# The Binary Vector as the Basis of an Inverted Index File

Donald R. KING: Rutgers University, New Brunswick, New Jersey.

*The inverted index file is a frequently used file structure for the storage of indexing information in a document retrieval system. This paper describes a novel method for the computer storage of such an index. The method not only offers the possibility of reducing storage requirements for an index but also affords more rapid processing of query statements expressed in Boolean logic.*

## INTRODUCTION

The inverted index file is a frequently used file structure for the storage of indexing information in document retrieval systems. An inverted index file may be used by itself or with a direct file in a so-called combined file system. The inverted index file contains a logical record for each of the subject headings or index terms which may be used to describe documents in the system. Within each logical record there is a list of pointers to those documents which have been indexed by the subject heading in question. The individual pointers are usually in the form of document numbers stored in fixed-length digital form. Obviously, the length of the lists will vary from record to record.

The purpose of this paper is the presentation of a new technique for the storage of the lists of pointers to documents. It will be shown that this technique not only reduces storage requirements, but that in many cases the time required to search the index is reduced. The technique is useful in systems which use Boolean searches. The relative merits of Boolean and weighted term searches are beyond the scope of this paper, as are the relative merits of the various possible file structures.

## THE BINARY VECTOR AS A STORAGE DEVICE

The exact form of each document pointer is immaterial to the user of a document retrieval system as long as he is able to obtain the document he desires. The standard form for these pointers in most automated systems is a document number. Note that each pointer is by itself a piece of information. However, if one thinks of a "peek-a-boo" system, the document pointer becomes simply a hole punched in a card. In this case the position of the pointer, not the pointer itself, conveys the information. The new technique presented in this paper is an extension of the "peek-a-boo" concept.

A vector or string of binary zeroes is constructed equal in length to the number of documents expected in the system. The position of each vector element corresponds to a document number. That is, the first position in a vector corresponds to document number one and the tenth vector position corresponds to document number ten. A vector is constructed for each subject heading in the system. As a document enters the system, ones are inserted in place of the zeroes in the positions corresponding to the new document number in the vectors for the subject headings used to describe the document. As an example, assume the following document descriptions are presented to a system using binary vectors:

| *Document Number* | *Subject Headings* |
|---|---|
| 1 | A, B, D |
| 2 | C, E |
| 3 | A, C |

The binary vectors for terms A, B, C, D, and E before the insertion of the indexing data would be as follows:

| *Subject Heading* | *Vector* |
|---|---|
| A | 000 . . . 0 |
| B | 000 . . . 0 |
| C | 000 . . . 0 |
| D | 000 . . . 0 |
| E | 000 . . . 0 |

After the insertion of the indexing information, the same vectors would appear as follows:

| *Subject Heading* | *Vector* |
|---|---|
| A | 101 . . . 0 |
| B | 100 . . . 0 |
| C | 011 . . . 0 |
| D | 100 . . . 0 |
| E | 010 . . . 0 |

The binary vector seems to have several advantages over the standard form of storage of document numbers in an inverted file. First, the records are of fixed length since the vectors are all equal in length to the expected number of documents in the system. Space may be left at the end of each vector for the addition of new documents. Periodic copying of the file may be used to expand the index records with additional zeroes added at the end of each record during the process. Consequently, unless there are limitations of size imposed by the equipment, only one access to the storage device will be needed to retrieve the index record for a term. The second advantage offered by the binary vector method appears in the search process. Most modern computers have a built-in capability of performing Boolean logical manipulations on binary digit vectors or strings. Thus, when Boolean operations are specified as part of a query, the implementation of the operations within the computer is considerably easier and faster for binary vectors than for the standard form of inverted files.

Other investigators of the use of the binary digit patterns or vectors have not fully explored its advantages and disadvantages. Bloom suggests, without an explanation or evaluation, the use of bit patterns as the storage technique for inverted files in large data bases in the area of management information systems.[1] Davis and Lin, again in the area of management information systems, propose bit patterns as the means of locating pertinent records in a master file.[2] They do not compare the method with other possible techniques. Sammon discusses briefly the use of binary vectors as a storage technique, but dismisses it on the basis that the two-valued approach obviates the possible assignment of weights to index terms in describing documents.[3] Gorokhov discusses the use of a modified binary vector approach in a document retrieval system implemented on a small Soviet computer.[4] Faced with the need to minimize storage requirements for his inverted file, Gorokhov concentrated on developing a technique for locating and removing strings of zeroes occurring in the binary vectors used within the system. Since these zeroes represent the absence of information they could be removed if there were a way to indicate the position in the original vector of the ones that remained. He proposed the removal of strings of zeroes and the inclusion of numeric place values with the remaining vector elements. His result is a file with variable-length index records. The abandoning of the pure binary vector obviates the process, and Gorokhov found it necessary to expand the vector elements into the original vector before logical operations could be applied. Even though he does not state so explicitly, Gorokhov seems to have found his method more efficient than the standard inverted file. Gorokhov's suggestion has led to the development of an algorithm for the compression of binary vectors.

Heaps and Thiel have also discussed the use of compressed binary vectors as the basis of an inverted index file.[5, 6] Aside from a brief description of the method for implementing the concept, they offer no comparison of the binary vector with the standard inverted file.

## STORAGE REQUIREMENTS

An immediate reaction to the concept of binary vectors is to state that they will obviously take more storage space than the standard inverted file. A closer study shows that this is not always the case. The storage requirements for the two types of files may be calculated as follows:

1. $M_{BV} = \frac{D \cdot N}{8}$ bytes (binary vector file)
2. $M_{SI} = D \cdot J \cdot K$ (standard inverted file)

where:

$M$ = Storage requirements in bytes
$D$ = Number of documents in the system
$N$ = Number of index terms in the system
$J$ = Average depth of indexing in the system
$K$ = Size in bytes of a document number stored in the file

Using equations 1 and 2 we find that the storage requirements for the binary vector file are, in fact, less than the requirements for the standard inverted file if $N < 8 \cdot J \cdot K$.

It is well known that the distribution of the use of index terms follows a logarithmic curve. In simple terms, one might say that a few terms are used very frequently and many terms are used infrequently. This condition implies that in a binary vector file the records for many terms will contain segments in which there are no "ones" in any byte. A method for removing these "zero" bytes is called compression.

## COMPRESSION ALGORITHM

The technique for the compression of binary vectors as described here is designed specifically for the IBM 360 family of computers and similar machines. The extension to other machines should be obvious.

Within the IBM 360 the byte, which contains eight binary digits, is the basic storage unit, and with the eight binary digits it is possible to store a maximum integer value of 255. For the purpose of describing a proposed compression algorithm for the binary vector in the IBM 360, the term *subvector* will be defined as a string of contiguous bytes chosen from within the binary vector. A *zero subvector* will be a subvector each of whose bytes contains eight binary zeroes. A *nonzero subvector* will be a subvector each of whose bytes contains at least one binary one. To compress a binary vector in the IBM 360 the following steps may be taken:

1. Divide the binary vector into a series of zero subvectors and nonzero subvectors. Subvectors of either type may have a maximum length of 255 bytes. For zero subvectors longer than 255 bytes, the 256th byte is to be treated as a nonzero byte, thus dividing the long zero subvector.
2. Each nonzero subvector is prefixed with two bytes. The first of the prefix bytes contains the count of zero bytes which precede the nonzero subvector in the uncompressed vector. The second prefix byte contains a count of the bytes in the nonzero subvector.
3. The compressed vector then consists of only the nonzero subvectors together with their prefix bytes.
4. A two byte field of binary zeroes will end the compressed vector.

The compression of the vectors creates variable-length records and removes the advantage of having records which are directly amenable to Boolean manipulation. The effect of file compression on such manipulation in the search process is not as severe as it might appear. For the search process, the compressed vector may be expanded into its original form. The process of expansion of the binary vectors is relatively simple, and since only those index term records which are used in a query need to be expanded at the search time, the search time is not significantly affected.

As an example of the use of the compression algorithm consider the following binary vector.

01100000/10000000/ seven zero bytes /00000001/10000000/ . . .

The slashes indicate the division of the vector into bytes. The vector might be read as indicating the following list of document numbers:

2, 3, 9, 80, and 81.

In a standard inverted file with each document number assigned three bytes of storage, fifteen bytes would be required to store these numbers. The compressed vector which results from the application of the algorithm is the following:

00000000/00000010/01100000/10000000/00000111/00000010/
00000001/10000000/ . . .

Again the slashes separate the vector into bytes. For the purpose of the following discussion consider each byte in a vector to be numbered sequentially beginning with byte one at the left.

In the uncompressed vector bytes one and two form a nonzero subvector. Consequently, the first four bytes in the compressed vector can be interpreted as follows:

| | |
|---|---|
| Byte one. | Binary zero indicating that no zero bytes were removed preceding this subvector. |
| Byte two. | Binary two indicating that the following nonzero subvector is two bytes long. |
| Bytes three, four. | Bytes one and two of the original vector. |

Bytes three through nine of the original vector are a zero subvector, and bytes ten and eleven form a second nonzero subvector. Consequently, the second four bytes of the compressed vector are interpreted as follows:

| | |
|---|---|
| Byte five. | Binary seven indicating that a zero subvector of seven bytes has been removed. |
| Byte six. | Binary two indicating that the following two bytes are a nonzero subvector. |
| Bytes seven, eight. | Bytes ten and eleven of the original vector. |

Thus the binary vector has been reduced from eleven bytes to eight bytes while the space required to record the document numbers in the standard inverted file remains fifteen bytes.

## MEMORY REQUIREMENTS FOR THE STANDARD INVERTED FILE AND THE BINARY VECTOR FILE

To compare memory requirements for the standard inverted file and the compressed binary vector file, we base our comparison on the total number of postings in the file. In the standard inverted file the storage space for the postings is equal to the number of postings times the length of a single posting, which is usually two, three, or five bytes. Memory requirements for the compressed binary vector file are more difficult to estimate because the distribution of document numbers within the record for each index term is not known. The fact that a single byte in the binary vector file may contain between zero and eight postings is extremely important. The worst possible case occurs if the postings in the binary vector are spaced in such a way that each nonzero byte contains only one posting, and these bytes are separated by zero bytes. Consider the following example:

. . . /00000000/00010000/00000000/00000100/ . . .

In this case the compression algorithm will remove the zero bytes, but will add two bytes (the prefix bytes) for each nonzero byte. The resulting compressed vector will be essentially the same length as the standard inverted file record if each posting is three bytes long in the standard inverted file. It might seem that the distribution of one posting per byte for the entire vector represents an even worse situation. It is clear that the compression algorithm will, in this case, not reduce the size of the vector. However, it must be remembered that in the standard inverted file each posting will require at least two bytes and perhaps three bytes. Thus, the length of the record in the standard inverted file is two or three times longer than the corresponding binary vector regardless of compression.

In data used in two model retrieval systems prepared to compare the standard inverted file and the binary vector file there are 6,121 documents with a total of 94,542 postings. An examination of the binary inverted file for the model systems discloses that there are only 55,311 nonzero bytes in the binary vector file. Thus there seems to be some form of clustering of the document numbers in each index term record. If each nonzero byte in this binary vector is isolated by zero bytes, two prefix bytes would be added for each byte. Thus the total memory requirements for the postings in the compressed file would be 165,933 bytes. Less storage space is required if some nonzero bytes are contiguous. On the other hand, the standard inverted file will require 189,084 bytes if a two-byte posting is used, or 283,626 bytes if a three-byte posting is used. Further study of the clustering phenomenon is needed.

## MODEL RETRIEVAL SYSTEMS

To test some of the conjectures about the differences between the standard inverted file and the binary vector file, two model systems were prepared for operation on an IBM 360/67. Details of the systems and PL/1 program listings are available elsewhere.[7] The data base used was obtained from the Institute of Animal Behavior at Rutgers University. In the data base 6,121 documents were indexed by 1,484 index terms. A total of 94,542 postings in the system gives an average depth of indexing of 15.4 terms per document. Both inverted files were stored on IBM 2314 disc storage devices.

To ease the problem of handling variable-length records in both files, the logical records for each index term were divided into chains of fixed-length physical records. For the standard inverted file a physical record size of 331 bytes was chosen. The entire file required 702,713 bytes including record overhead. For the uncompressed binary vector file a physical record size of 1,286 bytes was chosen to include overhead and space for up to 10,216 document numbers. When the compression algorithm was applied, with a physical record length of 130 bytes, the memory requirements for the binary vector file were reduced to 281,450 bytes, or 41 percent of the space required to store the standard inverted file.

A series of forty searches of varying complexities were run against both files. The "TIME" function of PL/1 made it possible to accumulate timing statistics which excluded input/output functions. Search times for the binary vector file include expansion of the compressed vectors, Boolean manipulation of the vectors, and conversion of the resultant vector into digital document numbers. The times for the standard inverted file are for the Boolean manipulation of the lists. The following points were noted in the analysis of the times:

1. In twenty-two of the forty queries for which comparative timings were obtained, the search of the binary vector file was faster, in one case by a factor of thirty-five. In the eighteen cases in which the search of the standard inverted file was faster, the search of the standard inverted file was at most 6.17 times faster.
2. The range of the total times for the binary vector file was .79 seconds to 9.72 seconds. The range for searching the standard inverted file was .15 seconds to 202.98 seconds. The fact that the search times for the binary vector file are within a fairly narrow range, in contrast to the wider range of times for searching the standard inverted file, has important implications for the design of an on-line interactive document retrieval system. In such a system it is important that the computer respond to users' requests not only rapidly but consistently. The narrower range of the search times provided by the binary vector file will assist in producing consistent times.
3. The search times for the binary vector file, exclusive of expansion and conversion times, are unaffected by the number of postings con-

tained in the index terms used in a query. On the other hand, the number of postings in the records used from the standard inverted file appears to cause the differences in search times for that file.

To test the conjectures that

1. search times for the binary vector file are related to the number of index terms in the query, and
2. search times for the standard inverted file are related to the number of postings in the index terms in the query,

a correlation analysis was performed. The following correlation coefficients were obtained:

| *Variables* | *r* |
|---|---|
| Number of terms in query and search times for the binary vector file. | .960 |
| Number of postings in query terms and search times for standard inverted file. | .979 |

The relationships indicated above are significant at the .001 level. No attempt was made to compute an average search time per term for the binary vector file or average search time per posting for the standard inverted file. Such times would have meaning only for the model systems.

## SUMMARY

The binary vector is suggested as an alternative to the usual method of storing document pointers in an inverted index file. The binary vector file can provide savings in storage space, search times, and programming effort.

## REFERENCES

1. Burton H. Bloom, "Some Techniques and Trade-Offs Affecting Large Data Base Retrieval Times," *Proceedings of the ACM* 24 (1969).
2. D. R. Davis and A. D. Lin, "Secondary Key Retrieval Using an IBM 7090-1310 System," *Communications of the ACM* 8:243–46 (April 1965).
3. John W. Sammon, *Some Mathematics of Information Storage and Retrieval* (Technical Report RADC-Tr-68-178 [Rome, New York: Rome Air Development Center, 1968]).
4. S. A. Gorokhov, "The 'Setka-3' Automated IRS on the 'Minsk-22' with the Use of the Socket Associative-Address Method of Organization of Information" (Paper presented at the All-Union Conference on Information Retrieval Systems and Automatic Processing of Scientific and Technical Information, Moscow, 1967. Translated and published as part of AD 697 687, National Technical Information Service).
5. H. S. Heaps and L. H. Thiel, "Optimum Procedures for Economic Information Retrieval," *Information Storage & Retrieval* 6:137–53 (1970).
6. L. H. Thiel and H. S. Heaps, "Program Design for Retrospective Searches on Large Data Bases," *Information Storage & Retrieval* 8:1–20 (1972).
7. D. R. King, "An Inverted File Structure for an Interactive Document Retrieval System" (Ph.D. dissertation, Rutgers University, 1971).