ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

Data Compression Methodologies for Lossless Data and Comparison between Algorithms

Shrusti Porwal, Yashi Chaudhary, Jitendra Joshi, Manish Jain
Department of Computer Science and Engineering
Vedaant Gyan Valley, Village Jharna, Mahala - Jobner, Link Road,
Jaipur Ajmer Express Way, NH-8, Jaipur, Rajasthan (India)-303007 (India)-303007

***Abstract* — This research paper provides lossless data compression methodologies and compares their performance. Huffman and arithmetic coding are compare according to their performances. Data compression is a process that reduces the data size, removing the excessive information. Shorter data size is suitable because it simply reduces the cost. The aim of data compression is to reduce redundancy in stored or communicated data, thus increasing effective data density. Data compression is important application in the area of file storage and distributed system because in distributed system data have to send from and to all system. So for speed and performance efficiency data compression is used. There are number of different data compression methodologies, which are used to compress different data formats like text, video, audio, image files. There are two forms of data compression "lossy" and "lossless", in lossless data compression, the integrity of data is preserved.**

Index Terms— **Data compression, Huffman Coding, Run Length Encoding, Arithmetic Encoding.**

I. INTRODUCTION

Data compression is a process that reduces the data size, removing the excessive information and redundancy. Why shorter data sequence is more suitable? –the answer is simple it reduces the cost. Data compression is a common requirement for most of the computerized application [1]. Data compression has important application in the area of file storage and distributed system. Data compression is used in multimedia field, text documents, and database table. Data compression methods can be classified in several ways. One of the most important criteria of classification is whether the compression algorithms remove some part of data which cannot be recovered during decompression. The algorithm which removes some part of data is called lossy data compression. And the algorithm that achieve the same what we compressed after decompression is called lossless data compression [2]. The lossy data compression algorithm is usually use when a perfect consistency with the original data is not necessary after decompression. Example of lossy data compression is compression of video or picture data. Lossless data compression is used in text file, database tables and in medical image because law of regulations. Various lossless data compression algorithm have been proposed and used. Some of main technique sare Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding. In this paper we examine Huffman Coding and Arithmetic Encoding and give compression between them according to their performances.

II. HUFFMAN CODING FOR DATA COMPRESSION

A Huffman Coding is more sophisticated and efficient lossless data compression technique. In Huffman Coding the characters in a data file are converted to binary code. And in this technique the most common characters in the file have the shortest binary codes, and the least common have the longest binary code [3]. To check Huffman Coding's work we assume that we have a text file and we have to compress it through Huffman Coding, the characters in the file have the following frequencies shown in figure 1:

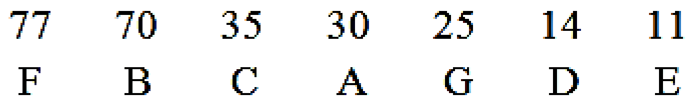
A: 30
B: 70
C: 35
D: 14
E: 11
F: 77
G: 25

Fig 1: Frequencies characters in the file

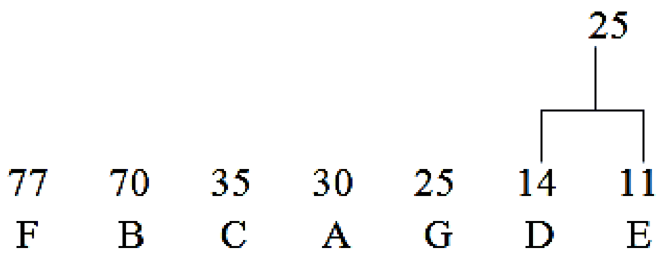
ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

We have characters from A to G and corresponding frequencies.

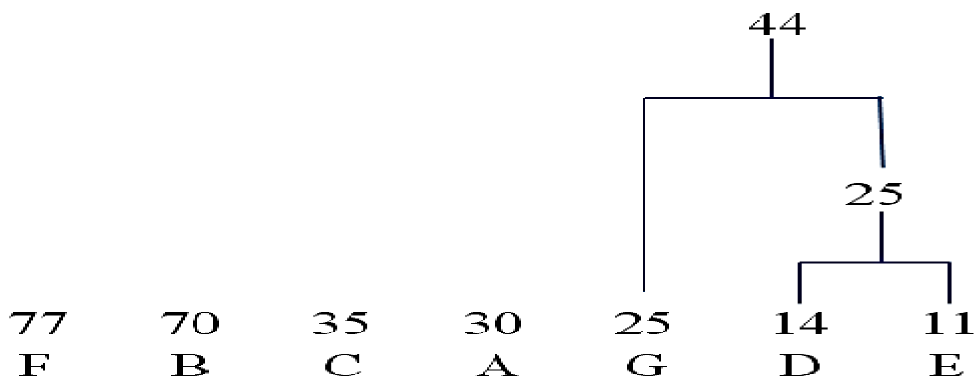
Step 1: In first step of building a Huffman Code order the characters from highest to lowest frequencies of occurrence as follows:



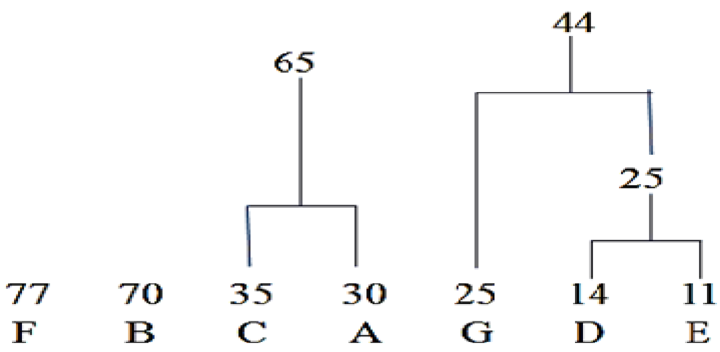
Step 2: In second step of building a Huffman code we take two least-frequent characters and logically grouped them together, and then their frequencies are added. In our example, the D and E characters have grouped together and we have combined frequency are 21:



This begins the construction of a "binary tree" structure. Now we again select the two elements with the lowest frequencies, and the lowest frequency is D-E combination and G. we group them together and add their frequencies. This is new combination of frequency 44:

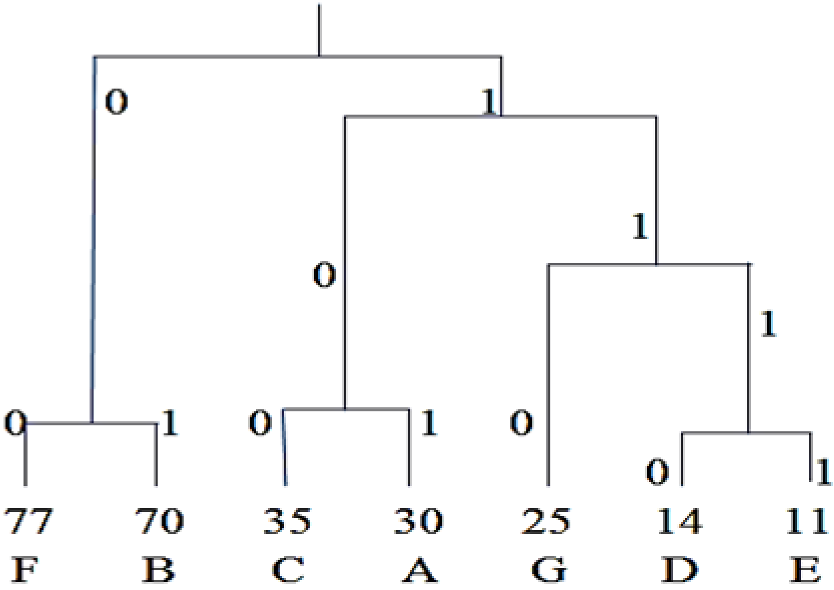


Continue in the same way to select the two elements with the lowest frequency, group them together, and then add their frequencies, until we reach to all elements and remains only one parent for all nodes which is known as root. In third iteration, the lowest frequency elements are C and A:



ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

Step 3: In third step we do labeling the edges from each parent to its left child with the digit 0 and the edge to right child with 1. The code word for each source letter is the sequence of labels along the path from root to leaf node representing the letter. Now final binary tree will be as follows:



Tracing down the tree gives the "Huffman codes", with the shortest codes assigned to the character with greater frequency shown in figure 2:

F:	**00**
B:	**01**
C:	**100**
A:	**101**
G:	**110**
D:	**1110**
E:	**1111**

Fig 2: Huffman codes with the shortest codes

The Huffman codes won't get confused in decoding. The best way to see that this is so is to envision the decoder cycling through binary tree structure, guided by the encoding bits it reads, moving top to bottom and then back to the top.

III. ARITHEMATIC ENCODING FOR DATA COMPRESSION

Arithmetic encoding is the most powerful compression techniques. This converts the entire input data into a single floating point number. A floating point number is similar to a number with a decimal point, like 4.5 instead of $\mathbf{4}^{1}/_{2}$. However, in arithmetic coding we are not dealing with decimal number so we call it a floating point instead of decimal point [4]. Let's take an example we have string:

BE_A_BEE

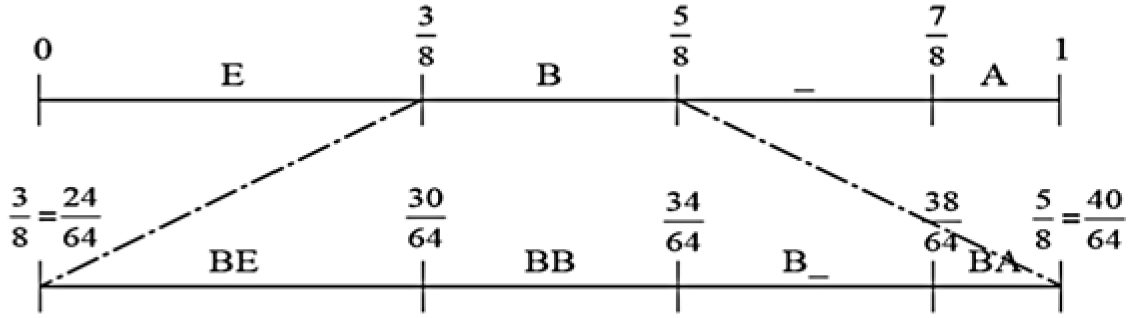
And we now compress it using arithmetic coding.

Step 1: in the first step we do is look at the frequency count for the different letters:

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

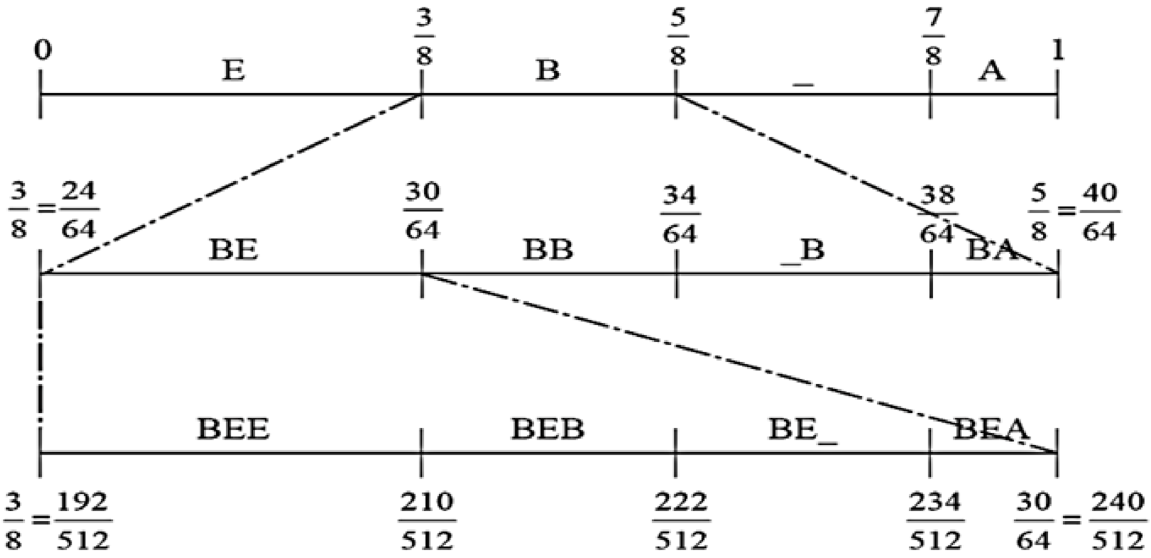
E	B	_	A
3	2	2	1

Step 2: In second step we encode the string by dividing up the interval [0, 1] and allocate each letter an interval whose size depends on how often it count in the string. Our string start with a 'B', so we take the 'B' interval and divide it up again in the same way:



The boundary between 'BE' and 'BB' is 3/8 of the way along the interval, which is itself 2/3 long and starts at 3/8. So boundary is 3/8 + (2/8) * (3/8) = 30/64. Similarly the boundary between 'BB' and 'B_' is 3/8+ (2/8) * (5/8) = 34/64, and so on.

Step 3: In third step we see next letter is now 'E', so now we subdivide the 'E' interval in the same way. We carry on through the message….And, continuing in this way, we eventually obtain:



and continuing in this way, we obtain:

So we represent the message as any number in the interval
[7653888/16777216, 7654320/16777216]

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

However, we cannot send numbers like 7654320/16777216 easily using computer. In decimal notation, the rightmost digit to the left of the decimal point indicates the number of units; the one to its left gives the number of tens: the next one along gives the number of hundred, and so on.

So

$7653888 = (7*10^6) + (6*10^5) + (5*10^4) + (3*10^3) + (8*10^2) + (8*10) + 8$

Binary numbers are almost exactly the same, only we deal with powers of 2 instead of power of 10. The rightmost digit of binary number is unit (as before) the one to its left gives the number of 2s, the next one the number of 4s, and soon.

So

$110100111 = (1*2^8) + (1*2^7) + (0*2^6) + (1*2^5) + (0*2^4) + (0*2^3) + (1*2^2) + (1*2^1) + 1$

$= 256 + 128 + 32 + 4 + 2 + 1 = 423$ in denary (i.e. base 10).

IV. Measuring compression performances

Performance measure is use to find which technique is good according to some criteria. Depending on the nature of application there are various criteria to measure the performance of compression algorithm. When measuring the performance the main thing to be considered is space efficiency [5]. and the time efficiency is another factor.

Since the compression behavior depends on the redundancy of symbols in the source file, it is difficult to measure performance of compression algorithm in general. The performance of data compression depends on the type of data and structure of input source. The compression behavior depends on the category of the compression algorithm: lossy or lossless. Following are some measurements use to calculate the performances of lossless algorithms.

Compression ratio: compression ratio is the ratio between size of compressed file and the size of source file.

$$\text{Compression ratio} = \frac{\text{Size after compression}}{\text{Size before compression}}$$

Compression factor: compression factor is the inverse of compression ratio. That is the ratio between the size of source file and the size of the compressed file.

$$\text{Compression factor} = \frac{\text{Size before compression}}{\text{Size after compression}}$$

Saving percentage calculates the shrinkage of the source file as a percentage.

$$\text{saving percentage} = \frac{\text{size before compression} - \text{size after compression}}{\text{size before compression}} \%$$

Compressed pattern matching: compressed pattern matching is the process of searching of pattern in compressed data with little or no decompression shown in following table.

Table 1: Comparison between arithmetic and Huffman coding methodologies

COMPRESSION METHOD	ARITHMETIC	HUFFMAN
Compression ratio	Very good	Poor
Compression speed	Slow	Fast
Decompression speed	Slow	Fast
Memory space	Very low	Low
Compressed pattern matching	No	Yes
Permits Random access	No	Yes

ISSN: 2319-5967
ISO 9001:2008 Certified
International Journal of Engineering Science and Innovative Technology (IJESIT)
Volume 2, Issue 2, March 2013

V. CONCLUSION

In this paper we have find out that arithmetic encoding methodology is very powerful over Huffman encoding methodology. In comparison we came to know that compression ratio of arithmetic encoding is better. And furthermore arithmetic encoding reduces channel bandwidth and transmission time.

REFERENCES

[1] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000.

[2] Burrows M., and Wheeler, D. J. 1994. A Block-Sorting Lossless Data Compression Algorithm. SRC Research Report 124, Digital Systems Research Center.

[3] Ken Huffman. Profile: David A. Huffman, Scientific American, September 1991, pp. 54–58.

[4] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

[5] Cormak, V. and S. Horspool, 1987. Data compression using dynamic Markov modeling, Comput. J., 30: 541–550.

[6] Cleary, J., Witten, I., "Data Compression Using Adaptive Coding and Partial String Matching", IEEE Transactions on Communications, Vol. COM-32, No. 4, April 1984, pp 396-402.

[7] Mahoney, M., "Adaptive Weighting of Context Models for Lossless Data Compression", Unknown, 2002.

[8] Bloom, C., "LZP: a new data compression algorithm", Data Compression Conference, 1996. DCC '96. Proceedings, p. 425 10.1109/DCC.1996.488353.

[9] Capocelli, M., R. Giancarlo and J. Taneja, 1986. Bounds on the redundancy of Huffman codes, IEEE Trans. Inf. Theory, 32: 854–857.

[10] Kaufman, K. and T. Shmuel, 2005. Semi-lossless text compression, Intl. J. Foundations of Computer Sci., 16: 1167-1178.

[11] Pu, I.M., 2006, Fundamental Data Compression, Elsevier, Britain.

[12] D. S. Taubman and M. W. Marcellin, JPEG2000: Image Compression Fundamentals, Practice and Standards, Kluwer Academic Publishers, Massachusetts, 2002.

AUTHOR BIOGRAPHY

Shrusti Porwal, B. Tech., M.Tech (Pursuing) from Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan, India. She is member of IAE, IAE-Societies.

Yashi chaudhary, B. Tech., M.Tech (Pursuing) from Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan, India. She is member of IAE, IAE-Societies.

Jitendra Joshi, B. Sc. M.C.A. M. Tech. Ph .D (Pursuing), is an Assistant Professor of computer science and engineering, Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan, India. He has over seven years of experience in teaching computer science and engineering to graduate and post graduate students. He is member of IAE, IAE-Societies, IAS, CSI, IEEE, Computer Society of Australia and Computer Society of Singapore. He has published two books viz Data Structure and Algorithm, Computer Architecture and Micro Processor. He has published many research papers in his field of interest, viz. Algorithms, Artificial Intelligence, Neural Network, E-Commerce, Sensor Network and Network and Security.

Manish Jain B. Tech from Rajasthan Technical University, Kota, Rajasthan.