INFORMATION AND CONTROL 3, 211–230 (1960)

# The Automatic Construction of a Glossary

ANDREW J. T. COLIN

*Department of Numerical Automation, Birkbeck College, London*

In this paper the term "glossary" refers to a list, in alphabetical order, of all the different words used in any text, together with a statement of how many times each word occurs in the text.

The manual construction of a glossary is usually made with the aid of a card index, where one card is used to note the occurrences of each different word in the text. The procedure consists of attempts to look up each consecutive word of the text in the card index; if the word is found, its present occurrence is noted on the card; but if it is not found, a new card is made out and inserted into the index.

The automatic construction of a glossary is similar in principle, but the logical rules by which the construction is made are more complicated because the store of an electronic computer, unlike a card index, is limited in size and, in general, is not large enough to contain the entire glossary of a text of average length. There are many ways of overcoming this difficulty and two of the more reasonable ones are discussed below.

## I. THE PARTIAL GLOSSARY METHOD

It is possible to overcome the problem of the limitation of storage capacity by dividing the different words in the text into a number of classes, or groups, each of which is known to be small enough to fit entirely into the computer store, and by making a separate partial glossary for each class of words. The partial glossaries can then be combined to produce the entire glossary for the text being analyzed. The construction of one partial glossary is called a "cycle," and the basic cycle pattern is shown in Fig. 1, which is self-explanatory.

## II. THE "OVERFLOW" METHOD

Another method of overcoming the storage space difficulty is to use the output medium of the computer as an auxiliary store. Thus, any information obtained from the text being analyzed for which there is no room in the main store of the computer is punched on an "overflow" tape. When the first "storeful" of information has been printed, the in-

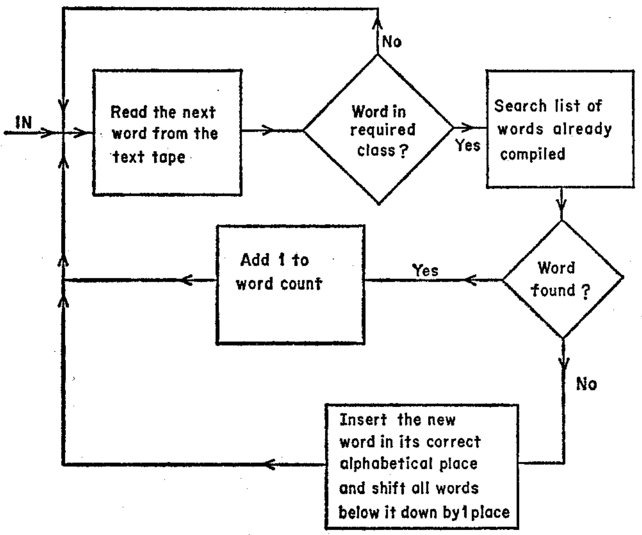
IN
Read the next word from the text tape
Word in required class ?
No
Yes
Search list of words already compiled
Word found ?
Yes
Add 1 to word count
No
Insert the new word in its correct alphabetical place and shift all words below it down by 1 place

FIG. 1

formation on the overflow tape is collected and sorted by treating it as a new "text." This process continues until no further overflow tape is produced. Each stage of the process is called a "cycle," and the outputs of the different cycles can be combined to produce the entire glossary for the text.

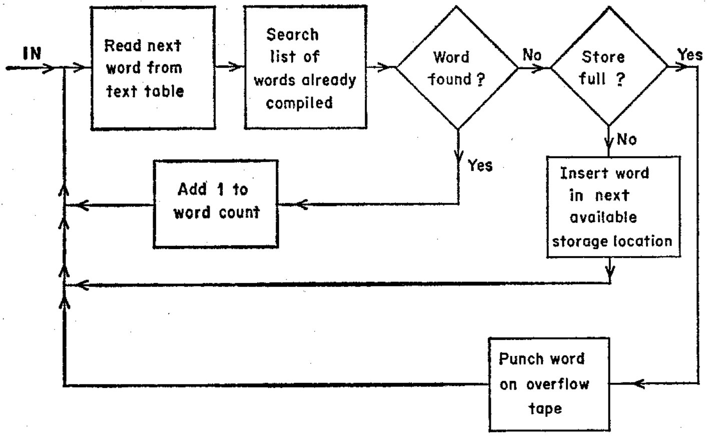
IN
Read next word from text table
Search list of words already compiled
Word found ?
No
Store full ?
Yes
No
Yes
Add 1 to word count
Insert word in next available storage location
Punch word on overflow tape

FIG. 2

Three variants of the overflow method (called methods 2a, 2b, and 2c) are shown diagrammatically in Figs. 2, 3, and 4.

Method 2a collects the different words in the text in the order in which they occur, until the store is full; whereas method 2b collects new words

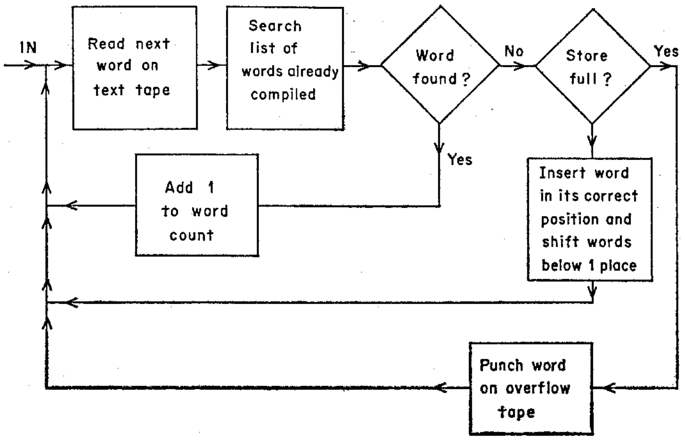
IN
Read next word on text tape
Search list of words already compiled
Word found ?
No
Store full ?
Yes
Yes
Add 1 to word count
Insert word in its correct position and shift words below 1 place
Punch word on overflow tape

FIG. 3

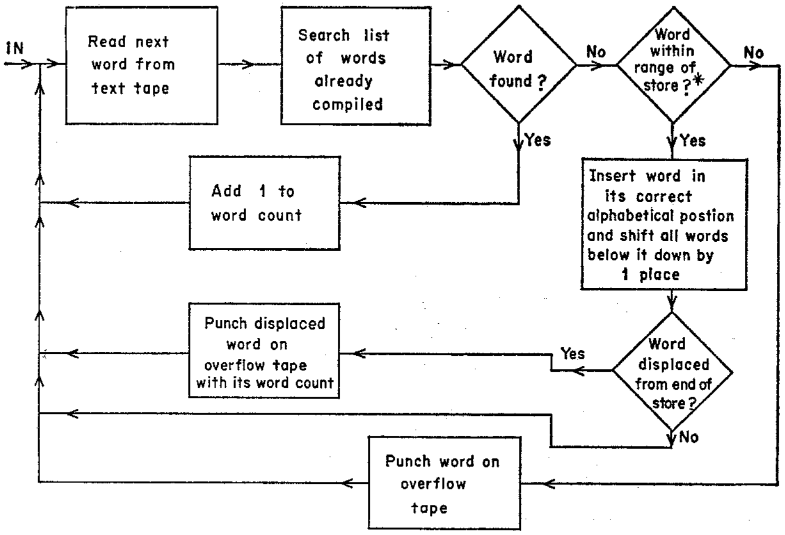
IN
Read next word from text tape
Search list of words already compiled
Word found ?
No
Word within range of store ?*
No
Yes
Yes
Add 1 to word count
Insert word in its correct alphabetical postion and shift all words below it down by 1 place
Punch displaced word on overflow tape with its word count
Yes
Word displaced from end of store ?
No
Punch word on overflow tape

FIG. 4

* i.e., Does the store have any blank locations left? If not, is the alphabetical position of the text word before that of the last word in the store?

and stores them in their correct alphabetical order, until the store is full. Both methods 2a and 2b ignore all new words which occur in the text after the store is full, so that their results are not in true alphabetical order; if method 2b is used, although the words collected in each cycle are in alphabetical order, the alphabetical position of the words obtained by the different cycles overlap, so that to obtain a true alphabetical order for the whole glossary, the partial glossaries have to be meshed.

However, method 2c ignores no new word in the text whose correct alphabetical position is between two of the words stored, so that, although the insertion of a new word into the store when it is already full will cause the last word in the store and its word count to be displaced from it and therefore necessitate its punching out on the overflow tape together with its word count, the final results produced by method 2c will be in true alphabetical order. For this reason, method 2c is the only variant of the overflow method which is considered any further.

Since methods 1 and 2c will eventually produce identical results, the preferred method will be the one which takes the least time to do so. If certain crude assumptions are made, it is possible to derive approximate analytical expressions for the times taken by the two methods.

In the Mercury, the times required for dictionary searching and organization of the process are very small compared with the times needed for the operations defined above. Theoretically an allowance should be

TABLE I

DEFINITIONS OF SYMBOLS USED IN THE ANALYSIS

| Symbol | Meaning | Value in Mercury |
|---|---|---|
| $T$ | Total number of words in text | |
| $w$ | Number of different words in text | |
| $z$ | Zipf's constant for the text[a] | |
| $i$ | Average time required to read in one word of the text | 40 msec |
| $s$ | Average time required to shift a word in the store down by one place | 1.5 msec |
| $p$ | Average time required to punch one word | 300 msec |
| $n$ | Maximum capacity of store (words) | 8160 |

[a] Defined as value of (rank) × (frequency), see "Mechanical Resolution of Linguistic Problems," by Booth, A. D., Brandwood, L., and Cleave, J. P. Butterworths, 1958.

made in the analysis for the time for which the machine is idle while new tapes are being inserted into the tape reader and similar operations are being carried out; but in practice it is found that the rewinding of overflow tapes, and selection of new text tapes can be done by the operator while the machine is running, so that the actual time taken to change tapes is very small: it is never more than 10 sec, and with some dexterity on the part of the operator, it can be as low as 3 sec. Any glossary program is unlikely to require the changing of more than, say, 100 tapes. Allowing 6 sec for each change, the total idle time will be about 10 min, which is very small compared with the total running time of the program.

## Method 1

Let $q$ be the average number of different words in each class. The number of cycles is $w/q$; the input time for each cycle is $iT$ msec; the average number of words which need to be shifted for the $j$th new word to be fitted into the store is $j/2$. Therefore, the total number of shifts required per cycle is

$$\sum_{j=1}^{q} \frac{j}{2} \simeq \frac{q^2}{4}.$$

Therefore the total shifting time per cycle $\simeq sq^2/4$ msec. The punching time for output is $pq$ msec. Therefore the approximate total time $t$ is given by

$$t = \frac{w}{q}\left(iT + \frac{sq^2}{4} + pq\right). \qquad (1)$$

The only parameter in Eq. (1) which may be adjusted by the programmer is $q$, the number of words in each class. To find the value of $q$ which leads to the shortest time for the program, put

$$\frac{dt}{dq} = w\left(\frac{s}{4} - \frac{iT}{q^2}\right) = 0.$$

Therefore,

$$q^2 = \left(\frac{4iT}{s}\right). \qquad (2)$$

This gives rise to the unexpected result that a value of $q$ which is as near as possible to $n$ does not necessarily give the fastest results.

## Method 2

Consider the first cycle of a program using this method. The total reading-in time is $iT$ msec.

The rest of the analysis must be considered in two parts, the first part corresponding to the state of the program when the store is not yet full, and the second to the state when the store is full.

*Part 1* (*store not yet full*)

The total shifting time is $n^2/4$ (by analogy with the analysis of method 1). The overflow punching time is nil. Assume that Zipf's law holds good for the text. Then, the probability that any word in the text is the word of rank $a$ is $z/Ta$. Therefore, the probability of $k$ consecutive words not being the word of rank $a$ is

$$\left(1 - \frac{z}{Ta}\right)^k \simeq \left(1 - \frac{kz}{Ta}\right)$$

when $z/Ta$ is small. Therefore the expected position in the text of the word of rank $a$ can be evaluated if we put

$$\left(1 - \frac{kz}{Ta}\right) = \frac{1}{2}.$$

This gives

$$k = (Ta/2z).$$

Very approximately, the new words will be collected in the order of their ranks, so that the last word to be fitted into the store without displacing another word will be approximately of rank $n$. Therefore the total number of words processed during the first state is $Tn/2z$. Therefore, the total number of words not yet processed is

$$T\left(1 - \frac{n}{2z}\right).$$

The number of new words not yet processed is $(w - n)$. Therefore, the number of "not new" words still left is

$$T\left(1 - \frac{n}{2z}\right) - (w - n).$$

*Part 2* (*store full*)

When the second state begins, the words stored range uniformly through the alphabet, so the probability of any word in the text being within the range of the store is 1. But, at the end of the second state, the store contains the first $n$ different words in the text, so the probability of any words in the text being in the range of the store is $n/w$. Therefore, the average probability for this during the second state is

$$\left(1 + \frac{n}{w}\right)\Big/2.$$

Therefore, the average probability for a word *not* being in the range of the store is

$$\left(1 - \frac{n}{w}\right)\Big/2.$$

Consider first the $(x - n)$ new words left on the tape. The number of new words which will be in the range of the store is

$$\left(1 + \frac{n}{w}\right)(w - n)/2.$$

The average position of these words in the store is $\frac{1}{2}n$th location. Therefore, the amount of shifting required is

$$\frac{n}{4}\left(1 + \frac{n}{w}\right)(w - n) \text{ shifts,}$$

and the time required is

$$\frac{sn}{4}\left(1 + \frac{n}{w}\right)(w - n) \text{ msec.}$$

Each new word encountered in the second state will either be punched itself, or displace another word, so that the total punching time required is $p(x - n)$ msec.

Now, the number of "not new" words which will not be within the range of the store is

$$\frac{1}{2}\left(1 - \frac{n}{w}\right)\left[T\left(1 - \frac{n}{2z}\right) - (w - n)\right]$$

and the punching time required for these is

$$\frac{p}{2}\left(1 - \frac{n}{w}\right)\left[T\left(1 - \frac{n}{2z}\right) - (w - n)\right].$$

Finally, the time required for punching out the $n$ new words found is $pn$ msec. The total time $t$ required for the first cycle is, therefore,

$$t = iT + \frac{sn^2}{4} + \frac{sn}{4}\left(1 + \frac{n}{w}\right)(w - n)$$
$$+ px + \frac{p}{2}\left(1 - \frac{n}{w}\right)\left[T\left(1 - \frac{n}{2z}\right) - \left(w - n\right)\right].$$

As Zipf's law no longer holds for the words now left on the overflow tape, the calculation of the time required for subsequent cycles is almost impossible to determine analytically, but a reasonable assumption is that each cycle will take roughly half the time of the previous one, so that, extremely approximately, the total time for the entire program $t_2$ is $t_2 = 2t$.

Table II gives a numerical comparison of the two methods discussed, using Joyce's "Ulysses" as a sample text. For this text $T = 260{,}000$, $w = 30{,}000$ and $z = 26{,}000$.

Although the assumptions made in the foregoing analysis are so crude that the resulting figures for the necessary time can only be assumed to be correct to within an order of magnitude, one conclusion emerges clearly, namely, that although method 2c, if used exactly in the form described above, is inferior to method 1, it would be very much superior if it were possible to eliminate the shifting time. The shifting time in both methods is roughly proportional to $n^2$; so that if we can artificially reduce the apparent size of the store by a factor $k$, without actually changing the number of words stored in it, we shall reduce the shifting time by a factor of $k^2$.

This artificial reduction can be made by subdividing the store of the computer into a large number of small self-contained stores, each of which is used for one small class of words. If the rms size of the small stores is $p$, then $k = n/p$.

If we take a typical value of $p = 48$, we find that the shifting time

TABLE II

| Method | $q$ optimum | Input time (hr) | Shifting time (hr) | Punching time (hr) | Total (hr) |
|---|---|---|---|---|---|
| 1 | 5247 | 17 | 17 | 3 | 37 |
| 2c | — | 5 | 61 | 17 | 84 |

required in method 2b falls from 61 hr to practically zero, making a great reduction in the total time required.

This method of reducing shifting time could, in principle, be applied to method 1 as well, but this would not be practicable as the programmer would have not only to divide the words in the text to be analyzed into classes, but he would have to divide each class into many subclasses, with the certainty that the size of each of the subclasses would not exceed the space allotted to it.

From this discussion it is evident that one of the best methods of constructing a glossary is method 2c, with the modification described. The next section of this paper discusses the details of a program using this method.

## III. THE PROGRAM

This program was written to make a glossary of King Alfred's translation of "Orosius," a work in Anglo-Saxon about 45,000 words long, but it could easily be adapted for use with any text in Roman characters.

The text is typed on a teleprinter using the method of coding described in the companion paper, the line and page arrangements being kept the same as they are in the source text. As the text is, in general, too long for one physical tape, it is punched on several tapes, each of which ends with the symbol /. As the standard word length in the machine is fixed at 16 letters, no word in the teleprinted text must exceed this length. Since longer words are very rare, this requirement is not objectionable.

It may be that certain groups of words are not required in the glossary, although they may need special treatment later in the linguistic analysis of the text; all the words in these groups are prefixed by certain letters (or other symbols) which never occur initially in the rest of the text. The groups of words so labeled in the present analysis are shown in Table III.

All the words required in the glossary are divided into classes, where

TABLE III

| | |
|---|---|
| Preposition | *J* |
| Adverbs | *K* |
| Proper nouns | *Q* |
| Conjunctions | *X* |
| Demonstrative pronouns | *V* |
| Personal pronouns | *Z* |

each class consists of all the words which have a certain initial letter pair. Each class is stored in a subdivision of the store called a "leaf." Each leaf is conveniently made into a whole number of sectors, and is "labeled" by the particular letter pair of its class.

It is clearly uneconomical to allot the same number of sectors to each leaf, as words in certain classes occur much more frequently than words in other classes.

To decide on the number of sectors to be allotted to each leaf, an estimate is made of the relative number of different words in each class, putting 1 for small classes. These numbers are then adjusted so that their sum is equal to the total number of sectors available. Each one will then indicate the number of sectors required for the leaf.

The letter pair labels of the various leaves, together with the proposed leaf sizes, are punched on a "leaf directory" tape. Part of a leaf directory print-out is shown in Fig. 5. During the execution of the program, the address and size of each leaf is stored in a "leaf directive."

Before the main program is entered, the leaf directives are constructed from the leaf directory tape; each one consists of two integers, which indicate the location of the first sector of the leaf and the number of sectors in the leaf. Each leaf directive is kept in a location in the second half of the computing store, whose address is numerically equal to a simple transformation of the letter pair label of the leaf to which the directive refers. All the locations which do not contain leaf directives are filled with the code 0-0.

When the leaf directives have been assembled, one of the text tapes is placed in the reader, and the main part of the program started. A flow diagram of this part of the program is shown in Fig. 2.

```
AB1.AC2.AD2.AE6.AF2.AG2.AH1.AI4.AJ1.AL2.AM5...
BA2.BE5.BI1.BO1.BU2.BY1.
CA1.CE1.CH4.CI2.CO1.CU2.CY1.
DA1.CE1.DH2.DI1.DO1...
...
...
...
.
.
.
.
ZI2.ZO1.ZU1.ZY1./
```

FIG. 5

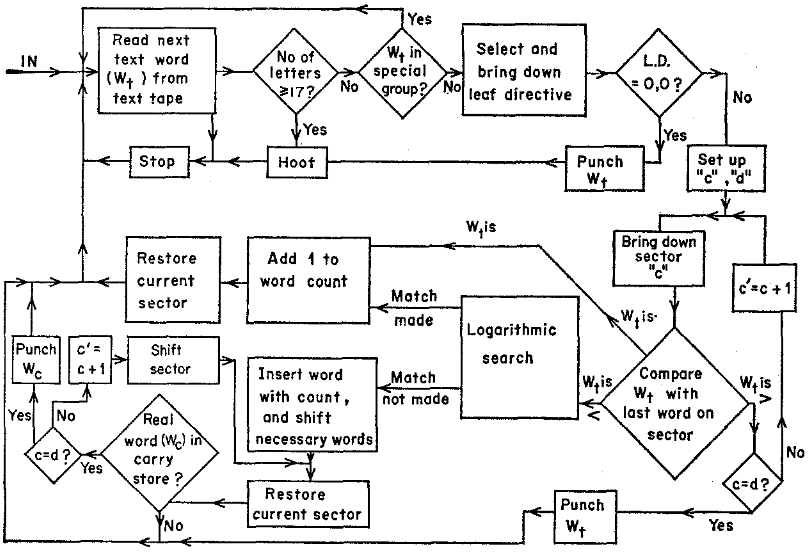

FIG. 6

As mentioned previously, the standard word length in the machine is 16 letters, or 8 letter pairs; but since each leaf is used to store words starting with a given letter pair, this letter pair can be deduced from the leaf and need not be stored in the leaf. The resulting spare location is used to count the number of times the word has occurred in the text (the so-called "word count").

A detailed flow diagram of the main part of the program is shown in Fig. 6.

The main program begins with a routine which reads in the next word from the text tape (the "text word") and assembles it, in machine form, in a convenient location. The routine ignores all symbols which are not parts of words (e.g. ? or CR LF) except for two; these are:

1. The symbol → following a word. If this symbol is encountered, the machine reads and stores the number immediately following it. (The reason for this will be evident later.)

2. The symbol /. This symbol signifies "end of text tape," and makes the machine stop, so that either the next text tape may be placed in the reader or the output routine (q.v.) started.

As any text word is being read in, the letters are counted; should they exceed the permissible total of 16, the machine is made to hoot and stop, so that the offending word may be noted, and the text tape moved on to the next word.

When the reading operation has finished, the initial letter of the text word is examined; should it prove to be one labeling a special group, the word is ignored, and the next word read in.

Otherwise, the word is one required in the glossary, and the search in the dictionary of words which have already been compiled is begun. The search takes place in three stages, which are respectively direct, linear, and logarithmic.

The initial stage consists of finding the leaf to which the text word belongs. This is done by suitably transforming the initial letter pair of the text word, and using the result to B-modify an order which brings down the directives from the second half of the computing store. If the leaf directive brought down is 0-0, there is no leaf for the word; in this case the word is unacceptable to the machine, and is punched on the overflow tape.

Otherwise, the leaf directive is used to form two numbers, namely $c$, the current sector, which is initially the number of the first sector of the leaf, and $d$, which is the last sector of the leaf.

Then, the second or linear part of the search is concerned with finding which sector of the leaf the text word should be on. The current sector is brought into the computing store, and the text word is matched against the last word on this sector. This match may indicate one of three things:

1. The text word is greater than the last word on the sector.
2. The text word is less than the last word on the sector.
3. The text word is exactly equal to the last word on the sector.

Case 1 implies that the search for the correct position of the text word must take place further on in the leaf, so if the current sector is not the last one in the leaf, the sector number is increased by one and the matching process repeated. If, however, the current sector is the last one, this fact implies that the leaf is full and that the text word is outside its range. In this case, the text word is punched on the overflow tape.

Cases 2 and 3 indicate that the correct position of the text word is on the current sector. Case 2 leads to the third part of the dictionary search, which is a logarithmic one within the current sector. The routine which carries out the search indicates the correct position of the text word in the sector and also whether the text word has been exactly matched with a word on the sector or not.

If an exact match has been made, either by the logarithmic search routine or by the previous stage of the search (case 3), 1 is usually added to the word count, and the current sector restored to the drum. However, it is necessary to insert a safety device here, for counting in the word count registers is carried out modulo 1024, and should any word occur more than this number of times in the text it would give rise to an incorrect count. Therefore, before 1 is added, the word count is examined and if it is 1023, 1 is not added, but the word is punched on the overflow tape.

If, however, an exact match is not made the word is inserted in its correct position in the sector, and all the words below it moved down by one place. The word count associated with the word is set to the number which followed the word on the text tape; or, if there was no number, the count is set to 1. The "word" which is displaced from the end of the sector is placed in a "carry store." The current sector is restored to the drum and the "word" in the carry store examined. If this "word" consists entirely of ones, it signifies a "blank location," and the shifting process may stop, but if there is a real word in the carry store, and the current sector is not the last of the leaf, shifting must continue. Therefore the next sector of the leaf is brought down from the drum, and the shifting and examination process repeated. However, if the sector which has been shifted by one place is the last one in the leaf, the word in the carry store, if it is a real word, is one for which there is no room in the leaf, so this word, followed by the sign $\rightarrow$ and its present word count, is punched on the overflow tape. It is necessary to punch the word count because the word may have occurred several times in the text while it was still in the store, and this is the only way not to lose this information. The word count will then be read in by the input routine when the overflow tape is processed.

When the entire text has been processed the output routine is entered. This routine is trivial in comparison to the main part of the program and merely punches out all the words in the store, together with their word counts.

When the output routine has finished, the entire program is repeated using the overflow tape as a new text; and the cycles are repeated until the overflow tape contains only words unacceptable to the machine.

Table IV shows the times taken for each stage of the process, when the glossary for King Alfred's "Orosius" was made.

The large number of overflow tapes produced were due almost entirely to a gross underestimation of the number of words starting with GE-.

TABLE IV

| Stage of process | Time (min) |
|---|---|
| Initial input of text | 90 |
| First output | 40 |
| Input of first overflow tape | 30 |
| Second output | 10 |
| Input of second overflow tape | 3 |
| Third output | 1 |
| Input of third overflow tape | 1 |
| Fourth output | 1 |
| Total = 2 hr 56 min | |

The number of sectors allowed for this leaf was 15, whereas there are enough different words in this class to fill about 50 sectors.

In the entire text there were about ten words unacceptable to the machine. Some were unacceptable because a leaf had not been provided for them (these were mainly Latin words), and some because they had been wrongly or badly punched on the text tape.

In addition to the computer time taken to make the glossary, about three weeks were spent in teleprinting the text, and roughly twenty hours writing and testing the program.

One year was the time estimated to be necessary for the manual construction of the glossary.

# PART II

## The Automatic Construction of a Concordance

A concordance is a list, in alphabetical order, of all the different words in a text; each word being followed by a number of references, consisting of page and line numbers.

In the automatic construction of a concordance, the difficulties associated with storage are considerably greater than in the construction of a glossary. The two main problems are:

1. Since a concordance contains a great deal more information than a glossary of the same text, the store of the computer can, in general, only contain a small part of all the information required at any time; and

consequently, even if the full capacity of the store is used, a large number of "cycles" are required to obtain the complete concordance.

2. Efficient use of the store is difficult to achieve because the amount of storage space required for the references to each word in the text is initially unknown.

There is clearly no way in which the first difficulty may be overcome without sacrificing some information, but if a concordance were to be made directly from a text about which nothing were initially known, there would be two general methods by which the second problem might be overcome.

1. The store could be divided into a number of "blocks" and each different word and its references stored in a different one of them. The blocks would all have to be of equal size, because the word which each block would eventually contain would not be initially known; the blocks could either each be large enough to contain all possible references to any word in the text, or else they could be made smaller and used in conjunction with an "overflow" scheme.

2. Alternatively, the information from the text could be stored compactly by inserting each item in its correct place, and shifting all the items below it down to make room for it. This could be done by a method similar to one of those described for the construction of a glossary.

Both these methods have very serious drawbacks. Method 1 would make very inefficient use of the store, and would therefore increase further the already large number of cycles required to complete the concordance.

Method 2, on the other hand, although it would make efficient use of the store, would consume much time merely shifting information from one place in the store to another, since every reference to every word would require a shifting process.

There are many possible schemes based on these methods or combinations of them, a few of which lead to some increase in the speed of the program, but no matter what method were adopted, the construction of a concordance from a text about which nothing is initially known would be an extrenely time-consuming task.

However, if a glossary of the text to be analyzed is available, a program can be designed which overcomes the drawbacks of both the methods mentioned above, but retains their advantages. Using the information contained in the glossary, the store can be split up into blocks, each of which is used to store the references to a known word. Since a word count is available, the blocks can be made of exactly the right capacity, so that the store can be used at its full efficiency. Furthermore, each reference obtained from the text can be placed directly in the location prepared for it, so that shifting is completely obviated.

A program designed on these lines is described below.

## THE CONCORDANCE PROGRAM

Each cycle of the concordance program has three phases. The first phase reads in the glossary of the text to be analyzed, and divides the store into blocks accordingly. Each block contains three items; firstly, a "label," which consists of the word whose references are being collected in the block; this word is not stored in its standard machine form of 8 letter pairs, but, to save storage space, in its most compact form, i.e., $\left[\frac{l+1}{2}\right]$ letter pairs, where $l$ is the number of letters in the word. The second item consists of sufficient 10-bit locations to store all the expected references to the word, and the third item is a marker to indicate that a new block has started. The marker is placed just before the labeling word: it is necessary because the blocks are of unequal length, and it consists of one 10-bit machine word of which the five most significant bits are set to "11111," and the five least significant bits indicate the number of letters in the labeling word. The code "11111" is chosen for the five most significant bits of the marker so that it will not be confused with an alphabetical letter pair. If this form of marker is used, page or line references higher than 991 (11110,11111 in binary) cannot be stored because they would be confused with the marker.

For example, suppose that the word GLASS occurs three times in the text being analyzed. The block for this word will be set up as shown below (in binary notation):

| | | |
|---|---|---|
| 11111,00101 | Marker | |
| 00111,01100 | GL | Labeling word |
| 00001,10011 | AS | |
| 10011,00000 | S- | |
| 00000,00000 | | |
| 00000,00000 | | |
| 00000,00000 | | Space for expected references |
| 00000,00000 | | |
| 00000,00000 | | |
| 00000,00000 | | |

For convenience of access the blocks are arranged into groups of two sectors each. Each group is called a "sector pair" and no block is allowed to overflow from the end of one sector pair to the next.

The program reads in the glossary tape, constructs the blocks, and assembles them into their sector pairs, starting with the first available sector pair on the drum. Any block which will not fit into a sector pair is made the beginning of the next sector pair, and the spare locations on the previous sector pair are filled with the code 11111, 11111. If any word has so many expected references that it requires more than 1 sector pair to store them, it is split up into a number of different blocks, each of which is up to one sector pair long. Words in this category are called "multiblock" words.

The first word on each sector pair is stored in standard machine form in the second half of the computing store, in an address which corresponds to that of the sector pair. If the first block of any sector pair is not the first block for a particular word, the word which is recorded in the computing store is a fictitious word generated by increasing the real word by the smallest possible amount. For example, if the word "THE" requires many blocks of storage, the words stored would be:

| | |
|---|---|
| First word stored: | TH E- -- -- -- -- -- -- |
| Subsequent words stored: | TH E- -- -- -- -- -- -A |

However, the labeling words stored in the blocks themselves would be all the same. The reason for this arrangement will become apparent later.

The process of setting up the blocks continues until either the store is full, or the glossary tape comes to an end. The first and last words in the store are placed in an easily accessible place. If the program stops because the store is full, the glossary tape is marked at the place it stopped, so that it may be inserted at the same place when the next cycle is begun.

## PHASE 2

This part of the program collects the references from the next tape, and places them in the predetermined locations in the store. A flow diagram of this is shown in Fig. 7.

Phase 2 begins with a routine which reads in the next word of the text counting the number of letter pairs which it contains, and also keeps a record of the current page and line references. The current line refer-

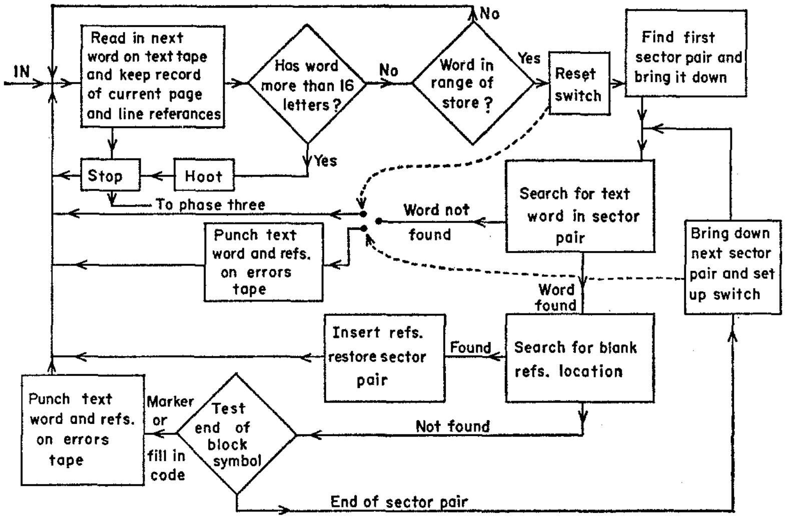
IN
Read in next word on text tape and keep record of current page and line referances
Has word more than 16 letters?
No
Yes
Word in range of store ?
No
Yes
Reset switch
Find first sector pair and bring it down
Stop
Hoot
To phase three
Word not found
Search for text word in sector pair
Punch text word and refs. on errors tape
Word found
Bring down next sector pair and set up switch
Insert refs. restore sector pair
Found
Search for blank refs. location
Punch text word and refs. on errors tape
Marker or fill in code
Test end of block symbol
Not found
End of sector pair

FIG. 7

ence is increased by 1 wherever the "line feed" symbol is encountered on the text tape, and the page reference is set equal to the last page number read from the tape. A new page number automatically resets the current line reference to 1. The routine also contains a device to stop the machine if the "end of tape" symbol is read, and a safety measure which makes the machine hoot and stop if a word which contains more than 16 letters is encountered.

When a word has been read in, it is compared with the first and last words in the store to determine whether it is within the storage range for the particular cycle of the program. If it is not, it is ignored and the next word read in. Otherwise, the search for the block to which the word belongs is started. The search takes place in two stages.

The first stage is a logarithmic one carried out among the words in the second half of the computing store, and its results indicate either the sector pair on which a "single block" word is stored, or the sector pair on which the first block of a "multiblock" is stored. The reason for the storage of the fictitious words mentioned previously is that it is essential to pick out the first block of a multiblock word. If a logarithmic search were applied to a dictionary where the required word occurs several times, the actual occurrence of the word which it would select would not necessarily be the first one, but would depend on the precise position in the store of the group of words. This difficulty can be overcome by slightly increasing all the words except the first as indicated.

When the correct sector pair has been selected, it is brought down into the computing store, and the second stage of the search is entered.

As the blocks in the sector pair are of unequal length, the search for the correct one must be a linear one, but it is speeded up by only comparing the text word with those labeling words which have the same number of letters as the text word. These words are found by means of their markers.

When the correct block is found, the current page and line references are recorded in the first available blank locations and the current sector pair restored. It may be, however, that there are no blank locations left in the block. If the block ends with the marker for the next block or with the "fill-in" code 11111,11111, this signifies that an error has occurred and that the block is the wrong size. In this case the text word and its current references are punched on an "errors" tape. If, however, the block ends at the end of a sector pair, this usually signifies that a word is a "multiblock" one. In this case the next sector pair is transferred to the computing store, a "switch" is set up, and the search begins again.

It may be that the programmer has erased certain very common words from the glossary, because their concordances were not required. When these words are encountered in the text, they will not be found in the store, although they may be in the storage range.

If a word is not found in the first sector pair within which the search is made, it is one of the erased words and is simply ignored. If, however, a word is not found on the second sector pair which is searched, this implies that an error was made in setting up the block for the word, and that this error happened to make the block end at the end of the sector pair. Therefore, when the block to which the word belongs was found to be full, the program mistakenly assumed that the word was a multiblock word and the next sector pair was brought down and searched. To allow for this very unlikely occurrence, the switch which was set up when the second sector pair was brought down causes the program to punch the word on the error tape.

Phase 2 of each cycle continues until the entire text has been processed.

Phase 3 prints out the partial concordance made during phases 1 and 2. Each word is followed by all the page and line references which have been collected, and, if a blank location is found, the symbol "?" is printed.

If the glossary program and the concordance program which form the subjects for this paper are used together, they provide an incomplete, but nevertheless useful error detecting and correcting device. By means of the "?" 's on the output and the error tape all errors except (a) incorrect punching of text, (b) complete omission of a word in the glossary, and (c) incorrect recording of references, may be detected and some of them corrected. If the program should produce an objectionably large number of detected but uncorrected errors, it is possible to run a further cycle of the program using as a glossary those words about which there is some doubt and giving a very large block size to each one.

## CONCLUSION

In general, full concordances of texts are of more use for linguistic analysis than glossaries. Nevertheless, it is the author's opinion that a glossary of a text is well worth making, even if it is only used to make an efficient concordance program. It is thought that the method described is probably one of the most efficient ones available if a machine similar to Mercury is to be used; but this would not be so if a machine specially designed for linguistic analysis were available. Such a machine would have two special characteristics:

1. It would have a very large store, in which information was very rapidly accessible.
2. It would be able to shift all the information from a given point downwards by a number of locations in one short operation.

Received: March 25, 1960.