NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

THE USE OF FRAGMENTS FOR DATA COMPRESSION

by

SUDESH GUPTA

A thesis submitted to the Faculty of Graduate Studies and Research, in partial fulfillment of the requirements for the degree of Master of Computer Science.

Department of Computer Science
Concordia University
Sir George Williams Campus
Montreal, Quebec

SEPT. 1977

ABSTRACT

THE USE OF FRAGMENTS FOR DATA COMPRESSION

SUDESH GUPTA

The purpose of this thesis is to study the storing of large amounts of data in an economical way by fragmentation. Fixed length and variable length text, as well as word fragments, are discussed. An algorithm has been developed to generate variable length dictionary fragments which give some preference to the longest fragments with a frequency exceeding a given threshold. It has been found that the word dictionary size is much larger than the fragment dictionary. The techniques discussed take into consideration that the compressed data must be completely reversible.

ACKNOWLEDGEMENTS

I wish to express my sincere appreciation to Prof. H. S. Heaps my supervisor for his advice and guidance throughout the course of research and preparation of this thesis. I am also thankful to my family and my husband for providing the needed encouragement.

My special thanks to my friends for their assistance during the writing of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Data storage and retrieval problems arise in many applications. They are encountered by business professionals who review periodic reports that arrive from various departments within a plant or from plants within a company. Enormous amounts of storage allocations, and frequent retrieval of periodicals, books and other bibliographies, are performed by librarians each year. The library in its traditional concept is oriented more to the provision of documents than to the supply of information. But this is changing. More and more people are interested in having fast access to more and more reliable information.

There has been an information explosion in all scientific disciplines. Thus two important problems in information retrieval are i) the organization of information and the optimization of storage, and ii) reduction of the time of retrieval of relevant documents. This thesis is intended as a contribution to the first problem.

Salton (1968) describes extensive file organization and retrieval methods. Collmeyer (1970) gives a critical analysis of retrieval performance for selected file organization techniques. One of the simplest file organization is the sequential file structure in which information is stored on

magnetic tape in alphabetic order with respect to some field. The retrieval of a record involves rolling the tape backward or forward until the desired record may be retrieved. Any additions and deletions require a complete rewrite of the entire sequential file. As explained by Chapin (1969), sequential file organization typically necessitates extensive and frequent sorting operations to reorder the file. In many installations the total sorting time is as high as one-quarter or one third of the total computer usage.

It is often desirable to store and retrieve records using more than one key. This is most easily accomplished by sorting the data in duplicate files with the first file sorted by one key and the second by a second key. This duplication, however, uses twice the storage space and involves double the maintenance cost, since records to be updated must be accessed and changed in both files. The method becomes useless if more than two keys are used for storage and retrieval of file records. It is not practical to store the same file as many times as the number of keys.

One alternative to the sequential file is one organized in the form of a multilist file. Associated with each key is a pointer to the list of records that have that key. Only the beginning address of each list occurs in the sequential index directory. The successive records of a particular list are obtained by means of the pointer that

resides in each record and points to the next record containing that key. The major advantages of this file structure are programing simplicity and updating flexibility as explained by Lefkovitz (1969). However, the disadvantages of the multilist structure are: i) that in order to respond to a multikey query, the records corresponding to the shortest list must be brought in core, even though only a small intersection of the lists satisfies the query, and ii) the total file requires much more storage space than occupied by the actual data, because of the pointers that reside within the records.

It is desirable to use file structure in which the only records brought into core are the ones that satisfy the query. This allows a rapid search and requies less storage space than required for the multilist file. One of such file structures is the inverted index file. As explained by Lefkovitz (1969) the inverted index structure allows a shorter time for initial and successive responses to queries. It is therefore appropriate to describe the details of the inverted index structure.

INVERTED INDEX FILE

In the inverted index organization each search key appears in a key directory. Corresponding to each key in the key directory there exists a pointer in a position

SMITH G. DETROIT 33 ANALYST
JONES S. ROYAL OAK 26 ANALYST
ABEL B. WARREN 19 PROGRAMMER
ABEL A. DETROIT 26 PROGRAMMER

Four Records in Sequential File

Key-Directory	Position Directory
Analyst	1
Programmer	3
19	5
26	6
33	8
Detroit	9
Royal Oak	11
Warren	12
Abel A.	13
Abel B.	14
Jones S.	15
Smith G.	16
//////////	17

Inverted Index
1
2
3
4
3
2
4
1
1
3
2
3
4
3
2
1

FIG. 1.1 Inverted Index File Organization

directory to point to the first record number in an inverted index list of numbers of records which contain that key. The record numbers are stored in the inverted index as explained by Lefkovitz (1969), Dodd (1969) and Lowe (1968). The complete file may be stored sequentially. The Fig. 1.1 shows an inverted file organization for four records.

In an inverted index organization it is possible to process a query using the key directory and the inverted index; the sequential data base is accessed only after determining the record numbers of the relevent records. For example, let us assume that a request for information on all the 'Analysts' who live in 'Royal Oak' is recieved. The 'analyst' entry in the index shows that record numbers 1 and 2 contain this key. An examination of the 'Royal Oak' entry in the index shows that the second record satisfies the required key. Application of the 'AND' operation on these two lists [(1,2),(2)] shows that only the record number 2 satisfies the intial request. An access of the sequential file may then be made and the record having the name 'Jones' retrieved.

Hence, retrieval can be done efficiently with the inverted file organization;but it requires additional storage for the inverted index and the key directory as explained by Cardenas (1973, 1975). It is possible to reduce the total storage by coding each key in the key directory and storing

12 6 5 1	11 7 4 1	10 8 3 2	9 6 4 2

Coded Sequential File

Key-Directory	Position Directory
Analyst	1
Programmer	3
19	5
26	6
33	8
Detroit	9
Royal Oak	11
Warren	12
Abel A.	13
Abel B.	14
Jones S.	15
Smith G.	16
////////////	17

Inverted Index
1
2
3
4
3
2
4
1
1
3
2
3
4
3
2
1

FIG. 1.2 Inverted Index File Organization With coded Data Base

a sequential data base that contains the coded representation of each key. If the indexes of the key directory are assigned as the codes of the corresponding entries (keys) in the directory then the records (fig. 1.1) and their coded form will be as shown in Fig. 1.2.

The size of the dictionary (key directory) depends on the number of words present in the data base. It is a well known empirical law, which has been proved analytically by Mandelbrot (1953), that if D_w is the number of words in the data base and N_w is the total number of words in the data base, then

$$D_w = E\ N_w^B$$

where B and E are constants determined by the data base characteristics and $0 < B < 1$.

The above equation indicates that the number of dictionary entries increases as the data base grows, and there is no upper bound to the size of the dictionary. Even if the most frequent words are stored in core, there will be many more words to be stored in random access devices and their decoding will require disk accesses. As the data base increases there will be new entries in the dictionary, many of which will be words that occur infrequently. It is also well known, Zipf's law (1949), that if the different words are ranked in order of decreasing frequency then their frequency is related to the rank as follows:

$$f(r) = k/r$$

where f(r) is the frequency of word of rank r and k is a constant, which can be determined from the equation

$$N_w = k \sum_{r=1}^{D_w} 1/r.$$

As described by Booth (1967) it follows that 50% of the different words occur only once in the data base, 16.66% occur twice and 100/[n(n+1)]% occur only n times. Thus coding the 50% infrequent words that appear only once in the data base would hardly save any storage but would greatly increase the size of the dictionary required for coding and decoding of the sequential data base. Thus it is important to investigate suitable codes to obtain maximum possible compression of the data base with a dictionary of moderate size.

First we survey briefly some of the important techniques that may be used in the compression of a data base and then give the scope of the thesis.

A coding scheme for compression of text in document data base has been described be Heaps and Thiel (1970) and subsequently been analyzed by Heaps in (1972). The compression is achieved by use of a dictionary in which each uncoded word is stored only once. Each word in the data

base is expressed in coded form. For a term of frequent occurrence the space required to store the repetition of the code is significantly less than that required to store the repetitions of uncoded term. The above coding scheme, gives a compression ratio 25% for 100,000 different words, but gives rise to a large dictionary.

Several authors have considered the use of the most common words or phrases as codeable elements for compression. Wagner (1973) has developed a method for compact generation of diagnostic messages in a compiler. This technique depends on manual selection of a set of text strings which are common to one or more error messages. These phrases may be stored without redundancy, and adoption of the method for PL/C compiler messages showed a minimum saving of 28% approximately. White (1967) has obtained compression of printed English text by dictionary encoding through storage of the most frequently occuring words, phrases, and in addition letter and letter combinations. The designed encoder for this scheme obtains the longest possible match between the input text and dictionary entries, and generates a code which contains fewer binary digits than the entry itself. Extensive experiments with this scheme indicates that 50% compression can be obtained for a broad type of English language text.

In order to reduce the storage requirements of an exhaustive word coding scheme, Schwartz (1963) has designed a split dictionary which contains, not only frequent words, but stems and suffixes from which the less common words can be synthesized. The experimental dictionary implemented by him contained 3,849 words and stems and 36 suffixes from which an additional 2,261 different words could be synthesized.

Colombo and Rush (1969) have discussed the use of word fragments in computer based retrieval systems. The fragments are stored in a pair of dictionaries; the first one contains all the shortest word fragments listed alphabetically together with the set of words in which each occurs; the second one contains words listed alphabetically together with the fragment which uniquely represent them. This pair of dictionaries enables the selection of appropriate fragments both for 'generic' retrieval and retrieval of specific words.

Rickman and Gardner (1973) have shown that bigrams can be used for automatic indexing. Their experimental results reveal that the term predicted by using bigrams are effectively the same as those predicted by using both bigrams and longer letter strings. An alternate technique, in which commonly occuring digrams are replaced by codes of shorter length, has been discussed by Snyderman and

Hunt (1970). They obtained a compressed text by using EBCDIC 8 bit codes which are not present in the data base. The digrams are chosen so that their initial letters belong to a set of 8 'master' characters which are selected primarily by frequency but arranged to include vowels. The second letter of the digram belongs to a set of 21 combining characters. Thus a total of 168 digram combinations are obtained. The decision to use such a combination of 8 master and 21 combining characters is not based on any theory but purely on a trial and error basis. They obtained a reduction of about 35% on 200,000 records.

An improved version of digram coding has been studied by Schieber and Thomas (1971), but the basic technique is again to use EBCDIC 8 bit codes. The algorithm used by Snyderman and Hunt does not base the selection of these two-character pairs (called digrams) on their frequency of occurrence in the data. The technique described by them is an improved and extended concept of encoding digrams on the basis of frequencies. The technique is based on the principal of taking two alphabetic characters that frequently appear in combination and replacing them with one unused special character code. Their experimental data base conatins 40,000 bibliographic records taken from the Central Library and Documentation branch of International Labour Office, Geneva. As only 58 characters are used in the data

base, 198 codes are available. Saving one character for some special function, 197 most frequent digrams are selected for coding, and a reduction of 43.5% has been obtained.

A common problem in the publishing and banking industry is the generation of a set of keys to be used for subsequent searching of the data base, with the input words preserved in the keys. One solution is to truncate the words to fixed length, usually by dropping the rigthmost letters. The uniqueness of the resulting keys as a function of the key length has been studied by Lowe (1971) and the effectiveness for retrieval by Lowe (1967).

Bourne and Ford (1961) have discussed thirteen coding techniques for abbreviation of words to standard length and have illustrated them by application to two small data bases of 2,082 and 8,184 terms. It is regarded as desirable that each coded word should be decipherable as a string of characters with some mnemonic similarity to the uncoded word. This criterian becomes of little concern, however, if the code is computer decodable into its English spelling. Four compression techniques have also been discussed by Nugent (1968). Dolby, has described a method of variable length name compression and a method based on truncation. He states that it is desirable to retain at least five characters in the truncated code. An advantage of compression based on truncation and character supression

is that the coding is an operation performed directly on the original word without the necessity for search through a large table. As a result, the information discarded cannot be reconstructed from the coded term.

Gottlieb et al (1975) have surveyed different techniques for compression. In particular, they compared differencing and statistical encoding, and results based on examination of large insurance files show that, for a variety of file structures, the stastistical compression methods are more generally applicable than differencing, but at a higher cost of CPU time.

Clare et al (1972) have proposed the use of variable length character strings as codeable elements. These strings are not limited to be substrings of whole words, provided they are part of the record. For this reason, the fragments are called 'Text fragments'. Note that the text fragments may contain embedded blanks. Clare et al place an additional restriction on the text fragments that are to be included in a dictionary, namely that the dictionary fragments should occur in the data base with approximately equal frequencies. These dictionary fragments are intended to be used as the indexing, compression and retrieval elements. The fragments below a certain threshold are also included in the dictionary. However, it is not advantages to replace a fragment of low frequency by a code because of the fairly high cost of

including it in the dictionary which must be kept in an easily accessible, and therefore more expensive, storage medium.

A method of selection of equifrequent fragments has been described by Schuegraf and Heaps (1973). Use of equifrequent character strings has considerable merits. As has been observed by the authors, this technique i) allows maximum compression to be achieved by use of fixed length codes, ii) requires fixed length buckets for storage of documents numbers within the inverted index, and iii) permits the use of a fixed size dictionary and allows more freedom of design. They have observed that the use of fixed length codes for fragments of text gives much greater compression than the use of fixed length codes based on run length coding as described by Lynch (1973).

Data compression is of interest in data processing because it offers cost saving and the potential for increased capacity in mass storage devices and channels. As observed by Ruth (1972) the one characteristic of compression which separates business data from temetry data is that no loss or change of data can be tolerated in business applications. The compression techniques discussed in this thesis all allow a reduction of the size of files, but probably at the price of increased CPU activity needed for compression and decompression. Compaction of data may be achieved by means of any

technique which reduces the size of the physical representation of the data including, perhaps, a reduction in the relevant information. Compression of data is a compaction technique which is completely reversible and hence does not reduce the amount of information.

The file oriented technique described in this thesis are compression techniques which are easy to implement in a general fashion. In each technique a dictionary of distinct fragments is maintained and codes are assigned for these fragments. We consider text fragments as well as word fragments, and fixed length codes as well as variable length codes. Thus the four different compression techniques described in this thesis refer to the four possible combination of fragments (word/text) and codes (fixed/variable).

In Chapter 2 of the thesis a description of the MARC tape record format is given. While a familiarity with the semantics of a file may be necessary for maxmial compaction, the compression techniques have the advantage of automatic applicability to a variety of files. Thus only the format and not the semantics of MARC tapes is relevant for our purpose.

In chapter 3 we describe compression by use of fixed length fragments. The experiment is done by using MARC tape samples to generate the set of text/word fragments of a

particular length N ($2 \leq N \leq 8$). For each N, the corresponding fragment dictionary is then used to code the data base by assigning either fixed length codes or variable length codes.

In chapter 4 procedures are given for the selection of variable length fragments. A certain threshold frequency acts as a parameter whose value determines the number of distinct fragments. The selection algorithm has been designed to give some preferance to the choice of the longest fragments with a frequency greater than the threshold. Use of a fragment dictionary, and coding of the data base by using concatenation of non-overlapping fragments, allow data compression. We notice from the experiments that the dictionary size in this case is much smaller than the word dictionary (refer chapter 3). Hence it may be possible to store the dictionary in core which leads to a rapid decoding of the record that satisfy a query.

CHAPTER 2
MARC TAPE FORMAT AND EXPERIMENTAL DATA

In this Chapter we describe the data base used for testing the algorithms used to process text and word fragments. We have selected a section of MARC (Machine Readable Catalog) tapes issued by the U. S. Library of Congress in 1969.

2.1 MONOGRAPH DESCRIPTION

The MARC tapes are available as either seven or nine track tapes. Each physical record on the tape is less than or equal to a maximum of 2048 characters. The information pertaining to a single monograph may consist of one or more physical records, called a logical record.

Each monograph description contains a leader, followed by a record directory, followed by a one-character field terminator, followed by a set of control fields and followed by a set of variable length fields. The difference between the control fields and the variable fields is that the latter contain internal tags to indicate the position of subfields. Each control field and each variable field contains a field terminator as its last character, except that in the last variable field, the field terminator is replaced by a record terminator. The general format of a monograph is illustrated in Fig 2.1.

The leader is a fixed length field for all the records and consists of 24 characters. The subfields of the leader are shown in Fig. 2.2.

The record directory consists of a series of fixed-length subfields (12 characters each) which contain the identification tag (T_j), the length (L_i) and the starting character position (P_i), in the record of each of the variable or control fields. The description of the record directory entries is shown in Fig. 2.3.

The control fields contain alphanumeric data elements, many of which have a fixed length. All the control fields end with a field terminator (FT).

The variable fields are made up of variable length alphanumeric data. All the fields end with a field terminator code, except the last field in the logical record. The format of control and variable length fields is given in Fig. 2.4. The format and description of the variable length field is given in Fig. 2.5. For a detailed description of MARC tapes refer The Library of Congress MARC manuals (1970).

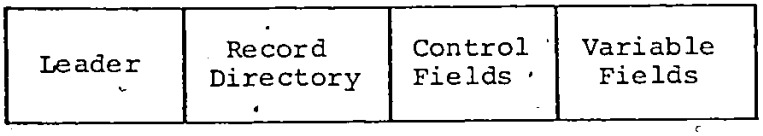
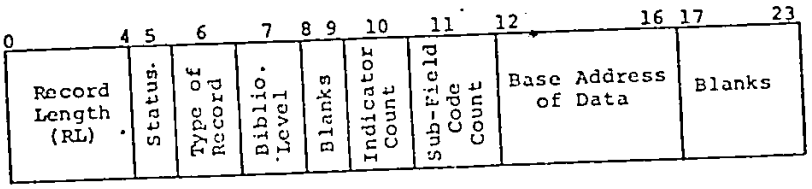


FIG. 2.1 MONOGRAPH FORMAT



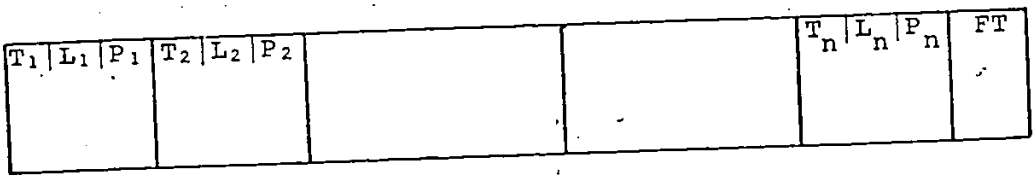
(a)

Name of Leader Data Element	Character Position in the Record	Description
RL (Record Length)	0 - 4	Total number of characters in the logical record including itself
Status	5	n, c or d n = new c = corrected or revised d = deleted
Legend		
a) Type or record	6	a = language material, printed
b) Bibliographic level	7	m = monograph
c) Two blank characters	8 - 9	
Indicator Count	10	Each variable field begins with two characters called indicators which provide certain information about the data which follows.
Subfield Code Count	11	Each subfield within a variable length field is identified by a two-character code
Base Address of Data	12 - 16	Indicates the position in the record at which the set of control fields begin
Seven Blank Character	17 - 23	Not used at present

(b)

FIG. 2.2 LEADER

(a) Outline of Leader
(b) Contents of Leader



(a)

Name of Record Directory Data Element	Number of Characters	Description
Tag (T_i)	3	The numeric symbol which identifies the field
Field Length (L_i)	4	The number of characters in the field identified by the tag
Starting Character Position (P_i)	5	The character position in the record of the first character of the field, relative to a base address

(b)

FIG. 2.3 RECORD DIRECTORY

(a) Outline of Directory
(b) Contents of Directory

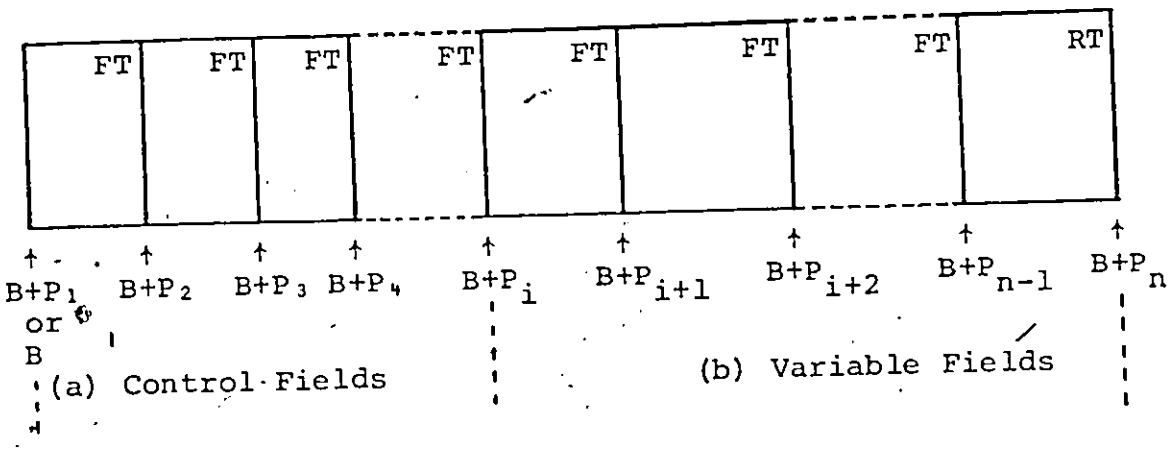
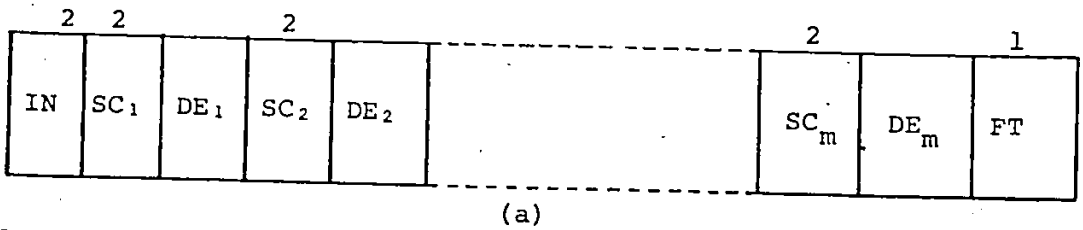


FIG. 2.4 CONTROL AND VARIABLE FIELDS



(a)

Name of Variable Fields Data Elements	Length	Description
Indicators (IN)	2	Each variable field will begin with two characters which provide descriptive information about the field
Subfield Codes (SC_i)	2	The subfield code precedes each data element in a field and identifies the data element. $ is the first character
Data Element (DE_i)	-	Contents of the subfield or data element, which may be of variable length

(b)

FIG. 2.5 OUTLINE OF VARIABLE FIELDS

(a) Outline of Fields
(b) Contents of Fields

TABLE 2.1 MARC TAPE (ISSUE 1969), USING 646 RECORDS

Tag	Field				
	Minimum Length	Maximum Length	Frequency	Total No.of Characters	Average Length
1	12	12	646	7752	12.00
8	40	40	646	25840	40.00
15	14	14	1	14	14.00
20	13	20	42	553	13.17
40	34	46	2	80	40.00
41	7	16	33	327	9.91
50	7	33	646	12038	18.63
51	24	24	1	24	24.00
82	7	26	632	7510	11.88
100	15	55	520	13846	26.63
110	19	112	79	4375	55.38
111	62	112	11	854	77.64
130	29	29	1	29	29.00
240	10	93	25	953	38.12
245	8	369	646	51553	79.80
250	10	44	124	1762	14.21
260	18	208	646	29496	45.66
300	16	68	643	17839	27.74
360	8	27	290	2455	8.47
410	29	69	8	379	47.37
440	22	86	30	1131	37.70
490	10	123	164	6807	41.51
500	10	279	261	17304	66.30
504	24	113	279	9923	33.41
505	18	713	27	7468	276.59
520	77	238	13	1784	137.23
600	13	79	67	2434	36.33
610	21	68	29	1182	40.76
611	31	31	1	31	31.00
630	12	57	10	302	30.20
650	8	81	659	20155	30.58
651	11	58	17	578	34.00
652	10	74	119	4426	37.18
700	10	64	167	5722	34.26
710	20	121	83	4565	55.00
730	11	66	5	176	35.20
740	16	72	43	1356	31.53
810	37	126	23	1728	75.13
840	43	62	3	159	53.00

FIELD STATISTICS

2.2 EXPERIMENTAL DATA DESCRIPTION

The data base used for experiments consists of 646 logical records. The Table 2.1 indicates the frequency of occurence of each of the tags, minimum length of each field, maximum length of each field, total number of characters in each field and average length of each field in the data base.

The textual data used in the experiment consists of the contents of each control field excluding end of field mark (FT) and the contents of each variable field excluding indicators, subfield codes and end if field. The leader and record directory are not part of the textual data.

The textual portion of the 646 logical records amounts to 229,309 characters, of which 48 are distinct. Table 2.2 illustrates the distinct characters and the frequencies of their occurence.

A word, in the experimental data, is any string of non blank characters chosen from the textual data. Blank characters are used as word separators. The textual data yielded 34056 words of which 10,829 were different.

The characteristics of the experimental data are shown in Table 2.3 and the word length distribution is shown in Fig. 2.6.

TABLE 2.2 MARC TAPE (ISSUE 1969), USING 646 RECORDS

CHARACTER	FREQUENCY	CHARACTER	FREQUENCY
ø	31356	Y	3664
E	14383	8	3024
I	11257	2	2907
A	11244	3	2798
N	11202	F	2503
S	10204	5	2127
O	10053	7	1903
R	9905	4	1888
.	9451	W	1721
T	9110	V	1448
L	7025	K	1218
C	6597	-	1185
0	6343	(	1073
1	6263	)	1065
9	5164	J	675
D	4974	/	634
H	4715.	X	511
6	4435	=	352
P	4420	↑	279
U	4411	Z	236
M	4187	;	234
,	3963	ß	152
B	3721	Q	127
G	3584	^	18
		♦	

Character frequency count of textual data of MARC tapes, using 646 records. This count includes every field of each record, except Leader, Record directory, FT, Indicators, and Subfield codes.

TABLE 2.3 MARC TAPE (ISSUE 1969), USING 646 RECORDS

Length	Number of Words	Number of Different Words
1	694	22
2	4716	240
3	4634	793
4	3626	883
5	4269	1205
6	3614	1313
7	2859	1276
8	3154	1894
9	1783	974
10	1666	856
11	1400	448
12	496	317
13	458	183
14	148	100
15	260	71
16	50	45
17	26	24
18	89	82
19	49	48
20	30	23
21	15	15
22	6	5
23	4	3
24	6	5
25	2	2
27	2	2

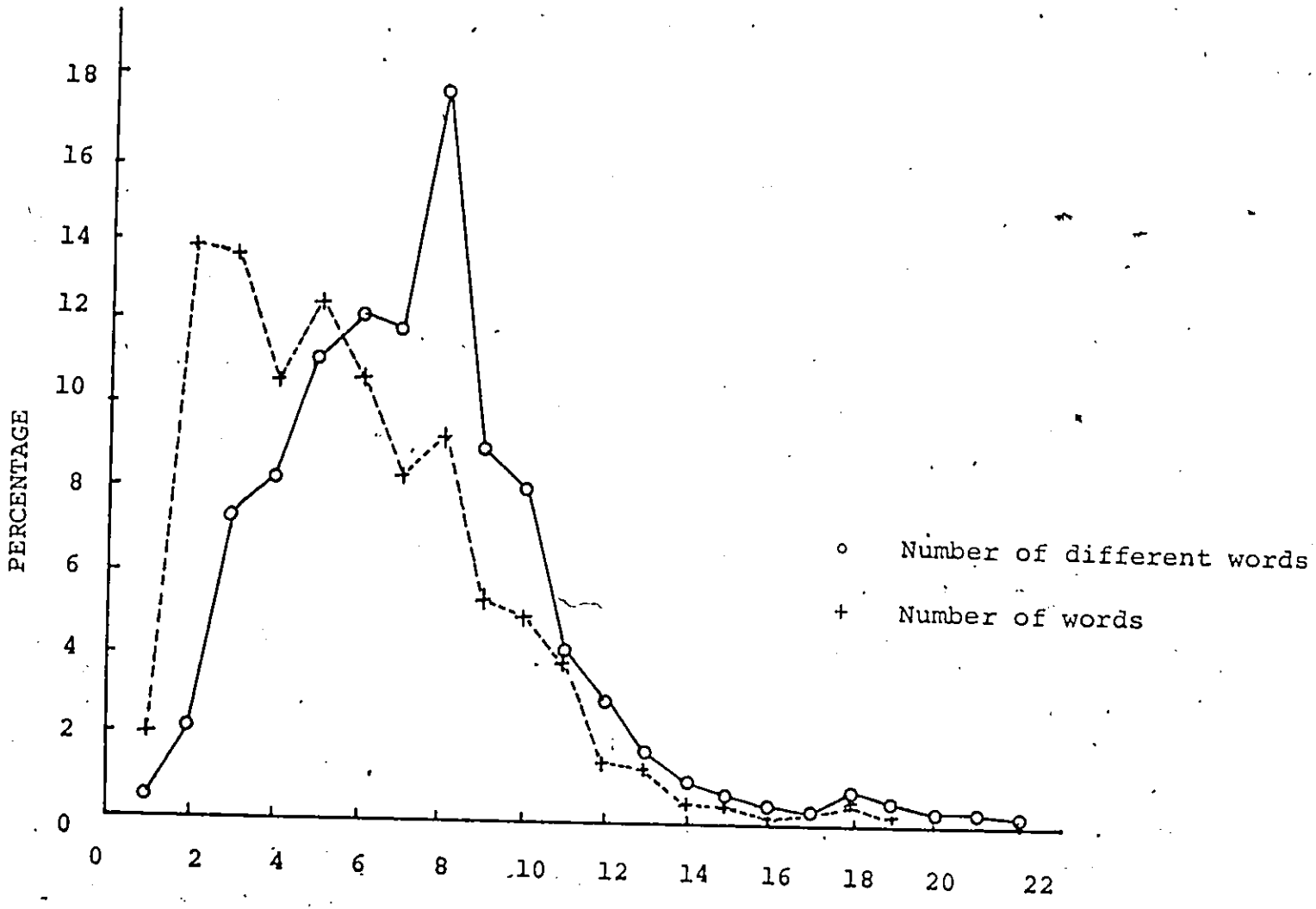


FIG. 2.6 WORD LENGTH

CHAPTER 3

FIXED LENGTH FRAGMENTS

This chapter will discuss methods for coded representation of a data base using a suitable dictionary to store fragments of the words or texts. Before we consider the methods for such a representation we require the following preliminary definitions:

3.1 DEFINITIONS

Definition 1 Suppose a group of M characters (including blanks) is partitioned into K sub-groups, such that each sub-group contains the same number N of characters. Each sub-group will be called a fragment of fixed length N.

Definition 2 Instead of partitioning a given set of characters into equal size we can have a partition resulting in fragments of different lengths. Such fragments will be called variable length fragments.

Definition 3 A group of characters of which no character is a blank will be called a word.

Definition 4 A text will consist of a group of words such that one or more blanks separate two adjacent words.

Definition 5 A word, fragmented according to definition 1 and padded with right blanks if necessary, will result in fixed length word fragments.

Definition 6 A text, fragmented according to definition 1, will result in fixed length text fragments.

Definition 7 Each word or text fragment can be represented uniquely by means of a numeric code (section 3.3). If the number of bits in such a coded representation is the same for each fragment, then we call the code a fixed length code.

Definition 8 If the number of bits required for a unique representation varies from fragment to fragment (as in the case of Huffman codes), then we call the code a variable length code.

3.2 CODING METHODS

Based on the previous definitions we give below some possible methods of coding a given data base:

Method 1 In this method we partition the data base into fixed length text fragments and store the data base in a coded form by using fixed length codes for each text-fragment.

Method 2 The only difference between this method and the previous method is that here we use variable length codes instead of fixed length codes for each text-fragment.

Method 3 This is the same as method 1 except that instead of text-fragments we use word-fragments.

Method 4 . This is the same as method 2 except that instead of text-fragments we use word-fragments.

Method 4 This will assign fixed or variable length codes for each word in the data base.

Example 1 Consider the text:

BIBLIOGRAPHICAL REFERENCES INCLUDED IN NOTES

Let N = 3 be the length of each fragment. Then the fixed

length word-fragments are:
BIB, LIO, GRA, PHI, CAL, REF, ERE, NCE, SØØ, INC, LUD, EDØ, INØ, NOT, ESØ.

In the actual implementation of the codes for the data base we generate a fragment of blanks to mark the end of each word the length of which is an exact multiple of N. This is necessary during the decoding process of the data base and for exact reproduction of it in the textual form.

Since the length of the first word in the text of Ex. 1 is a multiple of N (=3), a fragment of blanks will be generated after CAL during the implementation. Thus, fragmenting the text into word-fragments will result in sixteen, instead of fifteen fragments.

For the same value of N the fixed length text-fragments are BIB, LIO, GRA, PHI, CAL, ØRE, FER, ENC, ESØ, INC, LUD, EDØ, INØ, NOT, ESØ. Thus there are fifteen text-fragments in total and we need exactly fifteen codes, one for each fragment. In adopting this scheme, no fragment of blanks needs to be generated.

3.3 DATA BASE REPRESENTATION USING DICTIONARY AND CODED DATA

The five methods which we described in the previous section are all minor variations of the general method which makes use of a dictionary and a coding scheme. The dictionary consists of distinct fragments (word or text) generated by scanning the given data base. The data base can be coded either by using fixed length codes for the fragments in the dictionary or by variable length codes for the fragments. Suppose there are D fragments in the dictionary, then $j=\lceil \log_2 D \rceil$ bits are required to give a unique address of a location in the dictionary. Thus any fragment f_j in the dictionary can be coded in terms of the address c_j which is a fixed length code.

Alternately, if we use minimum redundancy codes (Huffman codes) for the fragments in the dictionary, then we will keep the fragments as well as their codes in the dictionary. Whatever coding scheme we use, the method of coding the data base is simply to replace every fragment in the data base by the corresponding code. We shall see in the following sections the advantages and disadvantages of the five methods.

3.4 STORAGE REQUIREMENTS

3.4.1 TEXT FRAGMENTS

If method_1 is used, we partition the data base into text-fragments such that the length of each fragment is N. Let D_N be the number of different fragments of length N in such a partition of the data base. If c is the number of bits required to represent a character in the machine, the storage for the dictionary of distinct fragments would be cND_N bits. If $2^{t-1} \leq D_N < 2^t$, then we need $t = \lceil \log_2 D_N \rceil$ bits to code a fragment in the dictionary. Thus the total space (in bits) necessary to store the dictionary and the coded data base is $T_f(N) = cND_N + t\lceil N_c/N \rceil$, where N_c is the number of characters in the data base. In particular, for N=1, the total storage (in bits) is $cD_1 + N_c \lceil \log_2 D_1 \rceil$.

As described in chapter 2, the experimental data base consists of 229,309 characters of which 48 are different. If such a data base is stored by using fixed length codes for each character then the space requirements are as shown in Table 3.1.

TABLE 3.1

Number of characters in the data base N_c	229,309
Number of different characters D_N	48
Average code length/character c	.6 bits
Space for coded data	1,375,854 bits
Space for dictionary	288 bits
Space for dictionary and coded data $T_f(1)$	1,376,142 bits

SPACE REQUIREMENTS USING CHARACTER CODES

Now we discuss the storage requirement when variable length codes are used to represent fixed length text-fragments. As mentioned earlier, let N be the length of each fragment and D_N be the number of distinct fragments in the dictionary. An efficient coding scheme is to assign codes for the fragments depending on the probability of their occurance. One such scheme is due to Huffman (1952).

Let $P_j(N)$ be the probability of occurance of the jth text-fragment ($1 \leq j \leq D_N$) in the data base. Then the average information content of each group of N characters when coded without reference to adjacent fragment is given by

$$H(N) = - \sum_{j=1}^{D_N} P_j(N) \log_2 P_j(N).$$

This is also the average code length in bits of the most compact codes that could be used to represent the text-fragment. It has been observed Heaps (1972) that for large data bases such a coding scheme is not suitable because of the time and the space requirements during the decoding process.

Thus the total space (in bits) required for the coded data base and the dictionary is

$$T'_f(N) = cND_N + H(N) \lceil N_c / N \rceil$$

Since the actual data used consists of a sequence of substrings selected as explained in chapter 2, we find that we do not have exactly N_c/N fragments for a given string consisting N_c characters. We represent the actual number of fragments of length N by NF_N.

Fig 3.1 illustrates the decrease in the total number of fragments as N increases from 2 to 8. Note however, as shown in Fig 3.2, the number of different fragments increases in the range $2 \leq N \leq 6$ and thereafter decreases slowly. Fig 2.3 shows the average number of bits required to represent a fragment of length N, using both fixed and variable length codes. In both cases this number increases for $2 \leq N \leq 5$ and then remains constant.

In Table 3.2, we give the storage requirements for the coded data base using fixed length as well as variable length codes for fixed length text fragments of size N, $2 \leq N \leq 8$. We infer from the results that the storage requirement for the data base is minimum, when N=3, whether we use fixed length codes or variable length codes. This is graphically represented in Fig 3.4.

We can conclude from Table 3.1 and 3.2 that the amount of storage required to represent the dictionary and coded data for a text of given size is less when the text is fragmented into strings of length N ($2 \leq N \leq 8$)

rather than of strings of length one.

TABLE 3.2

N	NF_N	D_N	$\lceil \log_2 D_N \rceil$	H(N)	$T_f(N)$	$T'_f(N)$
2	117,320	1,267	11	9	1,305,724	1,071,084
3	80,260	7,199	13	11	1,172,962	1,012,442
4	61,937	15,157	14	12	1,230,886	1,107,012
5	50,942	18,168	15	13	1,309,170	1,207,286
6	42,481	18,874	15	13	1,316,679	1,231,717
7	37,815	18,391	15	13	1,339,647	1,264,017
8	33,940	17,479	15	13	1,348,092	1,280,212

TEXT FRAGMENT STATISTICS

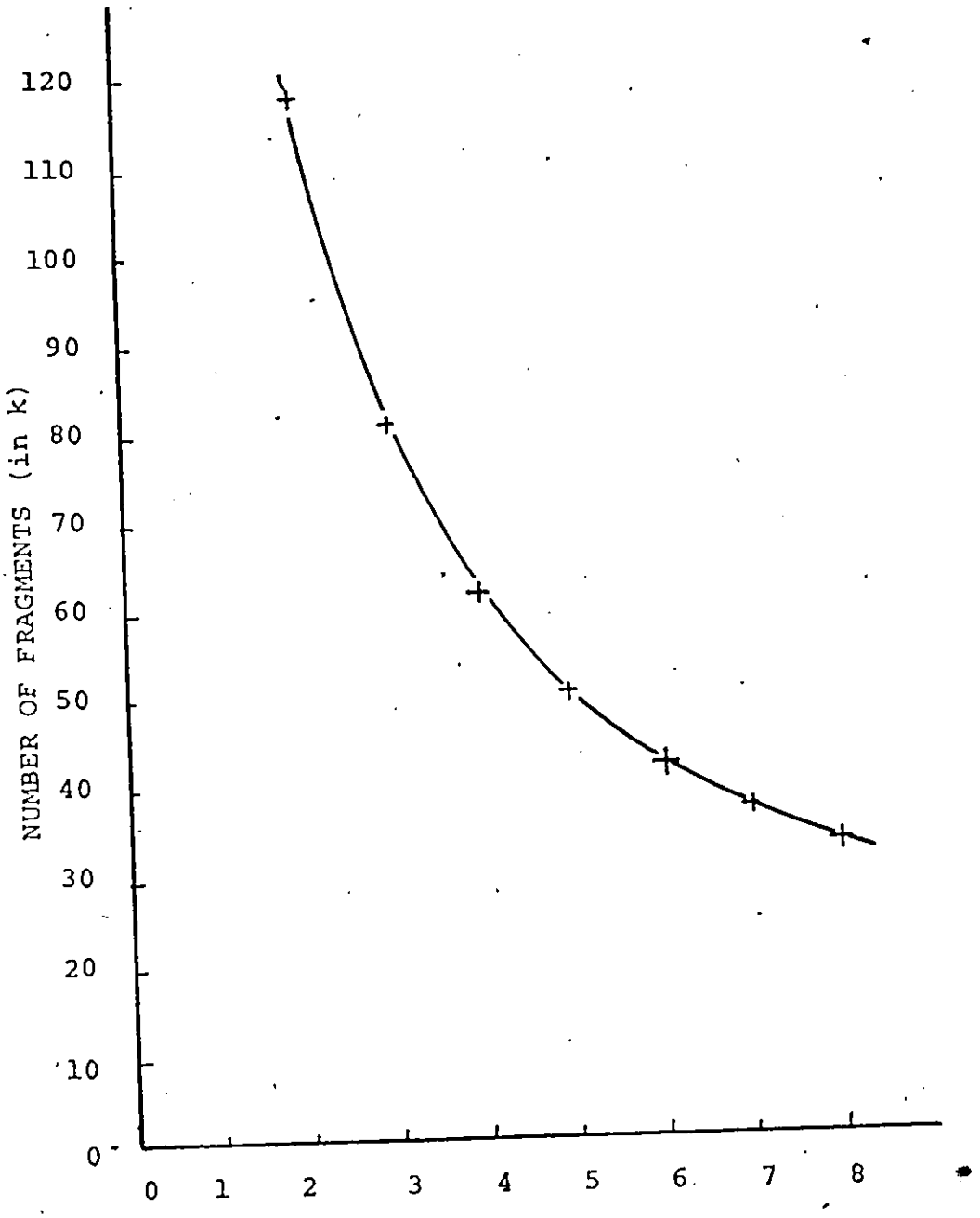


FIG. 3.1 TEXT FRAGMENT LENGTH (in characters)

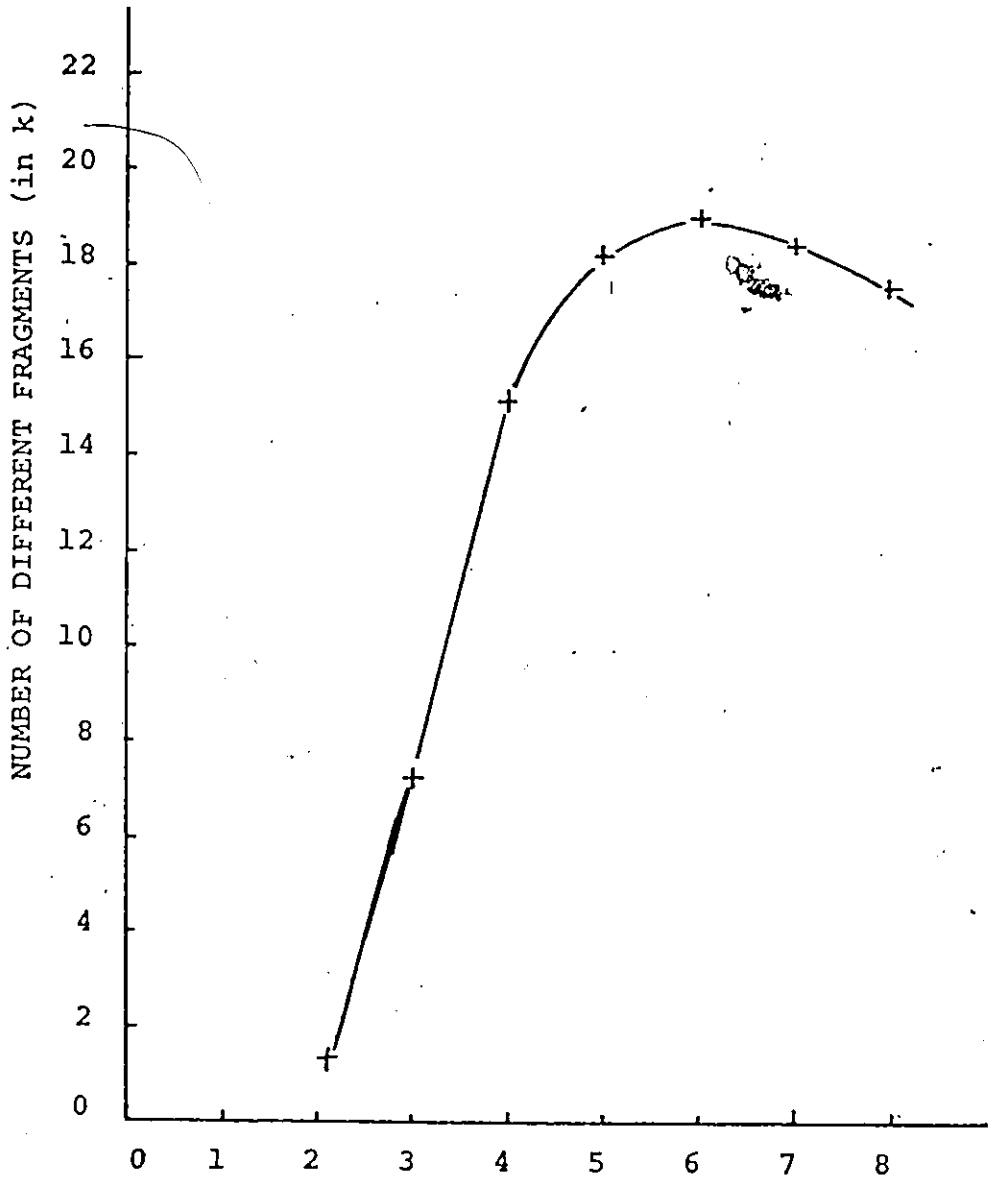


FIG. 3.2 TEXT FRAGMENT LENGTH (in characters)

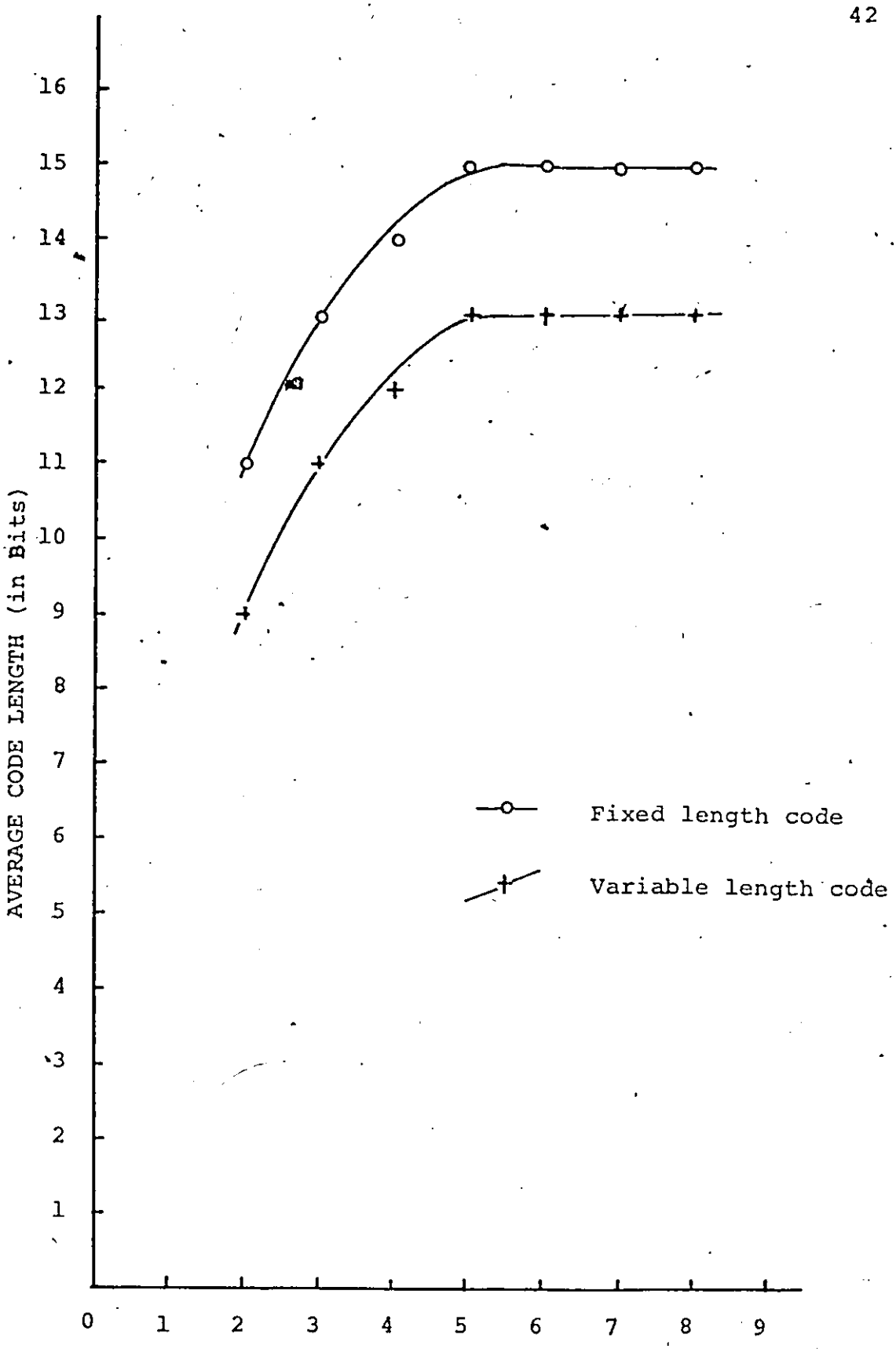


FIG.3.3 TEXT FRAGMENT LENGTH (in characters)

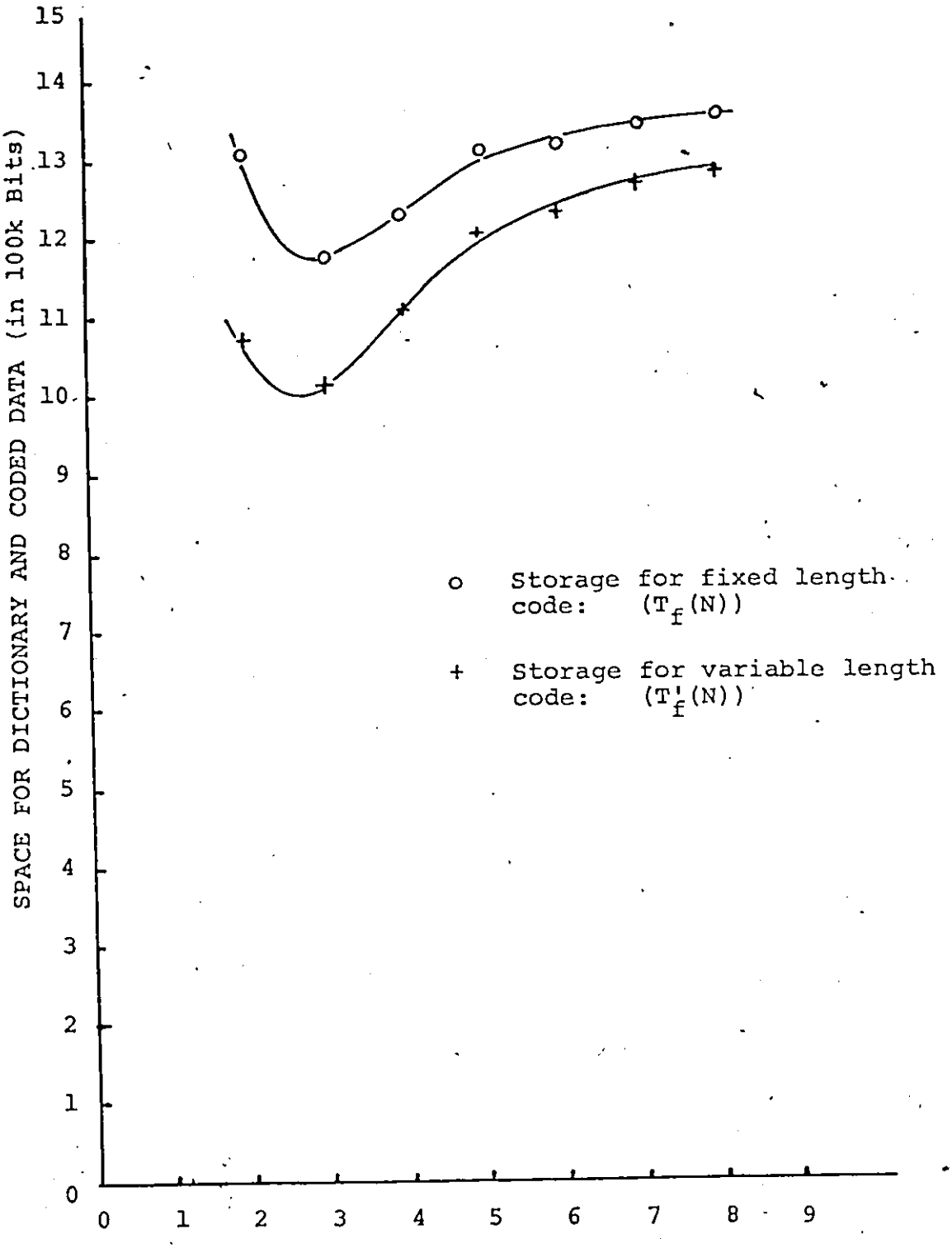


FIG. 3.4 TEXT FRAGMENT LENGTH (in characters)

3.4.2 WORD FRAGMENTS

We will consider fragmenting the words of the data base into fixed length fragments with a view towards efficient coding and economy of storage. We achieve efficiency in retrieval since most of the time the search in the data base is word oriented. Schemes of text-fragmenting the data base as explained in the previous section are context dependent in the sense that the text-fragments of a particular word depend on the predecessor and successor words. In contrast, word fragmenting schemes are context free. Thus the number of different word-fragments will be less than the number of different text-fragments for a given subset of the data base.

Each word in the data base is partitioned into fragments of length N, followed if necessary by a fragment of a smaller length to terminate the word without inclusion of right blanks. The choice between keeping word-fragments, each of length exactly N (padding with blanks, if necessary), and keeping those word-fragments without blanks, is crucial in terms of storage for the coded data base. To economise on storage the stored word fragments will be of varying lengths M where $M \leq N$.

Let $D_N(M)$ be the number of different word-fragments of length M. Let T(N) be the total number of word-fragments in the data base. Let Q(M) be the probability that the word chosen from the data base is of M characters. Then

$$T(N) = N_W \{ \sum_{1}^{N-1} Q(M) + 2 \sum_{N}^{2N-1} Q(M) + \ldots\ldots \} ,$$

where N_W is the total number of words in the data base. We must consider the following cases.

Case 1:

By padding fragments of length smaller than N with blanks, we keep a dictionary consisting of $D_N(N)$ fragments of length N. Thus the number of bits required to store the dictionary is $cND_N(N)$. The number of bits required to give a unique code to each item in the dictionary is $t = \lceil \log_2 D_N(N) \rceil$. Hence, the total storage for dictionary and the coded data base is

$$S_f(N) = cND_N(N) + tT(N).$$

Case 2:

Instead of padding each word-fragment of length M ($M \leq N$), we can store fragments of different lengths. We store fragments of length j in DIC_j. Let the number of different fragments in DIC_j be $D_N(j)$. If $T_N(j)$ is the number of fragments, each of length exactly j, in the data base, then the total number of word-fragments in the data base is

$$T(N) = \sum_{j=1}^{N} T_N(j)$$

Also, the number of bits required to give a unique code to each item in DIC_j is $t_j = \lceil \log D_N(j) \rceil$. We also require $\lceil \log_2 N \rceil$ bits to give a unique code to identify the N dictionaries. These bits could preceed each coded fragment and during the decoding procedure they could be checked to determine to which dictionary the corresponding fragment belongs. The total space required to store the dictionaries and coded data base is

$$S''_f(N) = c \sum_{M=1}^{N} M D_N(M) + \sum_{j=1}^{N} t_j T_N(j) + T(N) \lceil \log_2 N \rceil$$

Case 3:

This is the same as Case 1, except that we use variable length codes, instead of fixed length codes. Thus the total space required to store the dictionary and the coded data base is

$$S'_f(N) = cND_N(N) + T(N)\ H(N),$$

where H(r) is the average length of the Huffman codes for fragments of length r.

Case 4:

This is the same as Case 2, except that we use variable length codes. It is easy to see that the total space required to store the dictionary and the coded data base is

$$S'''_f(N) = c \sum_{M=1}^{N} MD_N(M) + \sum_{j=1}^{N} H(j)T_N(j) + T(N)\lceil \log_2 N \rceil$$

We have applied each of the above methods to the data base described in Chapter 2. The results are shown in Fig. 3.5 to 3.8 and in Table 3.3.

Fig. 3.5 illustrates the decrease in total number of word fragments as N increases from 2 to 8. However, we observe from Fig. 3.6 that the number of different fragments increases as fragment length varies from 2 to 8. This increase is rapid for $2 \leq N \leq 4$ and relatively slow for $5 \leq N \leq 8$.

As we would expect, the total number of characters in the dictionary becomes greater when we pad each fragment with blanks to maintain fixed length, as shown in Fig. 3.7, this difference becomes significant only for fragment lengths greater than 4.

Fig. 3.8 presents a summary of relative space requirements for each of the four cases we have just described.

In conclusion, the storage required for the data base is less when variable length codes are assigned to word fragments. The use of separate dictionaries for fragments of different lengths, further reduces space requirements. We should note, however, that the extra storage required for fixed length codes may well be compensated by the speed gained in the decoding process.

It may be observed that the use of word fragments requires more storage in all four different methods than does the use of text fragments, the difference being very significant for $N \geq 4$. For text fragments, it was found that minimum storage resulted when $N = 3$. In contrast, for word fragments the minimum storage space results for values of $N \geq 6$. Using fixed length codes, the use of word fragments with $N = 7$ or 8 requires approximately 85% of the storage required when using text fragments with $N = 3$.

TABLE 3.3 WORD FRAGMENTS

N	T(N)	$D_N(N)$	$ND_N(N)$	$\sum_{M=1}^{N} MD_N(M)$	$\lceil H(N) \rceil$	$\lceil \log_2 D_N(N) \rceil$	$S_f'''(N)$	$S_f'(N)$	$S_f''(N)$	$S_f(N)$
2	124,805	1,217	2,434	2,391	9	11	1,058,060	1,137,849	1,341,726	1,387,459
3	87,739	5,764	17,292	16,658	11	13	1,036,842	1,068,881	1,218,882	1,244,359
4	69,707	9,933	39,732	37,088	11	14	943,267	1,005,169	1,106,498	1,214,290
5	59,757	10,514	52,570	45,964	11	14	956,060	972,747	1,087,574	1,152,018
6	52,027	10,538	63,228	52,840	11	14	904,914	951,665	1,023,929	1,107,746
7	47,281	10,717	75,019	59,803	11	14	878,716	970,202	998,416	1,112,048
8	43,989	10,810	86,480	66,230	11	14	878,212	1,002,759	975,557	1,134,726

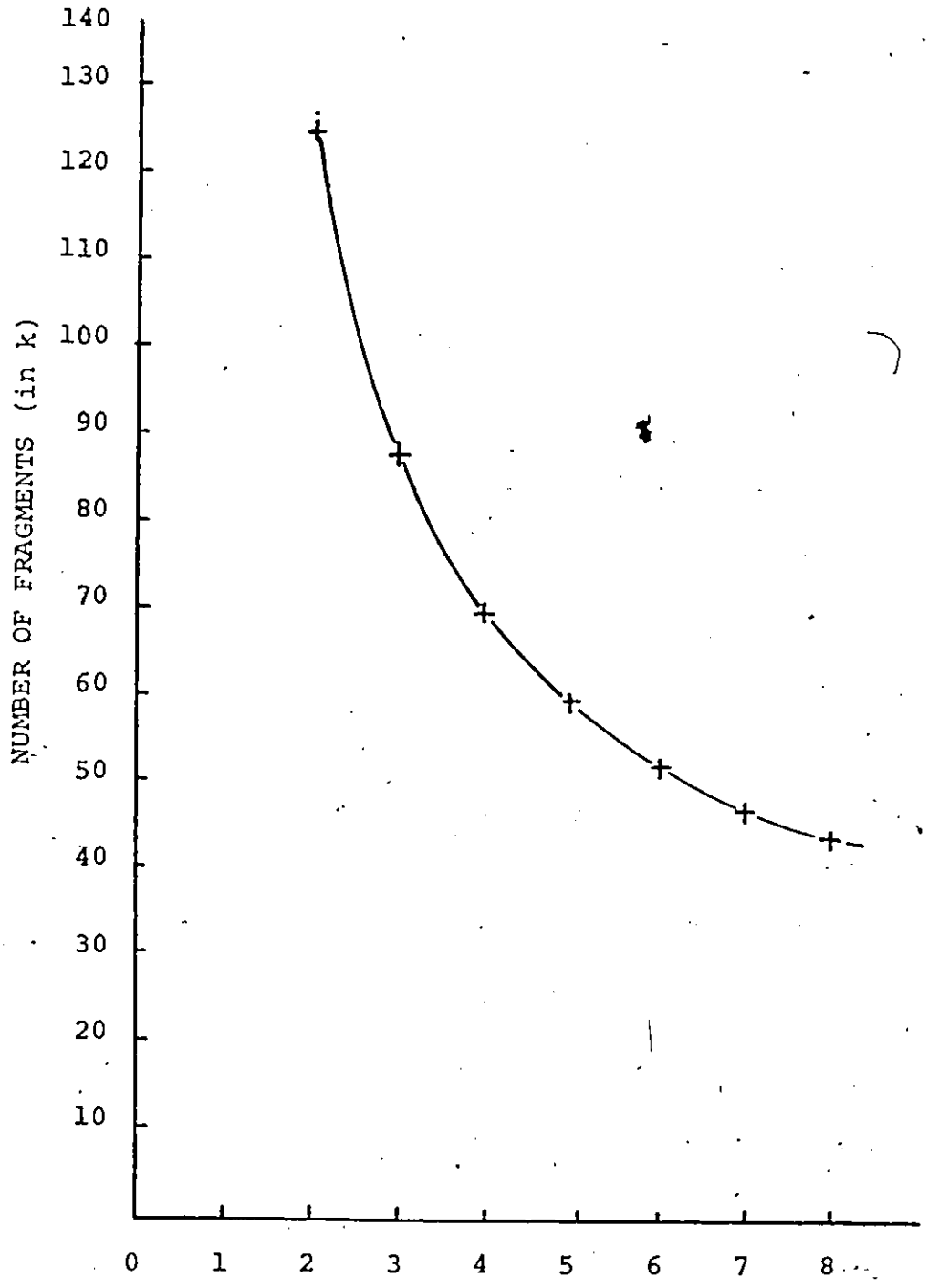


FIG. 3.5 WORD FRAGMENT LENGTH (in characters)

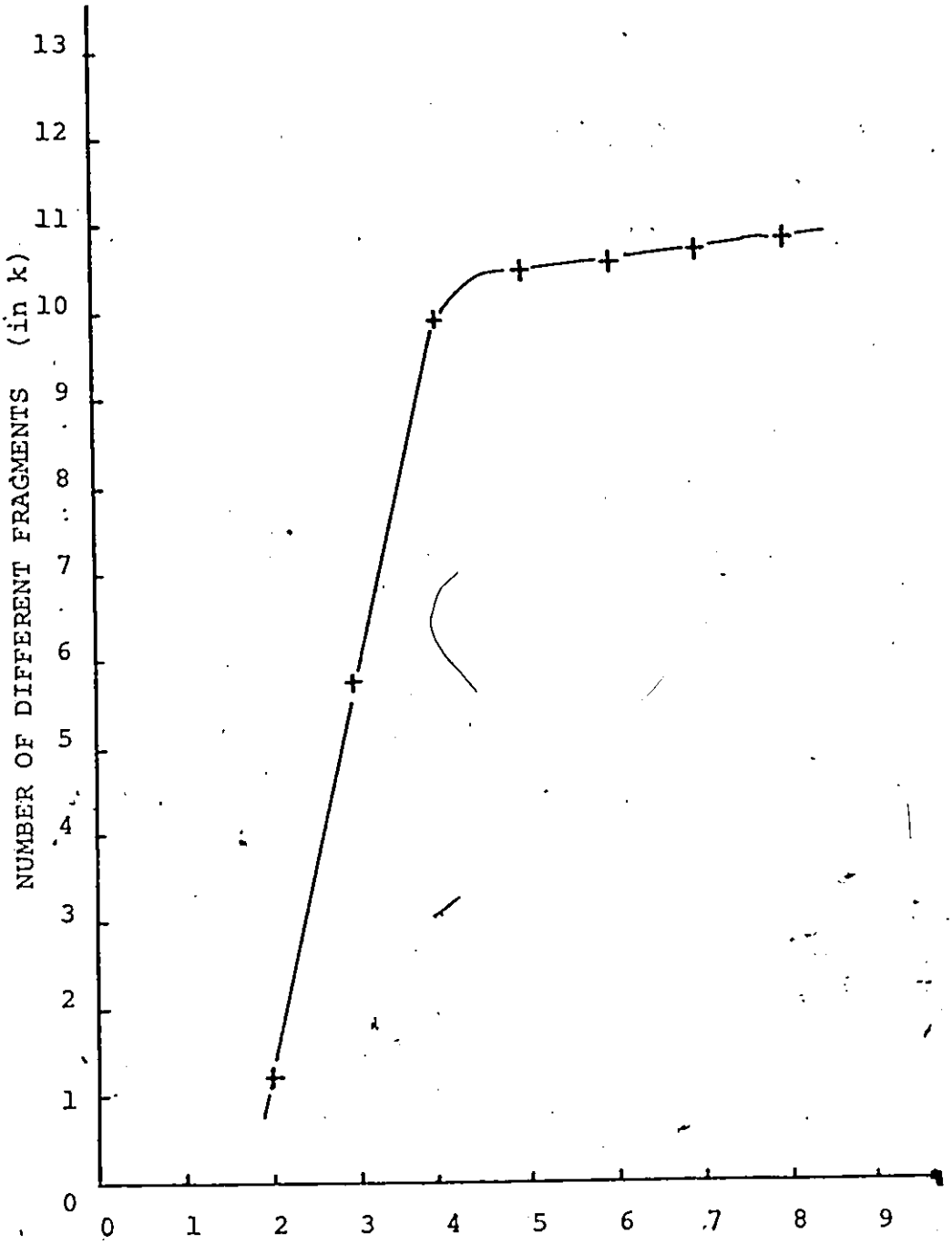


FIG. 3.6 WORD FRAGMENT LENGTH (in characters)

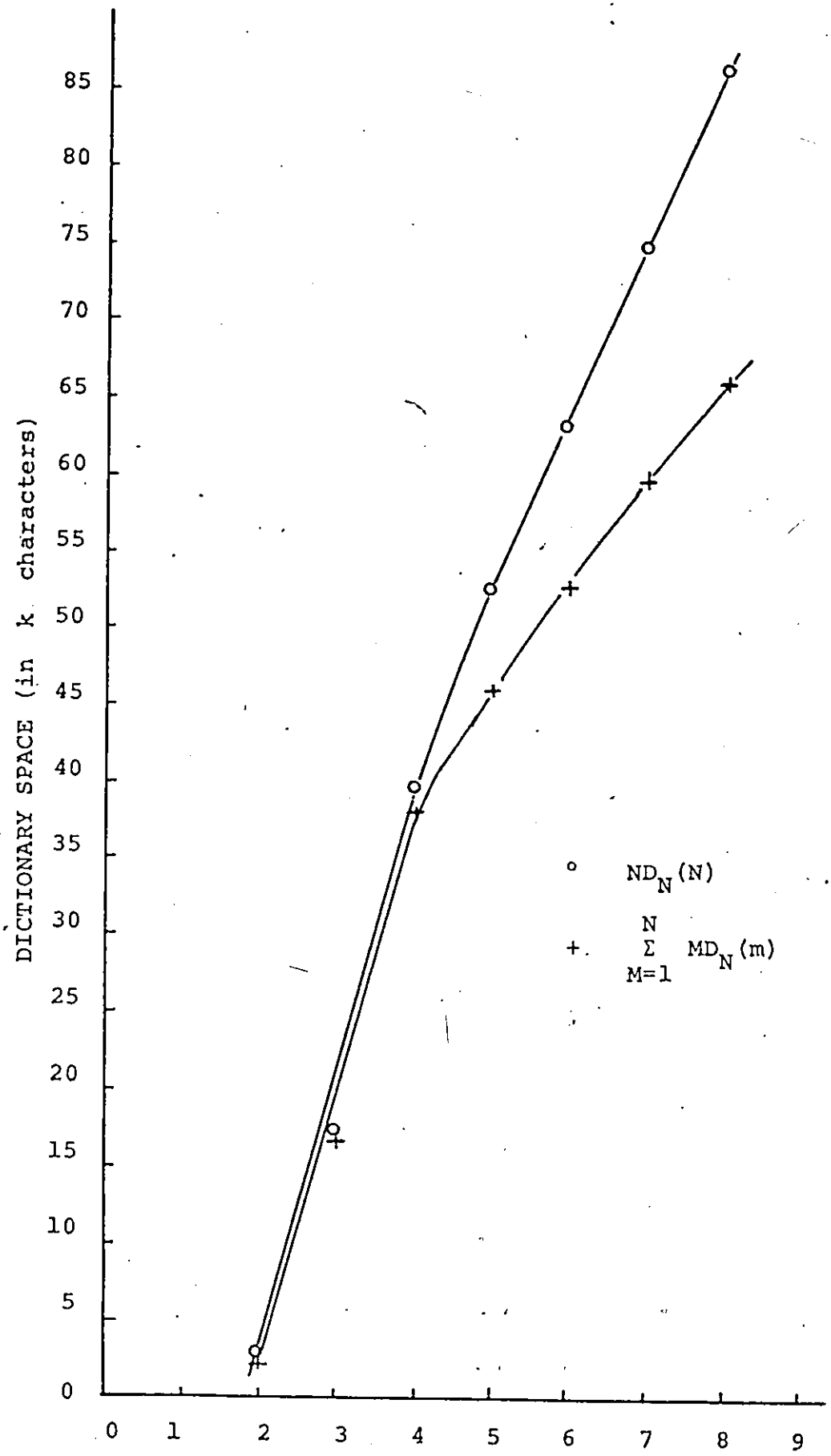


FIG. 3.7 WORD FRAGMENT LENGTH (in characters)

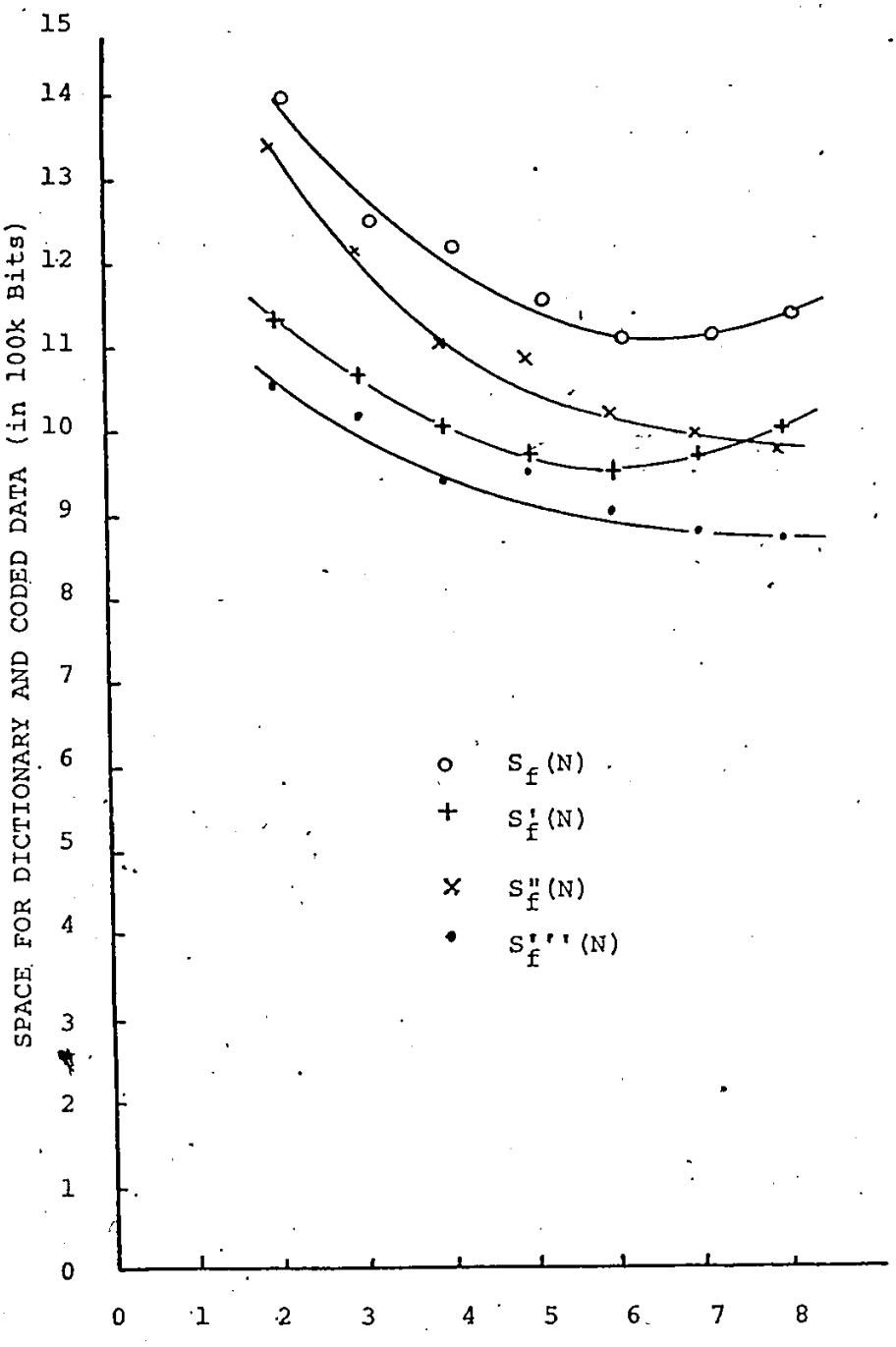


FIG. 3.8 WORD FRAGMENT LENGTH (in characters)

3.4.3 WORDS

So far we have considered coding schemes for text and word fragments representations of a data base. In this section we consider each word as a unit and discuss the storage requirements on the basis of coding schemes for these basic units.

By scanning the entire data base we may form a dictionary consisting of different words of the data base. Based upon the size of the dictionary or on the frequencies of the words in the dictionary, we assign codes to the distinct words in the dictionary.

Let D_w be the number of words in the dictionary, N_w be the number of words in the data base and A_w be average number of characters per word. Since words in the dictionary may be of different lengths, we use a special character (say a blank) as an end of word mark. Thus, the space required to store the dictionary is

$$c\ (A_w + 1)D_w$$

The space required for the coded data base depends on the coding sheme. If we use fixed length coding scheme, then the space is

$$tN_w \quad , \text{ where } \quad t = \lceil \log_2 D_w \rceil .$$

If we use a variable length Huffman coding scheme, then the space used for the coding is

$$H_w N_w \quad , \text{ where } \quad H_w = - \sum_{i=1}^{D_w} P_w(i) \log_2 P_w(i),$$

and $P_w(i)$ is the probability of occurrence of the i^{th} dictionary word in the data base.

Thus, the total space required for the dictionary and coded data base, when fixed length codes are assigned to distinct words is:

$$S_w = c(A_w + 1) D_w + tN_w$$

When we use variable length codes the total space required is

$$S'_w = c (A_w + 1) D_w + H_w N_w$$

Experimental results, obtained by applying this technique to a section of the MARC tapes are shown in Table 3.4.

N_w : Number of words = 34,056

D_w : Number of different words = 10,829

A_w : Average word length = 5.813

S_c : Space for coded data base with fixed length codes = 476,784 bits

S' : Space for coded data base with variable length codes = 374,616 bits

S_d : Space for dictionary = 442,640 bits

S_w : Total space with fixed length codes = 919,424 bits

S'_w : Total space with variable length codes = 817,256 bits

H_w : Average number of bits for variable length coding scheme = 11

$\lceil \log_2 D_w \rceil$: Number of bits for fixed length coding scheme = 14

Table 3.4

Concluding Remarks

In this chapter we have investigated text-fragments, word-fragments and words as codeable elements of a data base. The results of our experiments on MARC tape as a model data base have been given in the preceeding Tables. From the tables we infer that the total space required for dictionary and coded data base using a variable length coding scheme for words is the least among all the other methods. This, however, does not mean that this scheme is suitable for storage of any data base. We also find that the change in the size of the dictionary is considerably more when words are used as codeable elements as compared to using word-fragments as codeable elements. Thus we find that, when the data base remains static over a long period of time, words can be used with advantage as codeable elements. The results indicate that the size of the word-fragment dictionary increases rather slowly and the total space requirement per word-fragment is optimum when the length of the fragment is 6 and variable length codes are assigned.

As has been mentioned in Chapter 1, the number of different words D_w and the total number of words N_w in the data base are related by $D_w = EN_w^B$ where E and B are constants depending on the data base and $0 < B < 1$. This implies that the dictionary size increases with the increase in the size of the data base. The results in Table 3.5 substantiate this.

When we consider word fragments, instead of words as codeable elements, we observe a similar but strictly improved situation. Compared to the increase in the word dictionary size, we find the increase in the number of different word fragments to be considerably smaller. It may be noted that the statistics given in Table 3.5 satisfy a relation similar to the one due to Mandelbrot (1953).

$$D_w(N) = E(N)\,[N_w(N)]^{B(N)} \qquad (3.1)$$

where

N	B(N)	E(N)
2	.188	135.0709
3	.4259	45.4767
4	.6883	4.5476

These results have been obtained using least square fit on the data shown in Table 3.5 and using the formula (3.1). We also observe that B(N) and E(N) are related by

$$\log_e E(N) = c_1 \exp(-c_2\, B(N)),$$

$$c_1 = \exp(2.1434), \quad c_2 = 2.3692$$

TABLE 3.5

RATE OF INCREASE IN WORD FRAGMENTS

N_w	21,187	22,374	34,056
$N_w(2)$	37,172	74,097	124,805
$N_w(3)$	28,644	57,123	87,739
$N_w(4)$	25,062	50,043	69,707
D_w	4,445	7,723	10,829
$D_w(2)$	971	1,127	1,217
$D_w(3)$	3,586	4,875	5,764
$D_w(4)$	4,880	7,663	9,933

CHAPTER 4

VARIABLE LENGTH FRAGMENTS

It has been noted by Booth (1967) that in many textual data bases different words occur only once. However, the total space saving due to compression of infrequent words in the data base is negligible. As mentioned in the introduction of this thesis, and discussed in considerable detail by Scheugraf and Heaps (1973) and Clare et al (1972), it seems advantageous to use particular variable length fragments as codeable elements.

In this chapter we propose an algorithm to discard certain character strings that occur in the data base with low frequencies and will retain only the longest variable length fragments whose frequencies exceed a given threshold, t. The dictionary that stores all the non-overlapping fragments will also contain all members of the character set. In addition, we also store those prefixes whose frequencies lie within a small range of t. This attempts to maximize the average fragment length of the set of fragments in the dictionary and allows the whole data base to be represented as a concatenation of elements of the dictionary. The dictionary is thus 'complete' in the sense that its elements may be concatenated to represent the data base and also any further extensions of it due to update or corrections.

We first informally describe the problem of partioning the data base into variable length fragments of frequencies greater than a given threshold t. Let us suppose that the probabilities $p(c_1)$, $p(c_1\ c_2)$ and $p(c_2/c_1)$ (this is the probability that c_2 immediately follows c_1) have been found experimentally. We denote fragment names by α, their predicted frequencies by $f(\alpha)$ and $K(\alpha)$ their actual frequencies.

Initially we store all single letter fragments and their frequencies in a dictionary. We also store in this dictionary all two letter fragments for which $f(\alpha) > t$. We set $K(\alpha)$ to zero for all α in the dictionary.

The selection process consists of making several passes through the data base, each pass adding fragments to the dictionary. The selection process is terminated if during the previous pass, no more fragments are selected by the procedure. Since identical actions are done in all passes, we explain a single pass through the data base.

The data base is scanned from left to right (starting with the next character to be scanned) until a fragment α of the dictionary can be matched with a string in the data base. At this stage, we compute prediction frequency to decide on the possibility of extending α and thus extracting a fragment of bigger length. This process of scanning the next character,

computing prediction frequency and comparison with the threshold t is continued until we hit a character which if included in the string α would make the prediction frequency smaller than or equal to t. If the fragment α is not already in the dictionary then it is stored in the dictionary and $K(\alpha)$ is incremented by 1. The selected fragment α is considered to be removed from the data base. Repeatedly performing this process, the entire data base can be reduced to an empty set by successive removal of fragments. The reduction of data base to a null set marks the end of one pass through the data base. At the end of each pass, for every α in the dictionary for which the actual frequency $K(\alpha)$ greater than t, the predicted frequency $f(\alpha)$ is set to $K(\alpha)$; for all other fragments the predicted frequency to zero. For all fragments α in the dictionary, $K(\alpha)$ is reset to 0.

We will first describe the data structure used for building variable length fragments and this is done in Section 4.1. We give a formal description of the algorithms to select and generate variable length fragments in Section 4.2. Finally, in Section 4.3, we give specific results concerning variable length fragments selected by the algorithms from a section of the MARC tapes, the storage for the dictionary, and the coded data base.

4.1 DATA STRUCTURE

We use a tree structure for storing fragments during the selection procedure. A tree consists of a finite set of one or more nodes such that there is one specially designated node called the root that has no pointers directed to it, and all other nodes have single pointers directed to them. All nodes may contain any finite number of pointers directed away from them. A node which has no pointer directed away from it is called a terminal node.

A level of a node with respect to the tree is defined by saying that the root has level 1, and the level of any other node is one more than the level of its father. Thus in Fig. 4.1 the nodes L, I, T and A are at level 3 and node U is at level 5. Thus the height of the tree is five.

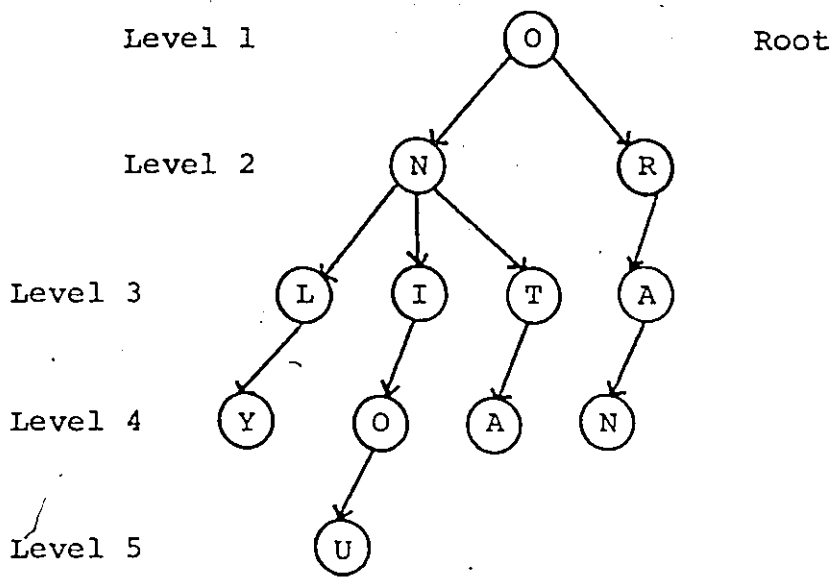


FIG. 4.1 Tree

An ordered set of zero or more disjoint trees is called a forest. Thus, in FIG. 4.1, if root O is removed the tree becomes a forest consisting of two disjoint trees with N and R as roots.

A binary tree consists of a finite set of nodes, in which each node has either 0, 1 or 2 pointers directed from it. In the case where there is one pointer in a node we distinguish between the right and the left pointers. If every node of a tree has 0 or 2 pointers, we call it a pure binary tree.

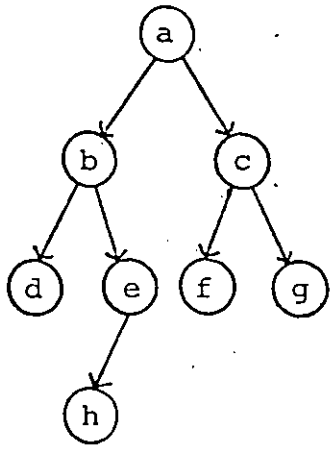


FIG. 4.2 Binary Tree

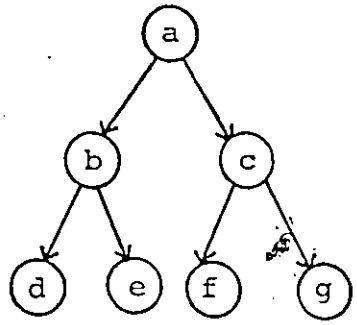


FIG. 4.3 Pure Binary Tree

In the above representation of a tree, the number of pointers per node is variable. This gives rise to variable node size, which may involve extra computational time during implementation of the algorithm which uses this structure. Therefore we transform the tree of Fig. 4.1 into the pure binary tree as shown in Fig. 4.4.

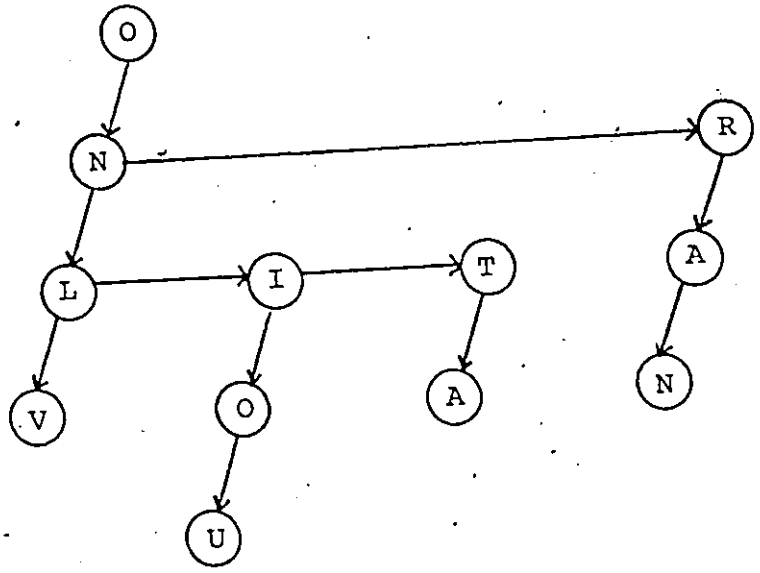


FIG. 4.4 Representation of tree in Fig. 4.1 in Binary tree form

In Fig. 4.4, each node at level X, is connected by a single left pointer to one of its sons at level X-1 to which sons at level X-1 are connected by a single chain of right pointers. This representation enables us to use a fixed length node size.

DEFINITION

Let $p(C_1C_2)$ be the probability of occurance of the character pair C_1C_2 when the data base is partitioned into character strings of length two. The probability that C_2 is immediately following by C_1 is thus:

$$p(C_2|C_1) = \frac{p(C_1C_2)}{p(C_1)}$$

provided $p(C_1) \neq 0$.

If this probability is conditional only on the preceeding character and not on the preceeding string, then the frequency f of a string $C_1C_2C_3$ is given by:

$$f(C_1C_2C_3) = f(C_1C_2)\ p(C_3|C_2) \qquad (3.1)$$

where $f(C_1C_2)$ represents the frequency of the pair C_1C_2.

4.2 SELECTION ALGORITHMS

In this section we describe the algorithms which build up the data structure to store variable length fragments and their frequencies. The algoritm CFQ will set up the root nodes in the forest T1 and store the frequencies of distinct characters. The algorithm PAIR builds up the second level of tree structure to store distinct bigrams and their frequencies in T2. Then the algorithm VLF builds the forest of trees by selecting and storing variable length fragments in T2.

Since the hash function H is 1-1 mapping, a character C_i is not explicitly stored in the root node T1, but we store only the frequency $f(C_i)$ in T1. For example in Diagram 4.1 we show the storage for the four variable length fragments ONLY, ONIOU, ONTA and ORAN during a certain pass of the algorithm VLF. During a subsequent pass, if the fragment ORANG is selected, then a new node will be inserted, as shown in Diagram 4.2.

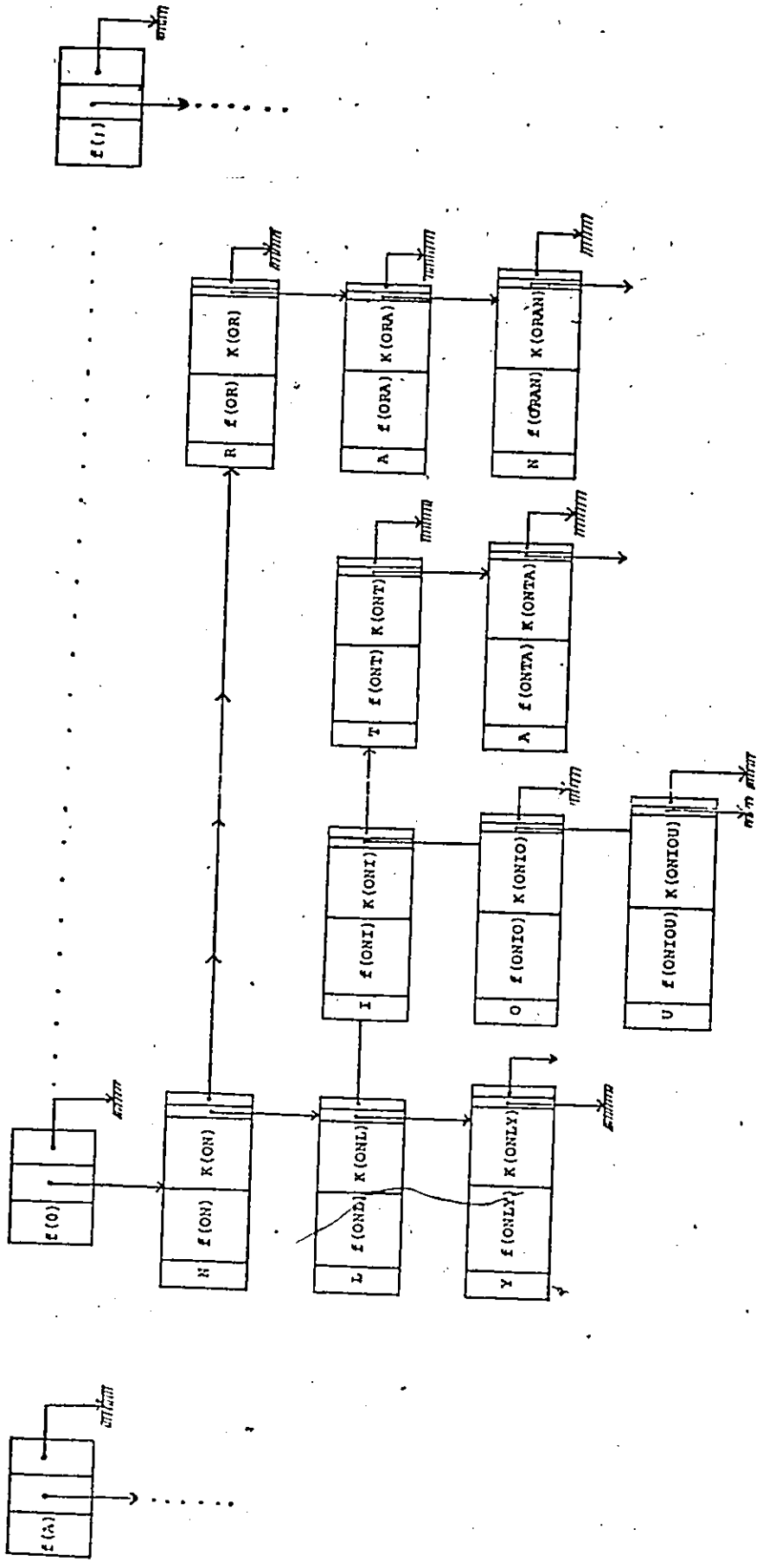

£(1)
R
£(OR)
K(OR)
A
£(ORA)
K(ORA)
N
£(ORAN)
K(ORAN)
T
£(ONT)
K(ONT)
A
£(ONTA)
K(ONTA)
I
£(ONI)
K(ONI)
O
£(ONIO)
K(ONIO)
U
£(ONIOU)
K(ONIOU)
£(0)
N
£(ON)
K(ON)
L
£(ONL)
K(ONL)
Y
£(ONLY)
K(ONLY)
£(A)

DIAGRAM 4.1

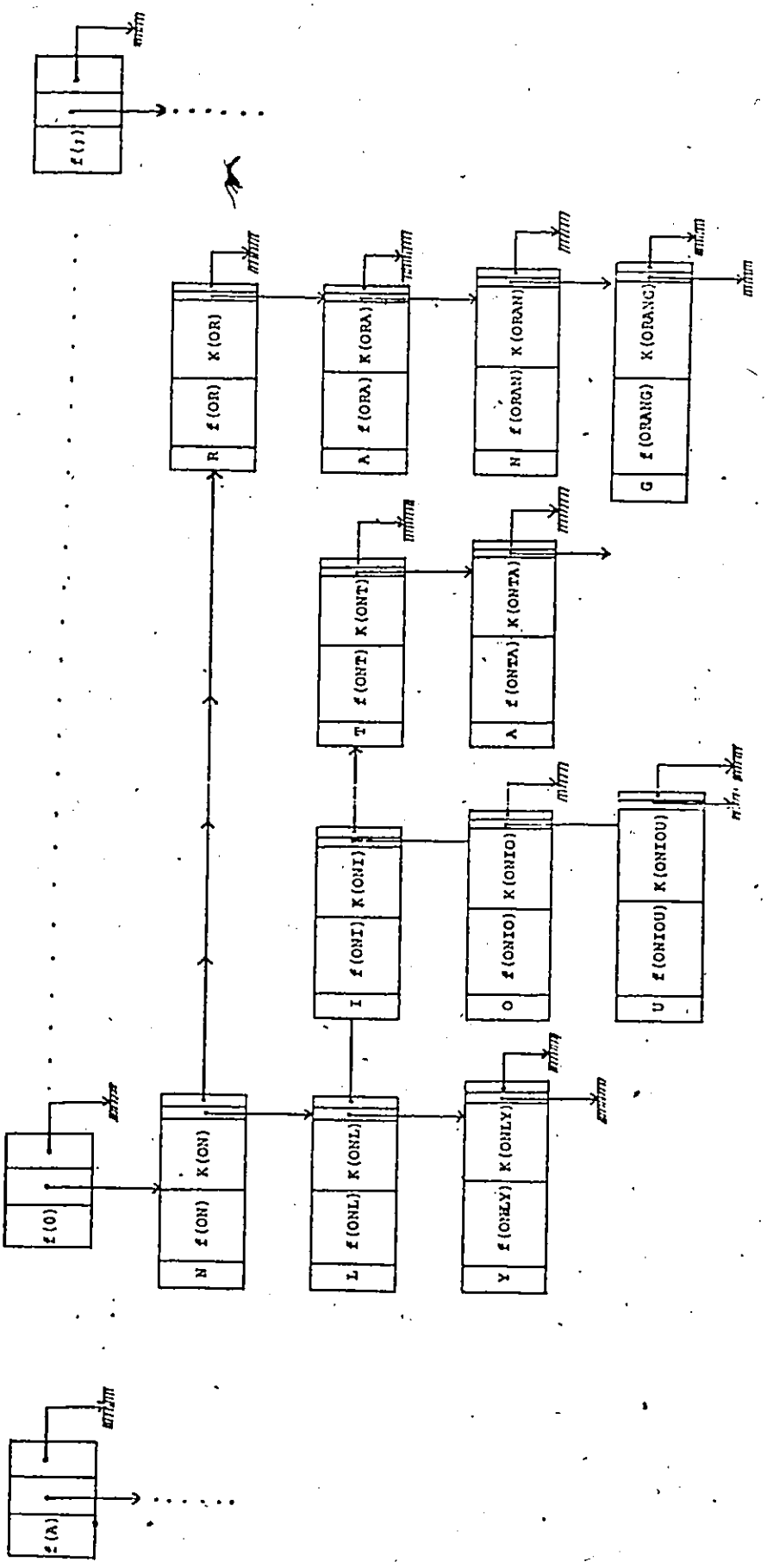

f(1)
R
f(OR)
K(OR)
A
f(ORA)
K(ORA)
N
f(ORAN)
K(ORAN)
G
f(ORANG)
K(ORANG)
T
f(ONT)
K(ONT)
A
f(ONTA)
K(ONTA)
I
f(ONI)
K(ONI)
O
f(ONIO)
K(ONIO)
U
f(ONIOU)
K(ONIOU)
f(0)
N
f(ON)
K(ON)
L
f(ONL)
K(ONL)
Y
f(ONLY)
K(ONLY)
f(A)

DIAGRAM 4.2

Let us assume the existance of a function NEXTCHAR, which when called repeatedly gives the next character from the data base. By invoking this function we compute the frequencies of distinct characters, say, $C_1, \ldots C_m$ in the data base and store the frequencies as in the following algorithm.

ALGORITHM CFQ

Let there be N roots in the forest, whose each root (say the $i^{\underline{th}}$) will have two fields f(i) and DOWN(i) (see diagram 4.1). The field f(i) will contain the frequency of a character C_i and will be used in algorithm VLF; the field DOWN(i) is a pointer to a node of another table T2 (see algorithm PAIR).

Let H be a Hash function, defined as follows:

$$H(C_i) = i$$

where

$$H(C_i) \neq H(C_j)$$

for all distinct characters C_i, C_j in the data base.

STEP 1. (Initialize) Set fields f and DOWN to zero in all roots of the forest.

STEP 2. (Store frequencies of each character in T1:)

For each C_i, $1 \leq i \leq m$

Set f(i) ← frequency of C_i.

Let us assume the existance of a function BIGRAM, which when called repeatedly gives the next two characters from the data base. By calling this function, we compute the frequencies of distinct character pairs, say $\alpha_1, \alpha_2 \ldots \alpha_u$. (eg. $\alpha_i = C_i C_{i+1}$, $\alpha_{i+1} = C_{i+1} C_{i+2}$). We store those character pairs α_i for which $f(\alpha_i) > t$, by using the following algorithm.

ALGORITHM PAIR

Let H be the number of nodes at level two in the forest, where each node will have fields CH, f, K, LLINK and RLINK as shown in Diagram 4.1. The significance of these fields is as explained below:

The field CH stores one character (say C_1); the f field stores the frequency of a fragment αc_1 as computed by the formula (3.1), where α is uniquely determined from the root to father node of this node; LLINK is a pointer to the node which stores a character C_2 such that $\alpha C_1 C_2$ is a fragment selected by the algorithm; the RLINK points to a node which contains a character which is an alternate successor of α and the K field of this node stores the actual frequency of occurrence of αC_1.

STEP 1. Set fields f, K, LLINK, RLINK to zero and CH to blank in all nodes of table T2. Set i ← 0 and AV ← 0.

STEP 2. $i \leftarrow i + 1$, let $\alpha_i = C_r C_s$

$r \leftarrow H(C_r)$

If DOWN(r) $\neq 0$; go to 3.

Otherwise

Set AV $\leftarrow$ AV + 1

DOWN(r) $\leftarrow$ AV

CH(AV) $\leftarrow C_s$

f(AV) $\leftarrow$ frequency of pair α_i;

and go to step 2.

STEP 3. X $\leftarrow$ DOWN(r)

If RLINK(X) $\neq 0$ set

X $\leftarrow$ RLINK(X) and repeat this step.

Otherwise

AV $\leftarrow$ AV + 1

RLINK(X) $\leftarrow$ AV

CH(AV) $\leftarrow C_s$

f(AV) $\leftarrow$ frequency of α_i.

and go to step 2.

ALGORITHM VLF

This algorithm selects the longest fragments with frequencies >t, a given threshold, from a given data base. Let

$$C_1, C_2, \ldots\ldots\ldots\ldots C_N$$

be the data base. Let P(X;Y) denote the probability that character X is immediately followed by Y.

Step 1

(Initialize pointers) Let I and K be pointers to the first character of the data base. Let E be the pointer to the last character of the data base.

Step 2

(Fragment of length > 1)

$p \leftarrow \text{DOWN}(H(C_I))$, If $p = 0$ set $I \leftarrow I + 1$ and repeat this step; otherwise $K \leftarrow I$

Step 3

(End of data base?)

$K \leftarrow K + 1$; If $K > E$, go to step 8.

Step 4

(Find the longest fragment)

If $CH(p) \neq C_K$ then (find match among the alternatives)

If RLINK(p) = 0 then

Set: $L \leftarrow K \leftarrow K - 1$ and go to step 5.

(We have found the longest fragment) Otherwise

Set: $P \leftarrow RLINK(p)$ and repeat step 4.

Otherwise (search next level)

If LLINK(p) = 0 then

Set: $L \leftarrow K$ and go to step 5.

(We have found the longest fragment, otherwise).

Set: P = RLINK(p) and go to step 3.

Step 5

(Find the frequency) Let

$$\alpha = C_I \ldots\ldots\ldots C_L$$

be the longest fragment of the data base just found. Use the algorithm FINDF to find the frequency (FREQ) of the fragment .

Step 6

(The fragment length is already maximum) If

FREQ > t go to step 7

Otherwise let

K ← I ← I + 1 and go to step 2.

Step 7

(Find the longest possible fragment of data base)

$K \leftarrow K+1$, if $K > E$ go to step 8.

$L1 \leftarrow K$, if $(FREQ * P(C_{L1}; C_L)) > t$

Then

$FREQ \leftarrow FREQ * P(C_{L1}; C_L)$

$L \leftarrow L1$

$a \leftarrow aC_L$

Use the algorithm STORE to store and its frequency, and repeat this step.

Otherwise

(Remove the fragment from the data base)

Use the algorithm INCK to increase $K(a)$.

$K \leftarrow I \leftarrow L1$ and go to step 2.

Step 8

(Completion of one pass)

The entire data base has been reduced to a null set removal of successive fragments. The values of $f(a)$ are predictions of the frequencies $K(a)$. Reset for all fragments a in D:

$$f(\alpha) = \begin{cases} K(\alpha) & \text{if } K(\alpha) > t \\ 0 & \text{if } K(\alpha) \leq t \end{cases}$$

$$K(\alpha) = 0$$

Step 9

Repeat step 1 through step 9 until no more fragments can be selected.

ALGORITHM STORE

This algorithm stores a given fragment $\alpha, f(\alpha)$ and $k(\alpha)$ in the dictionary D. Let

$$\alpha = C_i C_{i+1} \ldots\ldots\ldots\ldots C_m$$

Step 1

(Initialize pointers) I and M are the pointers to the beginning and the end of the fragment . i.e.,

$$I \leftarrow i \quad ; \quad M \leftarrow m.$$

(K points to the current character)

$$X \leftarrow \text{DOWN}(H(C_I)) \quad ; \quad K \leftarrow I + 1$$

Step 2

(Is the Fragment found in the dictionary?) If $K > M$, stop ; otherwise go to step 3..

Step 3

(Find a match among the alternatives) If $CH(X) \neq C_K$ and $RLINK(X) \neq 0$, set $X \leftarrow RLINK(X)$ and repeat step 3.

Step 4

(Match is not found in the alternates; hence store C_k as one of the alternates) If $CH(X) \neq C_k$ and $RLINK(X) = 0$, set

$AV \leftarrow AV + 1$ (Pointer to next free location)

$RLINK(X) \leftarrow AV$

$CH(AV) \leftarrow C_k$

$X \leftarrow AV$

and go to step 6. Otherwise go to step 5.

Step 5

(Look at next level)

If $CH(X) = C_k$ and $LLINK(X) \neq 0$

Then let $X \leftarrow LLINK(X)$, $K \leftarrow K + 1$ and go to step 2;

otherwise go to step 6.

Step 6

(Store the rest of the string in D)

For values of J : $K+1, K+2, \ldots, M$

DO

$AV \leftarrow AV + 1$, $LLINK(X) \leftarrow AV$,

$CH(AV) \leftarrow C_j$, $X \leftarrow AV$

Then set: $f(AV) \leftarrow FREQ$
$k(AV) \leftarrow 0$
and terminate

ALGORITHM INCK

This algorithm increases the counter $K(\alpha)$ by 1 of a fragment α stored in the dictionary D.

Let

$$\alpha = C_j C_{j+1} \cdots\cdots\cdots\cdots\cdots C_\ell$$

Step 1

(Initialize pointers) J and L are the pointers to the beginning and end of the fragment α. i.e.,

$$J \leftarrow j \quad ; \quad L \leftarrow \ell$$

(K points to the current character)

$$X \leftarrow \text{DOWN}(H(C_J)) \quad ; \quad K \leftarrow J+1$$

Step 2

(Fragment of length 1) If $K > L$,

$K(H(C_J)) \leftarrow K(H(C_J)) + 1$ and stop;

otherwise go to step 3.

Step 3

(Find a match among the alternates)

If CH(X) $\neq$ C_K, set X $\leftarrow$ RLINK(X) and repeat this step. (Always a match will be found.)

Step 4

(Next character) K $\leftarrow$ K + 1

If K > L (end of fragment α) then K(X) $\leftarrow$ K(X) + 1 and stop; otherwise (look at the next level) X $\leftarrow$ LLINK(X) and go to step 3.

ALGORITHM FINDF

This algorithm finds the frequency of a fragment α stored in the dictionary D.

$$\alpha = C_i C_{i+1} \cdots\cdots\cdots\cdots C_\ell$$

Step 1

(Initialize pointers) I and L are pointers to the beginning and the end of the fragment α. i.e.,

$$I \leftarrow i \quad ; \quad L \leftarrow \ell$$

(K points to the current character)

$$X \leftarrow \text{DOWN}(H(C_I)) \quad ; \quad K \leftarrow I + 1$$

Step 2

(Fragment of length 1) If $K > L$, frag $\leftarrow f(H(C_I))$ and stop ; otherwise go to step 3.

Step 3

(Find match of C_K among the alternates)

If $\text{CH}(X) \neq C_K$, set $X \leftarrow \text{RLINK}(X)$ and repeat this step. (Always a match will be found.)

Step 4

(Next character) $K \leftarrow K + 1$

If $K > L$ (end of fragment α) freq $\leftarrow f(x)$ and stop; otherwise (look at the next level) $X \leftarrow$ LLINK(X) and go to step 3.

4.3 STORAGE REQUIREMENTS

In this section we estimate the storage reqired for the dictionary and the coded data base. Let t be the given threshold.

Case 1. The total dictionary space DIC is partitioned so that the j^{-th} part, called DIC_j, will store variable length fragments of length exactly j. Let $D_t(j)$ and D_t denote the number of fragments in DIC_j and DIC. If $T_t(j)$ and T_t denote the number of fragments of length j and the total number of fragments in the data base, then

$$T_t = \sum_{j=1}^{H} T_t(j)$$, where H is the maximum length of the tree as in algorithm VFL. Thus the total space required to store the dictionary and the coded data is given by (Ref. Ch 3, Sec. 3.4.2).

$$S_v(t) = c \sum_{M=1}^{H} MD_t(M) + \sum_{j=1}^{H} t_j T_t(j) + T_t \lceil \log_2 H \rceil$$

where

$$t_j = \lceil \log_2 D_t(j) \rceil$$

and $\lceil \log_2 H \rceil$ represents the number of bits required to give a unique code to H dictionaries.

Case 2. In this case, we store all variable length fragments in one dictionary. A special character (say a blank) is used to mark the end of each fragment in the dictionary DIC. Thus the number of bits required to store the dictionary and the coded data base is:

$$S_v'(t) = c \sum_{M=1}^{H} (M+1) D_t(M) + T_t \lceil \log_2 D_t \rceil$$

TABLE 4.1 SAMPLE 1

LENGTH	NO. OF WORDS	NO. OF DIFF. WORDS
1	347	20
2	2548	164
3	2403	503
4	1809	510
5	2203	748
6	1953	801
7	1507	742
8	1561	1029
9	936	562
10	893	500
11	737	279
12	278	199
13	245	113
14	79	54
15	140	48
16	31	31
17	18	17
18	52	50
19	25	24
20	20	15
21	11	11
22	3	3
23	3	3
24	3	2
25	1	1
27	1	1

Total number of words = 17,807

Total number of different words = 6,430

TABLE 4.2 SAMPLE 2

LENGTH	NO. OF WORDS	NO. OF DIFF. WORDS
1	347	18
2	2168	172
3	2231	491
4	1817	567
5	2066	684
6	1661	729
7	1352	725
8	1593	1031
9	847	527
10	773	451
11	663	252
12	218	153
13	213	88
14	69	54
15	120	30
16	19	17
17	8	7
18	37	33
19	24	24
20	10	9
21	4	4
22	3	3
23	1	1
24	3	3
25	1	1
27	1	1

Total number of words = 16,249

Total number of different words = 6,075

VARIABLE LENGTH FRAGMENTS

TABLE 4.3 SAMPLE 1, t = 30

LENGTH	NUMBER OF FRAGMENTS	DIFFERENT FRAGMENTS
1	10,111	47
2	25,203	444
3	11,819	440
4	1,145	31
5	607	8
6	60	1

TABLE 4.4 SAMPLE 1, t = 50

LENGTH	NUMBER OF FRAGMENTS	DIFFERENT FRAGMENTS
1	12,837	47
2	28,466	361
3	9,152	243
4	1,081	13
5	408	3
6	60	1

VARIABLE LENGTH FRAGMENTS

TABLE 4.5 SAMPLE 2, t = 30

LENGTH	NUMBER OF FRAGMENTS	DIFFERENT FRAGMENTS
1	9651	47
2	22,695	426
3	10,500	384
4	987	24
5	649	9
6	45	1

TABLE 4.6 SAMPLE 2, t = 50

LENGTH	NUMBER OF FRAGMENTS	DIFFERENT FRAGMENTS
1	13,247	47
2	25,736	328
3	7,329	198
4	1,092	15
5	532	6
6	45	1

TABLE 4.7

THRESHOLD (t)	FRAGMENTS		SPACE IN BITS		
	TOTAL	DIFFERENT	DICTIONARY	CODED DATA	TOTAL SPACE
30	48,945	971	14,550	548,245	562,795
50	52,004	668	9,426	567,584	577,010
30	44,527	891	13,188	497,773	510,961
50	47,981	595	8,353	519,645	528,003

SPACE REQUIREMENT USING DIFFERENT DICTIONARIES

TABLE 4.8

THRESHOLD (t)	FRAGMENTS		SPACE IN BITS		
	TOTAL	DIFFERENT	DICTIONARY	CODED DATA	TOTAL SPACE
30	48,945	976	17,754	489,450	507,204
50	52,004	668	13,422	520,040	533,462
30	44,527	891	17,886	445,270	463,156
50	47,981	595	11,928	479,810	491,738

SPACE REQUIREMENT USING ONE DICTIONARY

4.4 CONCLUSIONS

We have used two different samples of MARC tapes for our experiments. The algorithms of section 4.2 were applied to each sample of the data base in order to select variable length fragments, and the storage space was determined when all the fragments of different lengths are kept in one dictionary. The space requirements when fragments of different lengths are accomodated in different dictionaries have been determined. The results may be sumarized as follows:

The first sample consists of 17,807 words of which 6,430 are different (Refer Table 4.1). If these words are coded by assigning codes to each different character, the space required for the dictionary and the coded data is 715,530 bits. The second sample consists of 16,249 words of which 6,075 are different. In this case the space required to store the dictionary and the coded data base when single characters are used as codable elements is 661,800 bits (Refer Table 4.2).

For the threshold value 30 and 50 the total number of fragments and the total number of different fragments are shown in Tables 4.3 to 4.6. The different fragments shown in these tables form a 'complete' dictionary. Table 4.7 shows the storage required to store the coded data base and the dictionaries to store fragments of variable sizes

(Refer Case 1). From this we observe that for sample 1 and t = 30, the total storage is 562,795 (in bits) and for t = 50, the storage required is 577,010 bits. Similarly for sample 2, the storage required for t = 30 is 510,961 bits and for t = 50, is 520,003 bits.

Table 4.8 gives the statistics on the storage requirement when fragments of variable length are stored in a single dictionary (Refer Case 2). We observe that for sample 1 and t = 30 the total storage is 507,204 bits and for t = 50 the storage required is 533,462 bits. Similarly for sample 2 the total storage required for t = 30 is 463,156 bits and for t = 50 is 491,738 bits. It is clear from Tables 4.7 and 4.8 that for a given threshold value and for a given sample, the storage of variable length fragments in a single dictionary requires less storage than the storage of variable length fragments in different dictionaries.

The amount of saving in storage achieved by using one dictionary as opposed to different dictionaries has been computed for both the samples and the results may be summarized as follows:

1) If different dictionaries are used for fragments of variable lengths, there is 21.35% of saving for sample 1 with t = 30, while a saving of 19.36% is achieved for t = 50. However for sample 2, with t = 30 there is 22.8% of savings and a saving of 20.22% is avhieved with t = 50.

2) If all fragments are stored in one dictionary, the savings achieved for sample 1 and t = 30 is 29.11%, while a saving of 25.45% is achieved when t = 50. Similarly for sample 2 and t = 30 there is a saving of 30% in storage, while for t = 50 the saving is 25.7%.

We remark that the amount of storage gained for sample 1 by the use of a single dictionary, as opposed to use of different dictionaries for fragments of different lengths, varies between 6.09% and 7.76%; while for sample 2, the net percentage gained is bewteen 5.48 and 6.2. Thus the space required to store the data base is less when fragments are used as codable elements than when single characters are used as codable elements.

CHAPTER 5

CONCLUSIONS

CONCLUDING REMARKS

We have investigated compression techniques for a data base using fragments as codable elements. The experimental data base consists of bibliographic information obtained from a section of MARC tapes issued by the Library of Congress. Word fragments which are substrings of words as well as text fragments which are strings possibly extending over word boundaries are considered as codeable elements.

In chapter 3 the advantages of using fragments instead of words as codable elements are discussed. It has been shown that the word dictionary size is much larger than the size of fragment dictionary. In the case of fixed length fragments, the dictionary size increases as N, the fragment length, increases. For $N \leq 3$, the dictionary size is much smaller than word dictionary and can be kept in core. It is noticed that fixed length fragment dictionary increases as data base increases. But the rate of increase is much smaller for N=3 than the rate of increase for a word dictionary. Hence the fragment dictionary can be kept in core for fast coding and decoding.

The fixed length fragments selected from the data base are assigned either fixed length or variable length codes. It is experimentally found that when fragments as opposed to characters are used as codable elements, the amount of compression achieved is 36.18% when the fragment length N=8 and 24.66% when N=3. Since the dictionary size for N=3 is much smaller than that for N=8, and hence can be kept in core for fast decoding. It is concluded that both economy of storage (24.66% compression) and speed of retrieval is achieved for N=3.

Use of particular variable length fragments selected to have frequencies greater than a given threshold t has been explained in chapter 4. Algorithms for selection of fragments having this property are also described. It is found that the dictionary size is always smaller than the word dictionary size and hence can be kept in central memory. It is also found through experiments that the total space required to store dictionary and the coded data base changes by a small amount as the threshold value t changes. The choice of t which is optimal in terms of total storage requirement is not obvious from our results. It is felt that the choice of t must be left to the designer , since this may depend on other characteristics of the data and system resources.

RECOMMENDATIONS

The use of fragments in information retrieval has been the subject of study for several researchers. The present study has treated only one of several aspects, i.e. using fragments codable elements for data compression. Despite the limited scope of the present investigation, the study has led to several questions which are worthy of further study.

The only input to the algorithm which generates variable length fragments is t, the threshold. It is found in our investigation that the total space used for dictionary and coded data base is a function of t. However it is not obvious how the change in t affects the total storage. Thus we can ask, given s, the total storage space available, what is the choice of t so that the dictionary and the coded data base requires at most s amount of storage? Another question is, what choice of t will give rise to maximum compression (i.e. minimal storage for dictionary and coded data?)

How will the addition deletion of records in a data base affect the distribution of fragments? We have found that additions to a data base increase the number of different fragments, but at a much slower rate than the increase in the number of different words.

Is it possible to have hierarchical storage organization of fragments which can be used for efficient retrieval purposes? If so then fragments are useful both for compression as well as for retrieval.

REFERENCES

[1] Barton, I.J., Creasey, S.E., Lynch, M.F., Shell, M.J., "An Information Theoretic Approach to Text Searching in Direct Access Systems," Private Communication, M.F. Lynch.

[2] Booth, A.D., "A 'Law' of Occurrence for Words of Low Frequency," Inf. and Contr. Vol. 10, (1967), pp. 386-393.

[3] Bourne, C.P., Ford, J.E., "A Study of Methods for Systematically Abbreviating Englush Words and Names," JACM, Vol. 8, (1961), pp. 538-552.

[4] Bryne, J.G., Mullarney, A., "A Survey of Text Compression," Paper presented IV Cranfield Intl. Conf. on Mech. Inf. Storage and Retrieval Systems, (July 1973).

[5] Cardenas, A.F., "Evaluation and Selection of File Organization. A Model and System," Comm. ACM, Vol. 16, (1973), pp. 540-548.

[6] Cardenas, A.F., "Analysis and Performance of Inverted Data Base Structures," CACM, Vol. 18, No. 5, (May 1975), pp. 253-263.

[7] Chapin, N., "A Comparison of File Organization Techniques," Proc. 24th ACM National Conference, (1969), pp. 273-283.

[8] Clare, A.C., Cook, E.M., Lynch, M.F., "The Identification of Variable Length, Equifrequent Character Strings in a Natural Language Data Base," Computer J. Vol. 15, (1972), pp. 259-262.

[9] Collmeyer, A.J., Schemer, J.E., "Analysis of Retrieval Performance for Selected File Organization Techniques," Proc, Fall Joint Computer Conf. (1970), pp. 201-210.

[10] Colombo, D.S., and Rush, J.E., "Use of Word Fragments in Computer-Based Retrieval Systems," J. Chem. Doc., Vol. 9, No. 1, (Feb. 1969), pp. 47-50.

[11] Dodd, G.G., "Elements of Data Management," Computing Survey (June 1969), pp. 117-133.

[12] Dolby, J.L., "An Algorithm for Variable Length Proper Name Compression," Jr. Library Automation, Vol. 3, (1970), pp. 257-275.

[13] Gottlieb, D., Hagerth, S.A., Lehot, P.G.H., and Rabinowitz, H.S., "A Classification of Compression Methods and Their Usefulness for a Large Data Processing Center," Nat. Comp. Conf., (1975), pp. 453-458.

[14] Heaps, H.S., "Storage Analysis of a Compression Coding for Document Data Bases," INFOR, Vol. 10, (1972), pp. 47-61.

[15] Heaps, H.S., and Thiel, L.H., "Optimization Procedures for Economic Information Retrieval," Inform. Stor. Retr. Vol. 6, (1970), pp. 137-153.

[16] Huffman, D.A., "A Method for Construction of Minimum Redundancy Codes," Proc. IRE, Vol. 40, (1952), pp. 1098-1101.

[17] Information Systems Office, Library of Congress MARC Manuals, used by the Library of Congress Information Science and Automatic Division, American Library Assoc., Chicago, (1970).

[18] Lefkovitz, D., File Structure for On-Line Systems. Spartan Books, New York, (1969).

[19] Lowe, T.C., "Direct Access Memory Retrieval Using Truncated Record Names," Software Age, Vol. 1, (Sept. 1967), pp. 28-33).

[20] Lowe, T.C., "The Influence of Data Base Characteristics and Usage on Direct Access File Organization," JACM, Vol. 15, No. 4, (1968), pp. 535-548.

[21] Lowe, T.C., "Effectiveness of Retrieval Key Abbreviation Schemes," J. ASIS, Vol. 22, (1971), pp. 374-381.

[22] Lynch, M.F., Petrie, J.H., and Snell, M.J., "Analysis of the Microstructure of Titles in the INSPEC Data-Base," Inform. Stor. Retr., Vol. 9, (1973), pp. 331-337.

[23] Mandelbrot, B.M., An Informational Theory of the Statistical Structure of Language, Proc. Symp. Appl. Communication Theory, W. Jackson, Ed., Butterworth, London, (1953).

[24] Nugent, W.R., "Compression Word Coding Technique for Information Retrieval," J. Library Automation, Vol. 1, (1968), pp. 250-260.

[25] Rickman, J.T., Gordner, H.W., "On-Line Index Perm Predictions Using Bigran-Term Association," Proc. ACM Nat. Conf., (1973), pp. 262-268.

[26] Ruth, S.R., Villers, J.M., "Data Compression and Data Compaction," Tech. Rep. AD 723525, (1972).

[27] Ruth, S.R., Kneutzer, P.J., "Data Compression of Large Business Files," Datamation, Vol. 18, (Sept. 1972), pp. 62-66.

[28] Salton, G.A., "Computer Evaluation of Indexing and Text Processing, "J. ACM, Vol. 15, (1968), pp. 8-36.

[29] Salton, G.A., Automat: Information Organization and Retrieval. McGraw-Hill, New York, (1968).

[30] Schieber, W.D., Thomas, W.G., "An Algorithm for Compaction of Alphanumeric Data, "J. Library Automation, Vol. 4, (1971), pp. 198-206.

[31] Schipma, P.B., "Term Fragment Analysis for Inversion of Large Files," Paper presented at the Assoc. of Sci. Inf. Centre Meeting, Washington, D.C. (Feb. 1971).

[32] Schuegraf, E.J., "The Use of Equifrequent Fragments Retrospective Retrieval Systems. Ph.D. Thesis, Univ. of Alberta, Edmonton, Canada, (1974).

[33] Schuegraf, E.J., Heaps, H.S., "Selection of Equifrequent Word Fragments for Information Retrieval," Inf. Stor. Retr., Vol. 9, (1973), pp. 697-711.

[34] Schuegraf, E.J., and Heaps, H.S., "A Compression Algorithms for Data Base Compression by Use of Fragments as Language Elements," Inform. Stor. Retr.

[35] Schwartz, E.S., and Kleiboemer,A.J., "A Language Element for Compression Coding," Info. and Control, Vol. 10, (1967), pp. 315-333.

[36] Schwartz, E.S., "A Dictionary for Minimum Redundancy Coding," J. ACM, Vol. 10, (1963), pp. 413-439.

[37] Snyderman, M., Hunt, B., "The Myriad Virtues of Text Compaction," Datamation, Vol. (Dec. 1970), pp. 36-40.

[38] Thiel, L.H., and Heaps, H.S., "Program Design for Retrospective Searches on Large Data Bases," Inf. Stor. Retr., Vol. 8, (1972), pp. 1-20

[39] Wagner, R.A., "Common Phrases and Minimum Space Text," Comm. ACM, Vol. 16, No. 3, (March 1973), pp. 148-152.

[40] White, H.E., "Printed English Compression by Dictionary Encoding," Proc. IEEE, Vol. 55, No. 3, (March 1967), pp. 390-396.

[41] ZIPF, G.K., Human Behaviour and Principle of Least Effort; Addison-Wesley, Cambridge, Massachusetts, (1949).