.

The next 1,500 most frequent terms are also stored in core in alphabetic order, and each term is stored with a pointer that points to the inverted file.

3- Segment three.

Since the Zipf's law is assumed to be true for the distribution of terms used in question searches, it has been decided to isolate the very infrequent terms of the dictionaries and to store them in a segment four. Thus segment three will contain all the remaining terms.

An estimate of the number of terms that may be stored in segment three is computed by using the values of the frequencies of occurrence of the very infrequent terms for general English text as presented previously, in which 50%, 16.7%, and 8.3% of the different terms in the title attribute occur only once, only twice, and only three times respectively. The number of terms in both segments three and four may then be calculated to have the values shown in the table:

	INITIAL D	SEGM. 3	SEGM. 4
title	40,000	8,485	29,999
keyword s.heading	(23,166; 51,801)	(4,275; 11,436)	(17,375; 38,849)
author	18,677	3,154	14,550

Segment three of the dictionaries contains terms arranged in alphabetic order and stored in secondary storage in the following manner: An Index Table is stored in core and contains 26 elements, each associated with one of the letters "A" to "Z". Each element of the Index Table stores a pointer that points to a block that contain terms beginning with the corresponding letters.

4- Segment four.

The terms that appear only once, twice, or three times in the text of a particular attribute are not stored anywhere. By use of a hash function the key associated with any term is used to locate a block in the inverted file that contains the document numbers in which the term appears.

3.1.2. Space required to store segments. Bucket capacity.

1- Segment one.

16 terms * 15 bytes/term = 240 bytes is required in core for title terms, and 288 bytes are required for terms of other attributes.

2- Segment two.

1,500 terms * (15bytes/term + 3bytes/pointer) = 27,000 bytes in core.

3- Segment three.

Terms will be grouped according to their first letter. In order to estimate how many terms begin with a given initial letter the study of Dewey [3] is considered in order to produce an estimate of the average number of file accesses needed to find a term in the segment. A record size of 15 bytes/term and 3 bytes/pointer to the inverted file is assumed, and also a variable block size of maximum size 5,076 bytes.

For the four dictionaries the space required to store such a segment with the preceding specifications for the four dictionaries is shown in the following table:

	NUMBER TERMS	BYTES NEEDED	NUMBER BLOCKS
title	8,485	152,730	46
keyword & s.heading	(4,275; 11,436)	(76,950; 205,848)	(32; 58)
author	3,154	56,772	28

4- Segment four.

No storage is needed for allocation of this segment.

3.1.3. Search logic and handling of overflow buckets.

The same assumptions are adopted as in the previous scheme and it is supposed that, in practice, there is likely to be a tendency for search terms to be located in the region between the very high frequency terms and the very low frequency terms. It is desirable to use a scheme that gives the best possible response time under such conditions.

1- Segment one.

Whenever a term is to be searched for, a binary search is first applied to the 16 most frequent terms. If the term is not found it is necessary to search for it in segment two. The maximum number of comparisons needed to look for a term is

$\log_2 16 = 4.$

2- Segment two.

To locate a term in this segment a binary search is also applied in core. The average maximum number of comparisons is given by

$\log_2 1{,}500 = 10.55.$

3- Segment three.

When a term has not been found in either segment one or segment two then the in-core index table must be accessed in order to find the pointer that supplies a link to the block that contains terms beginning with a particular letter. The number of records needed to store all the terms beginning with a given letter may vary among different letter groups, and a binary search is applied to the number of blocks required for any specific letter stored in the index-table. Once a block is brought into core a comparison is performed between the term being searched for and the highest ranking term in the block in order to determine if the search term is present. If the highest term of the block is found to be of higher rank than the searched term then another block of the group must be retrieved by the binary search. Otherwise, if the result of the comparison is "less than", another comparison must be performed with the lowest rank term in the block; if the result of this comparison is "less than" it is required to access a lower ranked block on which to perform a binary search. On the other hand, if the result is "more than" then the binary search must be applied to the terms of this block.

Estimation of the average maximum number of both file accesses and comparisons in core needed to locate a term in this segment may be made as follows. Selecting a variable length block size of maximum 5,076 bytes, and a fixed term length of 15 bytes plus 3 bytes for the pointer to the inverted file, shows that the average number of file accesses is 1.23 for title terms, between 1.04 and 1.31 for keyword and subject headings, and 1 for author names. The average maximum number of comparisons in core for title terms is approximately

$(1.5 \text{ comparisons} * 1.23 \text{ accesses}) + \log_2 (5{,}076/18) =$

$2.47 + 8.13 = 10.61.$

Applying the same formula to the remaining attributes shows that the averages are between 9.69 and 10.09 for keywords and subject headings, and 9.63 for author names.

4- Segment four.

Each time that a term has not been found in the previous segments the term value is operated on by the hash function H1(x), which transforms it into the location address of the block of the inverted file that contains the document numbers in which the specific term appears.

The storage of the inverted files is structured in such a way that for the title term, for instance, there will be 7,499 different blocks available to store the 29,999 terms of the title attribute. Consequently, each block may store all the document numbers associated with four terms. In order to estimate an appropriate minimum block size it is useful to compute the average value of the frequency of occurrence of the group of terms of segment four in the following manner: For title terms:

$1(20{,}000/29{,}999) + 2(6{,}666/29{,}999) + 3(3{,}320/29{,}999) = 1.43$

1.43 multiplied by 4 terms/block is equal to 5.72.

Thus, a minimum block size of 6 document numbers is chosen. Such an organisation is implemented by taking advantage of collisions that result on application of the hash function H1(x) with a load factor of 4.0004 (29,999/7,499), and assuming that the transformed keys are uniformly distributed among the 7,499 blocks available for hash addresses. The probability of a collision occurring at any given address is

$C = 1 - (1 + 4.0004) * e^{-4.0004} = .9084$

Furthermore, there exists the possibility that in practice some identifiers have a common suffix or prefix or are simple permutations of other identifiers, and as a result may contribute to the clustering of 4 terms in a block when the division method is used for the hash function.

As a result of such organisation no ~~disc~~ accesses are needed for any term of segment three, and therefore no update algorithms for dictionaries are required since segments, one, two, and three are to remain unchanged during the operational life of the system. Consequently a sequential search must be performed during the output decoding phase in order to eliminate any documents not relevant to the specific search. The storage, in bytes, required for the dictionaries of the data base is shown in the following table:

	SEGM. 1		SEGM. 2		SEGM. 3		SEGM. 4	
	core	disc	core	disc	core	disc	core	disc
title	240		27,000		156	152,730		
keyword & s.heading	288		27,000		156	(76,950; 205,848)		
author	288		27,000		156	56,776		

3.1.4. File accesses and required comparisons in core.

	SEGM. 1		SEGM. 2		SEGM. 3		SEGM. 4		
	f.a.	c.c.	f.a.	c.c.	f.a.	c.c.	f.a.	c.c.	a.o.
title		4		10.5	1.23	10.6			1
keyword & s.heading		4		10.5	(1.04; 1.31)	(9.6; 10.1)			1
author		4		10.5	1.	9.63			1

f.a.=file accesses, c.c.=max.comparisons in core, and a.o.= arithmetic operations.

3.1.5. Statistical distribution of descriptors in queries.

It is assumed that question terms are distributed according to the Zipf's law, and in order to estimate the probability of a term being in each different segment of the dictionaries the following calculations are made:

$$\text{Pr(find a term in core)} = A * \sum_{i=1}^{1516} (1/Ri)$$

$$= A * (\log_e 1{,}516 + .5772) = .7071$$

Pr(find a term in segm. 3) =

Pr(find a term in the 9,974) − Pr(find a term in core) =

$$A * \sum_{i=1}^{9974} (1/\ Ri) - .7071 = A * (\log_e 9{,}974 + .5772) - .7071 =$$

$= .8757 - .7071 = .1686$

Thus the probability of having to search in segment four is .1243.

The above probabilities have been calculated by assuming the figures for title term. Although it is known that there may exist some variations in the assumption for the other attributes it is believed that this is not likely to seriously affect the performance of the system. Such estimations allow a prediction of the probable use of each dictionary segment during the search phase, and hence allows an evaluation of the overall performance of the scheme number three.

In conclusion, the probabilities that a term searched for is contained in a particular segment is as follows:

SEGMENT ONE	.2997
SEGMENT TWO	.4073
SEGMENT THREE	.1686
SEGMENT FOUR	.1243

3.2. Dictionary updating.

3.2.1. Updating algorithm and procedures to handle overflow areas.

An important property of the scheme presented above is the avoidance of any direct updates of the dictionaries since the segments one, two, and three remain static. All new terms added as a result of periodic updates are assigned to segment four. However it is known that no storage is allocated for segment four, and by application of H1(x) to the term value its appropriate document number is stored in the corresponding inverted file block only if such a new document number does not already appear in the block. Thus, it is necessary to implement an efficient update algorithm in order to reduce the required processing of question logic and the sequential search to a reasonable level so that a minimum response time may be obtained without undue storage cost.

Evidently, in theory, the smaller the load factor used for the H1(x) the lesser is the need to perform both excessive processing of the question logic and further sequential searches. However, it is supposed that the terms of segment four are infrequent ones, and only modifications may be found in the case that some of the terms initially present, or some of the new terms added to the dictionaries, become very popular as question terms and hence correspond to inverted lists that continue to grow as further new document items are added to the data base. Thus the performance of the H1(x) depends on the method used to handle overflows in the inverted files. Several different structures could be suggested but it is believed to be beyond the scope of this thesis to analyze in depth the most suitable inverted file structure for such a dictionary organisation. Nevertheless, a practicable algorithm to handle the overflow areas may be discussed briefly as follows: In order to keep the number of file accesses to a low level the inverted file blocks may be created to have a minimum number of free slots at creation time in order to enable storage of the new document items added to the data base and also to facilitate the classification of the document numbers of such blocks in ascending order whenever an addition is made.

Although it is believed that a continual reclassification of such inverted file blocks may prove to be very costly, it is considered necessary to perform such since in order to obtain a relative fast response time the inverted lists are to be classified permanently. Otherwise the results of an unclassified inverted file are very difficult to predict, and the degradation of the system may reach high levels in a short period of time. It is advisable that such updates be made periodically and not concurrently during the search time.

There are two situations that contribute to an increase in deterioration of the performance of the above organisation. The first is that because of the new terms added to the dictionaries the initial concept of 4 terms per block is destroyed, and as a consequence more further sequential search is required. The second is that as new document items are included as members of a block group then rearrangement of the inverted lists is necessary in order to obtain an acceptable response time.

The following scheme is proposed to handle the overflow items:

1- There is an external overflow area organised in such a manner that every initial inverted block contains the corresponding overflow block, and there is a fixed space available to store 5 new document numbers.

2- Since the initial inverted blocks are supposed to be of variable length, some periodic reorganisation must be performed to transfer the new 5 document numbers to the initial block classified in alphabetic order in order to decrease the number of file accesses from 2 to 1.

3.2.2. Retrieval performance. (Reorganisation)

The proposed reorganisation is based on the fact that any inverted file system requires appropriate periodic reorganisation in order to maintain a relative fast response. However, it is difficult to suggest an optimal interval of time between reorganisations since it may vary accordingly to the specific application.

3.3. Evaluation. Advantages and disadvantages.

The clustering of very infrequent terms has been used in order to reduce the dictionary sizes, taking advantage of the assumption of their very infrequent appearance in the search queries, and at the same time to present an organisation that allows a relative fast response time while minimizing the further processing that is generated.

The advantages in the implementation of this scheme are:

1- Reduction of the dictionary size.

2- Reduction of the inverted files since no repetition of document numbers occurs in the inverted lists of the infrequent term groups.

3- Reduction of the number of accesses to the dictionaries and inverted files.

As a consequence, the scheme has the following disadvantages:

1- Increase of sequential file accesses. Also it is necessary to search the sequential file whenever infrequent terms appear in search queries.

2- The infrequent terms are stored in the sequential data base in uncoded form and the entire sequential data base is stored using the following scheme:

the 16 most frequent	00xxxx
the 1,024 next most frequent	1xxxxx 0xxxxx
the 32,768 next, until required	1xxxxx 0xxxxx 0xxxxx
the uncoded terms	01wwww

where wwww indicates the length in characters.

This coding scheme allows a partial saving of storage for the sequential data base.

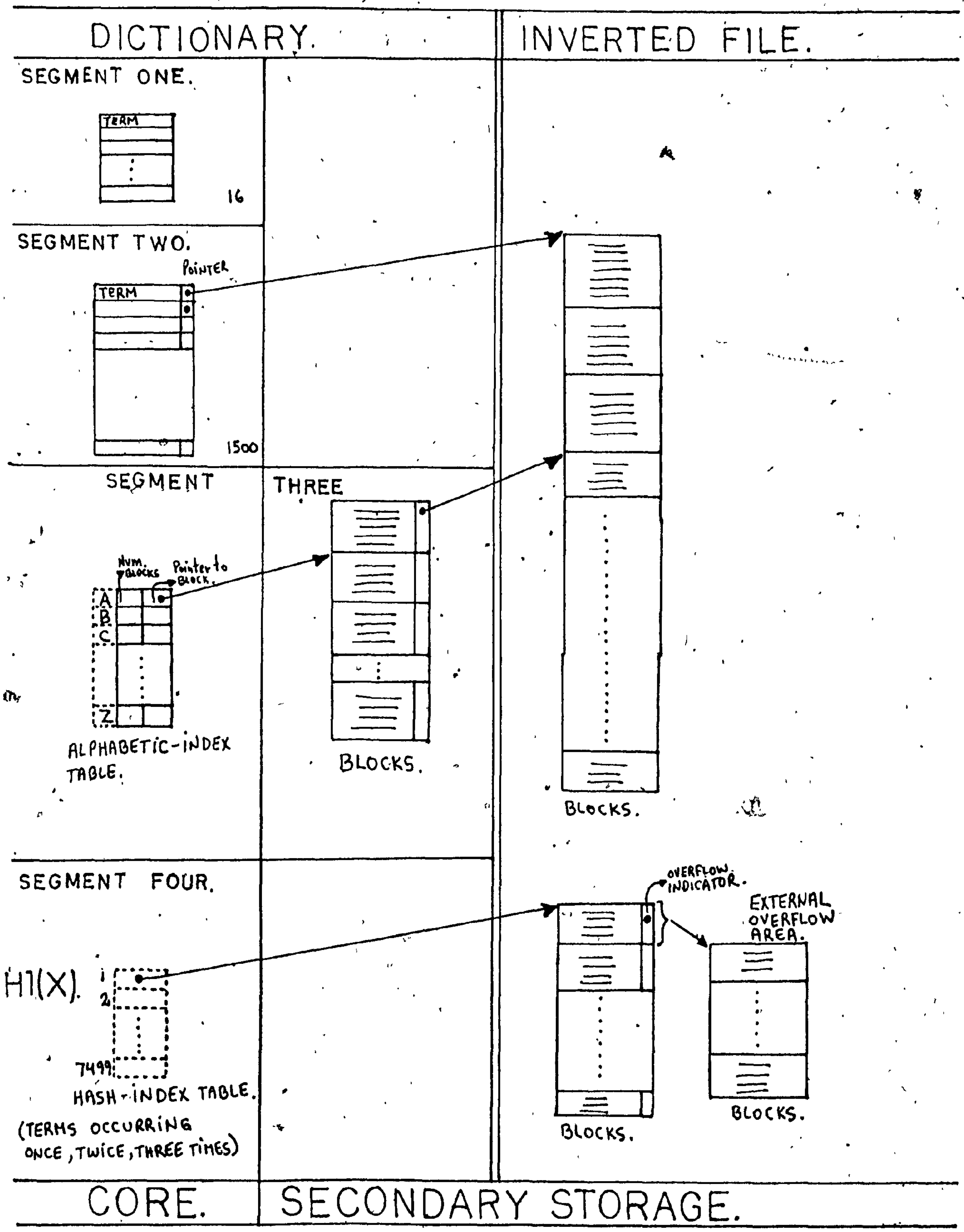


FIG. 3.3. SCHEME NUMBER THREE.

CHAPTER FOUR.

4. Comparative analysis of the three proposed schemes with respect to the question logic processing.

In the previous chapter an evaluation of each scheme was presented and consideration was given to the possible advantages and disadvantages that arise in the implementation of such dictionary structures.

The purpose of this chapter is to compare the proposed schemes according to some factors that affect the question logic processing. The factors include the following:

4.1. Number of disc accesses, and required comparisons in core, needed to search the dictionaries for a given term.

4.2. Number of required Boolean operations.

4.3. Processing of the sequential data base.

4.4. Average maximum time of processing per question term.

4.1. Number of disc accesses, and required comparisons in core, needed to search the dictionaries for a given term.

Since question terms are supposed to be distributed according to the Zipf's law, and assuming that the dictionaries remain static, the probabilities that a term to be searched for is in core may be established theoretically to have the following values for each of the three schemes

SCHEME ONE.	.6096
SCHEME TWO.	.7346
SCHEME THREE.	.7071

Since each scheme presents a different method of searching for a given term in the remaining group of terms not stored in core, each scheme may generate some different further processing. Thus it is appropriate to include the following probabilities of requiring such further processing:

SCHEME ONE.	.3904
SCHEME TWO.	.2654
SCHEME THREE.	.2929

The following is a summary of the average numbers of both file accesses and required comparisons in core needed to locate a term in any dictionary:

	SCHEME 1						SCHEME 2			SCHEME 3					
	1	2	3		4		1	2	3	1	2	3		4	
	C	C	F	C	F	C	C	C	F	C	C	F	C	F	C
TIT	3.3	8.9	1.6	10.2	1	1;61	4	11		4	10.5	1.2	10.6		
KEY SUB	3.3	8.9	(1.2; 1.6)	(9.6; 10.3)	1	1;61	4	11		4	10.5	(1.0; 1.3)	(9.6; 10.1)		
AUT	3.3	8.9	1.2	9.6	1	1;61	4	11		4	10.5	1	9.6		

C=average comparisons in core and F=average number of file accesses.

With respect to the average maximum number of file accesses required to locate a term in any dictionary attribute, the scheme number two presents the best results. Unfortunately the consequences are an increase of both processing of question logic and in the sequential search of the sequential data base since only the 2,064 most frequent terms of every dictionary are stored physically in core and the remaining terms are not stored anywhere.

Scheme number three provides an intermediate result. The average maximum number of file accesses is 1.23 for title terms, and the probability of needing further processing is relatively small (.1243), and fortunately the length of the inverted lists may be supposed to be small since only very infrequent terms are included using the approach of clustering their corresponding document numbers in inverted lists.

Scheme number one involves the highest average number of file accesses to search for a term in any dictionary. However, there is no need for extra processing, but unfortunately all the terms of any dictionary must be stored physically, the 516 most frequent in core and the rest in secondary storage.

4.2. Number of required Boolean operations.

The average length of the inverted lists is estimated for title terms assuming that the probability of occurrence of terms in questions is according to the Zipf's law, and assuming that only terms with rank greater than 10 are searched for. The average length of the inverted lists is:

$$\begin{aligned} \text{Ave[Zipf]} &= \sum_{r=1}^{D} A/r\ (A*N)/r - \sum_{r=1}^{10} A/r\ (A*N)/r \\ &= A^2N\left(\sum_{r=1}^{D} 1/r^2 - \sum_{r=1}^{10} 1/r^2\right) \\ &= A^2N\ (\pi^2/6 - 1.54) \\ &= 800.84. \end{aligned}$$

In the other hand, if it is assumed that the probability of occurrence of terms in questions is according to the uniform distribution then the average length of the inverted lists for title terms with rank greater than 10 is:

$$\begin{aligned} \text{Ave[Uniform]} &= \sum_{r=1}^{D} 1/D\ (A*N)/r - \sum_{r=1}^{10} 1/D\ (A*N)/r \\ &= 1/D\ A*N\left(\sum_{r=1}^{D} 1/r - \sum_{r=1}^{10} 1/r\right) \\ &= 1/D\ A*N\ (11.17 - 2.92) \end{aligned}$$

= 18.43.

The average maximum number of comparisons in core that are required to be performed every time that two inverted lists are to be combined, assuming that question terms are distributed according to the Zipf's law, is estimated as follows:

1- Suppose a total of N terms of which D are different.

2- Term of rank r occurs in Lr=A*N/r documents where $A = 1/\sum_{r=1}^{D} 1/r = 1/(\log_e D + .5772)$.

3- Let Pr = the probability of occurrence of rth term in questions.

4- To combine the inverted lists of the terms of rank r and s requires a maximum number of comparisons, in the worst case, of <= Lr + Ls - 1.

5- The average number of comparisons for all possible choices of rank r and s is

Ave. <= Ave. (Lr + Ls - 1).

i- If case $term_r = term_s$ is excluded:

$$\text{Ave.} = \sum_{r=1}^{D-1} \sum_{s=r+1}^{D} Pr\ Ps\ (Lr + Ls - 1) \Big/ \Big(\sum_{r=1}^{D-1} \sum_{s=r+1}^{D} PrPs \Big)$$

$$= \Big(\sum_{r=1}^{D-1} \sum_{s=r+1}^{D} Pr\ Ps\ (Lr + Ls) \Big) \Big/ \Big(\sum_{r=1}^{D-1} \sum_{s=r+1}^{D} PrPs \Big) - 1$$

Now: $$\sum_{r-1}^{D-1} \sum_{s=r+1}^{D} PrPs\ (Lr + Ls) = 1/2 \sum_{r=1}^{D} \sum_{s=1}^{D} PrPs\ (Lr+Ls) - 1/2 \sum_{r=1}^{D} Pr^2\ 2Lr$$

$$= \frac{1}{2} \sum_{r=1}^{D} PrLr + \frac{1}{2} \sum_{s=1}^{D} PsLs - \sum_{r=1}^{D} Pr^2\ Lr$$

$$= \sum_{r=1}^{D} PrLr - \sum_{r=1}^{D} Pr^2Lr$$

$$\sum_{r=1}^{D}\sum_{s=1}^{D} PrPs = \sum_{r=1}^{D-1}\sum_{s=r+1}^{D} PrPs + \sum_{r=2}^{D}\sum_{s=1}^{r-1} PrPs + \sum_{r=1}^{D} Pr^2$$

$$= 2\sum_{r=1}^{D-1}\sum_{s=r+1}^{D} PrPs + \sum_{r=1}^{D} Pr^2$$

$$\therefore \sum_{r=1}^{D-1}\sum_{s=r+1}^{D} PrPs = 1/2 \sum_{r=1}^{D}\sum_{s=1}^{D} PrPs - 1/2 \sum_{r=1}^{D} Pr^2$$

$$= 1/2 - 1/2 \sum_{r=1}^{D} Pr^2$$

$$\text{Ave.} = \left(\sum_{r=1}^{D} PrLr - \sum_{r=1}^{D} Pr^2Lr\right) / \left(1/2 - 1/2 \sum_{r=1}^{D} Pr^2\right) - 1$$

a) If $Pr = \frac{A}{r}$ (Zipf distribution).

$$\text{Ave.} = \frac{A^2N \sum_{r=1}^{D} 1/r^2 - A^3N \sum_{r=1}^{D} 1/r^3}{1/2 - \frac{A^2\pi^2}{12}} - 1$$

$$\text{Ave.} = \frac{A^2N \frac{\pi^2}{6} - A^3N\,(1.17)}{1/2 - \frac{A^2\pi^2}{12}} - 1 = \frac{A^2N\left[\frac{\pi^2}{6} - A\,(1.17\right]}{1/2 - \frac{A^2\pi^2}{12}} - 1$$

$$\text{Ave.} = \frac{A^2N[1.64 - .10]}{1 - \frac{A^2\pi^2}{6}} - 1 = \frac{A^2N[1.54]}{1 - .013} - 1 = \frac{12,334.3}{.98} - 1$$

Ave. = 12,585

b) If $Pr = \frac{1}{D}$ (Uniform distribution)

$$\text{Ave.} = \frac{\frac{AN}{D} \sum_{r=1}^{D} 1/r - \frac{AN}{D^2} \sum_{r=1}^{D} 1/r}{1/2 - 1/2D} = \frac{\frac{N}{D} - \frac{N}{D^2}}{1 - 1/D}$$

$$\text{Ave.} = \frac{2(25 - .0006)}{1 - 1/D} = 49.99$$

ii – If case $term_r = term_s$ is not excluded:

$$\text{Ave.} = \sum_{r=1}^{D} PrLr + \sum_{s=1}^{D} PsLs - 1$$

a) If $Pr = \frac{A}{r}$ (Zipf distribution)

$$\text{Ave.} = 2A^2N \sum_{r=1}^{D} 1/r^2 - 1 = A^2N \frac{\pi^2}{3} - 1 = 26,348.5$$

(if D is large)

b) If $Pr = 1/D$ (Uniform Distribution)

$$\text{Ave.} = \frac{2AN}{D} \sum_{r=1}^{D} 1/r - 1 = \frac{2AN}{D}(1/A) - 1 = \frac{2N}{D} = 50$$

If it is assumed that only terms with rank greater than 10 are allowed to be searched for. Thus in case [i–a] is considered:

$$\text{Ave.} = 12,585 - \left(\frac{\sum_{r=1}^{10} PrLr - \sum_{r=1}^{10} Pr^2Lr}{1/2 - 1/2 \sum_{r=1}^{10} Pr^2} - 1 \right)$$

$$\sum_{r=1}^{10} PrLr = A^2N \sum_{r=1}^{10} 1/r^2 = A^2N * 1.5479 = 12,397.6$$

$$\sum_{r=1}^{10} Pr^2Lr = A^3N \sum_{r=1}^{10} 1/r^3 = A^3N * 1.17 = 838.6$$

$$\sum_{r=1}^{10} Pr^2 = A^2 \sum_{r=1}^{10} 1/r^2 = A^2 * 1.54 = .012$$

$$\text{Ave.} = 12,585 - \left[\frac{11,559}{.9876} - 1\right]$$

$$\text{Ave.} = 12,585 - 11,703.1$$

$$\text{Ave.} = 881.9$$

The average maximum number of comparisons in two inverted lists for title terms, assuming that question terms are distributed according to the Zipf's law is approximately 734.2. This average is obtained by supposing that every term has its corresponding inverted list as initially arranged in the scheme number one.

The average may be different for the scheme number two because of collisions generated by the hash functions. Fortunately most of the collision terms are infrequent, and therefore the average may not increase drastically. Unfortunately, as new document numbers are added to the inverted lists the average will increase proportionally.

The average may be affected in the scheme number three by the fact that inverted lists of four different terms are clustered by the use of a hash function. Fortunately, the frequency of occurrence of such terms will be only one, two, or three, and consequently the average may not increase rapidly.

4.3. Processing of the sequential data base.

The amount of required processing of the sequential data base during a question search obviously depends on the number of document numbers that result from the combinations of the inverted lists generated during the question processing. The number of terms in a question search may vary according to many factors and a formal study of this subject is not undertaken in the present investigation. However, an estimate has been made, as described above, of the average number of document numbers in inverted lists, assuming that questions terms are distributed according to the Zipf's law. This gives some indication of the behaviour of the three schemes with respect to sequential search of the sequential data base.

The implementation of the scheme number one does not require any extra processing of the sequential data base, since all the term values are stored physically either in core, or in secondary storage so that with each term there is stored the pointer that points to the corresponding inverted list.

During execution of the scheme number two it may often be necessary to apply an extra sequential search on the sequential data base since only the 2,064 most frequent terms are stored in core and no physical storage is used for the remaining terms. Although two hash functions are to be applied, each of them with a load factor of 1, collisions may still appear in the addresses that they generate. The collisions are a result of both overflow of the storage areas because of continuous update of the dictionary, and also the tendency that exists in practice for identifiers to have a common suffix or prefix or to be simple permutations of other identifiers.

Using the Zipf's law, it may be predicted that in case of a collision the length of the inverted lists may be relatively small since a high percentage of the infrequent terms that appear in the remaining group of terms are not stored anywhere. For instance, in the title terms approximately 80% of the terms not stored physically occur only once, or twice, or three times (37,936-29,999=7,936), and as mentioned in section 3.1.3 for scheme number two the frequencies of occurrence may range between 43.3 and 1 for title terms. In the other hand, it is believed that a high percentage of the author names not stored physically may have a very low frequency of occurrence and the percentage of infrequent authors may be higher than those of title terms.

In the case of scheme number three, the additional required sequential searches of the sequential data base are generated by the use of clustering of four different inverted lists. However, according to the Zipf distribution the probability of occurrence of very infrequent terms as question terms may remain very small. Consequently the additional sequential searches may be required very rarely if it is supposed that the dictionaries remain static. Nevertheless, in practice, there is a possibility that some infrequent terms may become very popular as question terms and then the required additional sequential searches may increase. Also it should be mentioned that some terms added to the dictionaries as a result of periodical updates may be popular as question terms, and since segments one, two, and three are to remain unchanged the need for extra sequential search increases since the inverted file lists of such popular terms are stored using the cluster approach.

4.4. Average maximum time of processing for a question term.

The average maximum time of processing for a question term may be evaluated in terms of both the total average number of file accesses, and the total average number of comparisons in core required in the dictionary and the corresponding inverted file to locate such a term and its appropriate inverted lists. Such a evaluation is given below by considering the estimated initial values of the data base, and assuming that the worst cases occur:

	SCHEME ONE				SCHEME TWO				SCHEME THREE			
	F.ACC		C. CORE		F.ACC		C. CORE		F.ACC		C. CORE	
	DICT	INV	DICT	INV	DICT	INV	DICT	INV	DICT	INV	DICT	INV
TIT	2.6	1	52.4			1.2	15		1.2	1	25.1	
KEY	(2.2	1	(51.8;			1.2	15		(1;	1	(24.1;	
SUB	2.6)		52.5)						1.3)		24.6)	
AUT	2.2	1	51.8			1.2	15		1	1	24.1	

where

F.ACC=file accesses, C. CORE=comparisons in core.

DICT=dictionary, INV=inverted file.

CHAPTER FIVE.

5. CONCLUSIONS.

Considering both the theoretical descriptive analysis of the performance of the three structures presented, and the typical characteristics of the three different attribute dictionaries of document retrieval systems, it is possible to describe the benefits gained by use of each particular scheme.

The scheme number two is well adapted for use with the author name attribute since authors with relative high frequency of occurrence are stored in core. Advantage is taken of the expected large number of authors that have a very low frequency of occurrence and hence have inverted lists of small length. Thus the number of additional sequential searches may be kept to a relative low level.

In keyword and subject heading attributes, it may be supposed that there is not a very large number of very infrequent terms, since most of the terms may be expected to be significant as document content descriptors. Also the stop words have been removed. It is supposed that the question terms in this attribute are distributed according to the Zipf's law, but some modifications must be made as outlined in section 1.5.2.1. Thus, it appears convenient to use the proposed structure of the scheme one since it allows all the terms to be stored according to their frequencies of occurrence in the four different dictionary segments. Each term is stored with a pointer that points to the corresponding inverted list.

On the other hand the organisation of the scheme number three provides a suitable structure for title terms. The most frequent title terms are stored in core. The group of terms placed between the most frequent terms and the very infrequent ones are stored in secondary storage. The very infrequent terms are not stored anywhere and the document numbers that correspond to them are clustered in order of four terms by use of a hash function. Since it is assumed that the question terms are distributed according to the Zipf's law, the probability of appearance of an infrequent term in a question search is very small, and consequently the additional sequential search of the sequential data base may not be required very often. Moreover, whenever it is required the length of the inverted lists are likely to be small. Also, it is significant to mention that some dictionary storage for of the title terms is saved by taking advantage of the fact that those very infrequent terms have small probabilities of occurrence as question terms.

The search formulation is a very important factor that may help to improve the overall performance of the particular search strategy. The search formulation, which is a statement of the requirements of acceptable documents, may be regarded as an index term profile of the request. A search involves the matching of this request profile against the appropriate dictionary of index terms. It may use alternative index term combinations in order to retrieve as much as possible of the relevant literature and as little as possible of the irrelevant. Moreover, the search formulation may be structured into varying levels of generality and very general subsearches will tend to produce an increase in the question logic processing and a consequent increase in the sequential search of the sequential data base.

For example, if the frequency of occurrence of the terms t1,t2,t3, and t4 is expected to be high, a question of the form:

```
1.t1,t2,OR
2.t3,t4,OR
3.1,2,AND
4.SEARCH,3.
```

may be reformed in the following manner in order to reduce the amount of required processing:

```
1.t1,t3,AND
2.t1,t4,AND
3.t2,t3,AND
4.t2,t4,AND
5.1,2,3,4,OR
6.SEARCH,5.
```

It is obvious that, with the type of searching discussed in the present thesis is necessary to have a very clear and detailed statement of the requester's information need. Thus the interface between user and system will have an important effect on the performance of the proposed schemes, and with any scheme a failure may result from inadequate interaction between the requester and the system.

REFERENCES.

1. J. Minker. Information Storage and Retrieval, A Survey and Functional Description. ACM- Special Interest Group on Information Retrieval. SIGIR. Fall 1977, Volume XII, Number 2.

2. L.B. Doyle. Information Retrieval and Processing. John Willey and Sons, Inc., 1975.

3. H.S. Heaps. Information Retrieval Computational and Theoretical Aspects. Academic Press, Inc., 1978.

4. T. Radhakrishnan and R. Kernizan. Lotka's Law and Computer Science Literature. Journal of the American Society for Information Science. January 1979. Volume 30. Number 1, pp. 51.

5. P.J. Dimsdale and H.S. Heaps. File Structure for an On-Line Catalog of One Million Titles. Journal of Library Automation. Volume 6/1, March 1973.

6. G. Herdan. The Advanced Theory of Language as Choice and Chance. Springer-Verlag, New York. 1966.

7. M.G. Lindquist. Growth Dynamics of Information Search Services. Journal of the American Society for Information Science. Volume 29/2, pp 65-76. March 1978.

8. M.L. Pao. Automatic Text Analysis Based on Transition

Phenomena of Word Occurrence. Journal of the American Society for Information Science. Volume 29/3, pp. 121-124. May 1978.

9. F.W. Lancaster. Vocabulary Control for Information Retrieval. Information Resources Press. Washington, D.C. 1972.

10. C.J. van Rijsbergen. Information Retrieval. Butterworths. 1975.

11. K. L. Montgomery. Document Retrieval Systems. Factors Affecting Search Time. Macel Dekher, Inc. N.Y. 1975.

12. C. T. Meadow. The Analysis of Information Systems. Melville Publishing Company. Los Angeles. CA. 1973.

13. G. Salton. Automatic Information Organisation and Retrieval. McGraw-Hill Book Company. 1968.

14. K.J. McDonell. An Inverted Index Implementation. The Computer Journal. Volume 20/Number 2, August 1975, pp. 116-122.

15. B.M. Nicklas and G. Schlageter. Index Structuring in Inverted Data Bases by TRIES. The Computer Journal. Volume 20/Number 4, pp. 321-324. July 1976.

16. A.F. Cardenas. Analysis and Performance of Inverted Data Base Structures. Communications of the Association for Computing Machinery. Volume 18/Number 5, May 1975. pp.

253-263.

17. E. Hill, Jr. Analysis of An Inverted Data Base Structure. Proccedings International Conference on Information Storage and Retrieval: SIGIR 1978. Rochester N.Y. pp. 37-64. May 1978.

18. S. Herner and K. J. Snapper. The Application of Multiple-Criteria Utility Theory to the Evaluation of Information Systems. Journal of the American Society for Information Science. Volume 6, pp. 289-296. November 1978.

19. R.M. Bird, J.B. Newsbaum, and J.L. Trefftzs. Text File Inversion: An Evaluation. Fourth Workshop on Computer Architecture For Non-Numeric Processing. August 1978, pp. 42-50.

20. I. A. Macleod. Towards an Information Retrieval Language Based on a Relational View of Data. Information Processing and Management. Volume 13, pp. 167-175.

21. R. Morris. Scatter Storage Techniques. Communications of the Association for Computing Machinery. Volume 11/Number 1/ January 1968.

22. M. Tainiter. Addressing for Random-Access Storage with Multiple Bucket Capacities. Journal of the Association for Computing Machinery. pp. 307-315. July 1963.

23. V.Y. Lum, P.S.T. Yuen, and M. Dodd. Key-to-Address Transform Techniques: A Fundamental Performance Study on Large Existing Formatted Files. Communications of the Association for Computing Machinery. April 1971, Volume 14/Number 4, pp. 228-239.

24. J.B. Grimson, and G.M. Stacey. A Performance Study of some Directory Structures for Large Files. Information Storage and Retrieval. Volume 10, pp. 357-364. July 1974.

25. A. Tenenbein and J. Weldom. Probability Distributions and Search Schemes. Information Storage and Retrieval. Volume 10, pp. 237-242. April 1974.

26. E. Horowitz and S. Sahni. Fundamentals of Data Structure. Computer Science Press, Inc., 1976.

27. D.E. Knuth. The Art of Computer Programming: Sorting and Searching. Vol. III. Addison Wesley, Reading, Mass. 1973.

28. F.W. Lancaster. Information Retrieval Systems. Characteristics, Testing, and Evaluation. John Willey and Sons. Inc., 1968.

29. B. Schneiderman. Optimum Data Base Reorganisation Points. Communications of the Association for Computing Machinery. Volume 16. Number 6. June 1973. pp. 362-365.

30. J.R. Bell. The Quadratic Quotient Method: A Hash Code Eliminating Secondary Clustering. Communications of the Association for Computing Machinery. Volume 13. Number 2. February 1970.

31. A. Tenenbein. Expected Number of Passes in a Binary Search Scheme. Information Storage and Retrieval. Volume 10. pp. 29-32. 1974.

32. W.D. Maurer. An Improved Hash Code for Scatter Storage. Communications of the Association for Computing Machinery. Volume 11. Number 1. pp. 35-37. January 1968.

33. J.G. Kollias. The Selection of Secondary File Organizations. Management Datamatics. Volume 5. Number 6. 1976.