ON THE SOLUTION OF
AN INFORMATION RETRIEVAL PROBLEM

Burnett H. Sams
Data Systems Center
Radio Corporation of America
Bethesda 14, Maryland

The problem was to formulate a system which would be capable of digesting an input stream of documents in such a manner as to be able to regurgitate selected information in response to interrogations by a number of research analysts. A number of comments on the problem are in order. Portions of such systems are computers and computer programs serving as information processors. The documents are assumed to contain formatted texts on a sufficiently restricted subject matter to permit mechanical recognition and analysis of what for the moment will be loosely called the informational content of the document. That the system is intended to have a number of users is significant and implies that the person who puts information into the system and the person who takes it out are not always the same. Either there are very tight conventions governing the storage and retrieval of information, or there is a mechanism for associating classes of input and output descriptions. As the number of users or the scope of subject matter increases, it becomes increasingly necessary to provide leeway on taking in information and to provide alternative routes for getting at information stored within the system. In larger systems it may become necessary to accommodate man-machine dialogue enroute to desired information.

This paper is concerned with the design and programming of information retrieval systems. Typical units for measuring the capacity of random-access storage and the size of computer programs are respectively $10^8$ characters and $10^9$ instructions. The computer for this application is taken to be a general-purpose, stored-program, digital computer having a high-speed, random-access memory for program storage, working storage, and data storage of $2^{16}$ - $2^{17}$ words. The internal operations are geared to manipulate strings of bits. The applications will generally require serial magnetic tapes in addition to random-access memories. The computer has programable input and output units capable of operating in a simultaneous manner. This simultaneous processing capability is coordinated with internal processing by a program interrupt subsystem which automatically saves the machine registers on an interruption.

The system discussed below is not an actual development but is an extension of developments leading to frontiers in information retrieval, programming, and computer technology. The ACSI-MATIC Program under contract with the Department of the Army, Office of the Assistant Chief of Staff for Intelligence is developing a major information system to support certain headquarters operations of the U.S. Army.

RETRIEVAL SYSTEMS

A distinction is made between a document retrieval system, an information retrieval system, and a collating information retrieval sys-
tem. In a document retrieval system, documents are stored as they are received, and the object is to retrieve documents having something in common. In an information retrieval system, documents are transformed and stored so as to render their “informational content” accessible to programs whose objective is to retrieve extracts from documents having something in common. In a collating information retrieval system, documents are not retained in the principal data store; instead, composites are formed of information extracted from many documents. Such a system has a potential for research on inference formation and input validation (Fig. 1).

<table>
<thead>
<tr>
<th>System</th>
<th>Storage</th>
<th>Retrieval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Documents</td>
<td>Documents</td>
</tr>
<tr>
<td>Information</td>
<td>Documents</td>
<td>Extracts</td>
</tr>
<tr>
<td>Collating</td>
<td>Composites</td>
<td>Extracts</td>
</tr>
</tbody>
</table>

Figure 1. Document retrieval, information retrieval and collating information retrieval systems are differentiated by their informational units of storage and retrieval.

In document retrieval systems generally, each document is indexed by a set of key terms. Likewise, each composite is indexed by a set of key terms. For example, if one wanted to know the current status of a particular hurricane, the document retrieval system would produce a few dozen weather reports each having something to say about the hurricane. The information retrieval system would produce extracts from the reports dealing with the hurricane. The collating system would have previously organized the data in the weather reports and would produce only the current status of the hurricane. If one asked about a tornado that was also discussed in those same weather reports, the document system would simply present the reports a second time. The information retrieval system would produce those extracts dealing with the tornado, and the collating system would limit itself to the status of the tornado.\(^9\)

INDEXING

Each system requires some kind of indexing scheme that can locate records (documents, extracts, or composites) within a short period of time. The simplest kind of indexing associates an index term with a set of records each of which contains the index term. One may wish a more extensive association. For example, if one wants to get information about the consumption of electricity in New Jersey, one would not want to ignore a document concerning the electric consumption in Jersey City. The interrogator knows Jersey City is in New Jersey. He always wants related information about Jersey City or Newark or Trenton when he wants information about New Jersey. The index term “New Jersey,” therefore, should subsume the index terms “Jersey City,” “Newark,” etc. This particular subsuming is tree-structured and one can devise various techniques for reflecting the desired relationships (Fig. 2). Sometimes a tree relationship of sub-

\[\text{United States}\]

\[\text{Trenton, Newark, Jersey City, New York City, Buffalo, Albany}\]

United States

New Jersey

Groton

New York

New York City

Buffalo

Albany

Figure 2. A section of a tree illustrating hierarchical relations among political units.

subsuming is not adequate. Suppose, in addition to having New Jersey subsume Jersey City, one also wants Greater New York to subsume Jersey City; Jersey City has now become a lattice point in the structure (Fig. 3). Process-

\[\text{United States}\]

New Jersey

Groton

New York

New York City

Buffalo

Albany

Trenton, Newark, Jersey City, New York City, Buffalo, Albany

Figure 3. A section of a graph illustrating a non-tree-structured relation.
ing over an encoding of a lattice or a general linear graph is far more difficult and time-consuming than processing over an encoding of a tree; thus it is inefficient to view all points as graph points for the sake of generality. Each problem may have its own particular kinds of index term associations which may affect efficiency, and it is likely that most problems will have more than one kind. It is concluded, therefore, that information retrieval systems must have the ability to identify index terms as belonging to a class of index terms which permits a set of subsuming relations between elements in the class. One might also permit a set of subsuming relations to be defined among the classes themselves.

As soon as one permits classes of index terms, there is simultaneously a need for means of identifying the class to which a given term belongs and a need to identify terms within a class to see if they are legitimate and to find subsumed index terms. When the number of terms in a class is finite, a simple glossary will suffice. Each entry in the glossary could indicate which of the remaining entries in the glossary are subsumed index terms. When frequent additions to the glossary are anticipated, any scheme should be avoided that depends on the sequence of items in the glossary for these subsumed term indicators. If, however, the number of items in the class is infinite, one needs a recognition procedure. For example, suppose a system indexes all transactions of $1000 or more by indexing each document under the exact amount of the transaction. Suppose further that for standard requests one subsumes under a given amount all other amounts within a half-dollar of the given amount. Then—for an interrogation using $1065.00 as an index term, all amounts greater than $1064.49 and less than $1065.50 are subsumed.

Although the number of possible index terms in a system may be infinite, the number of documents or composites indexed and the number of index terms which actually reference documents are finite but unbounded. Each information retrieval system needs a list of active index terms with their coupling to the records they index. Schemes based upon searching the entire record store for each interrogation are excluded from these considerations of a large data store. The combination of glossaries and an index list is called a thesaurus. The index list is distinct from the glossaries, and some classes having an infinite number of index terms may not require glossaries. By the very fact that index terms may be active or inactive at a particular time, it is necessary to have means for updating the index list. If the list is long, efficiency will dictate that a hierarchy of directories be incorporated. The insertion and removal of terms from the active index list and the corresponding maintenance of directories should be entirely automated.

In cases where the subsumed index terms are from an infinite set, one does not generate them, but rather one examines the index list to see if it contains any index terms that could have been so generated. Clever coding and ordering of the index list can substantially shorten this examination. Actually, it may be profitable to introduce more than one index list or even one list for each class of index terms.

In general, one does not expect to retrieve on a single index term, but rather on combinations. In fact one might expect to perform arbitrary Boolean functions on sets of index terms. In addition, one may desire some external control over the selection of subsumed index terms for any particular term in some pre-established way. In the example above where one indexed all transactions of amounts larger than $1000, one may want for a particular retrieval only those transactions with amounts greater than $100,000. Each index term, therefore, should permit an operation with parameters that can control the selection of subsumed index terms. The permissible operations must be defined for each index list.

In many cases, one needs to relate index terms of different classes for retrieval. If one wants to retrieve all employees who earn $5,000 and who have 2 children, it is not sufficient to use the two index terms connected by “and” since one may get employees with two children earning other than $5,000 or employees earning $5,000 with other than two children if more than one person is described in the indexed record.

It is necessary, therefore, to group index classes and to let the terms in the grouped classes index portions of a record; such portions will be called subjects. For example, consider a document (Fig. 4) that mentions factory #1 and lists the names of all its employees together with the salary each earns and the
number of children each has. Suppose in particular, there are three employees Abel, Baker, and Carter who earn $5,000, $10,000 and $5,000 and who have 2, 2, and 3 children respectively. If the document were given the number 100, one could index as illustrated in Fig. 5, where the index numbers may consist of one or two parts, the first referencing the document and the second referencing the employee in the document. Each person has become a subject within the factory; a sub-subject of the subject “Factory.” Actually, if more than one factory were mentioned, one would require three parts for each index number, one each for document, factory, and person. Note that the number of persons per factory is unbounded. The grouping of index classes, which parallels the scheme for assigning index numbers and which parallels the structure of subjects in documents, is tree-structured.

### DATA STRUCTURE

An information store and its interaction with indexing may now be described. The data structure is concerned with the storage of information collected on the various subjects of interest for a particular information retrieval system. The subjects, for example, may exhibit hierarchical relationships such as persons employed by factories or components within assemblies.

The informational units for processing and for retrieval are fields. Fields may possess subfields or be combined to form larger fields. A field may exhibit three kinds of variability; the possible values of a field may not all have the same size representation, or more important for present considerations, a field may require a variable length list of values which are considered either as additional subfields or as alternative values called replications. The smallest fields are called primitive fields. Each primitive field has a specified ordered set of values called primitive values. Fields are defined recursively as being either a primitive field or a list of fields; these latter fields are then called subfields. A field value, or more briefly value, is correspondingly either a primitive value or a list of subfield values. If some subfield is replicated, there is not a uniquely determined value. Accordingly, the scope of a field, or more briefly scope, is defined as the set of all values obtained by ranging over all combinations of subfield replications. The set of all possible field values is ordered in one or more ways as extensions of the orderings defined for each subfield. For example, the field “Personality name” may have the three subfields, “Last name,” “First name,” and “Middle name,” which are further broken down by “Characters” to permit such retrievals as “all electrical engineers in Detroit whose first name is Alan,” or alternatively, whose last name ends in the letter “y.” Note that a distinction is drawn between the field value “Smith” and the instance of the subject “Person” that is known by the name “Smith.” Subjects are organizational entities in correspondence with the objects, individuals, or abstract categories to which the descriptive information pertains. It is convenient to also distinguish—from the fields which describe subjects—the fields, called auxiliary fields, which describe the relationship between other fields and the subject. For instance, the population of a city may be accompanied by a date and reference source. In any implemented system there will also be control fields to compensate for the several kinds of format variability allowed.

It is now possible to define information as the holding of a given set of relationships among a set of fields with given values. The informa-
tional content of a document is meaningful as the totality of information resulting from an analysis of the document in terms of field structure and values. An analysis of each document is required in a collating system since documents are not retained in the main data store.

Some fields are distinguished as the generators of index classes. An index class generated by a field consists of the set of all possible field values; these values are called index terms of the index class—e.g.:

**Index class**: Manufacturer-Vehicle

**Index term**: Ford-Truck

The set of index terms is organized into a number of directed linear graphs (not necessarily trees or lattices) by a number of functions on the set of index terms whose values are pairs of sets of ancestor and descendent index terms respectively.

Other fields are distinguished as the generators of subject classes. The terms subject value and subject scope are carried over from the field definitions. An attribute of a subfield which is also a generator of a subject class will be referred to as a sub-subject attribute. The members of a subject class are units of storage corresponding to the subject scopes actually retained in the information store of the system. Subjects are the units of indexing. Each index term points to a number of subjects, and each subject is pointed to by at least one, and usually more, index terms.

In the index list, besides identifying a particular subject, it is necessary to identify the class to which the subject belongs; this enables distinguishing between the “Location” in which a person resides and the “Factory” in which he works. The choice of subject classes is matched to the choice of index classes so that the set of subjects to which a pair of index terms jointly apply is precisely the intersection of the two sets of subjects to which the index terms individually apply. Consider for example, the subject class “Automobile ownership” consisting of the two subject classes “Family” and “Auto” (Fig. 6) and an instance of “Automobile ownership” (Fig. 7). The items with asterisks are subject identifiers and the subject “Auto” is replicated.

In an information retrieval system it is possible that only a small fraction of the information is deducible from the index list-subject relationship. In the design of such a system, the statistics of processing and retrieval, the sizes of files, and the nature of the hardware configuration are prerequisite to making a reasonable determination of indexing structure and the organization of subject storage. In general, information will be stored in units of records which correspond to subjects (usually with sub-subjects) for which the statistics of record size are well matched to the storage media. For many applications the number of fields per record may be rather large, so that one would not want to reserve space in the record for fields which may never be given values in a particular record, and considering value replications, one cannot know in advance the number of field values to be stored in a given record.

Two elements of information storage, the field and the record, have been introduced. The primitive field is the smallest piece of data that will be named and manipulated. The record is the largest piece of data that will be manipulated as an entity. Records are composed of fields. Within a record, any field or combination of fields may be associated with other fields or

**Figure 6. Example of subject structure.**

**Figure 7. Example of data structure.**

From the collection of the Computer History Museum (www.computerhistory.org)
combination of fields provided the association is independent of the particular datum that "fills" the field and provided the associations can be arranged to represent a tree structure for the record. A field is a form that has a value. The value may be any of a set of terms defined by the application or it may be the vacuous symbol. A record is a tree structure of fields independent of the values of the fields. It is to be emphasized that the values for a given field are not required to be tree-structured. It is only the fields themselves that must be so structured in order to provide a scheme for assigning names to data sets stored in the record.

Any piece of information in the system has a name which is factorable into:

- Record identification
- Subfield replicate

The derivation of names follows directly from the description of records, subjects and fields. It is recognized that a given term might possess different descriptions for input, indexing, storage, processing, and output as demanded for processing and storage efficiency.

RETRIEVAL

When one submits an interrogation saying he wants all information about a particular hurricane, he is saying that he wants to restrict himself to documents about that hurricane. Actually, information is retrieved. The documents from whence the information derived may be retrieved to substantiate a claim or to furnish more detail than is formalized within the system. Similarly, if he wants information about all factories that have employees who earn $10,000 and who have 2 children and that have employees who earn $5,000 and have 3 children, he is likewise restricting himself to certain records. The terms that do the restricting, we call restrictors. Restrictors need not always be index terms. For example, one may not choose to use the date of a document as an index term, but one may still choose to restrict by that date. This assumes that one can find such a date in a given record and test it. This is no problem in an information retrieval system, but it requires some special organizing in a document retrieval system.

In an information retrieval system, one may have many restrictors that are not index terms. These terms must be tested after the appropriate record is selected via the index list. This is essentially a two-stage selection procedure. First using the index terms, one gets all records that are associated with the index terms and their subsumed terms. The choice of which restrictors to use for this purpose and the order in which to use them is not necessarily prespecified. This may be determined for each retrieval by applying heuristics based on previous retrieval efficiencies. The subjects obtained are then tested, and only those are chosen which satisfy the remaining restrictors.

In a document retrieval system one displays the whole document that has been retrieved and it is up to the interrogator to find what he wants. In an information retrieval system, one may specify those items (the extractors) he wants to see. In non-collating systems the extracted information must be culled to remove redundant and inconsistent information. It is necessary that the interrogator know what items it is possible for him to see; that is to say, he may extract only those items that can be recognized by the system. This, however, is nothing new, for the same requirement is imposed upon him in using restrictors. The extractors can be expressed in the same manner that the restrictors are expressed. If information is not present, the extractors or groups of extractors may be ignored.

With a little additional effort, one may extract only if the sought data is subsumed by an index term. For example, one may extract a location only if it is subsumed by "Michigan." This is not quite the same as a round-about restrictor. A more vivid interrogation that exemplifies this is: "Give me (extract) the names and addresses of (restrict) all persons in factory #1; also give their positions if (conditional extraction) they are among the professional staff." Without the conditional extraction, one could at best obtain two lists—one with the names and addresses of all employees and one with the names, addresses and positions of the..."
professional staff. Of course one can permit Boolean functions of extractors.

In collating systems, once one has a set of records to be displayed, there is the problem of sorting them in some specified manner. If the field being sorted on appears at most once in every record, there is no new problem. Suppose, however, each record from which we have extracted our data has as subject a particular apartment house in some city and suppose that among the data in the record are the names of all the tenants and the address of the apartment house. If the interrogation reads: “In alphabetical order, list the names and addresses of all people who live in apartment houses in that city,” artificial records are created for sorting with the address of the house duplicated for each person who lives in it. Further complications arise if more than one level of sorting is specified at one time.

The retrieved information is extracted as subsets of selected subjects. This information therefore has subject structure and must be printed so as to reflect that structure. That is, replications of a particular sub-subject or field are vertically aligned and are vertically spaced so as not to conflict with each other. This also applies to values and repeated fields that exceed allotted column widths.

PROCESSING

Among the information processing programs are three which characterize solutions of the information retrieval problem. The first program prepares an information structure which represents the terms and syntax of a document or interrogation in a form convenient for machine processing. The same language is used for couching requests and for preparing inputs. A thesaurus is used to identify terms, to resolve some problems of synonymy and ambiguity, and to associate general and specific meanings. The variety of inputs renders impractical a fixed data structure to accommodate any allowed input. Different classes of terms require different sets of programs to incorporate a term into the information structure, and individual terms may introduce processing variations.

The second program extracts from the data store information structures satisfying selection criteria. The types of field structures and the selection criteria will determine the sub-programs to be used for selection and the amount of working storage required.

The third program forms composites of information structures. A number of information structures related to a given one are selected as candidates for merging into one or more composites. This collating process may discover conflicting information which is then directed to programs which resolve the conflict. There is no inconsistency in retaining conflicting information. As additional related information enters the system, the conflicts will be reexamined until a decision can be made to resolve the conflict. Many programs are required to handle the many different situations that may arise. These programs are large and have dissimilar storage requirements.

The computer programs are large because the design problems do not admit easy solutions. The data is varied; the processing is intricate, and the programming is correspondingly complicated. Each option and each event considered adds to the sizes of the programs. The computer programs may be large and yet the result may be only a crude approximation of the desired intelligent behavior. More cases, more flexibility, and new levels of processing detail may all be required in order to achieve a fully useful result. Finally, the programs are large because the problem is large and changing; or rather, the programs are large because there is not a single problem but a series of problems arising out of changes in requirements, changes in technology, and changes in understanding the problem—throughout, the programs must remain responsive to change.10

By a large program is meant first one which translates into such a large number of machine instructions that the program cannot fit all at once into the main memory of the computer. But the size alone is not a sufficient criterion since the program might be organized to consist of a number of sub-programs which may be brought into the main memory in a simple sequential fashion. In order for largeness to be an inherent characteristic of the program, mere size must be related to program complexities.

Program complexities may arise from both the organization of the computational algorithm and the manner in which the program is executed. The flow of the code into the main memory of the computer may be complicated
by requiring the programming parts to be executed in a variety of arrangements; these arrangements may be determined by the computer data being processed. The execution of a code may be interrupted and the code removed from the main memory in order to allow another program to proceed; at a subsequent time the program is restored to memory in such a way as to resume the execution from the point of interruption. The program may instruct the computer to perform an elaborate computation or the data may possess intricate relationships; in the latter case the literature on list organization and list processing is relevant to information processing.

This brief excursion through representative processing was conducted to suggest:

1. That many programs are involved;
2. That the next program to be executed is a function of the information being processed;
3. That the programs have variable storage requirements.

The storage requirements of the programs are such that only a few of them may coexist in the computer's memory. Accordingly, one expects the proportion of input-output transmission devoted to programs as opposed to data to be high; perhaps two orders of magnitude higher than for representative data processing problems. The managing of program flow becomes a major part of the information retrieval problem.

PROGRAMMING SYSTEMS

Consider for a moment the development of data processing. Faced with mountains of data, first machines and later computers and computer programs were developed to order and maintain the data; eventually, generalized file control programs appeared, not to process data, but to merely transmit data to and from data processing programs. Now, faced with mountains of data and smaller mountains of programs, additional programs are needed to manage the programs which control and process the data. These second level programs constitute what is commonly referred to as an executive system.

As processing becomes more complicated and executive systems become more deeply enmeshed in the computation, the programs which make up the computation must adhere to more and more conventions. Before the point is reached at which the conventions become overburdening to the programmer, some of the work of following the conventions is passed over to compilers and other automated programming aids. Now if a compiler is to produce programs to be executed by a given executive system, the compiler is constrained by the executive system and they become part of a programming system which encompasses the design, construction, debugging, maintenance, and operation of programs. 1, 2

Returning to the management of programs for the information retrieval problem, there is an initial requirement for an executive system which interprets transfers between programs to insure that the program transferred to is in memory and, in case it is not in memory, to bring it into memory after saving the necessary registers. It must be noted that programs are not disjoint; in general, successive programs will have considerable overlap of common code and working areas. The definition of program must therefore be broad enough to include portions of code produced by compilations of different programs. Furthermore it is desirable to allow assigning work areas to programs as their requirements become known during execution.

An example will suggest the richness of program structure that is applicable to the information retrieval problem. A program in use occupies an interval in space-time. The spatial entities are memory sequences, input files, output files, other programs which this program might call into use, and such entities borrowed at execution time from programs already in use. Each entity may be further partitioned to take advantage of hardware configurations. For instance, a memory sequence may be partitioned to take advantage of non-contiguous memory locations; likewise a magnetic tape file may be partitioned into reels by physical necessity or to permit faster accessing. At the program execution level, time is single valued. Time is partitioned into execution phases during which some subset of the spatial entities are in use. As the execution progresses from one phase to the next, some spatial entities drop out of use and others come into use. The term phase refers to the spatial entities used during an interval of time; the spatial entities used
during a phase may be used again during a later time interval; phases are ordered with repetition by the computation to form execution time. Program execution time is organized by parallel processing and multi-programming techniques to form process time. Thus as applied to the entire program complex, time has a multi-valued character. At any given moment of computer time, calculation has proceeded to some point in some phase of each process. Concurrently, computation is proceeding in one phase, input-output transmission is going on in other phases, and the remaining processes are waiting in various states of readiness. These two components of time are controlled by different mechanisms. Process time is changed in response to hardware or external signals; phase time is changed in response to computation.

There is no intended implication as to how or when spatial entities are assigned to hardware facilities. The above remarks apply to those problems in which all assignments can be made by the programmer and compiler and to those problems in which some assignments are postponed to the time of loading and to the time at which they are required by the computation.6

The author is indebted to many persons for conversations which have contributed to the developing or testing of these ideas. Some have left their imprint upon the paper and are singled out for special acknowledgement: Anatol Holt for the introduction of phase and sequence into program structure, John Goodroe for the connection with multi-programming, and Bob Colilla for developing the sections on indexing and retrieval.

BIBLIOGRAPHY
