# THE RESEARCH OF DATA COMPRESSION ON IDMS *

*Nan-Wen Lee*[1]


## ABSTRACT

Many data processing applications involve storage of large volumes of alphabetic data such as names, addresses and numeric data such as climatic data, telephone numbers, salaries. In order to reduce the data storage requirements and the data communication costs, data compression technique is needed.

The data compression technique employed here is a modified Huffman coding method. Repeating types of characters are taken into consideration to reduce the redundancy in the data representation.


## 1. INTRODUCTION

### 1.1 Database Software Packages

Numerous universities and corporations are utilizing central databases with general purpose database software. The two software packages which are most commonly used are IMS, an IBM Corporation product, and IDMS, a Cullinet Corporation product. Both of these software packages run on large general purpose IBM or IBM compatible main frame computers. IMS is a hierarchical database system, and IDMS is a network database system with some relational features. Although these software packages are more efficient on general purpose computers than proposed relational systems, they both have very significant central processing unit requirements, have massive disk storage requirements, and move information to and from disk storage frequently and in large amounts.

Both the IMS and IDMS software packages have data compression and expansion programs. When data is stored in the database, the compression program is called. When data is retrieved from the database, the data is expanded. While both programs must be efficient, compression is usually done during batch loads of data with no effect on response time at terminals. However, expansion occurs during user access to the database from terminals so the expansion program must be very efficient.

### 1.2 Proposed Algorithm for Improved Database Compression

In 'Data Compression On A Database System', an August 1985 article in the ACM Communications, Gordon V. Cormack of the University of Manitoba described data compression and expansion algorithms which he believed to provide impressive data compression on central

*收稿日期：75年7月11日 送審日期：75年7月12日

1中央氣象局資料處理科技士

databases stored under IMS. Programs based on these algorithms have been used at several universities with significant improvement in data compression over the programs previously used with IMS.

### 1.3 Purpose of The Project

The purpose of this project is to analyze the data stored under IDMS and then write programs for these algorithms which can replace the data compression and expansion programs currently used under IDMS. The improvement in data compression for IMS resulted from the observation that data in the database tended to consist of long strings of the same type of character. For example, names and titles are long strings of alphabetic characters and numerical data is generally grouped. While Huffman coding is optimal with respect to average compression, and other ad hoc techniques take advantage of repeating strings of the same character, techniques currently used do not reflect repeating characters of the same type.

To carry out the project it was necessary to have a general understanding of Huffman coding since the data compression algorithm used in this project is a modified form of Huffman coding. A general understanding of IDMS was required in order to replace the currently installed programs. Programs were written to analyze data and determine the compression codes to be assigned to characters. These programs were tested and debugged using sample data. Then actual IDMS data was processed by the programs to determine the effectiveness of the data compression. Then data compression and expansion programs were written in IBM assembler.

Unfortunately, at this time there is very little data in the production IDMS system at The University of Wyoming. It was expected that the Human Resource System and Student Information System would be installed before the project was completed. But these installations were delayed. The amount of data should be adequate when the Human Resource and Student Information System are installed. When this occurs, the analysis program will again be run against the actual data by Computer Services staff. If the proposed algorithms provide more significant improvements in compression of the actual data, the currently used IDMS compression and expansion programs will be formally replaced by the IBM Assembler programs developed for this project. The code corressponding to the algorithms in the IBM Assembler programs can be used as is.

## 2. TYPES OF DATA COMPRESSION

Two techniques that can result in a more effective encoded data representation are logical and physical data compression. Both of them can result in reduced transmission time and a reduced storage requirement.

### 2.1 Logical Compression

Logical compression is utilized when the database software is designed. The software should be designed to reduce the amount of space occupied by redundant characters, and user transparent techniques should be used for condensed representation of data elements.

If data in logically compressed data

bases are transmitted between locations, transmission time will be reduced since fewer data characters are transmitted. While logical compression is an effective tool in minimizing the size of a data-base, it only reduces transmission time when logically compressed data is transmitted.

Logical compression is normally used to represent data bases more efficiently and the frequency of occurrence of characters or groups of characters is not considered.

The following example illustrates the use of logical compression when the database software is designed.

In a climatic data set, a 'date' field exists in a record, then we can use various notation to represent 01 April 1986:

(a) 01 april 1986 (character representation)
(b) 01 apr 1986 (character representation)
(c) 01 04 1986 (character representation)

We know the maximum value for DAY is 31, and 5 bits are enough to represent it; the maximum value for MONTH is 12, and 4 bits are enough to represent it. If we use 7 bits to represent YEAR, then a total of 127 years can be represented and relative years ranging from 1900 to 2027 are permitted. We can see that the use of binary strings to represent 'date' is very economical in use of space.

### 2.2 Physical Compression

Physical compression consists of encoding characters using fewer bits than the standard computer or data communications bit representations. Typically, this has been done to reduce the quantity of data prior to entering a transmission medium and the expansion of such data into its original format upon receipt at a distant location.

Physical compression takes advantage of the fact that when data is encoded as separate and distinct entities, the probabilities of occurrence of characters and groups of characters differ. Since frequently occurring characters are encoded into as many bits as those characters that only rarely occur, data reduction becomes possible by encoding frequently-occurring characters into short bit codes while representing infrequently-occurring characters by longer bit codes.

The use of physical data compression for disk storage reduces the amount of disk storage that must be purchased. Also, fewer disk accesses and lower data transfer to and from disk storage is a result. There are storage and processing costs associated with the programs which must be compared with the savings which result from the compression.

## 3. DATA COMPRESSION TECHNIQUES

The first interest in character compression techniques resulted from transmission of characters over communications lines. These techniques increased the amount of information which could be transmitted in a given period of time. A number of techniques have been employed to compress characters for transmission over communications lines. Null suppression (replacement of most of the null characters or blanks by a special ordered pair of

characters), bit mapping (a bit map, which is appended in front of an input string, can indicate the presence or absence of nulls and thereby can reduce the size of the string) and statistical encoding are three of these compression techniques.

Statistical encoding takes advantage of the probabilities of occurrence of single characters and groups of characters, so that short codes can be used to represent frequently occurring characters or groups of characters while longer codes are used to represent less frequently encountered characters and groups of characters.

Huffman coding is a statistical encoding method which was described in 1952. Huffman showed that given probabilities of occurrence of a set of characters, the average code length is minimized. As the use of computers and external storage increased, Huffman coding was applied to storage of characters in computer external devices. Other techniques for specialized data have evolved over the years.

### 3.1 Huffman Coding

The Huffman coding technique is optimal in that it results in a minimum average code length for a given set of characters. In addition, Huffman codes have a prefix property which means no short code group is duplicated as the beginning of a longer group. For example, if one character is represented by bit string 101, then there is no other character which will be represented by 10100 since 10100 will be interpreted as the combinations of 101 and 00. Because of this special feature, the compressed data stream (bits combinations) can be decoded immediately by tree node traversal if we read from left to right without examining the entire block of input data.

To use Huffman coding, each occurrence of a character is counted in order to determine the probabilities of occurrence of each character in the data.

We arrange the characters in descending order of probabilities. The two nodes with the smallest probabilities are paired to produce another node whose probability is the sum of the probabilities of paired nodes.

Before the next pairing, sorting of the new probabilities is needed to ensure the probabilities have descending order. Then two adjacent nodes with the lowest probabilities of occurrence are again paired. By repeating this step, we finally get a node whose probability of occurrence is 1.

By repeating the assignment of binary 0 and 1 to paired nodes at each pairing step, the codes for each character can be obtained by tracing back from the last node to the starting nodes.

Huffman code can be generated through the employment of a tree structure as illustrated in the following two figures:

| Character Set | Initial Probability | 1st Step | 2nd Step | 3rd Step |
|---|---|---|---|---|
| C1 | .35 | C1 .35 | C1 .35 | C1 .35 |
| C2 | .15 | C2 .15 | C2 .15 | C5678 .22 |
| C3 | .15 | C3 .15 | C3 .15 | C2 .15 |
| C4 | .13 | C4 .13 | C4 .13 | C3 .15 1 |
| C5 | .12 | C5 .12 | C5 .12 1 | C4 .13 0 |
| C6 | .06 | C6 .06 1 | C678 .10 0 | |
| C7 | .03 1 | C78 .04 0 | | |
| C8 | .01 0 | | | |

| 4th Step | | 5th Step | | 6th Step | |
|---|---|---|---|---|---|
| C1 | .35 | C25678 | .37 | C134 | .63 1 |
| C34 | .28 | C1 | .35 1 | C25678 | .37 0 |
| C5678 | .22 1 | C34 | .28 0 | | |
| C2 | .15 0 | | | | |

From the pairing operation above, Huffman codes and lengths for C1, C2, C3, . . . C8 are:

| character | codes | length |
|---|---|---|
| C1 | 11 | 2 |
| C2 | 00 | 2 |
| C3 | 101 | 3 |
| C4 | 100 | 3 |
| C5 | 011 | 3 |
| C6 | 0101 | 4 |
| C7 | 01001 | 5 |
| C8 | 01000 | 5 |

Huffman code for a particular character is selected so that its length is as close as possible to f ( –log p), where the base is 2, p is the probability of occurrence of that character in the text, and f(x) is the closest integer greater than or equal to x.

### 3.2 A Modified Huffman Coding Technique

The central databases stored under the IMS and IDMS databases typically have data items which predominantly group the same types of character. Names, titles and addresses consist mainly of alphabetic characters. Salaries, social security numbers, and financial data are primarily numeric characters.

For the project the Huffman coding technique is modified to take these repeating types of characters into consideration. Four data types were used, alphabetic, numeric, blank and other symbols. The data types used were intuitively selected. As experience with the actual data is acquired, it may be advantageous to change the data types. For each data type, a code is assigned to every kind of character appearing in the data. To compress a record, an alphabetic type code is assigned to the first character. Then the first character is checked to determine which type the first character actually was. Then the second character is assigned a code for the actual type of the first character. This process is repeated until the entire record has been compressed.

To expand a compressed record, it is known that the first character is a code of the alphabetic type. The code bits of the first character are used to determine, from the alphabetic type codes, what the first character is. The first character is used to determine the code type of the second character. The code bits of the second character are used to determine what the second character is. Then the code bits of the second character are used to determine the code type to be assigned to the third character. This process is continued until the entire record is expanded.

Modified Huffman coding is almost the same as Huffman coding, the only difference being the assignment of codes to characters for several different types. For the modified Huffman Coding technique, we count the number of times a character is followed by an alphabetic character, and for each total obtained we divide by the total number of characters in the data. This yields probabilities for

the alphabetic type. This process is repeated for each type, so that we have probabilities for each type. Then for each type, character bit codes are determined. The detailed process for this will be described in another section of the paper.

## 4. DATA ANALYSIS

### 4.1 Input

Data was provided in a sequential file with fixed length records which were unloaded from the data base. Each record contains a variable number of characters. This file is processed to generate a new sequential file in which each record has the number of characters followed by the characters to be analyzed.

### 4.2 Analysis Process

The main program includes 5 subprograms. The Frequency Analysis and Counts tables are generated by the main program. The Length table is produced by the subprogram 'leng'. The Codes table is produced by the subprogram 'codex'. The program 'sorting' is called for sorting of the counts before the Counts table is printed out. The program 'reseq' is called by program 'leng' for frequency sorting. The program 'rever' is called by program 'codex' for probability-sum sorting before the modified Huffman code is determined.

In the text analysis program (main program), we count the

(a) Total number of characters, i.e. number of characters in the text.

(b) Occurrence of each character in the test data file.

(c) The percentage of each character-occurrence in the appropriate data type and in the whole test data file, i.e. the percentage of occurrence 'A' for the alphabetic type, and in the whole text.

(d) The total number of occurrence of each character following a character of each data type.

This yields the probabilities of occurrence of each character of each data type by appropriate division operations.

The tables produced by this process are omitted in this paper.

The first character of each input record is always regarded as an alphabetic data type.

Applying the Huffman coding method, we determine Huffman codes for each character in the test data file. Then we count the length of each code. Actually, Huffman codes are not required for characters because we are going to use proposed coding method to generate compressed codes. However, we need those codes to calculate code lengths.

Note that codes may be different for the same character if the types are different.

The codes generated are stored in compression-code tables. Each data type has an associated table. The characters are sorted with the longest length first, and the shortest last. Initial table values are formed using the rule below:

( cl means the first character with the longest length, ℓ1 means the length of the character with longest length (cl) )

( cn means the last character with the shortest length, ℓn means the length of the character with shortest length (cn) )

| | | |
|---|---|---|
| c1 | ℓ1 | ℓ1 1's followed by (15–ℓ1) 1's. |
| c2 | ℓ2 | (first ℓ2 bits of previous value) – 1, then followed by |

(15–ℓ2) 1's.

cn ℓn (first ℓn bits of previous value) – 1, then followed by (15–ℓn) 1's.

Then the final table value is formed by shifting each of them (15–ℓi) times to the right, filling with zeros from left. Then a one is inserted in position (16–ℓi) from the right.

As an example:

| character | code-length | initial code (15 bits) | final code (16 bits) |
|---|---|---|---|
| T | 8 | 111111111111111 | 0000000111111111 |
| B | 6 | 111110111111111 | 0000000001111110 |
| A | 6 | 111101111111111 | 0000000001111101 |
| L | 5 | 111011111111111 | 0000000000111101 |
| J | 3 | 110111111111111 | 0000000000001110 |
| P | 2 | 101111111111111 | 0000000000000110 |

## 5. COMPRESSION

The code table are utilized in the compression program. Given a string of characters to be compressed, the alphabetic table is used to compress the first character (the first character of each record is always regarded as alphabetic data type) via character linear searching, and the subsequent characters are compressed may use another code table as determined by the data type of the previous character. Each character is represented by a 16-bit string in code table. The compressed code used in the compression process is contained in the rightmost several bits. Preceding each compressed code is a bit with value 1 (this 1 functions as the delimiter), and the rest of the bits are all padded with binary value 0's. The length of each actual code and the code itself are determined by following steps.

Determine the code length of a compressed character:

1. Set comparison value (i.e. comparator) cv to be 0000000000000100 (binary representation).
2. If the table value for a compressed character is v, compare cv with v.
3. If cv is greater than v, then the count is the length of the code to be used. Stop.
4. If cv is not greater than v, then add one to a count which is initially set to 1. Double cv. Go to 2.

The shift instruction and mask instructions are used to move compressed codes into output buffer, and then the number of compressed bits are moved into the record-length field.

## 6. EXPANSION

The bit code length of a character, which will be produced in the expansion process, is not known in advance. So, a binary tree structure is used in the

expansion program. Four binary trees are used for the four different data types. A code is expanded by examining the input bit string one bit at a time. If the current bit is 1, the search moves to the left son, otherwise to the right son. When a leaf is encountered, that means a bits combination for a character ends, a character should be produced, and the next input bit is the first bit of another character.

## 7. CONCLUSIONS

The method employed here achieve the saving in disk storage and disk access activity. Generally speaking, the compression method can be expected to reduce the size of raw data by approximately a factor of three. The costs on the execution of compression and expansion are recovered by fewer data transfer. This technique can be applied on any database system.

## 8. ACKNOWLEDGEMENTS

This paper is extracted from the author's research which was sponsored by The Central Weather Bureau, Republic of China.

The author wishes to express his sincere appreciation to Dr. William Walden, Director of Computer Services of The University of Wyoming, who directed this research. Appreciation is also given to Dr. Henry Bauer, Chairman of Computer Science Department for his valuable advice.

## 9. REFERENCES

1. Cormack, Gordon (1985): 'Data Compression on a Database System', Communications of the ACM, pp. 1336-1342.
2. Reghbati, H. K. (1981): 'An Overview of Data Compression Techniques', Computer, pp. 71-75.
3. Guiasu, Silviu (1977): 'Information Theory with Applications', McGraw-Hill Inc.
4. Held, Gilbert (1983): 'Data Compression, Techniques and Applications, Hardware and Software Considerations', John Wiley & Sons.
5. Johnson, David (1983): 'Structured Assembly Language for IBM Computers', Mayfield Publishing Company.

# 資料壓縮在 IDMS 上的研究

李　南　文

## 摘　要

在資料處理上，有的牽涉到大批的文字資料，如姓名、地址；有的則涉及數字資料，如氣候資料、電話號碼、薪資。爲了減少資料的儲存空間及降低資料通訊上的成本，資料壓縮技術是有必要採用的。

將資料中重複出現的文字以別種資料表示法表示後再加以儲存，可達到節省儲存空間的目的，此種表示法卽本文所採用的——修飾後的霍夫曼編碼法。