INFORMATION AND CONTROL 3, 231-247 (1960)

The Application of the Ferranti Mercury Computer to Linguistic Problems

MICHAEL LEVISON

Department of Numerical Automation, Birkbeck College, London

In a book published in 1958[1] (Booth *et al.*, 1958) and hereafter called MRLP Booth, Brandwood, and Cleave discussed at some length the various ways in which electronic computers could be applied to resolving linguistic problems, and gave an account of the results obtained at the Birkbeck College Computational Laboratory.

Since that time, the University of London has acquired a Ferranti Mercury computer which, though primarily designed for the solution of mathematical problems (and making use of floating point arithmetic), has a large immediate-access store and a great speed which make it very suitable for linguistic work. This acquisition has made it possible to put into practice on a full scale many of the ideas which the book contained.

The present paper elaborates upon some of these ideas with particular reference to the way in which they are affected by the design of the Mercury. The paper is divided into several sections and the first devoted to a description of the Mercury insofar as it affects the design of linguistic programmes. The basic ideas behind the dictionary searching methods described in Section V appeared in MRLP. The minor discrepancies between some of the mathematical results in this section and the corresponding ones in MRLP are due to the assumption in the latter that this size of dictionary is large. It will become apparent that this assumption cannot always be made in dealing with the Mercury.

I. DESCRIPTION OF THE FERRANTI MERCURY COMPUTER

The Mercury computer as at present installed at the University of London Computer Unit is a one-address, pure binary, floating point machine with a core store of 1024 40-bit registers and a magnetic drum backing store of 16384 40-bit registers. In the following text the terms "register" and "location" are used to refer to places in the computer stores; "word" will be used exclusively in the linguistic sense.

[1] $[x]$ is used throughout to denote "the integral part of x."

The core store is divided into 32 parts called "pages," numbered from 0 to 31, each containing 32 40-bit registers, while the backing store is divided into 512 similar parts, numbered from 0 to 511, known as "sectors." Access to the backing store is by means of "drum instructions" which transfer the contents of any sector to any page or vice versa. It is possible to isolate blocks of sectors to prevent them from being overwritten.

The 40-bit registers mentioned above are called "long" registers. Each may be considered as two 20-bit "medium" registers, and each of these may be considered as two 10-bit "short" or "half" registers.

Floating point numbers, when stored in the computer, occupy one long register each, 10 bits being used to represent the exponent, and 30 bits to represent the fractional part.

The computer has a 40-bit accumulator in which operations in floating point arithmetic may be performed. The contents of any long register may be copied into the accumulator and vice versa.

Each machine instruction occupies one medium register, the most significant 7 bits of which give the function, the next 3 the "B-digits" (whose use is explained below), and the remaining 10 either an address or an integer modulo 2^{10} (depending on the function). Instructions can be obeyed only in the first half of the core store (pages 0–15) and are obeyed sequentially unless a "jump" instruction is encountered. Access to the second half of the core store is by drum and accumulator instructions only.

The short registers may be used for storing 10-bit integers which may be considered either as integers (modulo 2^{10}) in the range 0 to 1023 or as integers (modulo 2^{10}) in the range -512 to 511, or as addresses.

The machine also has 7 10-bit "B-lines" (denoted by B1, B2, $\cdots$, B7) in which simple arithmetical and logical operations with integers (modulo 2^{10}) may be performed.

Instructions may be divided into two main types. In those instructions which operate on B-lines, the B-digits mentioned above specify the B-line on which the operation is to be performed. The other instructions are said to be "B-modifiable"—that is to say, the contents of a B-line are added (modulo 2^{10}) to the address part of the instruction immediately before the function is performed, the B-line (if any) by which the instruction is to be B-modified being specified by the B-digits. The machine may be considered as having an eighth B-line, B0, which is always zero regardless of which operations are performed on it.

In addition, there is an extra set of instructions which operate only on B7 and which are themselves B-modifiable. B7, when used with these instructions, is called the Short Accumulator or Sac.

The B-lines and Sac can be tested for sign or for equality to zero, and "jumps" made conditional on the results of the test.

Input and output are by means of five-hole tape to or from the five least significant digits of any specified short register.

A short "binary" input routine is kept on sectors 0 and 1 of the drum and cannot be overwritten. It is used to read in one of several comprehensive interpretive input routines. The one most suited to linguistic programs is called PIG F and is read to sectors 2 to 63 of the drum. This in its turn interprets the program tape and assembles the program beginning in sector 128. The program is then brought into pages 1 to 15 of the core store in sections called "chapters" determined by the programmer, each small enough (i.e. less than 960 instructions) to be contained in these 15 pages, and is entered.

The times of the instructions which concern the linguist are as follows:

B-line, Sac and Jump instructions	1 beat or 60 μsec
Accumulator copying instructions	2 beats or 120 μsec
Drum transfers:	
Sector transfer time	$7\frac{3}{4}$ msec
Average time of random access	$8\frac{3}{4}$ msec

After the transfer of an odd sector any even sector is in position for the start of transfer 1 msec later, and vice versa.

Input instruction	1 beat

But no further *input* instruction can be obeyed until 5 msec have elapsed.

Output instruction	1 beat

But no further *output* instruction can be obeyed until 30 msec have elapsed.

II. WORD STORAGE

In order to run linguistic programs, one must be able to code words of real language into numerical form for storage in the computer, and to compare words by subtraction for equality and for alphabetical order.

Now, the fact that the Mercury is a floating point machine implies that additions and subtractions into its accumulator are almost always accompanied by some shifting and a possible resultant loss of digits,

even if the exponents are equal and if "non-normalizing" addition and subtraction instructions are used. For linguistic purposes, however, it is essential that no such loss of digits occurs, and that any additions or subtractions required are performed in "fixed point" form. Such operations must therefore be performed 10 bits at a time in the B-lines or in Sac, the accumulator being used only as a momentary store, or to transfer the contents of one long register of the core store to another.

It is convenient to code the English alphabet into numerical form as follows:

$$A = 1, B = 2, C = 3, \cdots, Z = 26.$$

Each of these numbers may then be expressed by a group of 5 bits called a "binary character." For storage, words are divided into pairs of characters beginning at the left, each pair $\alpha\beta$ being stored in a short register in the form $2^5\alpha + \beta$. Such pairs of characters are called "letter-pairs." From what has been said above about the Mercury, it is clear that letter-pairs form a convenient basic unit in the machine comparison of two words.

For simplicity, words may be stored as a fixed number of letter-pairs (e.g., as 8 letter-pairs, stored in 2 long registers), the absence of a letter being denoted by a zero.

Thus, for example, the word "CERTIFICATE" may be written as the 8 letter-pairs:

CE, RT, IF, IC, AT, E–, ––, ––

and these stored as:

101, 596, 294, 291, 52, 160, 0, 0

$= 3.32 + 5, \cdots, \cdots, \cdots, \cdot\cdot, 5.32 + 0, 0.32 + 0, 0.32 + 0.$

The conversion from a sequence of binary characters on a tape (each character representing one letter) to this form is, of course, carried out by the computer itself.

It is evident that if each of a set of words is expressed as k letter-pairs in this manner (where k is some fixed number) and the resulting $10\,k$ bits considered in each case to be a $10k$-bit *positive* integer, then the ascending numerical order of these integers is precisely equivalent to the alphabetical order of the words they represent.

Notation: If word W_α is alphabetically before word W_β, we shall write $W_\alpha \ll W_\beta$, etc.

III. WORD COMPARISON

To compare two words of k letter-pairs each for equality or for alphabetical order, it is necessary to compare the letter-pairs of one to the letter-pairs of the other, one by one, starting with the most significant. The comparison continues until (1) two corresponding pairs are found which are unequal, or (2) two corresponding pairs are found which are equal and equal to zero, or (3) all the corresponding pairs have been compared and found equal.

In case (1) the alphabetically earlier pair belongs to the alphabetically earlier word. In cases (2), (3) the words are equal.

Let W_α, W_β be two words having as their k letter-pairs $(\alpha_0, \alpha_1, \cdots, \alpha_{k-1})$ and $(\beta_0, \beta_1, \cdots, \beta_{k-1})$ respectively. Then the above rules are represented diagrammatically in Fig. 1.

IV. LETTER-PAIR COMPARISON

It will be appreciated that in coding letter-pairs into the form previously described, the resulting 10-bit integers are considered to be in the range 0 to 1023. For the purpose of examination of sign however, 10-bit integers in the B-lines or Sac are considered to lie in the range -512 to 511, each number in this range being identical in form with that one in the range 0 to 1023 to which it is equivalent modulo 2^{10}.

It follows that, within the machine, an integer in the range 512 to 1023 is considered to be less than one in the range 0 to 511, whereas for linguistic purposes it is necessary to consider it as being greater.

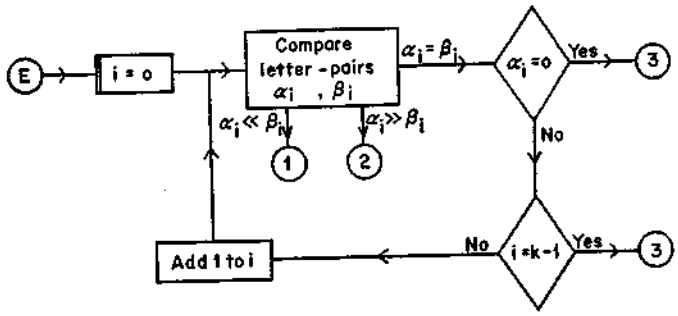

FIG. 1. Word comparison. 1, $W_\alpha \ll W_\beta$; 2, $W_\alpha \gg W_\beta$; 3, $W_\alpha = W_\beta$. (Diagram conventions: E is the the point of entry and the other circles the alternative points of exit of the programme. Blocks with a double line to the left contain the initial settings of the various parameters. Diamonds contain questions with Yes/No answers.)

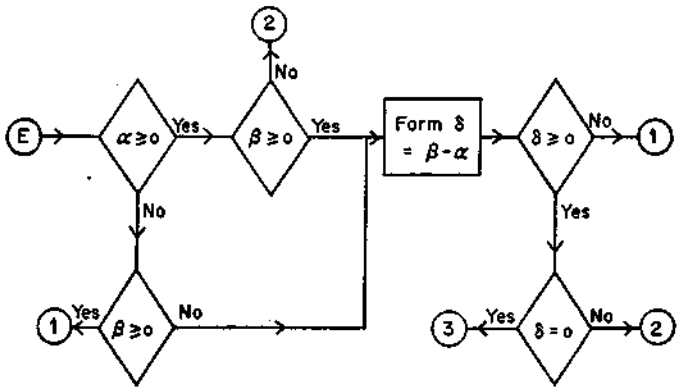

FIG. 2. Complete letter-pair comparison. 1, $\alpha \gg \beta$; 2, $\alpha \ll \beta$; 3, $\alpha = \beta$

The following rules must therefore be applied when comparing two letter pairs α, β, in order to overcome this difficulty:

1. If $\alpha < 0$ and $\beta \geqq 0$, then $\alpha \gg \beta$.
2. If $\alpha \geqq 0$ and $\beta < 0$, then $\beta \gg \alpha$.
3. If $\alpha \geqq 0$ and $\beta \geqq 0$, or if $\alpha < 0$ and $\beta < 0$, then $\alpha \gg$, $=$, or $\ll \beta$ according as $\alpha - \beta >$, $=$, or < 0.

These rules are represented diagrammatically in Fig. 2.

Of course, this complication only arises if it is necessary to compare the letter-pairs for alphabetical order. If it is merely required to determine equality or inequality, a much simplified form of comparison may be used (Fig. 3).

V. DICTIONARY SEARCHING

The purpose of any linguistic program is to read into the computer and to operate on the words of some text and to output some information about the text (e.g. a glossary, or a translation).

In the course of this it is sometimes found necessary to search a set of words (a "dictionary") previously stored in the computer in order (1) to discover whether or not a given "text word" is present in the dictionary, and/or (2) to obtain about the text word some information (e.g. its translation, or its part of speech) which has been stored side by side with the dictionary in the computer.

There are several methods available for dictionary searching and the basic ones are described and compared below.

A. The Direct Method

In this method, the dictionary words themselves are not stored at all. Instead, the information required about any dictionary word is stored in that location whose address bears some given arithmetical one-to-one relationship to the code number representing the word.

The method, though needing far the fewest number of machine instructions of any of the methods described, is nevertheless quite impracticable (except possibly if the "words" are restricted to one or two letters). The reason is that, for words of a given length, it requires the computer store to have as many locations as there are possible letter combinations of that length, while in fact only a very small fraction of these combinations—less than 1 in 10^9 if words are restricted to 12 letters—are actually words.

B. The Linear Method (Alphabetical Order)

The words of the dictionary are arranged in alphabetical order and stored in the machine. The text word is then compared with each in turn, starting with the first, using the word comparison illustrated in Fig. 1 with the letter-pair comparison illustrated in Fig. 2. until (1) a dictionary word is found which is the same as the text word, or (2) a dictionary word is found which is alphabetically later than the text word,

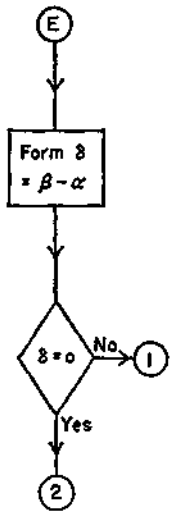

Fig. 3. Abbreviated letter-pair comparison. 1, $\alpha \neq \beta$; 2, $\alpha =$

or (3) the last word of the dictionary is passed without (1) or (2) occurring. In cases (2), (3) the text word is not present in the dictionary.

If the dictionary contains N words, the average number of word comparisons required by this method to locate a text word which is in it will be $\frac{1}{2}(N + 1)$. The average number required to detect the absence of a text word from it is $\frac{1}{2}N(N + 3)/(N + 1)$. These formulas are very approximate, being based on assumptions about the distribution of text words relative to the dictionary. (See Appendix.)

C. The Linear Method (Frequency Order)

This is similar to the previous method. The dictionary words are arranged, however, not in alphabetical order but in the order of probable frequency of occurrence.

The text word is again compared with each dictionary word in turn, this time until (1) a dictionary word is found which is the same as the text word, or (2) the last word of the dictionary is passed without (1) occurring.

In this method the abbreviated form of letter-pair comparison illustrated in Fig. 3 may be used, since it is only desired to ascertain whether or not dictionary and text words are exactly the same.

The average number of word comparisons to locate by this method a text word which is present in the dictionary is clearly less than the average number required by the previous method, being, in fact, $N/(\gamma + \log_e N)$, if Zipf's law holds. (See Appendix.) However, to determine if a text word is not present it is necessary to compare it with every word in the dictionary.

Furthermore, this method can hardly be used by the glossary-maker who, before making his glossary, would not know in which position in the dictionary to enter an unlocated text word, and who is anyway trying to arrange the text words in alphabetical order.

D. The Logarithmic (or Bracketing) Method

Once again let the dictionary contain N words, and let these be arranged in alphabetical order occupying "dictionary positions" 0 to $(N - 1)$.

The principle behind this method is as follows:—

Suppose we compare the text word, W_T, with the word D_H about half way down the dictionary. Then either

(1) $W_T = D_H$, or (2) $W_T \ll D_H$, or (3) $W_T \gg D_H$.

In case (1) we have located W_T in the dictionary. In case (2) we have discovered that W_T can only occur in the half of the dictionary before D_H. In case (3) W_T can only occur in the half of the dictionary after D_H.

In cases (2), (3) we repeat the whole process but with the appropriate half-dictionary replacing the whole, i.e., we compare W_T with the "quarter-way" word (case 2) or with the "three-quarter-way" word (case 3); and so on.

At each step either the text word is located in the dictionary or the range of the dictionary in which it can occur is halved. After, at most, $[\log_2 N]^1$ such steps this range has been reduced to one word only (if the text word has not been located) and a final word comparison indicates either that this word is the same as the text word or that the text word is not in the dictionary.

It will be seen later that the most convenient position for D_H is not, in fact, $\frac{1}{2}N$.

Now, the maximum number of dictionary words which one can search in m comparisons by this method is $2^m - 1$ (proved below), and it is found that complexities occur in programing the search if N is not of the form $2^m - 1$ (m integral).

To overcome these complexities we choose the integer m such that

$$2^{m-1} \leqq N < 2^m - 1$$

and expand the dictionary to contain $(2^m - 1)$ words by adding to it, in dictionary positions N to $(2^m - 2)$, "dummy" words, each consisting of letter-pairs 1023 (i.e., ensuring that, in any word comparison between a dictionary word W_D in this range and a text word W_T, we obtain $W_T \ll W_D$).

This, however, is not always possible or convenient. For example, suppose that the store were exactly large enough to hold a dictionary of 1500 words and that we wished to record a dictionary of 1450 words in it. Then we could not expand this dictionary to the required 2047 words by adding dummy words to it because there would be no *physical* locations in which to place the last 547 dummy words. Actually, in this case, it would be necessary to store dummy words only as far as the "three-quarter-way" position of the expanded dictionary (position 1535), since all the actual words occur before this. However, even this is not possible.

In such cases we can, instead, expand the dictionary *imaginarily*, i.e., no dummy words are actually added, but the dictionary is searched as

if it contained $2^m - 1$ words. Before taking any dictionary word D_p to compare it with the text word W_T, however, we examine its position number, p. If $p \geqq N$, no actual comparison takes place and we simply say $W_T \ll D_p$.

It is, of course, quite possible to combine these schemes and to store some dummy words physically and others imaginarily. In all cases the "half-way" position is $(2^{m-1} - 1)$, the "quarter-way" position is $(2^{m-2} - 1)$, and the "three-quarter-way" position $(3.2^{m-2} - 1)$. The dictionary positions with whose contents text words must be compared in successive steps are $P_1, P_2, \cdots, P_m$, where

$$P_1 = 2^{m-1} - 1, \qquad P_2 = P_1 \pm 2^{m-2},$$

$$P_3 = P_2 \pm 2^{m-3}, \cdots, P_m = P_{m-1} \pm 1,$$

or, in general,

$$P_1 = 2^{m-1} - 1, P_s = P_{s-1} + R_s (s = 2, 3, \cdots, m)$$

where $R_s = \pm 2^{m-s}$ and the sign is determined by the result at the preceding step.

The Logarithmic Search is illustrated diagrammatically in Fig. 4, the dotted section being included if imaginary dummy words are used.

The absence of a text word from the dictionary will, of course, be discovered in exactly m word comparisons. Also for each $s = 1, 2, \cdots, m$ there are just 2^{s-1} words of the dictionary reached with s comparisons.

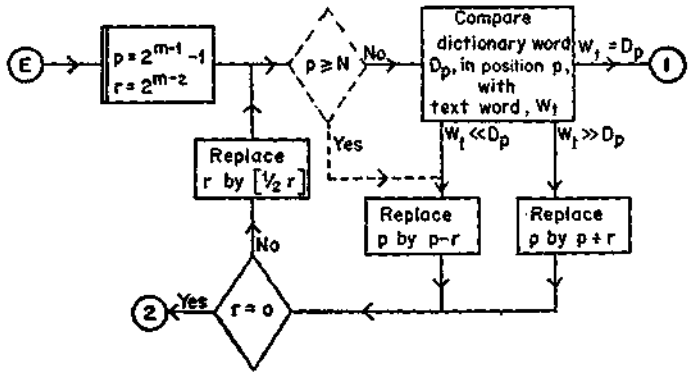

FIG. 4. Logarithmic dictionary search. 1, text word located in position p of dictionary; 2, text word not in dictionary.

This is why the maximum number of dictionary words which can be searched in m comparisons or less is $2^m - 1$, that is,

$$\sum_{s=1}^{m} 2^{s-1}.$$

Now, suppose $N \neq (2^m - 1)$, then some of the 2^{s-1} words may be dummy words. The number of actual words reached with s comparisons is

$$\left[\frac{N + 2^{m-s}}{2^{m-s+1}}\right].$$

This makes the average number of comparisons per located text word approximately $m - 1$.

There is a small refinement which might decrease this average very slightly. Provided that they are in correct alphabetical position, there is no reason why physical dummy words should be placed at the end of the dictionary. Thus, for example, if the dictionary has the sequence

$$\cdots, \text{THATCH, THAW, THE, THEATRE, } \cdots$$

we might put the dummy words between THAW and THE in the form THD. It might be possible, by this means, to defer to (say) the halfway position some very common word occurring just before it. Of course, one must choose as dummy words letter combinations which cannot be real words, especially in machine translation if a stem/ending analysis is in use. The refinement obviously cannot be used in making glossaries.

E. Refinements

Among the further refinements available in dictionary searching are those in which the dictionary is stored in several parts or "files."

For example, we might store the dictionary in two files comprising (1) common words, and (2) others; or in three files comprising (1) words of 4 letters and less, (2) words of 5 to 8 letters, and (3) words of more than 8 letters.

Each file is searched, of course, by one of the methods already described.

Of the various ways of partitioning the dictionary the one which concerns us must here is partitioning by initial letters. The dictionary is

divided into files in such a way that the (appropriate) file in wh
given text word can occur may be determined from the initial let
letter-pair of the text word.

There are two basic ways of linking a letter (-pair) with the a
priate file. One is to give the files numbers having a direct arithm
relationship to the letter (-pair). The other, less restricting, is to
ber the files in any convenient manner and to store a "file-directo
i.e., a special dictionary in which each "word" is an initial letter (
and the "information" is the number of the appropriate file.

It is usually quick and practicable to search the file directory b
direct method.

VI. DICTIONARY SEARCHING ON THE MERCURY

Up to now it has been assumed that each of the N dictionary
is quickly and readily accessible in the computer. This is in fact
with the Mercury, provided that the dictionary is small enough
contained side by side with chapters of the program in the core
(i.e., of the order of 100–200 words).

If, however, the Mercury is used to its best advantage and
2000–5000 words are stored, the words are no longer accessible in
ually. To reach them one must transfer an entire *sector* of words
drum to core store. As this takes a relatively long time it is essent
do it as infrequently as possible, and one must therefore try to tr
the appropriate sector for a given text word first time.

For this purpose we must arrange to partition the dictionary into
each sufficiently small to be contained on one sector. This will u
be done in one of the following ways:

1. We partition the dictionary according to initial letters (or l
pairs). The correct sector is brought to the core store by one of the
letter techniques described at the end of Section V,E. The latte
be complicated by the fact that some initial letters (pairs) are
common than others, and their words may not be containable
single sector.

2. We arrange the dictionary in alphabetical order and then
it into sufficiently small sections, recording the alphabetically e
word of each in order in the core store. A search among these
words" will indicate the sector on which the given text word can

Some space may possibly be saved by recording in the core sto
the earliest word of each sector but a short letter combination

divided into files in such a way that the (appropriate) file in given text word can occur may be determined from the initial letter-pair of the text word.

There are two basic ways of linking a letter (-pair) with th priate file. One is to give the files numbers having a direct arit relationship to the letter (-pair). The other, less restricting, is ber the files in any convenient manner and to store a "file-direc i.e., a special dictionary in which each "word" is an initial lette and the "information" is the number of the appropriate file.

It is usually quick and practicable to search the file directory direct method.

VI. DICTIONARY SEARCHING ON THE MERCURY

Up to now it has been assumed that each of the N dictionar is quickly and readily accessible in the computer. This is in fa with the Mercury, provided that the dictionary is small enoug contained side by side with chapters of the program in the co (i.e., of the order of 100–200 words).

If, however, the Mercury is used to its best advantage anc 2000–5000 words are stored, the words are no longer accessible *ually*. To reach them one must transfer an entire *sector* of word drum to core store. As this takes a relatively long time it is esse do it as infrequently as possible, and one must therefore try to t the appropriate sector for a given text word first time.

For this purpose we must arrange to partition the dictionary int each sufficiently small to be contained on one sector. This will be done in one of the following ways:

1. We partition the dictionary according to initial letters (or pairs). The correct sector is brought to the core store by one of the letter techniques described at the end of Section V,E. The latte be complicated by the fact that some initial letters (pairs) are common than others, and their words may not be containable single sector.

2. We arrange the dictionary in alphabetical order and then it into sufficiently small sections, recording the alphabetically e word of each in order in the core store. A search among these words" will indicate the sector on which the given text word can

Some space may possibly be saved by recording in the core sto the earliest word of each sector but a short letter combination

betically later than every word in one sector yet alphabetically earlier than every word in the next. This will reduce the length of the key-words.

In both of the above cases, once the files have been chosen, we may rearrange the words within them to suit any desired search method.

It is usually found convenient on the Mercury to take the fixed number of letter-pairs in a word to be 8. It is then possible to store 16 such words on any sector or page. If, however, the logarithmic method is used to search within the sector, it is more time-efficient to store only 15 ($= 2^4 - 1$). It is true that this leads to a wastage of 1 dictionary position in every 16, but in some programs uses are found for these spaces.

Special routines are devised to deal with words of 17 or more letters, but these are normally rare.

VII. SECTOR SEARCHING

We now come to consider which of the searching methods to use for a sector of words once it is in the core store. Naturally the choice is made on a time basis since the various methods produce identical results.

The time taken for a word comparison depends, of course, on the number of letter-pairs which it is necessary to compare. If the words being compared are the same, the average number of letter-pair comparisons required is $[(j + 1)/2]$ where j is the average number of letters per word of the text ($\simeq 5$ in most English texts). If the words are different, it will generally be sufficient to compare only the first letter-pair (i.e., two words selected at random generally have different initial pairs) but within the appropriate sector more will probably be necessary.

Reference to Figs. 2 and 3 reveals that the time taken for the abbreviated form of letter-pair comparison (Fig. 3) is constant, but that the time for the full form (Fig. 2) is highly variable and, on average, somewhat longer.

Let L,F be the average times for a single word comparison using respectively the full and abbreviated forms of letter-pair comparison. We will not consider here to what extent L,F depend on the fixed number of letter-pairs chosen for the words, but will assume this number to have been selected previously. It is clear, however, from the preceding paragraph that F is somewhat less than L (say, $\frac{1}{3}L \leqq F \leqq \frac{2}{3}L$). And the relative times taken to search the sector in the core store for a given text word (where the sector contains N ($= 2^m - 1$) words) are shown in Table I.

TABLE I

Method of search	Linear (alphabet)	Linear (frequency)	Logarithmic
Average time to locate text word in general dictionary position	$\frac{1}{2}(N+1)L + 2^{m-1}L$	$NF/(\log_e N + \gamma)$	$(m-1)L$
Average time to locate text word in worst dictionary position	NL	NF	mL
Average time to detect absence of text word from dictionary	$N(N+3)L/2(N+1)$ $\simeq 2^{m-1}L$	NF	mL

TABLE II

Method of search	Linear (alphabet)	Linear (frequency)	Logarithmic
Average time to locate text word in general dictionary position	$8L$	$4.6F$	$3L$
Average time to detect the absence of text word from dictionary	$8L$	$15F$	$4L$

If $N = 15 = 2^4 - 1$ (i.e., $m = 4$), Table I acquires the values now shown in Table II.

It will be seen from this table that the logarithmic method is quicker, on average, than the linear (alphabetical) method, but that there is little difference in speed between the logarithmic and the linear (frequency) methods. We have already seen, however, that the latter cannot be used in making glossaries. In addition, one further consideration prevails in machine translation when a stem/ending analysis is in use.

In a stem/ending analysis, if the text word is not located in the dictionary, its last nonzero letter is removed and the dictionary searched again for the new word (a "stem"). This process continues either until some stem is found in the dictionary or until no stem remains. The letters removed are called the "ending."

Suppose that the word has q nonzero letters and that the ending (if the stem is eventually located) has E letters. Then before locating the

TABLE III

	Linear (frequency)	Logarithmic	$(L(F) - \log)$ for $N = 15$
Average time to locate text word	$E(NF + S) + \frac{NF}{\log_e N + \gamma}$	$E(mL + S) + (m - 1)L$	$E(15F - 4L) + 4.6F - 3L$
Average time of complete failure	$qNF + (q - 1)S$	$qmL + (q - 1)S$	$q(15F - 4L)$

stem there will be E unsuccessful searches and one successful one. If no stem is found in the dictionary there will be q unsuccessful searches.

Suppose S to be the average time required to remove the last non-zero letter of a text word. Then the relative search times, assuming the dictionary to be arranged so that the sector appropriate to the text word is appropriate to every stem, are shown in Table III.

As it is probable that $4L < 15F$, and since the average value of E must be positive and is, in most languages, greater than 1, the scale is tipped slightly towards the logarithmic method. This is far more marked for any $N > 15$.

With all these preliminary factors decided, the way is clear to apply the Mercury to such problems as (1) the construction of glossaries, (2) the construction of concordances, and (3) machine translation and allied problems.

Appendix

THE DERIVATION OF THE MATHEMATICAL RESULTS OF SECTIONS V, B, C

A. Located Words

Let the rth dictionary word be denoted by W_r and let its relative frequency of occurrence be f_r. Suppose that T of the words occurring in a given text are located in the dictionary. Then, number of occurrences of W_r is given by

$$f_r T \Big/ \sum_{s=1}^{N} f_s$$

And the number of word comparisons required to reach W_r by any linear search $= r$. Therefore, the average number, A, of word comparisons/located text word is given by

$$A = \left(\sum_{r=1}^{N} rf_r\right) \Big/ \left(\sum_{r=1}^{N} f_r\right)$$

If the dictionary is arranged in frequency order, then $f_1 \geqq f_2 \geqq \cdots \geqq f_N$, and if, in addition, Zipf's Law holds,

$$rf_r = \text{const.} = M \text{ (say)}.$$

Therefore,

$$A = MN \Big/ \left(\sum_{r=1}^{N} M/r\right) = N/(\gamma + \log_e N).$$

DEFINITION: Let O be a fixed point on the straight line $\overrightarrow{OX}$, and suppose W_r to be represented by a particle of mass f_r on OX at a distance r from O. Then the center of mass of the particles representing the dictionary is called the "centroid" of the dictionary.

Now suppose the dictionary to be arranged in alphabetical order. Then $W_1 \ll W_2 \ll \cdots \ll W_N$, and $f_1, f_2, \cdots, f_N$ are to a large extent in random order. Thus common words, etc., will be found scattered over the dictionary and the centroid will be very approximately at the midpoint, $\frac{1}{2}(N + 1)$.

On this assumption,

$$\sum_{r=1}^{N} rf_r = \tfrac{1}{2}(N + 1).Nf,$$

where

$$f = \frac{1}{N}(f_1 + f_2 + \cdots + f_N).$$

Therefore,

$$A = \tfrac{1}{2}(N + 1).Nf/Nf = \tfrac{1}{2}(N + 1).$$

It should be noted that while an assumption is made about the distribution of common words in the (computer) dictionary, *no* assumption is made about their distribution over the alphabet. This is because there is no reason to suppose that the dictionary words are themselves spread evenly over the alphabet.

The approximation will be a good one if the dictionary contains (say) only 16 letter words, or uncommon words, since then $f_r \simeq f$ for all r.

B. UNLOCATED WORDS

DEFINITION: If W_α, W_β are any two letter combinations with $W_\alpha \ll W_\beta$, the set of all letter combinations x such that $W_\alpha \ll x \ll W_\beta$ is called the "alphabetical interval (W_α, W_β)."

Let W_0 denote the "null" word (every bit zero) and W' the word consisting of the single letter pair 1023.

Then if the dictionary is arranged in alphabetical order and a text word is not located in it, the text word must lie exactly in one of (W_0, W_1), (W_1, W_2), $\cdots$, (W_{N-1}, W_N), (W_N, W').

Now it is clear that in any (real) language certain initial groups of letters are more common than others so that the unlocated text words will probably occur more frequently in some alphabetical positions than in others. However, in these alphabetical positions there will probably be more dictionary words so that the alphabetical intervals will be correspondingly smaller.

It is therefore reasonable to expect that the unlocated text words will occur fairly evenly among the $(N + 1)$ alphabetical intervals.

Now if a text word is in (W_{s-1}, W_s) $(s = 1, 2, \cdots, N)$ the linear (alphabetical) search requires s word comparisons to detect its absence from the dictionary; if in (W_N, W'), N. Therefore the average number of word comparisons/unlocated text word

$$= \left(N + \sum_{s=1}^{N} s\right) \Big/ (N + 1)$$

$$= \tfrac{1}{2}N(N + 3)/(N + 1).$$

ACKNOWLEDGMENTS

The author would like to thank Dr. A. D. Booth for his encouragement and advice, and Miss J. Metherell who typed the script.

RECEIVED: MARCH 25, 1960.

REFERENCES

BOOTH, A. D., BRANDWOOD, L., and CLEAVE, J. P. (1958). "Mechanical Resolution of Linguistic Problems." Butterworths, London.