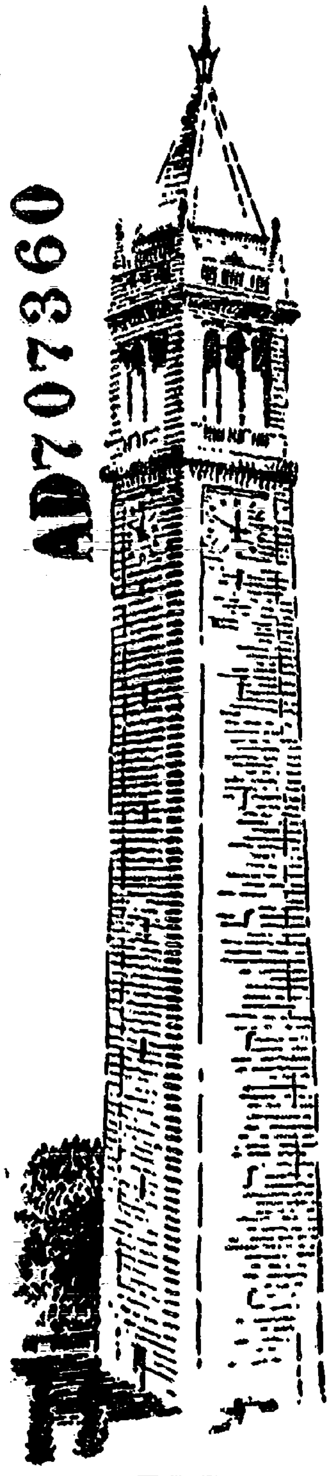


AN INFORMATION RETRIEVAL SYSTEM BASED ON SUPERIMPOSED CODING

by

John R. Files
and
Harry D. Huskey

10 July 1969

PROJECT GENIE REPORT NO. P-20
SUPPORTED BY ARPA CONTRACT SD-185

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley

28

AN INFORMATION RETRIEVAL SYSTEM BASED ON SUPERIMPOSED CODING

by

John R. Files

and

Harry D. Huskey

University of California

Santa Cruz

10 July 1969

P-20

The research reported on here was done at the University of California at Santa Cruz with partial support from Project Genie at the University of California at Berkeley (Contract SD-185 with The Advanced Research Projects Agency of the Department of Defense).

A method of coding a large file for information retrieval is discussed. Random superimposed coding of machine derived "roots" of the full v abulary is used to generate an easily updatable and very compact code file. No thesaurus or dictionary of terms is needed. High speed is made possible by the simplicity of the searching algorithm as well as the ability to make a search for several key words simultaneously. The simplicity of the search facilitates implementation of the system on a small computer with access to a large bulk storage device.

The cost of storing information in machine-accessible form has declined markedly in the last decade, and promises are such that one can look forward to having complete libraries available in such form. This places increased importance on algorithms which make it possible to search large files efficiently.

This paper describes an approach to this problem.

In practice, information in a large file can be more efficiently accessed if it is indexed in some manner. The method of indexing which will be discussed is particularly well suited for a file which:

1) is very dynamic with both deletions and additions frequently occurring.
2) contains an extensive vocabulary which is to be encoded.

Both of these characteristics are frequently found in files that are to be coded. A file of information on recently published articles about a given subject and a card catalogue for a large library are good examples of files which require a large amount of maintenance. If updating the index (code file) is expensive and time-consuming, updating is put off until it is felt that the performance of the system has deteriorated enough to justify the effort required to update it. Until the updating takes place, information which is no longer of use is still retrieved, and the new information, if present, is in a secondary file. Keeping a secondary file containing recent additions avoids the serious problem of not having new material available, but it does decrease the efficiency of the system since such a file must be searched separately each time an inquiry is made of the main file.

The ability to utilize an extensive vocabulary is also very important. In the proposed system the vocabulary to be used is derived directly from words used in the original documents, thereby eliminating the time-consuming and expensive practice of manually abstracting and choosing indexing terms. Machine generated derivatives of the original vocabulary retain more information about the original content of the item than does the manual system of assigning descriptors. In the manual case when selected descriptors are assigned to a document, associations of descriptors to words and to phrases are made. Such associations are not made in exactly the same manner by two trained indexers, and it is likely that the associations made by the average interrogator of an information retrieval system will be even more diverse. Because of this lack of uniformity in assigning descriptors it is desirable to allow each searcher to determine words and phrases that he wishes to associate with the concept on which he is doing a search. Postponing such associations until the time of the search can be accomplished only if the entire word content is preserved in the coded form.

Ease of update and freedom of vocabulary are not enough in themselves to make a coding algorithm worthwhile. Factors such as speed of access, ability to make searches for combinations of words and compactness of code file are also important considerations. All of these characteristics will be discussed for the coding scheme discussed below.

THE SYSTEM

The information retrieval system which was investigated can be divided into three components: preparation of the text, generation of the code file, and the searching procedure. A general outline of the first two components can be seen in Figure 1.

Since the form and format of the text to be used can be expected to vary greatly, the text is standardized as it is read in. Flags are set to indicate boundaries between records as well as at the ends of lines to make it easier to reproduce the document when it is retrieved. Also, as a measure to reduce the bulk of the file generated (text file) extra blanks in the input text are removed. In the pilot system the text file was generated from two sources: a bibliography of computer science and a listing of authors and titles from recent issues of The Computer Group News of the IEEE. Both of these texts were read, processed, and stored on a disk. The text file generated was 100,000 characters stored one character per byte.

Once the text file is generated coding can proceed. The text file is examined character by character until the end of a string which is to be coded (word) is encountered. The unit coded is a string of at least three alphabetic characters surrounded by non-alphabetic symbols (an English word). After the word is found it is compared with a list of non-content words. (i.e., the Delete List containing words such as: of, the, and etc). If the word is found in the Delete List there is no further processing of that word, and the next word is considered.

When a word is found that is not in the Delete List, the trimming algorithm is applied to reduce the word to a pseudo-root. Common endings such as s, ed, ing and compound endings such as fully (as in carefully) are removed. By removing endings, different forms of the same word are made into synonyms. For example, the words 'computer' and 'computers' will both be reduced to the base 'comput.' This derived root is then passed on to the coding procedure. (further discussion of trimming algorithm in Appendix C).

In the coding procedure, a code word is generated for each record. The code word can be thought of as a bit string containing N bits, all of which are initialized to zero at the beginning of the coding operation. When a trimmed word is to be coded into the code word, the numeric value of the letters in the trimmed word is summed, giving a number which is used to choose an element from the uniform distribution of integers between 1 and N. Thus the resultant integer (code value of the word) is generated by an algorithm which given the same trimmed word in the future will generate the identical code value for that word. By using a fixed arithmetic procedure to produce the code value for a word, the need for a dictionary of words and assigned code values disappears. This frees the large amount of storage which such a dictionary would occupy as well as saving the time required to search such a file.

If for a particular word the code value generated is K, then the K'th bit in the code word is set to one. The entire operation of finding a word, checking the Delete List to see if it should not be coded, trimming, and coding is repeated until the entire record is processed. The code word which is uniquely determined by the words in the record is then stored in a file (code file) along with a pointer to the beginning of the record in the text file. This procedure is repeated until all the records have been coded.

(Figure 1)

After coding the file is ready for searching. The searching program accepts any number of words, each of which is processed in the same manner as the words in the text file. It is looked for in the Delete List, trimmed, and used to generate a code value. This code value is then used to produce a query code in exactly the same way as the code words were produced in the code file. Upon generation of the query code the actual search may begin. Each code word in the code file is matched against the query code to see if the query code is a subset of it. (Here a bit string X is said to be a subset of another, Y, if when the I'th bit in X is one, the I'th bit in Y is also one. i.e., 1010 is a subset of 1011 while 0101 is not.) Each time that the query code is a subset of the code word, the pointer to the text file is used to gain access to the corresponding record which can be further processed to see not only if it contains the relevant words, but that the words are in the correct order.

The above is a brief description of the coding suggested for a file of an information scanning program. Some details such as the exact procedure for removing endings and the use of several independently generated code values to produce multiple code words for a given record, were not dealt with here. A more detailed treatment of these problems can be found in the appendix.

RESULTS

From the pilot system, data was gained on the performance of such a system of superimposed coding. When possible, the performance of the superimposed coding system will be compared with that of a threaded list and inverted file. (figures 2 and 3) The following factors received major consideration:

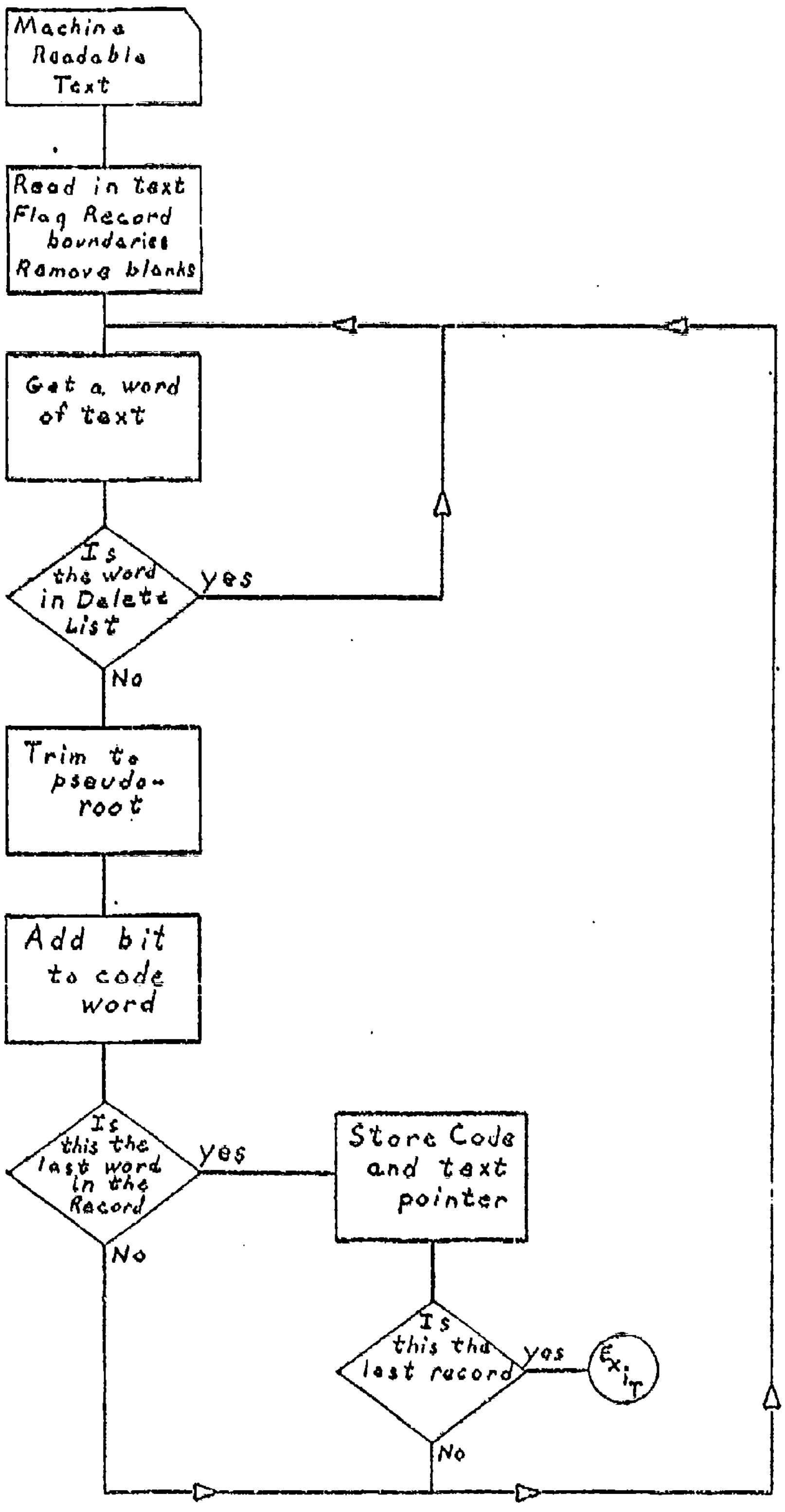


FIGURE 1 Coding Procedure

1) Ease of update
2) Effect of a large vocabulary
3) Amount and type of storage
4) Speed of search
5) Cost

Before making any comparisons it would be best to give a brief description of threaded lists and inverted files. An inverted file consists of two main parts, a vocabulary file and an occurrence file. As records are processed, each significant word is looked up in the vocabulary file. If the word has appeared before, it has associated with it a pointer to an area in the occurrence file; if not, then an area in the occurrence file is set aside for the word and a pointer to the first location in that area is entered in the vocabulary file. After this pointer is found, an entry is made in the first free location in the corresponding area of the occurrence file to indicate the record in which the word occurred.

(Figure 2)

The threaded list on the other hand, has the same type of vocabulary file, but the occurrence file is arranged in a different manner. The pointer in the vocabulary file now indicates a location associated with the first record containing the given word. This location in the occurrence file, in turn, contains a pointer to another location in the occurrence file associated with the second record which contains the word, and the pointer in this location points... Thus a linked list of all the occurrences of the word is formed.[2]

(Figure 3)

1) Ease of update

In the proposed system a record can be added or deleted very easily. To delete a record a search is performed which will retrieve the desired document. This produces not only the pointer to the record in the text file but the location of the record's code in the code file. The code word and pointer are removed from the code file, and their location is recorded as being free to be used for a new entry to the code file. The space that the text was occupying in the text file is now also free to contain new text. In order to add a record, which is the more common situation, the text of the new record is added to the text file in the first free location of a suitable size, or at the end. It is then processed in the same manner as all the other records have been. The generated code word and pointer is inserted in the first free space in the code list. Here no room is wasted since all of the code word and pointer combinations are of the same length. Thus any type of update in the code file will affect only the code for the record which is being changed.

The threaded list can be updated with slightly more effort. The problem, and a minor one, is that the records in the occurrence file are not all of the same length, making it necessary to see if there is enough room in a given free area to insert the new entry.

The inverted file on the other hand is far more difficult to update than either of the others. If a record is to be removed all that need be done is to delete all pointers to it in the occurrence file. The addition of a record however becomes a serious problem. If for every word in the record there is room for an additional pointer in the areas set aside for pointers to records containing that word, then the update is easy. But if there is no room, a secondary file must be set up. The number of such files will grow until it is felt that a thorough update should be made. Then the entire text file must be re-inverted to produce a new vocabulary and occurrence file. This is a very time-consuming and expensive project.

2) Effect of a large vocabulary

With the superimposed coding there is no problem associated with having an arbitrarily large vocabulary. This is true because the superimposed coding does not require a table of vocabulary words like the inverted and threaded list files do. Since the vocabulary file is not present and does not have to be searched, increasing the vocabulary neither lengthens the time required for a search nor

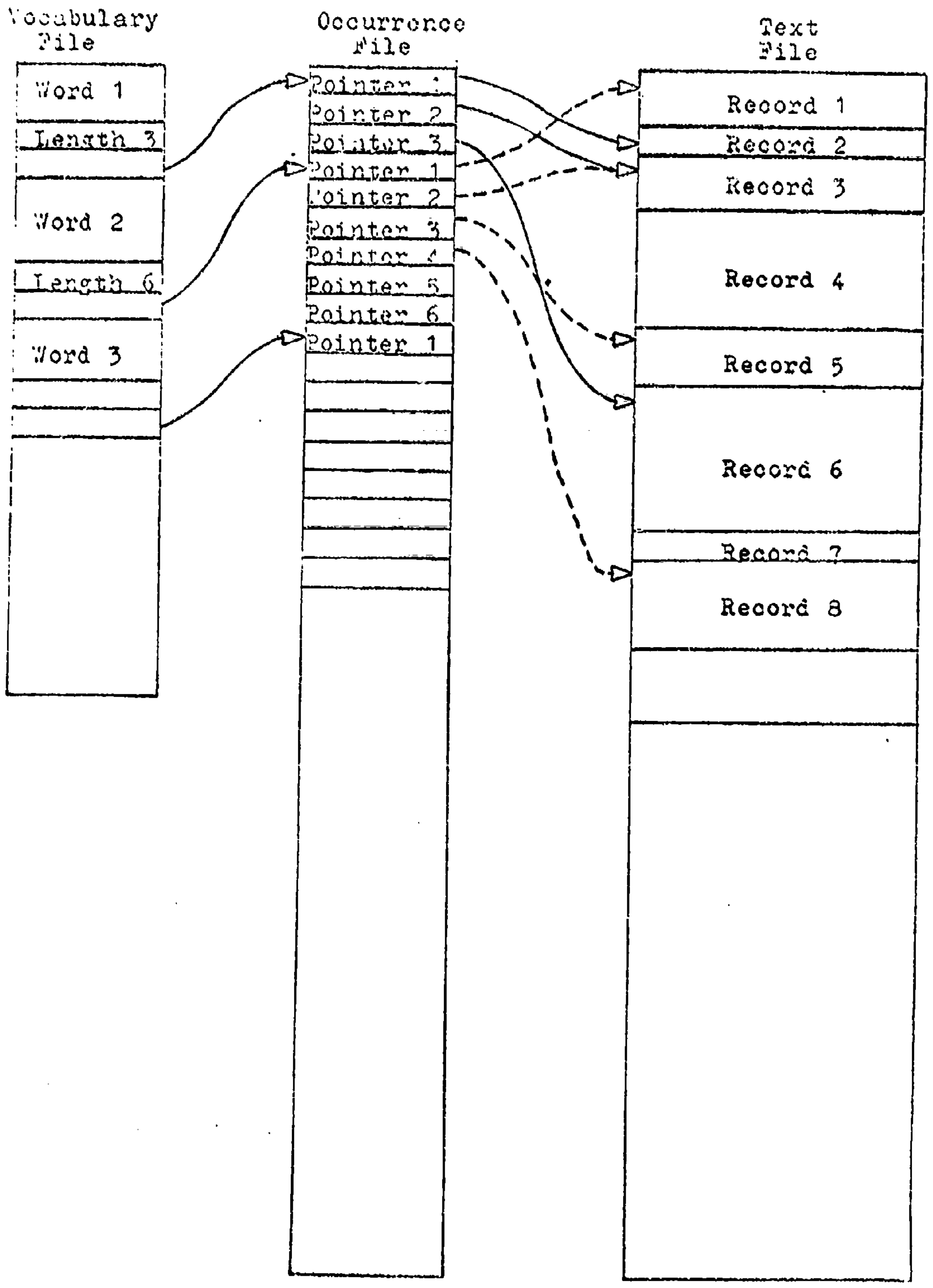


FIGURE 2 Inverted File

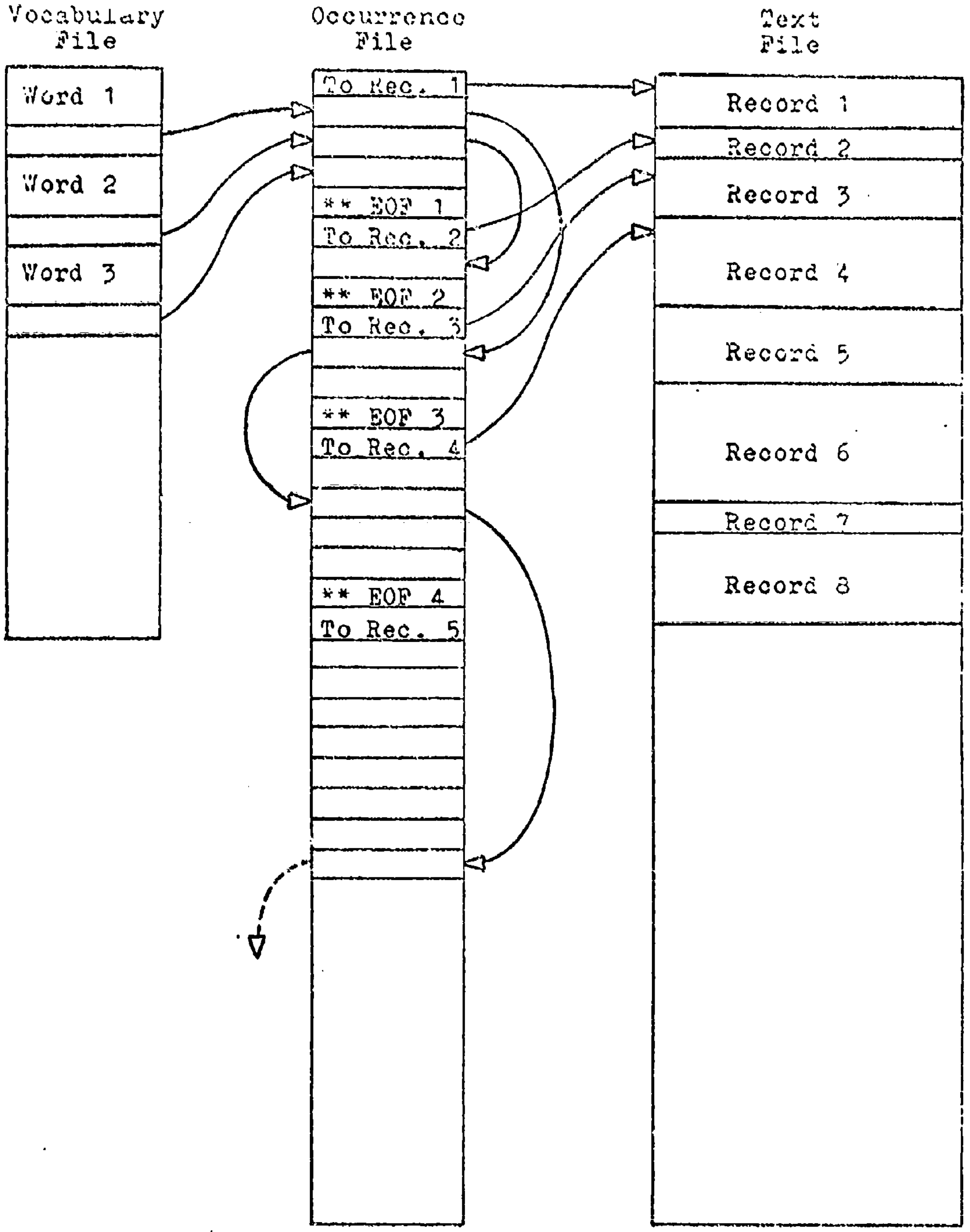


FIGURE 3 Threaded List

increases the amount of storage required to contain the coded information.

3) Storage requirements

The major advantage of superimposed coding lies in the great economy of storage. In the pilot program which was run, a text file of 100,000 bytes was used to produce a code file requiring 3,000 bytes. This reduction of 30 to 1 from the text to the code file is far better than the ratio obtained with the threaded list and inverted files. Such reductions are largest with small files such as the one experimented with, but substantial reductions do exist even in larger files. For example, assume that the text file consisted of 10,000,000 bibliographic entries, each containing 12 words which will be coded. Such an author-title entry was found to have roughly 300 characters in it, implying that the text file would be roughly 3×10^9 characters. Also assume that an average search contains at least three significant words. Such an assumption is made on the grounds that a search based on fewer words would tend to return more titles than would be of interest due to the very large size of the bibliography. From these two assumptions, utilizing considerations explained in Appendix B, it is found that the code file would consist of seven code words and one pointer for each record. Each of the code words is produced in a manner similar to the single code word mentioned before. Now, however, once the trimmed form of the word is found seven different procedures are applied to produce the pseudo-random number between 1 and N for each of the seven code words. Each of the code words will have 24 bits and the pointer will have 32 bits, thus indicating that each record will produce 25 bytes of code in the code file. The total size of the code file would then be 2.5×10^8 bytes, which still is a reduction of better than 10 to 1.

Such a reduction is far out of reach of an inverted file since each record in the text would have to have twelve 24 bit pointers pointing to it, and one 32 bit pointer from the record to the starting position of that record in the text file. This requires a total of 4×10^8 bytes and indicates only a portion of the room taken up by the inverted file. It does not include the vocabulary file which would be substantial, nor does it encompass the overhead of the occurrence file consisting of markers for the boundary between lists of pointers for a given word. Also it ignores the room which must be set aside for a linking pointer in case a new occurrence is to be added.

An additional advantage of the superimposed coding lies in the type of storage which can be used to store the code file. Since the file will be searched serially the storage media need not be random access. This permits the use of a cheaper sequential access storage device such as magnetic tape, which could greatly decrease the cost of such a system.

4) Speed of search

Evaluating the speed of a search using superimposed coding is difficult since the speed of any implemented system depends heavily on the characteristics of the storage media containing the code file as well as on the obvious consideration of the size of the text file. The search can be performed by reading the code file from bulk storage into addressable memory and comparison of the query codes with code words made by software. If this is done then the time required to search the code file can be cut to less than 6 x (the memory cycle time of the machine) x (the total number of code words in the code file.) This speed can be achieved due to the simplicity of the comparison which the software must make. The program only needs to test if X is a subset of Y by loading the accumulator with Y, doing a logical AND of the accumulator with a register which contains X, and testing to see if the accumulator equals X. When large text files are used, and there are several independently assigned code words for each record, time is saved by being able to reject a record when any one of the query codes fails to be a subset of the corresponding code word. By taking advantage of this a substantial amount of time can be saved. In the previously mentioned large file, with seven code words for each record and an average search of three words, more than 90% of the records would be rejected after only the first comparison was made. This means that there would be 36 memory cycle times (the time allotted for the 6 comparisons which did not have to be made) free to take care of the overhead in the searching program.

Even with this simple and fast searching procedure, a search does require longer than the threaded list or inverted file. Although the implementation of this technique in software is slower there are several methods that radically reduce the amount of time required to search the code file.

Since the algorithm for searching the code file is simple, the actual testing to see if X is a subset of Y can be done with very simple hardware. If the I'th bit of X is 1 and the I'th bit of Y is 0 for any of the values of I from 0 through 7, then X is not a subset of Y and the value of Z will be 1. If in no case is bit I of Y=0 and bit I of X=1, then X is a subset of Y and Z is 0.

(Figure 4)

Considering the speed of present day circuitry the time required to search a code file would be reduced to the time required to transfer the data from bulk storage. Since the hardware is so simple, it is practical to scan data from several sources simultaneously. An alternative to having the file searched externally would be to wire into read only memory the commands to test for a subset. By adding instructions to use the next code word and repeat the operation if the test fails, the search will proceed through core memory at a rapid rate making only one core access for each test. The end of the list of code words can be marked by a code word containing all ones. This has any possible query as a subset and would assure that the loop was interrupted at that point.

A second technique which would reduce the time required to search the file is to sort it in some manner. One such method which generates a superimposed 8 bit code from 24 bit code is discussed in Appendix A. Other methods such as carefully dividing code file into small groups and then doing a logical OR of the chosen code words to form rejector vectors have been suggested.[4]

Hardware to Test if
X Is a Subset of Y

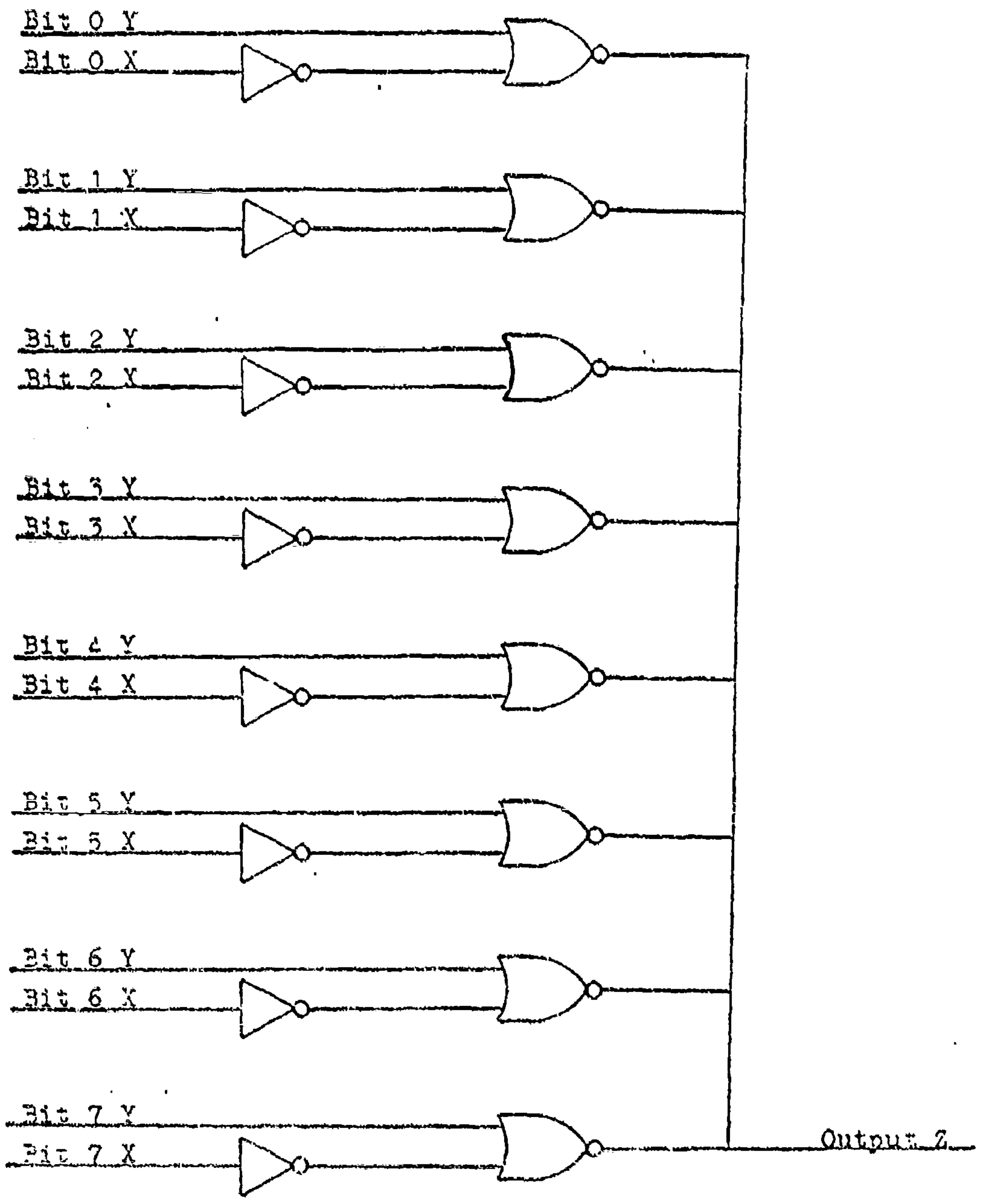


FIGURE 4

Hardware to Test if X is a Subset of Y

$$Z = (\bar{Y}_0 \wedge X_0) \vee (\bar{Y}_1 \wedge X_1) \vee \ldots \vee (\bar{Y}_7 \wedge X_7)$$

In comparing the speed of the search it should be noted that with superimposed coding and when searching for several words, the search for all of the words is carried out at once. In the threaded list and inverted file a search for several words is made by making a list of occurrences for each word and then finding the intersection of the lists. Due to this parallelism of the search superimposed coding can handle a multiple word search in a more efficient manner than the other two methods.

At first glance it appeared that searching the entire code file would preclude the use of superimposed coding on a large file. With more careful examination, however, it is apparent that this type of code file can be searched as rapidly as either the threaded list or the faster inverted file. Factors which lead to this conclusion include:

A) The code file search can easily be implemented in hardware. Such hardware is simple and very fast as well as being able to handle several streams of data simultaneously.

B) If several sequential access devices or a random access storage device is used then the code file may be structured to allow large blocks of the code file to be rejected with only one test.

C) The superimposed coded file is much more efficient at handling searches for records containing several desired keys.

5) Cost

The cost of implementing an information retrieval system utilizing the type of superimposed coding suggested would be substantially less than the cost of implementing a threaded list or inverted file using the same text file. The reasons for this stem from the reduced requirement for computational capability of the computer, as well as a substantial reduction in the amount of storage required for the coded information.

All three systems must dedicate a large amount of storage to the actual text. This, in all of the cases, can be either directly accessible to the computer such as a large disk file, or may be only machine referable such as a machine controllable microfilm display, like the proposed system at the University of California, Santa Cruz or the one being used as part of Project Intrex at M.I.T.[5] The difference of storage cost is not found in the storage of the text file but in the comparison of the cost of the storage of the code file of the superimposed coding system with the cost of storing the vocabulary and occurrence files of the threaded list and inverted file. The code file is smaller and can be stored in a sequential access device rather than a random access device. Both of these factors tend to reduce the cost of the system.

If scanning of the code file is implemented in hardware then the requirements on the computer become very small. All that it is responsible for is processing the words in the inquiry in order to generate the query codes, and then, while the search is in progress, stand by to store the pointers to the text file which the one or, possibly several, hardware scanners pass to it.

The trial program which processed the questions, generated the query codes and handled the searching in software, was substantially under 16,000 bytes of code on an IBM 1130 with no overlaying. Thus the requirement for expensive core storage is low. The cost of the hardware which would do the testing for the query code being a subset of the code word and its interfacing with the computer would be very small compared to the cost of the necessary storage devices.

One phenomenon which is found in the superimposed coding and not in some other forms of coding is the presence of spurious matches. These occur because, in a given code word the fact that the I'th bit is zero signifies that any word assigned the code value I is not in the record. The converse is not true. Since many vocabulary words would cause the I'th bit to be one, the I'th bit being equal to one, does not indicate that a specific word is present. By generating several independent code words for each record the number of times that superimposing will cause an irrelevant record to be retrieved can be made arbitrarily small. Take for example the case where twelve words were coded into seven 24 bit code words. In that case the probability that a record in which all seven of the query codes for a question were a subset of the code words, and none of the three words involved in the search were in the given record was 3×10^{-10}. (See formula in Appendix B, bd=.35, cw=2.8, qc=7)

Since the number of such spurious matches can be limited to any desired extent, although not entirely eliminated, it is convenient to perform some final verifying operation to assure that the words specified in the search are actually present. This verification in the case of the pilot program was accomplished as a side result of the check to see that the desired words occurred in the specified order. Consequently there was no penalty in making this extra check on the records which were retrieved.

The requirement that additional checking be done is not an unreasonable one. The fact that a document contains the words in which one is interested does not necessarily indicate that the document is of interest. Therefore any key word searching procedure can only be the first step of an information retrieval system. The job of a key word search is to quickly reject records that do not contain information of interest. In this sense any of the three types of key word information retrieval systems which have been mentioned are more properly information screening procedures which can rapidly eliminate a large portion of the text file as unlikely to contain relevant information. Such a system should be used to identify those records which warrant further and more extensive examination.

CONCLUSION

The method of superimposed coding which has been discussed is a simple and relatively inexpensive manner of scanning a large text file. With a simple check for spurious matches made after the search, such a system can stand alone as a key word information retrieval system. On the other hand since the actual scanning of the text can be easily and rapidly handled by peripheral hardware, the method is very attractive as a first stage screening method. Although the prospect of having to search the entire code file for every inquiry, at first glance, appears discouraging, the simplicity of the scanning algorithm and the ease with which searches can be carried out in parallel makes such a linear search very reasonable.

APPENDIX A

Besides implementation in hardware, measures can be taken to eliminate the need for searching the entire code file, thus reducing the required search time. One manner of doing this is to use the first code word of each record to generate a shortened code word for it. In the case of a 24 bit code word, the first three bits of the second level code word is the logical OR of the first three bits of the first level code word. Bits 4 through 6 could also be ORed and used as the second bit of the second level code word. Continuing this process an 8 bit second level code word is produced based on the bits 1 through 24 of the original code word. Since there are only 256 of these second level codes possible, with each record's first code word being mapped into one and only one of these classes, the file is partitioned into 256 sets characterized by the numbers 0 through 255. When it is time to search the code file, the element of the partition that the first query code belongs to is determined. If for example the query code is 000100000001000001000000 it would belong to set 84 (01010100). The only sets which would have to be searched would be those characterized by numbers which have 84 as a subset. (i.e., 11111111, 11111110, 11111100 would have to be searched, but 11111011 would not have to be examined further.) There would be only 32 out of the 256 sets which would have to be searched, thus the number of code words which would have to be compared with the query codes would be reduced. Using the scheme of coding 12 words into 24 bits would cause roughly 10% of the code file to be classified as 255 (11111111) and just over 3% to be classified by a number whose binary representation contains 7 ones and one zero. Due to the non-uniform distribution of the code words over the 256 sets, the reduction in the amount of the code file to be searched would not be the 7/8 suggested by the reduction in the number of sets which must be searched. The reduction would, however, be in the neighborhood of 30%. (3/8 of the sets whose binary representation has seven ones and one zero and 18/28 of those with six ones and two zeros can be eliminated.)

APPENDIX B

Since care was taken to assign the code values using numbers from a uniform distribution, the expected number of spurious matches can be predicted. By varying the length and number of the code words the frequency of spurious matches can be controlled. The number of spurious matches is a function of the bit density, bd (i.e., the number of ones in the code word divided by the number of bits in the code word); the number of code words per record, cw; the number of ones in the query code, qc; and the number of records which are coded into the code file N.

The expected number of spurious matches = $Nx(bd)^{cw \times qc}$

The number of bits used to code one record = cw x (the number of bits in the code word)

By keeping the number of bits used and the number of ones in a code word constant in the above two equations, it is found that the minimum number of spurious matches occurs when the number of bits in the code word is e times the number of ones in the code word. That is when the bit density is 1/e. The number of bits B to use for the code word when there are M words to be coded in each record is roughly 2.2M. This is found by considering that the probability that a given position will be left blank is $(1-1/B)^M$. The expected bit density would then be $1-(1-1/B)^M$. Setting this equal to the 1/e and solving for B yields the desired results.[3]

APPENDIX C

The trimming program was divided into three sections. The first step removes all 'e's, 'd's and 's's from the end of the word. These letters were removed since there are many words such as 'attractions' which have compound endings terminating in s, es, d, and ed. By removing these letters, in the above, the suffix, 'tion', is left on the end of the word where it can be easily identified and removed in a later section of the program. Once this operation is completed the endings 'er' then 'al' and then 'ly' are searched for and removed if found. This procedure removes endings such as the 'ally' on the end of 'functionally' and again is a technique to handle compound endings.

After the above two trimmings have been accomplished, the Trim List is consulted. Suffixes found in the Trim List are arranged in order by length, starting with the longest. The ending found in the list is compared letter by letter with corresponding letters on the end of the word remaining after the first two trimming stages have been completed. Since all of the 's's, 'e's and 'd's have been removed, the suffixes are in an unusual form. For example, 'ness' would have been trimmed to 'n' by the first stage of the trimming procedure. Also 'ance' appears as 'anc' in the Trim List.

The reason for having suffixes in this form can be seen by considering the problem of trimming the two words 'finance' and 'financed'. In the second case, when the 'ed' is found on the end of the word, it is difficult to decide if the 'ed' or just the 'd' should be removed. The decision was made to remove the 'ed'. This means that to trim 'financed', 'anc' must be in the Trim List. However, 'finance' which should be reduced to the same pseudo-root requires either the ending 'ance' to appear in the list or the 'e' removed before the ending is compared with endings in the Trim List. The second course of action was chosen because it reduces the length of the Trim List and makes the first step of the trimming operation very simple.

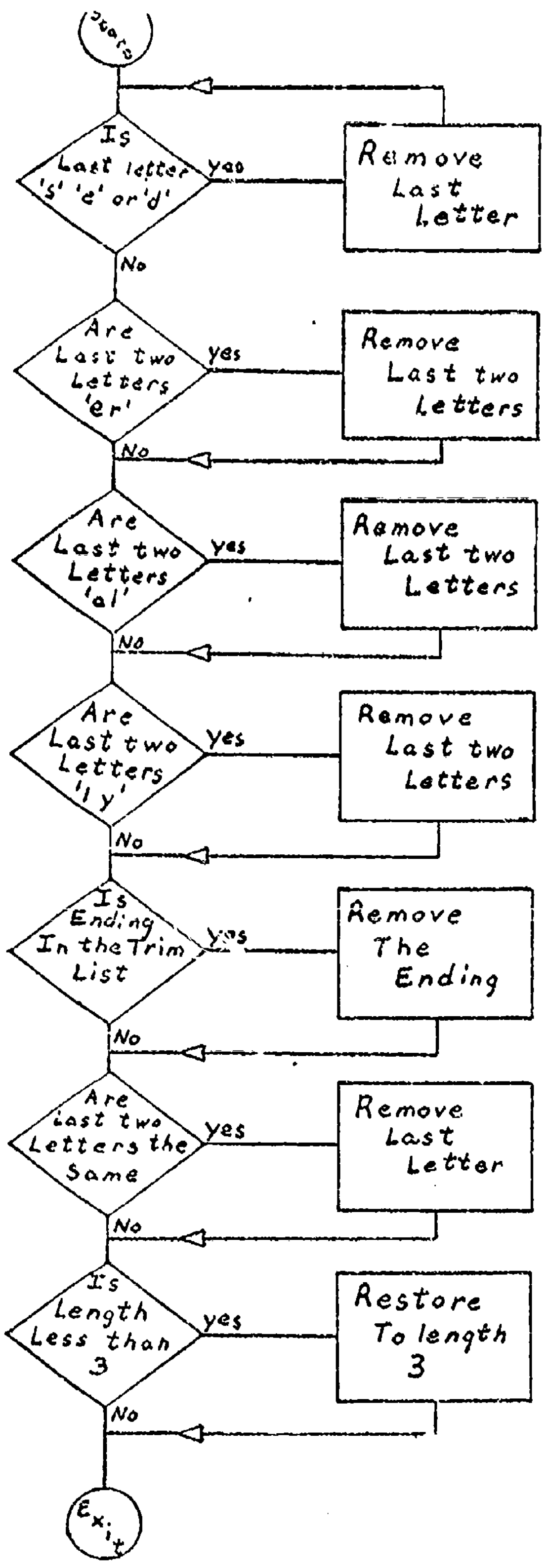
The comparison of the endings in the Trim List is continued until either the list is exhausted or a match is found and the ending removed. There are two more checks to be made on the trimmed word. First, the last two letters of the word are compared. If they are the same, then the last letter is removed. This is done so that a word such as 'trimming' will be cut back to 'trim'. First the 'ing' is removed to give 'trimm' and then the second 'm' removed to give the desired root.

The final action provides some protection against trimming words too severely. The word 'deeds' would be trimmed to nothing. To prevent such loss of information, any word which has been reduced to less than three letters is restored to a length of three. At this point the word is considered trimmed.

(Figure 5)

There is one major problem which occurs with the use of a trimming algorithm. Words which do not convey the same meaning can be reduced to the same root. An example would be that both 'information' and 'informal' are reduced to 'inform'. Such a result may be undesirable; it is unlikely that when searching for one of the words, the other would be of interest. Unfortunately the effect of this type of false retrieval could not be observed in the small pilot program. Such confusion of terms was rare due to the specialized nature of the text. In a system utilizing a larger text file containing a more generalized vocabulary, the number of such erroneous replies may become substantial. If a system utilizing a trimmed form of the vocabulary words is used for the first stage of an information retrieval system, the problem of such extra records is not a serious one, since the purpose of the search is to locate information-rich sections of the text. Further examination would determine whether the record is of interest or not.

The decision to utilize a trimming algorithm in the pilot program was based on the feeling that the error of failing to retrieve information was less tolerable than retrieving some irrelevant information.


Start
Is Last letter 's' 'e' or 'd'
yes
Remove Last Letter
No
Are Last two Letters 'er'
yes
Remove Last two Letters
No
Are Last two Letters 'al'
yes
Remove Last two Letters
No
Are Last two Letters 'ly'
yes
Remove Last two Letters
No
Is Ending In the Trim List
yes
Remove The Ending
No
Are last two Letters the Same
yes
Remove Last Letter
No
Is Length Less than 3
yes
Restore To length 3
No
Exit

TRIM LIST

ology	ful
ement	val
icant	ial
ition	cal
ation	ing
orial	enc
iting	anc
ating	iz
istic	ry
ancy	iv
ment	it
ient	at
ator	or
ical	er
ying	en
ary	al
eou	ag
est	id
ent	ic
ion	ab
ern	y
dom	n

APPENDIX D

DELETE LIST

a	had	what
i	his	will
am	how	with
an	may	being
as	nor	would
go	our	every
in	the	might
is	was	other
it	also	since
so	does	their
to	from	there
we	have	these
all	more	which
and	must	while
are	that	would
but	this	should
can	thus	another
for	ways	however
had	were	either
		without

BIBLIOGRAPHY

1. Casey, Robert S. et. al.: Carl S. Wise, "Mathematical Analysis of Coding Systems," Punched Cards, Their Applications to Science and Industry, Reinhold Publishing Company, New York, 1958.

2. Lowe, Thomas C.: Design Principles for an On-line Information Retrieval System, Doctoral dissertation submitted to the University of Pennsylvania, Philadelphia, 1966.

3. Mooers, Calvin N.: "Coding, Information Retrieval, and the Rapid Selector", American Documentation, 4:225, Oct. 1950.

4. Moore, Robert T.: "A Screening Method for Large Information Retrieval Systems", Joint Computer Conference, 19:259, 1961.

5. Overhage, Carl F. et. al.: Massachusetts Institute of Technology Project Intrex, Semi-annual Activity Report, 15 March 1968 to 15 September 1968, Cambridge Massachusetts, 1968.

6. Parker, Edwin B.: Spires (Stanford Public Information Retrieval Service), 1968 Annual Report to the National Science Foundation (Office of Science Information Service), Project GN 600, Project GN 742, January 1969.