An Overview on Data Compression: Lossless Techniques

[1]Ankush Choudhary,[2]Ashish Kumar Sharma,[3]Jyoti Dalal,[4]Leena Choukiker

[1234]M.Tech Student

[1234]Amity University Haryana

ABSTRACT-This paper describes what data compression is , why we need it and what are its different techniques. The advantages and disadvantages of the lossless techniques are listed. Different types of coding have been discussed. The steps for encoding and decoding various codes have also discussed.

Keywords-Huffman coding, Arithmetic coding, Run-length coding

I. INTRODUCTION

Data compression means storing data in a format that requires less space than usual. Data compression is particularly useful in communications because it enables devices to transmit or store the same amount of data in fewer bits. Data compression implies sending or storing a smaller number of bits. Although many methods are used for this purpose, in general these methods can be divided into two broad categories: lossless and lossy methods.

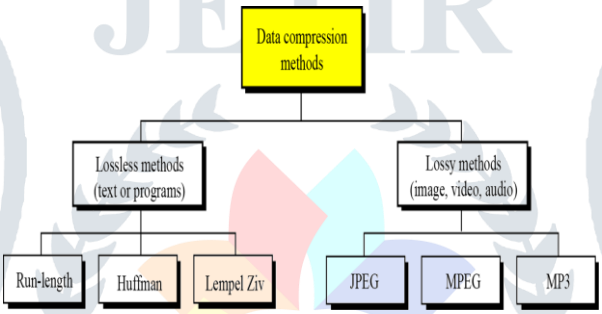

Figure 1:Data Compression methods

Now there are two reasons behind compressing the data : (1) all media (text, audio, video, graphics) has redundancy. So compression is the method to remove it. (2) Compression makes it possible for efficient storage and communication of media. Data compression is also widely used in backup utilities, spreadsheet applications, and database management systems. There are two methods of data compression: (1) Lossy Compression (2) Lossless Compression. With lossless compression, every single bit of data that was originally in the file remains after the file is uncompressed. All of the information is completely restored. This is generally the technique of choice for text or spreadsheet files, where losing words or financial data could pose a problem. The Graphics Interchange File (GIF) is an image format used on the Web that provides lossless compression.

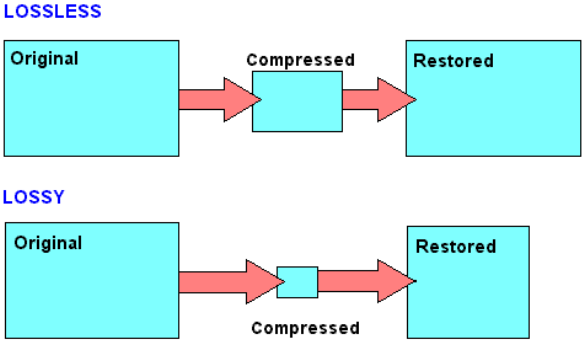

Figure 2: Lossless and Lossy compression

A. LOSSY COMPRESSION: Lossy compression reduces a file by permanently eliminating certain information, especially redundant information. When the file is uncompressed, only a part of the original information is still there (although the user may not notice it). Lossy compression is generally used for video and sound, where a certain amount of information loss will not be

detected by most users. The JPEG image file, commonly used for photographs and other complex still images on the Web, is an image that has lossy compression. Using JPEG compression, the creator can decide how much loss to introduce and make a trade-off between file size and image quality.

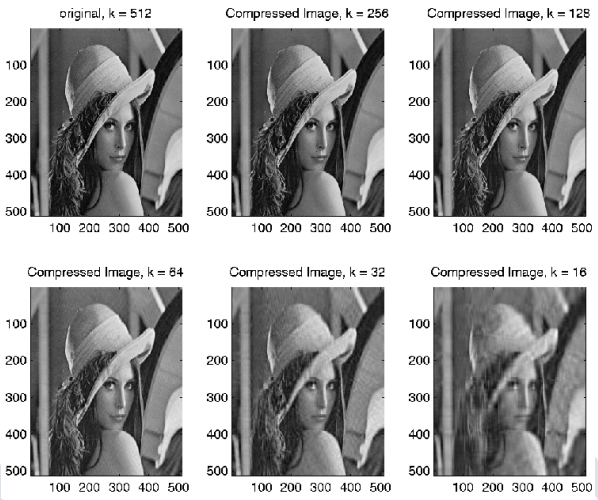

Figure 3: Example of lossy compression

B. LOSSLESS COMPRESSION: In lossless data compression, the integrity of the data is preserved. The original data and the data after compression and decompression are exactly the same because, in these methods, the compression and decompression algorithms are exact inverses of each other: no part of the data is lost in the process. Redundant data is removed in compression and added during decompression. Lossless compression methods are normally used when we cannot afford to lose any data. There are a number of methods of lossless compression. Some of them are discussed here.

B.1. HUFFMAN CODING: Huffman coding assigns shorter codes to symbols that occur more frequently and longer codes to those that occur less frequently. For example, imagine we have a text file that uses only five characters (A, B, C, D, E). Before we can assign bit patterns to each character, we assign each character a weight based on its frequency of use. In this example, assume that the frequency of the characters is as shown in Table 1.

Character	A	B	C	D	E
Frequency	17	12	12	27	32

Table 1

Now let us see how Huffman coding works.

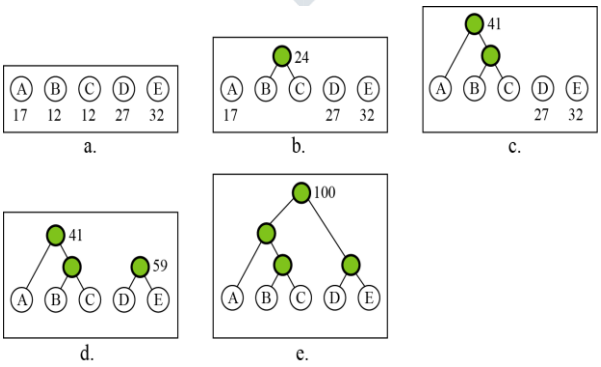

Figure 4: Working of Huffman coding

A character's code is found by starting at the root and following the branches that lead to that character. The code itself is the bit value of each branch on the path, taken in sequence.

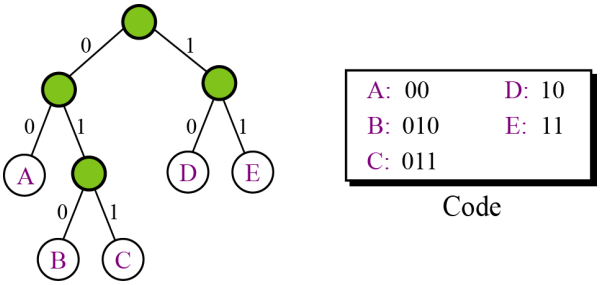

Figure 4: Final tree and code

Let us see how to encode text using the code for our five characters. Figure 5 shows the original and the encoded text.

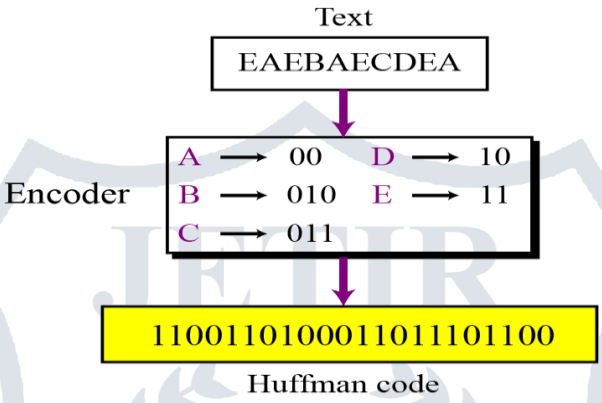

Figure 6: Huffman Encoding

The recipient has a very easy job in decoding the data it receives. Figure 7 shows how decoding takes place.

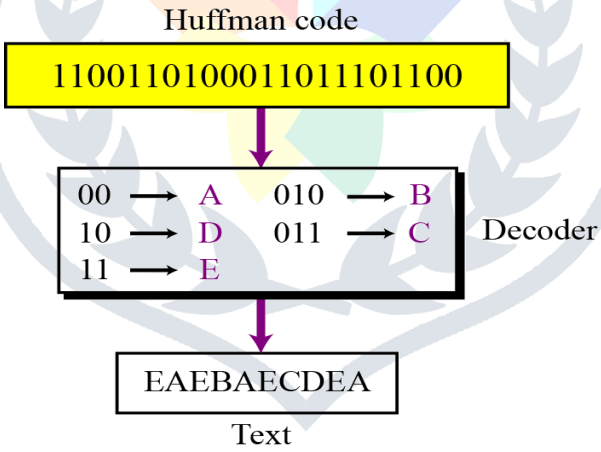

Figure 7: Huffman Decoding

Huffman coding is almost as computationally simple and produces prefix codes that always achieve the lowest expected code word length, under the constraints that each symbol is represented by a code formed of an integral number of bits.

B.2 SHANNON FANO CODING: [1] In the field of data compression, Shannon–Fano coding, named after Claude Shannon and Robert Fano, is a technique for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured). In Shannon–Fano coding, the symbols are arranged in order from most probable to least probable, and then divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned; symbols in the first set receive "0" and symbols in the second set receive "1". As long as any sets with more than one member remain, the same process is repeated on those sets, to determine successive digits of their codes. When a set has been reduced to one symbol this means the symbol's code is complete and will not form the prefix of any other symbol's code. Shannon–Fano coding is used in the IMPLODE compression method, which is part of the ZIP file format. Shannon–Fano does not always produce

optimal prefix codes; the set of probabilities {0.35, 0.17, 0.17, 0.16, 0.15} is an example of one that will be assigned non-optimal codes by Shannon–Fano coding.

Message	X1	X2	X3	X4	X5	X6	X7	X8
Probability	1/4	1/4	1/8	1/8	1/16	1/16	1/16	1/16

Table 2: Messages and probabilities

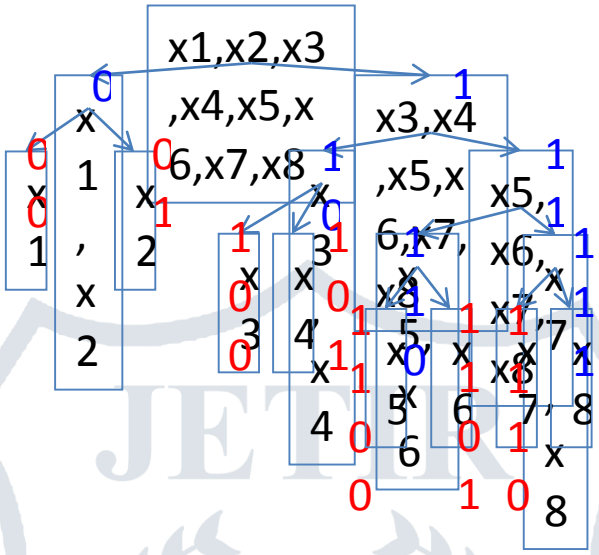

Figure 8: Shannon Fano Encoding example

B.3 ARITHMETIC CODING: Arithmetic coding, is entropy coder widely used, the only problem is it's speed, but compression tends to be better than Huffman can achieve. The idea behind arithmetic coding is to have a probability line, 0-1, and assign to every symbol a range in this line based on its probability, the higher the probability, the higher range which assigns to it. Once we have defined the ranges and the probability line, start to encode symbols, every symbol defines where the output floating point number lands. [1] In most situations, arithmetic coding can produce greater overall compression than either Huffman or Shannon–Fano, since it can encode in fractional numbers of bits which more closely approximate the actual information content of the symbol. However, arithmetic coding has not superseded Huffman the way that Huffman supersedes Shannon–Fano, both because arithmetic coding is more computationally expensive and because it is covered by multiple patents . Let us say we have

Symbol	Occurrence	Range
a	2	[0,0.5)
b	1	[0.5,0.75)
c	1	[0.75,1]

Table 3: Symbol , their frequency and range

We start to code the symbols and compute our output number. The algorithm to compute the output number is:

- Low = 0
- High = 1

Loop. For all the symbols.

- Range = high - low
- High = low + range * high range of the symbol being coded
- Low = low + range * low range of the symbol being coded

Where: Range, keeps track of where the next range should be and High and low, specify the output number.

Symbol	Range	Low Value	High Value

		0	1
b	1	0.5	0.75
a	0.25	0.5	0.625
c	0.125	0.59375	0.625
a	0.03125	0.59375	0.609375

Table 4: Symbol, Range and Values

The output number will be 0.59375.

B.4 RUN LENGTH CODING: Run-length encoding is probably the simplest method of compression. It can be used to compress data made of any combination of symbols. It does not need to know the frequency of occurrence of symbols and can be very efficient if data is represented as 0s and 1s.The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences.The method can be even more efficient if the data uses only two symbols (for example 0 and 1) in its bit pattern and one symbol is more frequent than the other.

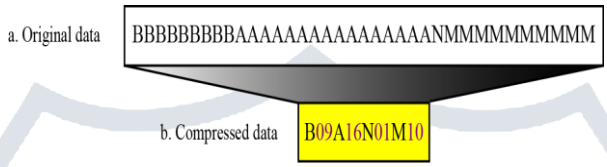

Figure 9: Run-length encoding example

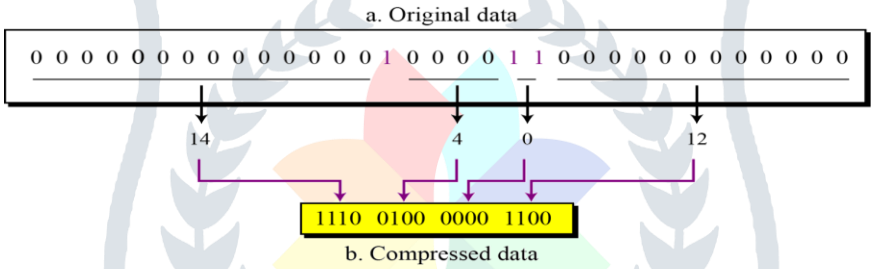

Figure 10: Run-length encoding for two symbols

II. CONCLUSION: Different lossless techniques have been mentioned here. In future work we will be implementing codes with the help of algorithms for all the various lossless techniques.

III. REFERENCES :

[1] http://en.wikipedia.org/wiki/Data_compression

[2] http://whatis.techtarget.com/definition/lossless-and-lossy-compression

[3] http://www.webopedia.com/TERM/D/data_compression.html

[4] http://www.data-compression.com/lossless.shtml

JETIR