DATA COMPRESSION TOOLKIT

By

ABHISHEK CHAUHAN -031414

GAURAV AGRAWAL -031257

PULKIT AGARWAL -031413

MAY-2007

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CERTIFICATE

This is to certify that the work entitled, "DATA COMPRESSION TOOLKIT" submitted by Abhishek Chauhan, Gaurav Agrawal, Pulkit Agarwal in partial fulfillment for the award of degree of Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Mr. Ravindara Bhatt
Lecturer
Department of CSE and IT
Jaypee University Of Information Technology

ACKNOWLEDGEMENT

In Accordance with our final project submission of 8th Semester (B.Tech Computer Science & Engg.), we were assigned to study and research on ongoing compression algorithms and making Data Compression Toolkit for same.

We would like to express our extreme gratitude to

Mr. Ravindara Bhatt

Lecturer

Department of CSE and IT,

Jaypee University Of Information Technology

For guiding us, being extremely helpful, patient and always being there whenever we were in doubt. Without his help, support and constant supervision, we would have never been able to complete our final project assignment successfully.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Compression is one of the most important aspect that plays not only an important role in today's Fast growing world of technology but has also become a necessity for all kind of business organizations even if they are a small co-operate serving only hand full of people.

Compression not only plays an important role in preserving the data but also lays an emphasis on increasing the performance of various systems in various fields.

The professionals do not only do compression but various home users also utilize it. There are many commercial software available for performing both compression and decompression depending upon the user's need. As more and more data is being flooded in various organizations everyday compression has become really crucial and critical.

1 Introduction to data compression

1.1 Preliminaries

Computers process miscellaneous data. Some data, as colures, tunes, smells, pictures, voices, are analogue. Contemporary computers do not work with infinite-precise analogue values, so we have to convert such data to a digital form. During the digitalisation process, the infinite number of values is reduced to a finite number of quantised values. Therefore some information is always lost, but the larger the target set of values, the less information is lost. Often the precision of digitalisation is good enough to allow us neglecting the difference between digital version of data and their analogue original. There are also discrete data, for example written texts or databases contain data composed of finite number of possible values. We do not need to digitize such types of data but only to represent them somehow by encoding the original values. In this case, no information is lost.

Regardless of the way we gather data to computers, they usually are sequences of elements. The elements come from a finite ordered set, called an alphabet. The elements of the alphabet, representing all possible values, are called symbols or characters. One of the properties of a given alphabet is its number of symbols, and we call this number the size of the alphabet. The size of a sequence is the number of symbols it is composed of.

The size of the alphabet can differ for various types of data. For a Boolean sequence the alphabet consists of only two symbols: false and true, representable on 1 bit only. For typical English texts the alphabet contains less than 128 symbols and each symbol is represented on 7 bits using the ASCII code. The most popular code in contemporary computers is an 8-bit code: some texts are stored using the 16-bit Unicode designed to represent all the alphabetic symbols used worldwide. Sound data typically are sequences of symbols, which represent temporary values of the tone. The size of the alphabet to encode this data is usually 2^8, 2^{16}, or 2^{24}. Picture data typically contain symbols from the alphabet representing the colures of image pixels. The colour of a pixel can be represented using various coding schemes. We mention here only one of them, the RGB code that contains the brightness of the three components red, green, and blue. The brightness of each component can be represented, for example, using 28 different values, so the size of the alphabet is 2^{24} in this case.

A sequence of symbols can be stored in a file or transmitted over a network. The sizes of modern databases, application files, or multimedia files can be extremely large. Reduction of the sequence size can save computing resources or reduce the transmission time. Sometimes we even would not be able to store the sequence without compression. Therefore the investigation of possibilities of compressing the sequences is very important.

1.2 What is data compression?

A sequence over some alphabet usually exhibits some regularities, what is necessary to think of compression. For typical English texts we can spot that the most frequent letters are e, t, a, and the least frequent letters are q, z. We can also find such words as the, of, to frequently. Often also longer fragments of the text repeat, possibly even the whole sentences. We can use these properties in some way, and the following sections elaborate this topic.
A different strategy to compress the sequence of picture data is needed. With a photo of night sky we can still expect that the most frequent colures of pixels is black or dark grey. But with a generic photo we usually have no information what color is the most frequent. In general, we have no a priori knowledge of the picture, but we can find regularities in it. For example, colures of successive pixels usually are similar, some parts of the picture are repeated.
Video data are typically composed of subsequences containing the data of the successive frames. We can simply treat the frames as pictures and compress them separately, but more can be achieved with analysing the consecutive frames. What can happen in a video during a small fraction of a second? We can assume that successive video frames are often similar.
We have noticed above that regularities and similarities often occur in the sequences we want to compress. Data compression bases on such observations and attempts to utilise them to reduce the sequence size. For different types of data there are different types of regularities and similarities, and before we start to compress a sequence, we should know of what type it is. One more thing we should mark here is that the compressed sequence is useful only for storing or transmitting, but not for a direct usage. Before we can work on our data we need to expand them to the original form. Therefore the compression methods must be reversible. The decompression is closely related to the compression, but the latter is more interesting because we have to find the regularities in the sequence.

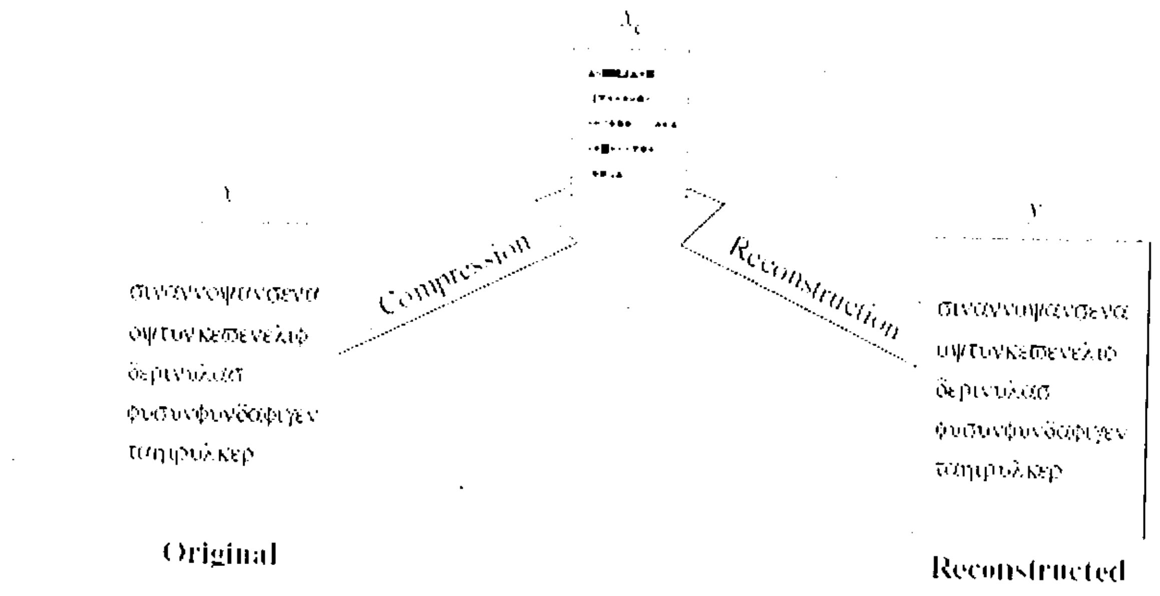


Fig 1.2.1 Compression and reconstruction

1.3 Measures of Performance

A compression algorithm can be evaluated in a number of different ways. We could measure the relative complexity of the algorithm, the memory required to implement the algorithm, how fast the algorithm performs on a given machine, the amount of compression, and how closely the reconstruction resembles the original.

A very logical way of measuring how well a compression algorithm compresses a given set of data is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression. This ratio is called the compression ratio. Suppose storing an image made up of a square array of 256×256 pixels requires 65,536 bytes. The image is compressed and the compressed version requires 16,384 bytes. We would say that the compression ratio is 4:1. We can also represent the compression ratio by expressing the reduction in the amount of data required as a percentage of the size of the original data. In this particular example the compression ratio calculated in this manner would be 75%.

Another way of reporting compression performance is to provide the average number of bits required to represent a single sample. This is generally referred to as the rate. For example, in the case of the compressed image described above, if we assume 8 bits per byte (or pixel), the average number of bits per pixel in the compressed representation is 2. Thus, we would say that the rate is 2 bits per pixel.

In lossy compression, the reconstruction differs from the original data. Therefore, in order to determine the efficiency of a compression algorithm, we have to have some way of quantifying the difference. The difference between the original and the reconstruction is often called the distortion. Lossy techniques are generally used for the compression of data that originate as analog signals, such as speech and video. In compression of speech and video, the final arbiter of quality is human. Because human responses are difficult to model mathematically, many approximate measures of distortion are used to determine the quality of the reconstructed

waveforms.

Other terms that are also used when talking about differences between the reconstruction and the original are fidelity and quality. When we say that the fidelity or quality of a reconstruction is high, we mean that the difference between the reconstruction and the original is small. Whether this difference is a mathematical difference or a perceptual difference should be evident from the context.

2 Information theory

Information is constantly send and received the form of text, speech, and images. Information is an elusive non-mathematical quantity that cannot be precisely defined, captured, and measured.

The standard dictionary definitions of information are as follows

- Knowledge derived from study, experience, or instruction.
- Knowledge of a specific event or situation; intelligence;
- A collection of facts or data.
- The act of informing or the condition of being informed, communication of knowledge.

The importance of information theory is that it quantifies information. It shows how to measure information, so that we can answer the question "how much information is included in this piece of data?" with a precise number! Quantifying information is based on the observation that the information content of a message is equivalent to the amount of *surprise* in the message. If I tell you something that you already know (for example, "you and I work here"), I haven't given you any information. If I tell you something new (for example, "we both have got an increase"), I have given you some information. If I tell you something that really surprises you (for example, "only I have got an increase"), I have given you more information, regardless of the number of words I have used, and of how you feel about my information.

3 Entropy

Shannon borrowed the definition of entropy from statistical physics to capture the notion of how

much information is contained in a and their probabilities. For a set of possible messages, *S* Shannon defined entropy as,

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$

where is *p(s)* the probability of message. The definition of Entropy is very similar to that in statistical physics—in physics *S* is the set of possible states a system can be in and *p(s)* is the probability the system is in state *s*.

Getting back to messages, if we consider the individual messages *s* belonging to *S*, Shannon defined the notion of the *self-information* of a message as

$$i(s) = \log_2 \frac{1}{p(s)}.$$

This self-information represents the number of bits of in-formation contained in it and, roughly speaking, the number of bits we should use to send that message. The equation says that messages with higher probability will contain less information (*e.g.*, a message saying that it will be sunny out in LA tomorrow is less informative than one saying that it is going to snow).

The entropy is simply a weighted average of the information of each message, and therefore the average number of bits of information in the set of messages. Larger entropies represent more information, and perhaps counter-intuitively, the more random a set of messages (the more even the probabilities) the more information they contain on average. Here are some examples of entropies for different probability distributions over five messages.

$$\begin{aligned} p(S) &= \{0.25, 0.25, 0.25, 0.125, 0.125\} \\ H &= 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8 \\ &= 1.5 + 0.75 \\ &= 2.25 \end{aligned}$$

$$\begin{aligned} p(S) &= \{0.5, 0.125, 0.125, 0.125, 0.125\} \\ H &= 0.5 \times \log_2 2 + 4 \times 0.125 \times \log_2 8 \\ &= 0.5 + 1.5 \\ &= 2 \end{aligned}$$

$$\begin{aligned} p(S) &= \{0.75, 0.0625, 0.0625, 0.0625, 0.0625\} \\ H &= 0.75 \times \log_2(\frac{4}{3}) + 4 \times 0.0625 \times \log_2 16 \\ &= 0.3 + 1 \\ &= 1.3 \end{aligned}$$

Note that the more uneven the distribution, the lower the Entropy.

In particular if message *p(A)* and *p(B)* are independent, the probability of sending one after the other is *p(A) p(B)* and the information contained in them is

$$i(AB) = \lg \frac{1}{p(A)p(B)} = \lg \frac{1}{p(A)} + \lg \frac{1}{p(A)} = i(A) + i(B) .$$

The logarithm is the simplest unction that has this property.

3.1 Joint entropy

The joint entropy of two discrete random variables X and Y is merely the entropy of their pairing: (X, Y). this implies that if X and Y are independent, then their joint entropy is the sum of their individual entropies.

For example, if (*X*, *Y*) represents the position of a chess piece — *X* the row and *Y* the column, then the joint entropy of the row of the piece and the column of the piece will be the entropy of the position of the piece.

$$H(X,Y) = \mathbb{E}_{X,Y}[-\log p(x,y)] = -\sum_{x,y} p(x,y)\log p(x,y)$$

Despite similar notation, joint entropy should not be confused with cross entropy.

3.2 Conditional entropy

The conditional entropy of *X* given random variable *Y* (also called the equivocation of *X* about *Y*) is the average conditional entropy over *Y*:

$$H(X|Y) = \mathbb{E}_Y[H(X|y)] = -\sum_{y\in Y} p(y)\sum_{x\in X} p(x|y)\log p(x|y) = \sum_{x,y} p(x,y)\log\frac{p(y)}{p(x,y)}.$$

Because entropy can be conditioned on a random variable or on that random variable being a certain value, care should be taken not to confuse these two definitions of conditional entropy, the former of which is in more common use. A basic property of this form of conditional entropy is that:

$$H(X|Y) = H(X,Y) - H(Y).$$

4 Lossy and lossless compression

4.1 Lossy compression

The assumed recipient of the compressed data influences the choice of a compression method. When we compress audio data some tones are not audible to a human because our senses are imperfect. When a human will be the only recipient, we can freely remove such unnecessary data. Note that after the decompression we do not obtain the original audio data, but the data that sound identically. Sometimes we can also accept some small distortions if it entails a significant improvement to the compression ratio. It usually happens when we have a dilemma: we can have a little distorted audio, or we can have no audio at all because of data storage restrictions. When we want to compress picture or video data, we have the same choice—we can sacrifice the perfect conformity to the original data gaining a tighter compression. Such compression methods are called lossy, and the strictly bi-directional ones are called lossless.

The lossy compression methods can achieve much better compression ratio than lossless ones. It is the most important reason for using them. The gap between compression results for video and audio data is so big that lossless methods are almost never employed for them. Lossy compression methods are also employed to pictures. The gap for such data is also big but there are situations when we cannot use lossy methods. Sometimes we cannot use lossy methods to the images because of the law regulations. This occurs for medical images as in many countries they must not be compressed loosely. Roughly, we can say that lossy compression methods may be used to data that were digitised before compression.

4.2 Lossless compression

When we need certainty that we achieve the same what we compressed after decompression, lossless compression methods are the only choice. They are of course necessary for binary data or texts (imagine an algorithm that could change same letters or words). It is also sometimes better to use lossless compression for images with a small number of different colours or for scanned text.

The rough answer to the question when to use lossless data compression methods is: We use them for digital data, or when we cannot apply lossy methods for some reasons.

This dissertation deals with lossless data compression, and we will not concern lossy compression methods further. From time to time we can mention them but it will be strictly denoted. If not stated otherwise, further discussion concerns lossless data compression only.

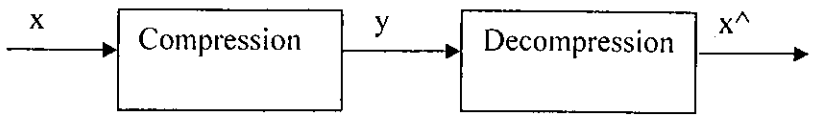


Fig. 4.1 Compression and Decompression

Where	X is Original file
	Y is Compressed file
	X^ is Uncompressed file

If $X = X^\wedge$ then, Lossless compression

If $X \neq X^\wedge$ then, Lossy compression

5 File format

A file format is a particular way to encode information for storage in a file. Since a disk drive, or indeed any computer storage, can store only bits, the computer must have some way of converting information to 0s and 1s and vice-versa. There are different kinds of formats for different kinds of information. Within any format type, e.g., word processor documents, there will typically be several different formats. Sometimes these formats compete with each other.

Types of file formats

- Video
- Audio
- Image
- Text

5.1 Video formats

A video format describes how one device sends a video pictures to another device, such as the way that a DVD player sends pictures to a television or a computer to a monitor. More formally, the video format describes the sequence and structure of frames that create the moving video image. Video formats are commonly known in the domain of commercial broadcast and consumer devices; most notably to date, these are the analog video formats of NTSC, PAL, and SECAM. However, video formats also describe the digital equivalents of the commercial formats, the aging custom military uses of analog video (such as RS-170 and RS-343), the increasingly important video formats used with computers, and even such offbeat formats such as color field sequential..Video formats were originally designed for display devices such as a CRTs. However, because other kinds of displays have common source material and because video formats enjoy wide adoption and have convenient organization, video formats are a common means to describe the structure of displayed visual information for a variety of graphical output devices.

5.1.1 *MPEG file structure*

The file structure is as follows:

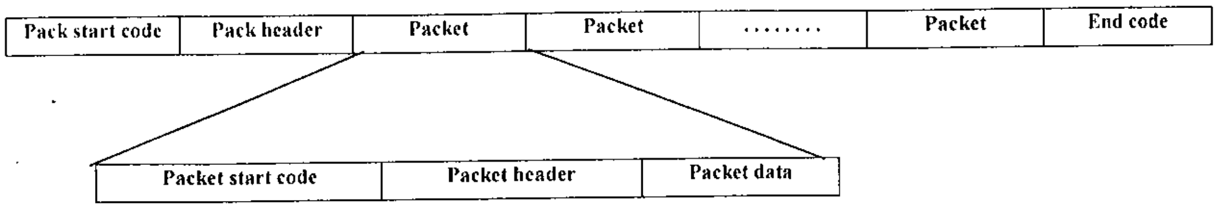


Fig. 5.1.1 MPEG File structure

5.2 Audio formats

An audio format is a medium for storing sound and music. The term is applied to both the physical medium and the format of the content – in computer science it is often limited to the audio file format, but its wider use usually refers to the physical method used to store the data. Music is recorded and distributed using a variety of audio formats, some of which store additional information.

5.2.1 *MP3 Audio Frame structure*

MPEG-3 Audio Frame				
Sync Word	Frame Header	Side Info	Main data	Ancillary Data

Fig. 5.2.1 MP3 structure

Sync Word is: 0xFFF (1111 1111 1111)

Frame Header: 18 bit structure

Side Info: variable bit length structure

Main Data: compressed sound packets.

Ancillary Data: Ignored by decoder, for inserting user defined data into the bit stream, e.g. Song Title etc. (NB. not ID3 tags).

Here is a presentation of the header content. Characters from A to M are used to indicate different fields. In the table below, you can see details about the content of each field.

AAAAAAAA AAABBCCD EEEEFFGH IIJJKLMM

A	Frame sync (all bits set)
B	MPEG Audio version ID
C	Layer description
D	Protection bit
E	Bit rate index
F	Sampling rate frequency index (values are in Hz)
G	Padding bit
H	Private bit. It may be freely used for specific needs of an application.
I	Channel Mode

Table 5.1: Detail about the header contents

5.3 Image formats

Image file formats provide a standardized method of organizing and storing image data. This article deals with digital image formats used to store photographic and other image information. Image files are made up of either pixel or vector (geometric) data, which is rasterized to pixels in the display process, with a few exceptions in vector graphic display. The pixels that comprise an image are in the form of a grid of columns and rows. Each of the pixels in an image stores digital numbers representing brightness and colo

5.3.1 *BMP file format*

BMP is a standard file format for computers running the Windows operating system. The format was developed by Microsoft for storing bitmap files in a device-independent bitmap (DIB) format that will allow Windows to display the bitmap on any type of display device. The term "device independent" means that the bitmap specifies pixel color in a form independent of the method used by a display to represent color.

Since BMP is a fairly simple file format, its structure is pretty straightforward. Each bitmap file contains:

- A bitmap-file header: this contains information about the type, size, and layout of a device-independent bitmap file.
- A bitmap-information header which specifies the dimensions, compression type, and color format for the bitmap.
- A colour table, defined as an array of RGBQUAD structures, contains as many elements as there are colours in the bitmap. The colour table is not present for bitmaps with 24 color bits because each pixel is represented by 24-bit red-green-blue (RGB) values in the actual bitmap data area.
- An array of bytes that defines the bitmap bits. These are the actual image data, represented by consecutive rows, or "scan lines," of the bitmap. Each scan line consists of consecutive bytes representing the pixels in the scan line, in left-to-right order.

BMP files always contain RGB data. The file can be:

- 1-bit: 2 colours (monochrome)
- 4-bit: 16 colours
- 8-bit: 256 colours.
- 24-bit: 16777216 colours, mixes 256 tints of Red with 256 tints of Green and Blue.

5.4 Text Formats

Basic text files contain data which is basically written in a language. The compression is achieved on various types of text files by applying various algorithms on various parameters such as repetition, ansii codes etc.

5.4.1 *Text File Structure*

A text file is simply a stream of characters.

- They contain human-readable text.
- The characters can represent numbers, words, or anything else.
- People can read and write them using a text editor.

There is no structure to a text file. People see text as having structure, though:

- pages, of
- lines, of
- characters

This structure by giving some characters special meanings:

- **EOL** - end of line
- **EOP** - end of page
- **EOF** - end of file

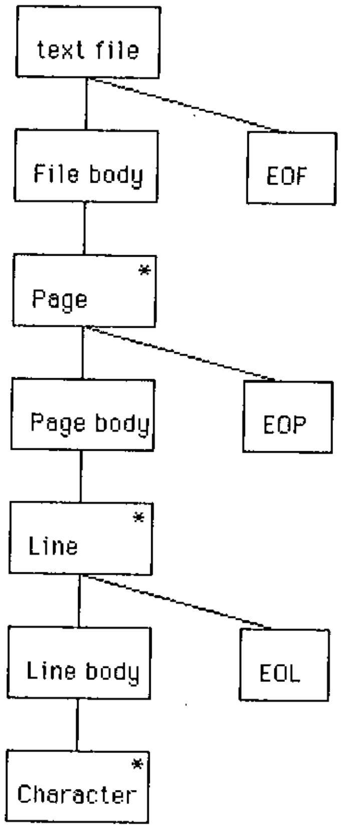


Fig. 5.4.1 Text file structure

6 Major techniques used in lossless and lossy compression

Lossless compression

Statistical techniques:

- Huffman coding.
- Arithmetic coding.

Dictionary techniques:

- LZW, LZ77.

Lossy compression

Major techniques:

- Vector Quantization.
- Wavelets.
- Fractal transforms.

6.1 Lossless compression

6.1.1 *Huffman coding*

The Huffman compression algorithm is named after its inventor, David Huffman, formerly a professor at MIT.

Huffman compression is a lossless compression algorithm that is ideal for compressing text or program files. This probably explains why it is used a lot in compression programs.

How Huffman compression works

Huffman compression belongs into a family of algorithms with a variable codeword length. That means that individual symbols (characters in a text file for instance) are replaced by bit sequences that have a distinct length. So symbols that occur a lot in a file are given a short sequence while other that are used seldom get a longer bit sequence.

A practical example will show you the principle: Suppose you want to compress the following piece of data:

ACDABA

Since these are 6 characters, this text is 6 bytes or 48 bits long. With Huffman encoding, the file is searched for the most frequently appearing symbols (in this case the character 'A' occurs 3 times) and then a tree is build that replaces the symbols by shorter bit sequences. In this particular case, the algorithm would use the following substitution table: A=0, B=10, C=110, D=111. If these code words are used to compress the file, the compressed data look like this:

01101110100

This means that 11 bits are used instead of 48, a compression ratio of 4 to 1 for this particular file.

Huffman encoding can be further optimized in two different ways:

- Adaptive Huffman code dynamically changes the code words according to the change of probabilities of the symbols.
- Extended Huffman compression can encode groups of symbols rather than single symbols.

6.1.2 *Arithmetic Coding*

Arithmetic coding is also a kind of statistical coding algorithm similar to Huffman coding. However, it uses a different approach to utilize symbol probabilities, and performs better than Huffman coding. In Huffman coding, optimal codeword length is obtained when the symbol probabilities are of the form $(1/2)^x$, where x is an integer. This is because Huffman coding assigns code with an integral number of bits. This form of symbol probabilities is rare in practice. Arithmetic coding is a statistical coding method that solves this problem. The code form is not restricted to an integral number of bits. It can assign a code as a fraction of a bit. Therefore, when the symbol probabilities are more arbitrary, arithmetic coding has a better compression ratio than Huffman coding. In brief, this is can be considered as grouping input symbols and coding them into one long code. Therefore, different symbols can share a bit from the long code. Although arithmetic coding is more powerful than Huffman coding in compression ratio, arithmetic coding requires more computational power and memory. Huffman coding is more attractive than arithmetic coding when simplicity is the major concern.

6.1.3 *LZW coding*

LZW is named after Abraham Lempel, Jakob Ziv and Terry Welch, the scientists who developed this compression algorithm. It is a lossless 'dictionary based' compression algorithm. Dictionary based algorithms scan a file for sequences of data that occur more than once. These sequences are then stored in a dictionary and within the compressed file, references are put where-ever repetitive data occurred. This compression algorithm maintains its dictionary within the data themselves.

Suppose you want to compress the following string of text: *the quick brown fox jumps over the lazy dog*. The word 'the' occurs twice in the file so the data can be compressed like this: *the quick brown fox jumps over << lazy dog*. in which << is a pointer to the first 4 characters in the string.

In 1978, Lempel and Ziv published a second paper outlining a similar algorithm that is now referred to as LZ78. This algorithm maintains a separate dictionary.
Suppose you once again want to compress the following string of text: *the quick brown fox jumps over the lazy dog*. The word 'the' occurs twice in the file so this string is put in an index that is added to the compressed file and this entry is referred to as *. The data then look like this: * *quick brown fox jumps over* * *lazy dog.*

How LZW works

LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.
The code that the LZW algorithm outputs can be of any arbitrary length, but it must have more bits in it than a single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0-255 refer to individual bytes, while codes 256-4095 refer to substrings.

Advantages and disadvantages

LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. Files that are compressed but that do not contain any repetitive information at all can even grow bigger!
LZW compression is fast.

6.2 Lossy compression

6.2.1 *Vector quantization*

Vector Quantization (VQ) is a lossy compression method. It uses a codebook containing pixel patterns with corresponding indexes on each of them. The main idea of VQ is to represent arrays of pixels by an index in the codebook. In this way, compression is achieved because the size of the index is usually a small fraction of that of the block of pixels. The main advantages of VQ are the simplicity of its idea and the possible efficient implementation of the decoder. Moreover, VQ is theoretically an efficient method for image compression, and superior performance will be gained for large vectors. However, in order to use large vectors, VQ becomes complex and requires many computational resources (e.g. memory, computations per pixel) in order to efficiently construct and search a codebook. More research on reducing this complexity has to be done in order to make VQ a practical image compression method with superior quality.

In data compression, vector quantization is a quantization technique often used in lossy data compression in which the basic idea is to code or replace with a key, values from a multidimensional vector space into values from a discrete subspace of lower dimension.

6.2.2 *Wavelet compression*

Wavelets are functions defined over a finite interval. The basic idea of the wavelet transform is to represent an arbitrary function *f(x)* as a linear combination of a set of such wavelets or basis functions. These basis functions are obtained from a single prototype wavelet called the mother wavelet by dilations (scaling) and translations (shifts).

The purpose of wavelet transform is to change the data from time-space domain to time-frequency domain which makes better compression results.

The simplest form of wavelets, the Haar wavelet function (see Figure 4) is defined as:

$$\psi(x) = \begin{cases} 1 & 0 \le x < 1/2 \\ -1 & 1/2 \le x < 1 \\ 0 & otherwise \end{cases}$$

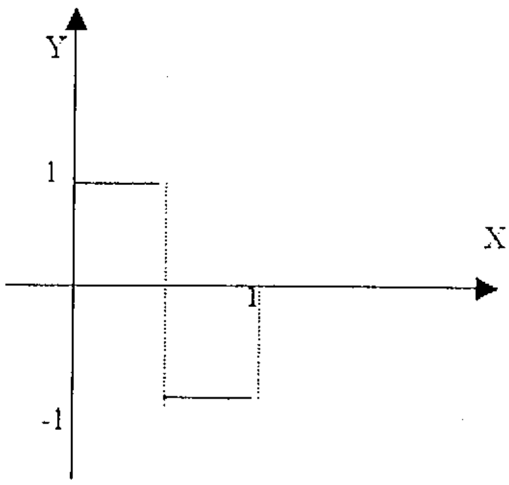


Fig 6.2.2.1 Haar Wavelet

The following is a simple example to show how to perform Haar wavelet transform on four Sample numbers. Assume we have four numbers

x(0) = 1.2 x(1) = 1.0 x(2) = -1.0 x(3) = -1.2.

Let us perform Haar wavelet transform on these four numbers.

$$\begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \begin{bmatrix} 2.2 \\ 0.2 \\ -2.2 \\ 0.2 \end{bmatrix}$$

Notice we can always do inverse transform from *x* to *y*:

$$\begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ y(3) \end{bmatrix}$$

If 0.2 is below our quantization threshold, it will be replaced by 0.
Then, reconstructed x will be [1.1, 1.1, -1.1, -1.1].

After first transform, we keep $y(1)$ and $y(3)$ at the finest level and iterate the transform on $y(0)$ and $y(2)$ again.

$$z(0) = y(0) + y(2) = 0 \text{ and } z(2) = y(0) - y(2) = 4.4.$$

Those four numbers become [0, 0.2, 4.4, 0.2]. After quantization, they could be [0, 0, 4, 0], which are much easier to be compressed.

6.2.3 *Fractal compression*

The application of fractals in image compression started with M.F. Barnsley and A. Jacquin. Fractal image compression is a process to find a small set of mathematical equations that can describe the image. By sending the parameters of these equations to the decoder, we can reconstruct the original image. In general, the theory of fractal compression is based on the contraction mapping theorem in the mathematics of metric spaces. The Partitioned Iterated Function System (PIFS), which is essentially a set of contraction mappings, is formed by analyzing the image. Those mappings can exploit the redundancy that is commonly present in most images. This redundancy is related to the similarity of an image with itself, that is, part A of a certain image is similar to another part B of the image, by doing an arbitrary number of contractive transformations that can bring A and B together. These contractive transformations are actually common geometrical operations such as rotation, scaling, skewing and shifting. By applying the resulting PIFS on an initially blank image iteratively, we can completely regenerate the original image at the decoder. Since the PIFS often consists of a small number of parameters, a huge compression ratio (e.g. 500 to 1000 times) can be achieved by representing the original image using these parameters. However, fractal image compression has its disadvantages. Because fractal image compression usually involves a large amount of matching and geometric operations, it is time consuming. The coding process is so asymmetrical that encoding of an image takes much longer time than decoding.

7 Selection of technique

Since our main objective was to implement lossless compression and decompression. We implemented Huffman compression in developed application. Huffman compression algorithm is one of the oldest compression algorithm. All the other compression algorithms are derived from it. The time and space complexity of Huffman compression algorithm is slightly less as compared to all other lossless compression algorithms. The implementation of Huffman algorithm is less complex as compared to other lossless algorithms and the results are quite optimal as compared to other lossless compression algorithms.

8 Design of application

8.1 Statement of Purpose

The Data Compression Toolkit facilitates the compression and decompression of any file. The compression and decompression is done on two basic parameters: algorithm complexity and amount of compression.

8.2 Context of the Application

Environment: User who acts as a controller and inputs the required inputs for the application to work. The application responds to the user by showing a status report.

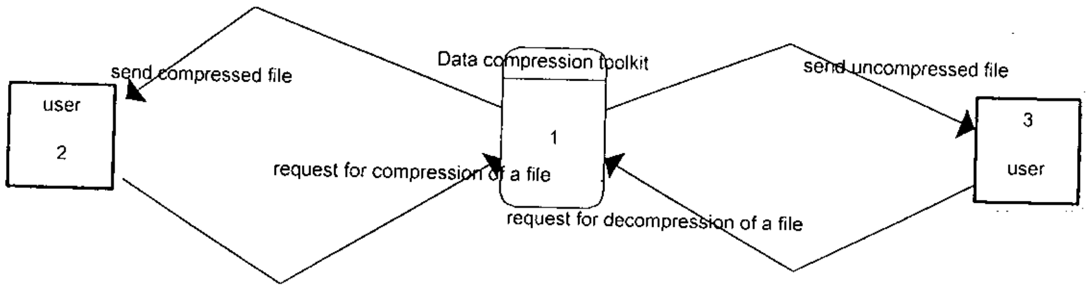


Fig 8.1 Context diagram

8.3 Event list

- Send file for compression (Flow Event)
- Send file for decompression (Flow Event)
- Receive compressed/encoded file (Flow Event)
- Receive decompressed/decoded file (Flow Event)

8.4 Cartesian Hierarchy

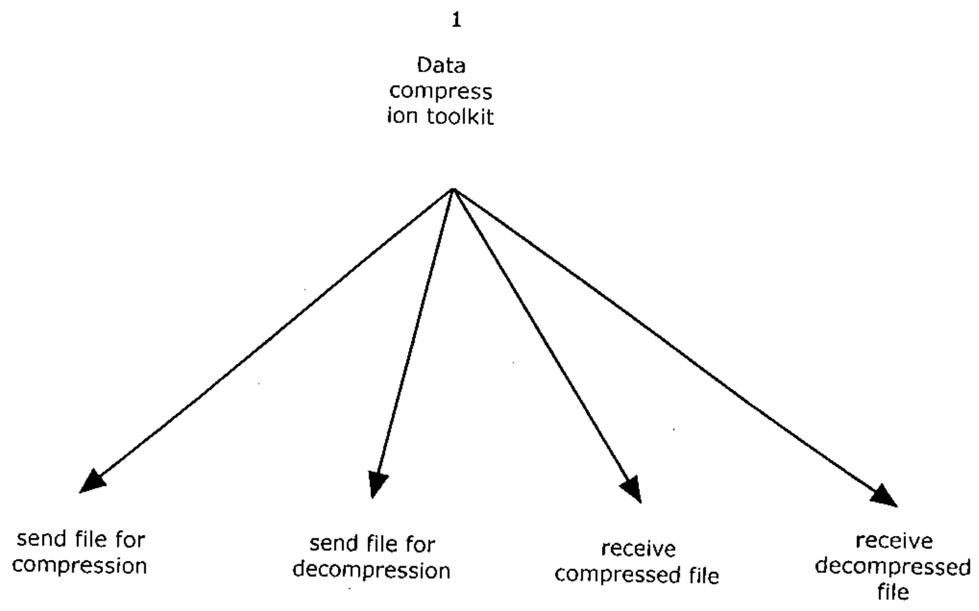

1
Data compress ion toolkit
send file for compression
send file for decompression
receive compressed file
receive decompressed file

Fig. 8.2 Cartesian hierarchy

8.5 Data flow diagram

Data Flow diagrams which are shown below are totally in consideration to the GUI of designed application. The internal details of the compression algorithm will be explained later on in the document.

8.5.1 *Compression*

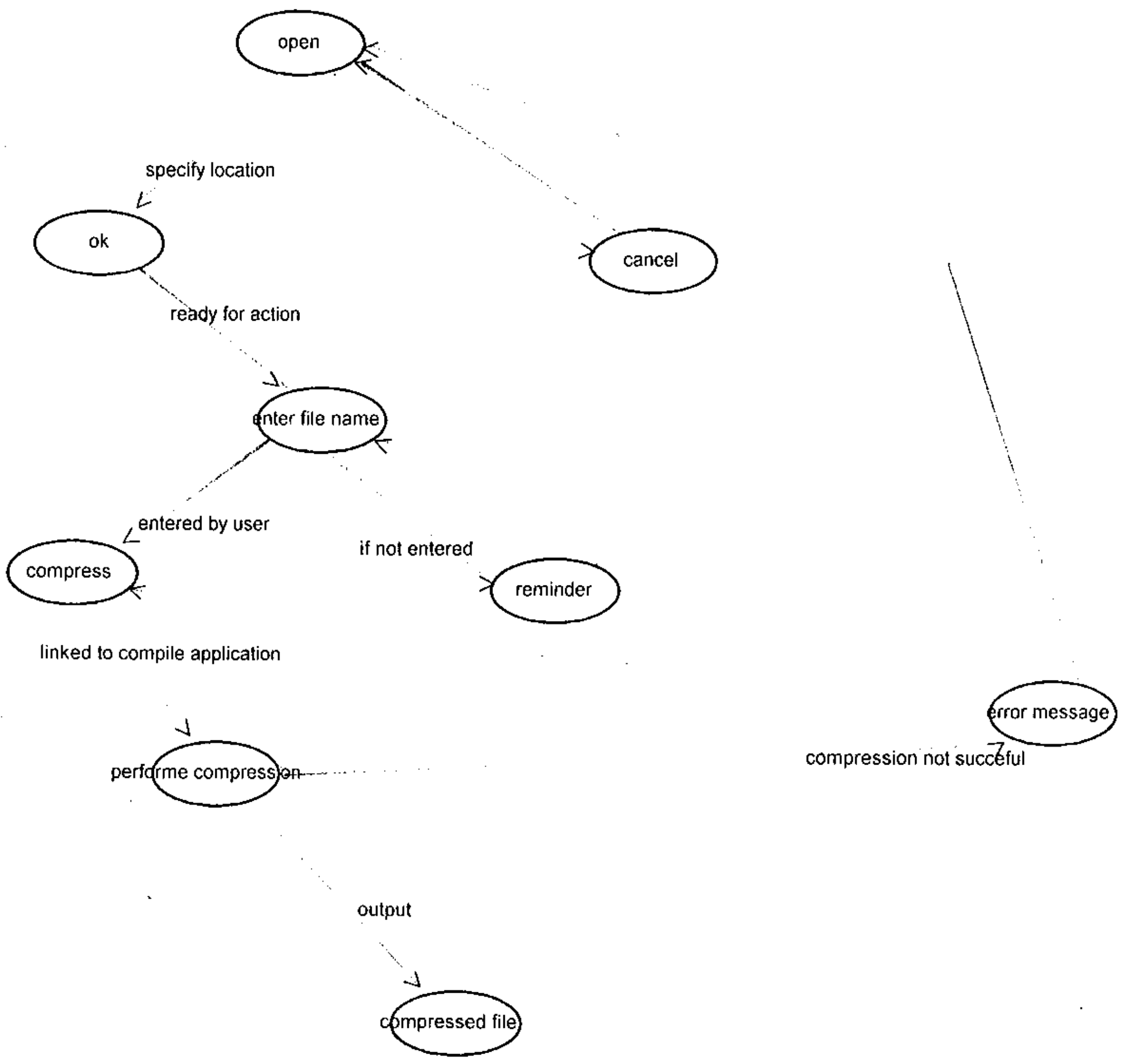


Fig.8.4 Data flow diagram for compression

8.5.2 *Decompression*

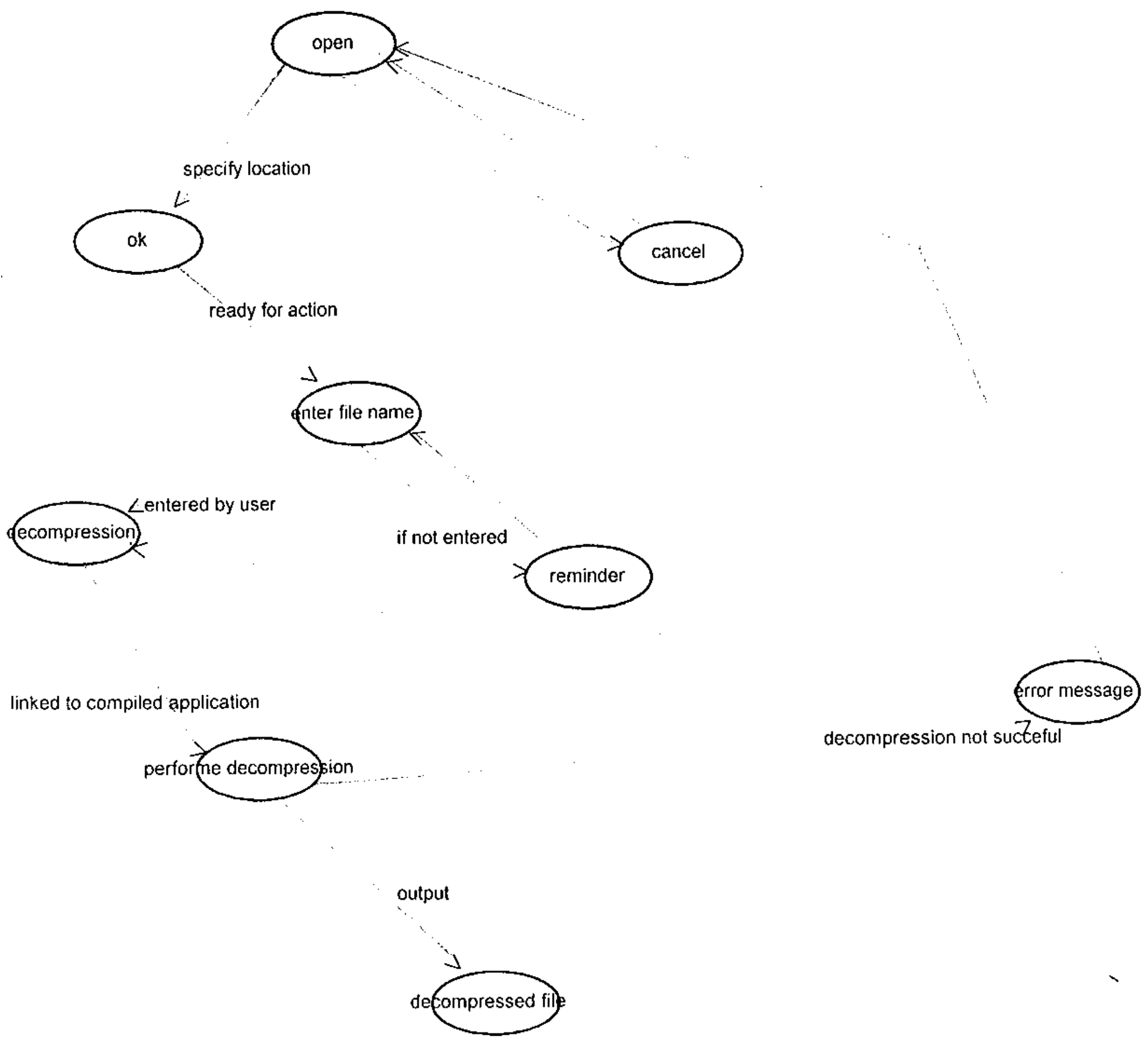
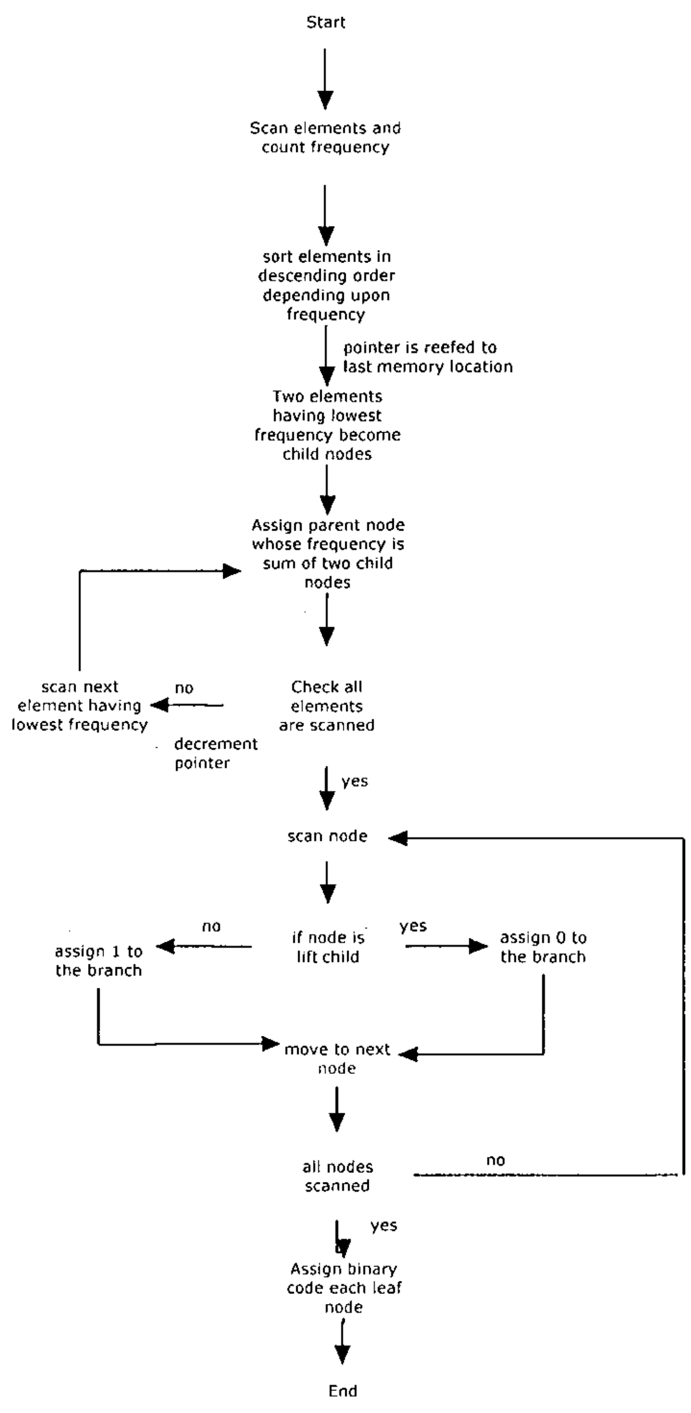


Fig.8.5 Data flow diagram for de-compression

9 Implementation Details

9.1 Creation of Huffman tree



Start
Scan elements and count frequency
sort elements in descending order depending upon frequency
pointer is reefed to last memory location
Two elements having lowest frequency become child nodes
Assign parent node whose frequency is sum of two child nodes
scan next element having lowest frequency
no
decrement pointer
Check all elements are scanned
yes
scan node
no
assign 1 to the branch
if node is lift child
yes
assign 0 to the branch
move to next node
all nodes scanned
no
yes
Assign binary code each leaf node
End

Fig 9.1.1 Flowchart of creation of Huffman tree

9.2 Compression

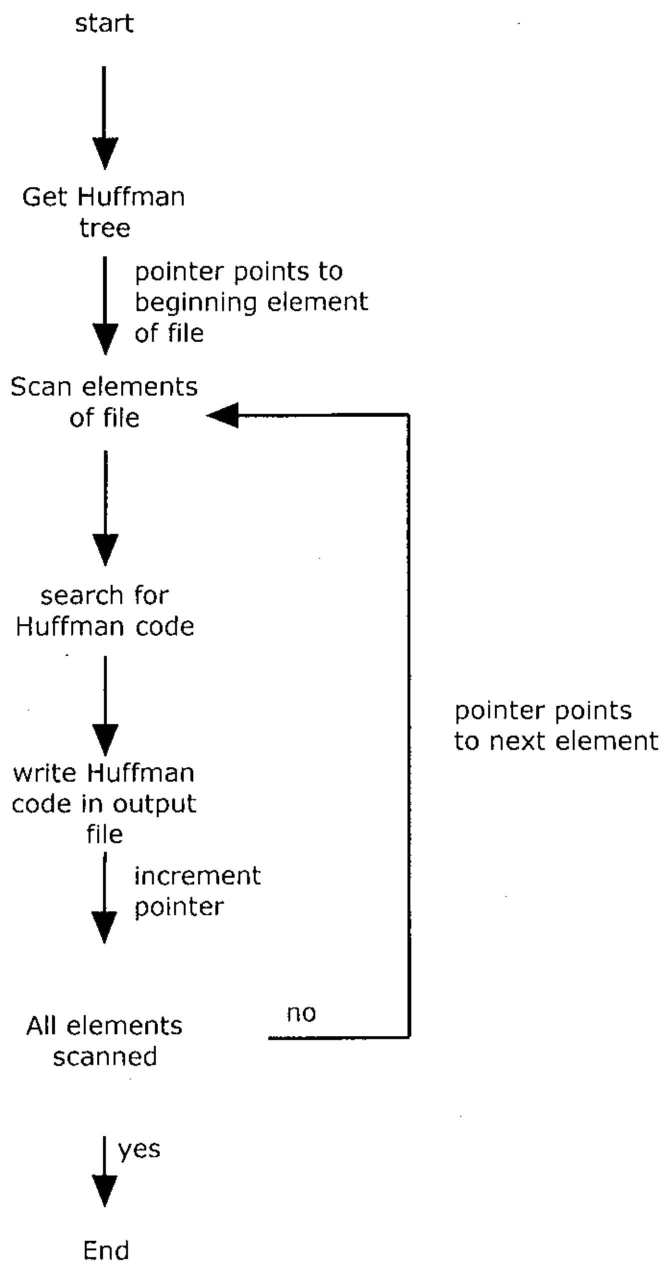
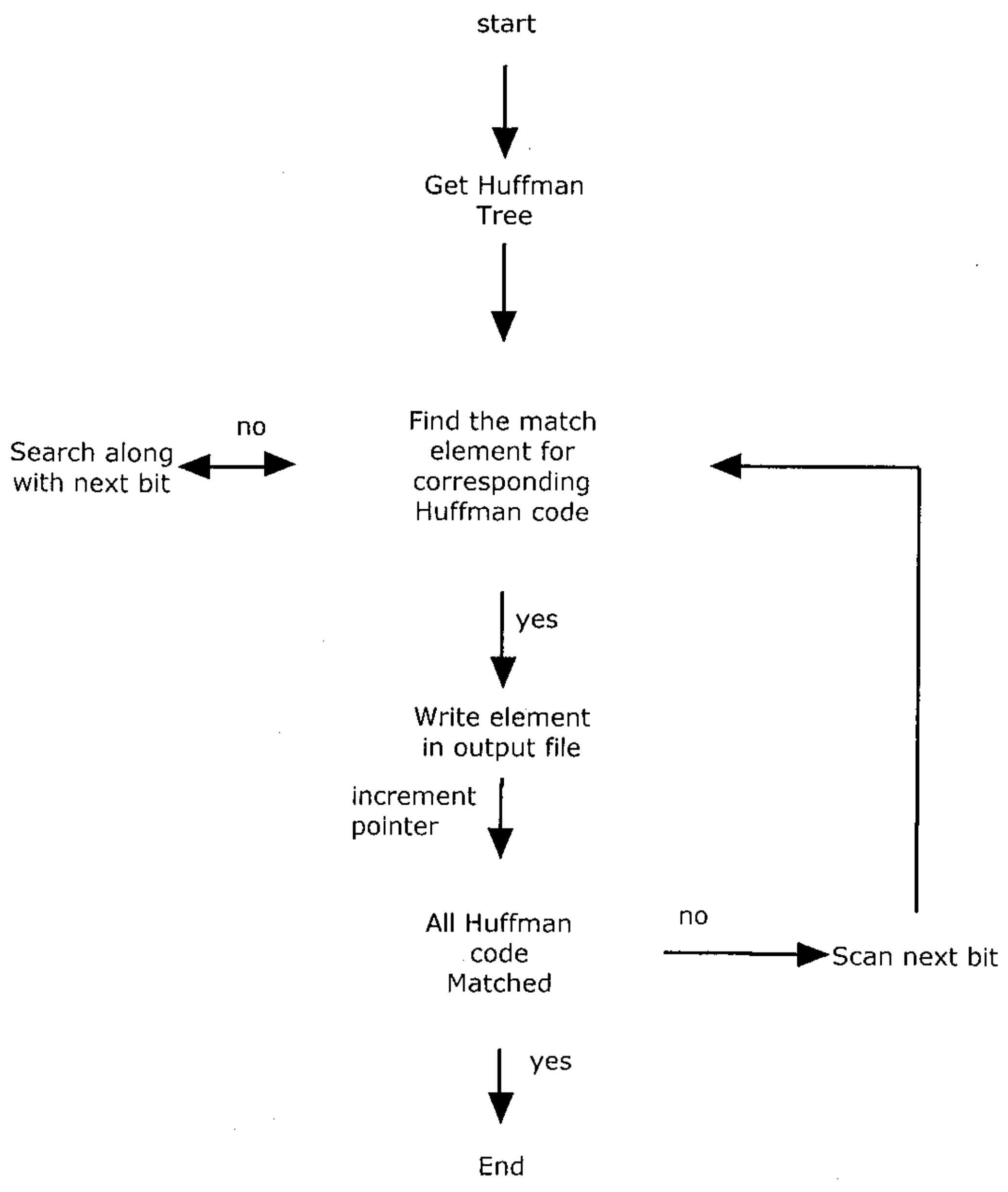


Fig 9.2.1 Flowchart for compression

9.3 Decompression



start
Get Huffman Tree
Find the match element for corresponding Huffman code
no
Search along with next bit
yes
Write element in output file
increment pointer
All Huffman code Matched
no
Scan next bit
yes
End

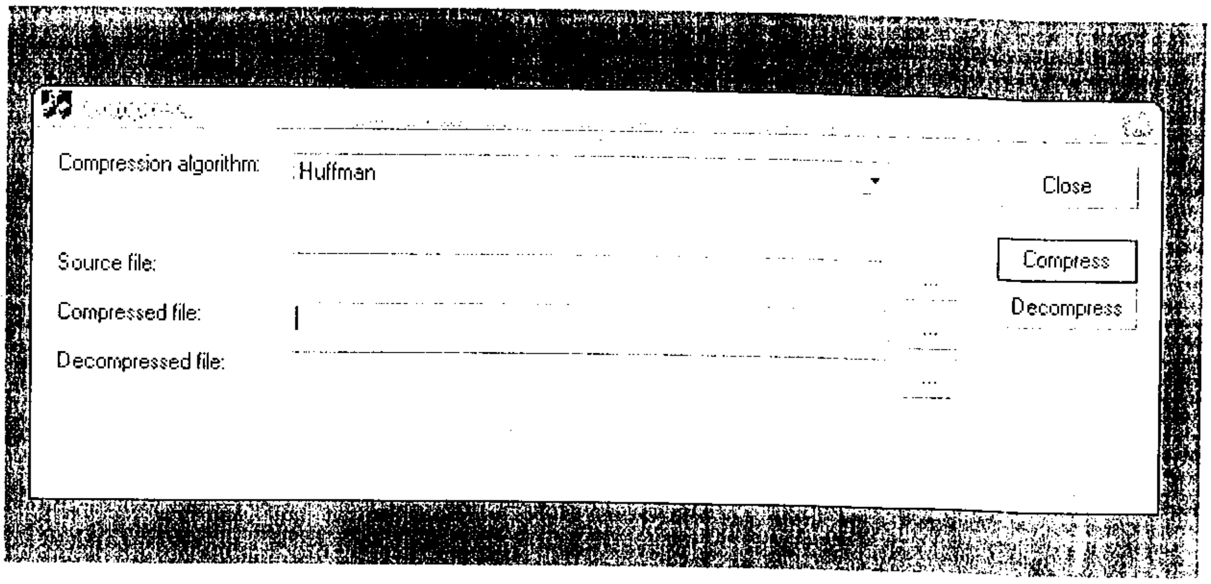
Fig 9.3.1 Flowchart for decompression

10 Screen shots and testing cases

Below are the screen shots of the designed application while working on to the various file formats

Initial Configuration of the Main Application

Screen shot of when application is started


Compression algorithm: Huffman

Source file:

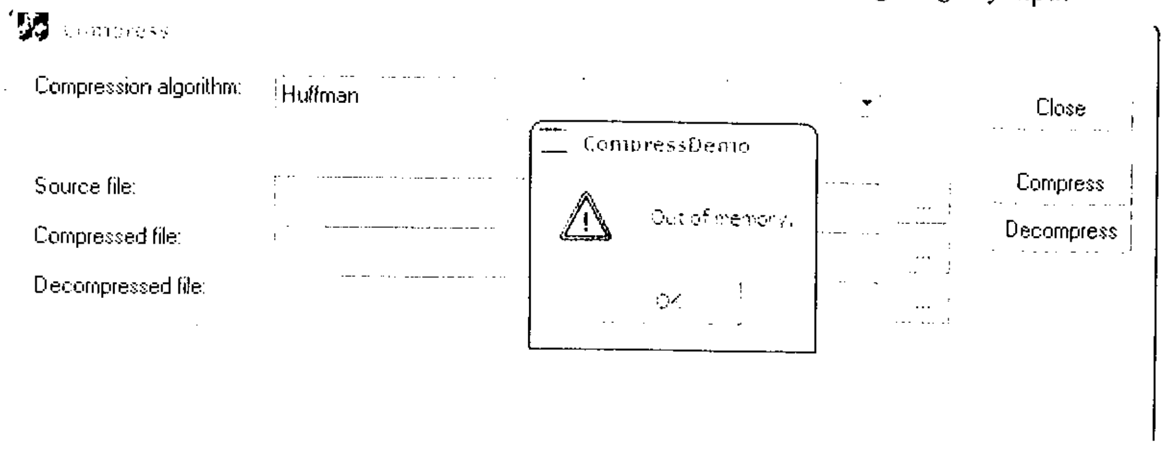
Compressed file:

Decompressed file:

Fig. 10.1 GUI of application

Error

Screen shot of when application is executed to perform a task without giving any input


Compress

Compression algorithm: Huffman

Source file:

Compressed file:

Decompressed file:

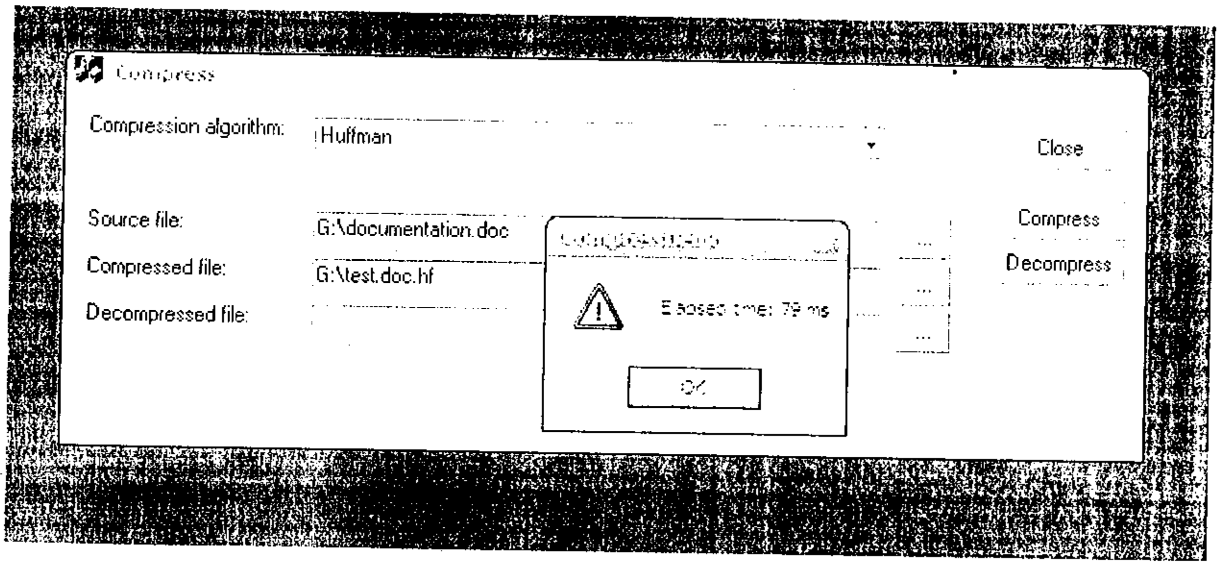
CompressDemo

Out of memory.

Fig. 10.2 when application is executed to perform a task without giving any input

Below are the screen shots that were taken when application was used to perform both compression and decompression on various types of file formats

10.1 Text file


Compress

Compression algorithm: Huffman

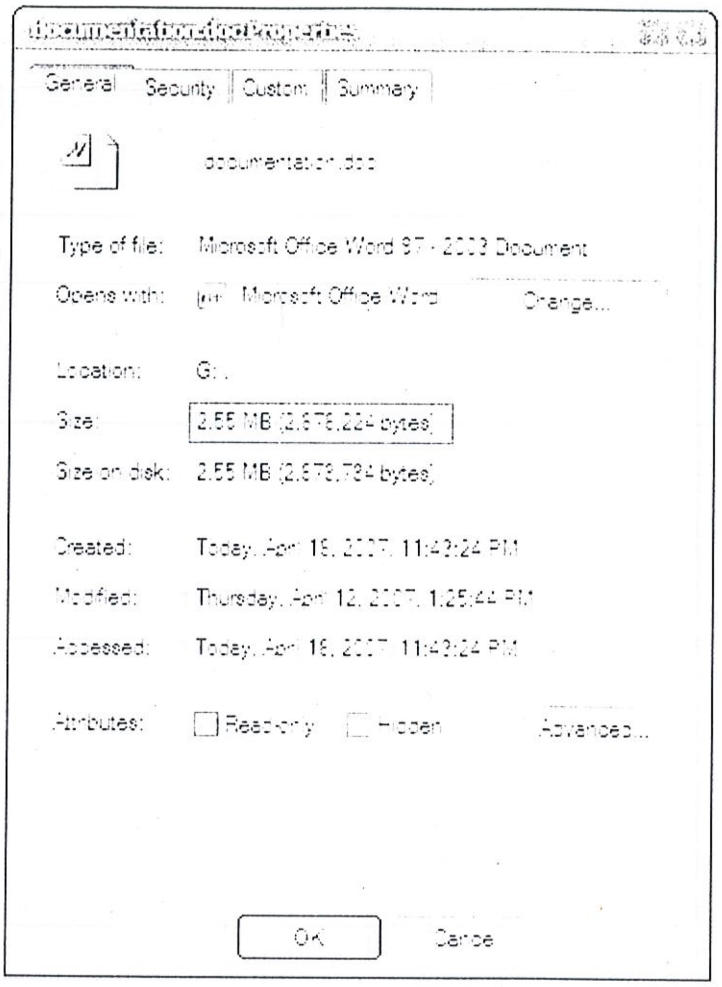
Source file: G:\documentation.doc

Compressed file: G:\test.doc.hf

Decompressed file:

Elapsed time: 79 ms

Fig. 10.1.1 We compressed the text file of .doc extension over here



documentation.doc Properties

documentation.doc

Type of file: Microsoft Office Word 97 - 2003 Document

Opens with: Microsoft Office Word

Location: G:\

Size: 2.55 MB (2,676,224 bytes)

Size on disk: 2.55 MB (2,678,784 bytes)

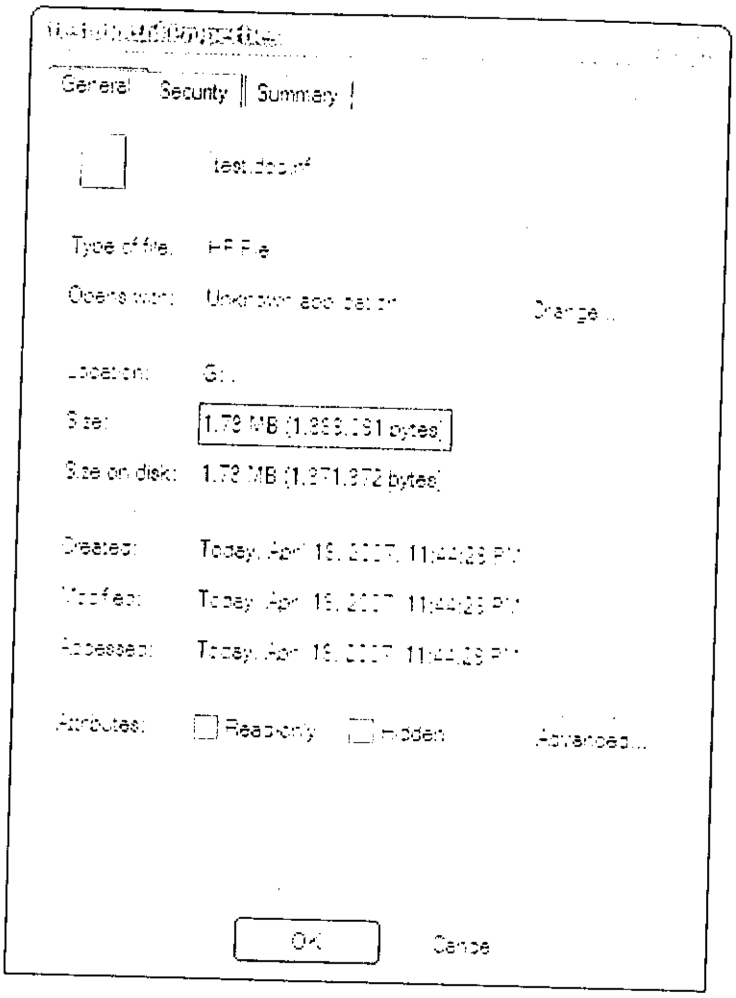
Created: Today, April 18, 2007, 11:43:24 PM

Modified: Thursday, April 12, 2007, 1:25:44 PM

Accessed: Today, April 18, 2007, 11:43:24 PM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.1.2 Highlighted text shows the original size of the text file



test.doc.hf

Type of file: HF File

Opens with: Unknown application

Location: G:\

Size: 1.79 MB (1,869,391 bytes)

Size on disk: 1.79 MB (1,871,872 bytes)

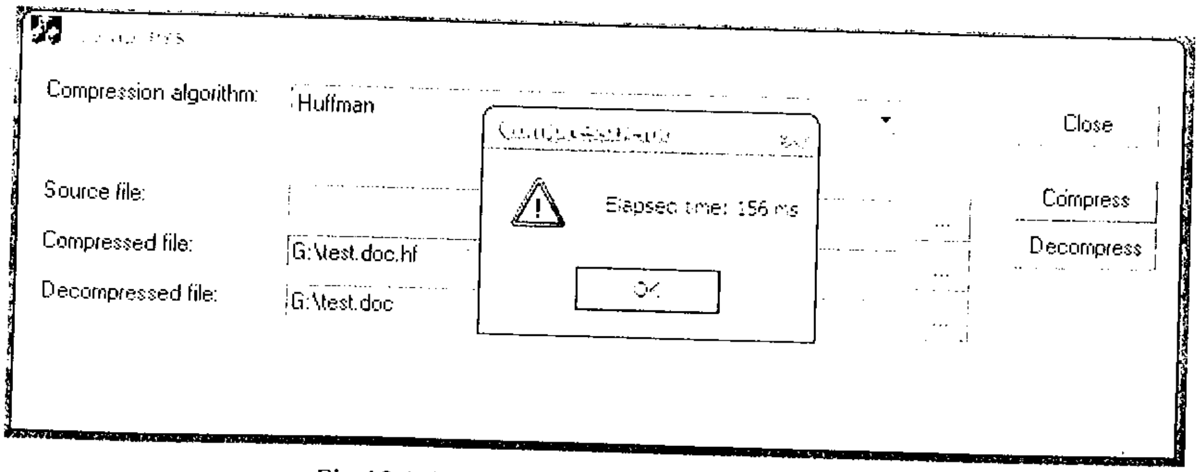
Created: Today, April 19, 2007, 11:44:29 PM

Modified: Today, April 19, 2007, 11:44:29 PM

Accessed: Today, April 19, 2007, 11:44:29 PM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.1.3 Highlighted text shows the size after the compression of the text file



Compression algorithm: Huffman

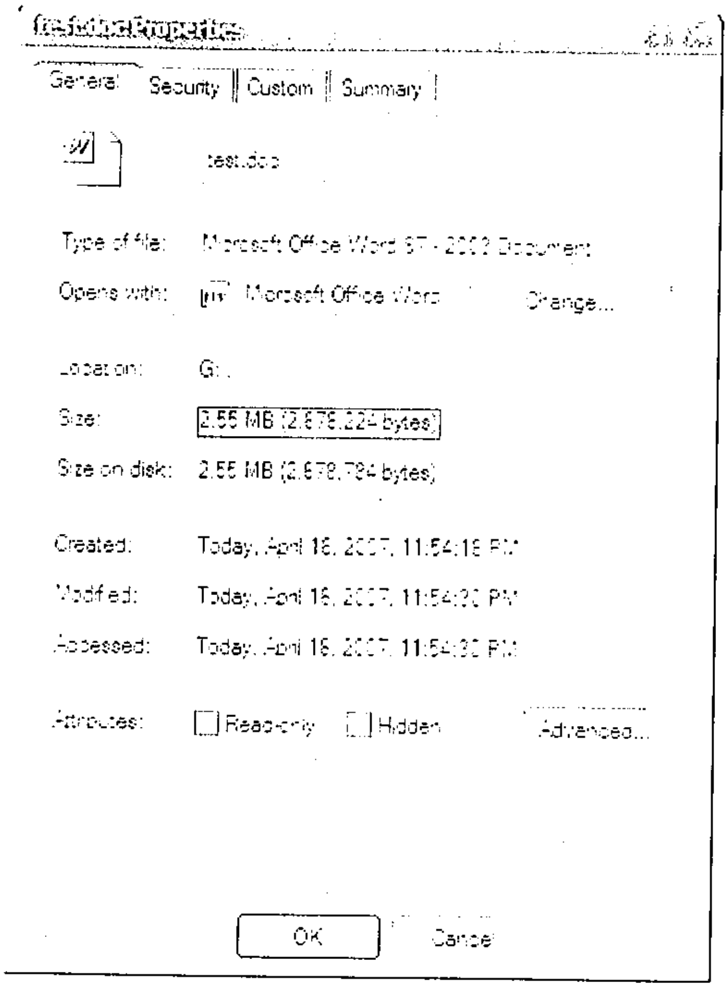
Source file:

Compressed file: G:\test.doc.hf

Decompressed file: G:\test.doc

Elapsed time: 156 ms

Fig.10.1.4 The same file was decompressed



test.doc Properties

General | Security | Custom | Summary

test.doc

Type of file: Microsoft Office Word 97 - 2003 Document

Opens with: Microsoft Office Word

Location: G:\

Size: 2.55 MB (2,678,224 bytes)

Size on disk: 2.55 MB (2,678,784 bytes)

Created: Today, April 18, 2007, 11:54:18 PM

Modified: Today, April 18, 2007, 11:54:30 PM

Accessed: Today, April 18, 2007, 11:54:30 PM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.1.5 we got the original file with the same size as the size was not decreased it showed Lossless compression.

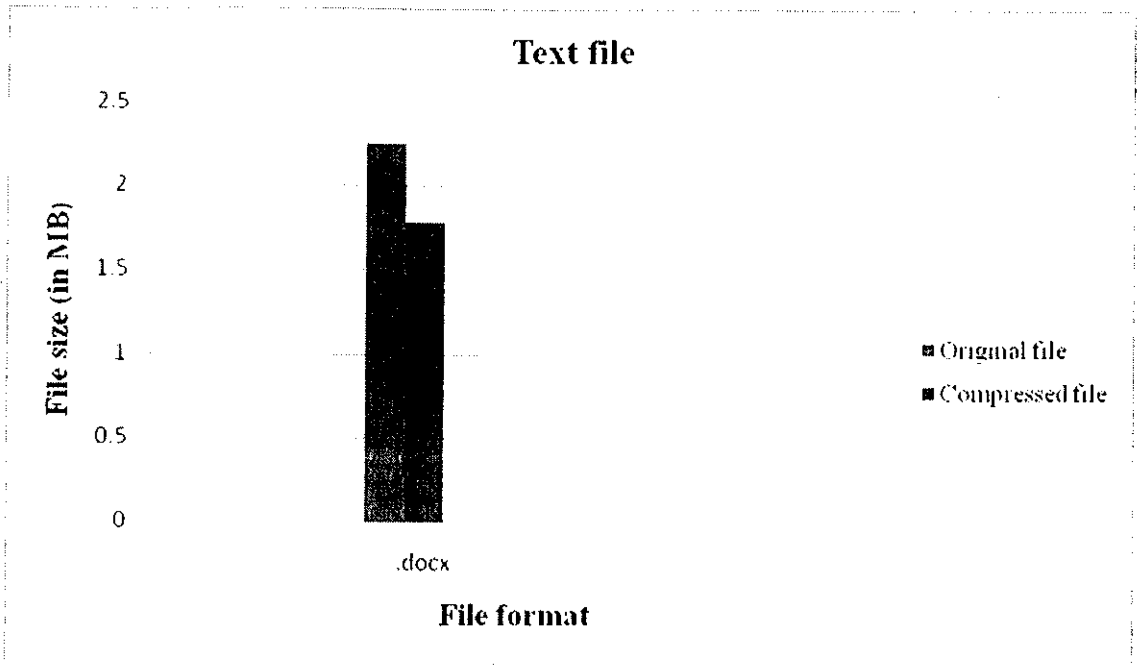
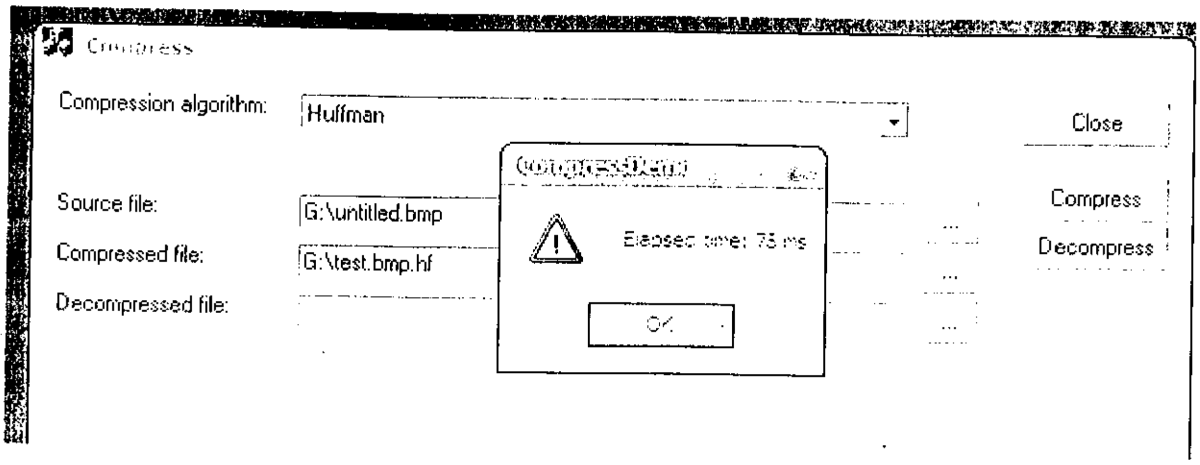


Fig.10.1.6 Comparison between original and compressed file.

10.2 Image file


Compress

Compression algorithm: Huffman

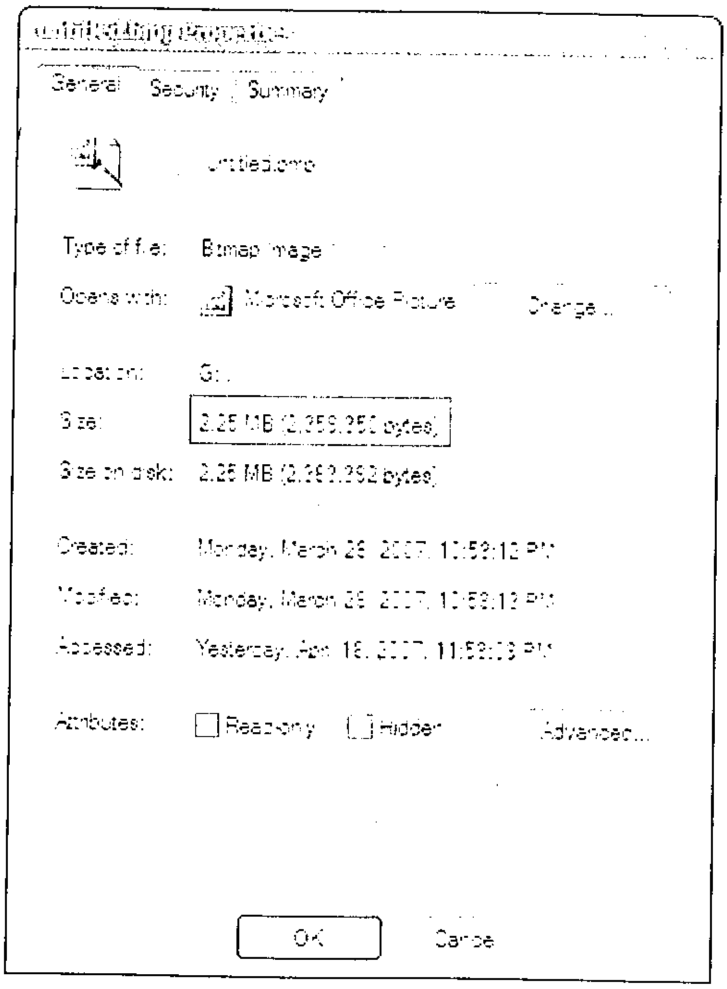
Source file: G:\untitled.bmp

Compressed file: G:\test.bmp.hf

Decompressed file:

Elapsed time: 78 ms

Fig.10.2.1 we compressed the image file of .bmp extension over here


untitled.bmp Properties

General | Security | Summary

untitled.bmp

Type of file: Bitmap Image

Opens with: Microsoft Office Picture

Location: G:\

Size: 2.25 MB (2,359,350 bytes)

Size on disk: 2.25 MB (2,363,392 bytes)

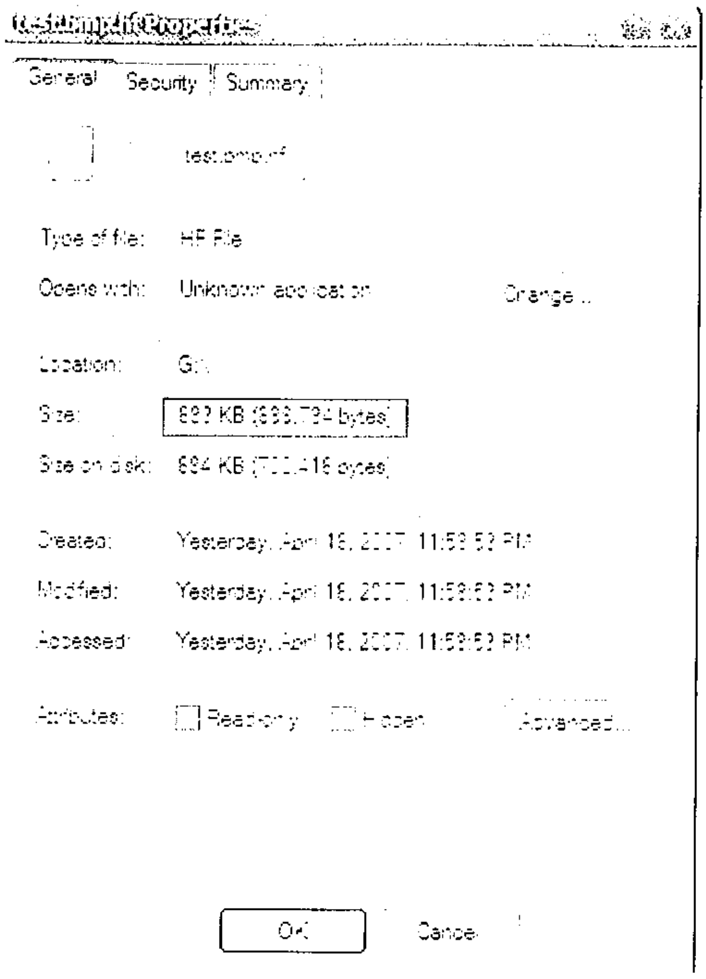
Created: Monday, March 26, 2007, 10:53:12 PM

Modified: Monday, March 26, 2007, 10:53:12 PM

Accessed: Yesterday, April 18, 2007, 11:59:03 PM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.2.2 Highlighted text shows the original size of the image file



test.bmp.hf Properties

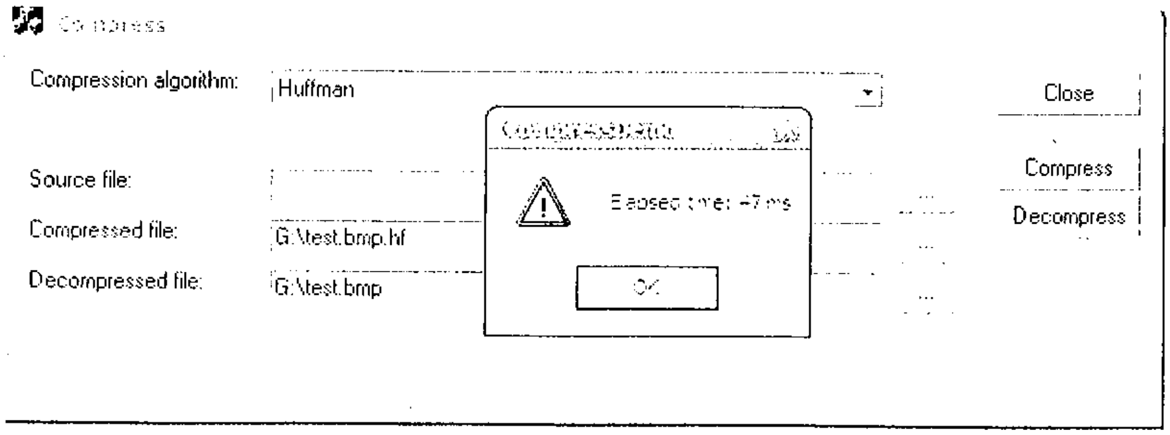
General | Security | Summary

test.bmp.hf

Type of file: HF File
Opens with: Unknown application Change...
Location: G:\
Size: 693 KB (699,794 bytes)
Size on disk: 694 KB (700,416 bytes)
Created: Yesterday, April 18, 2007, 11:59:53 PM
Modified: Yesterday, April 18, 2007, 11:59:53 PM
Accessed: Yesterday, April 18, 2007, 11:59:53 PM
Attributes: ☐ Read-only ☐ Hidden Advanced...

OK Cancel

Fig.10.2.3 Highlighted text shows the size after the compression of the bmp image file



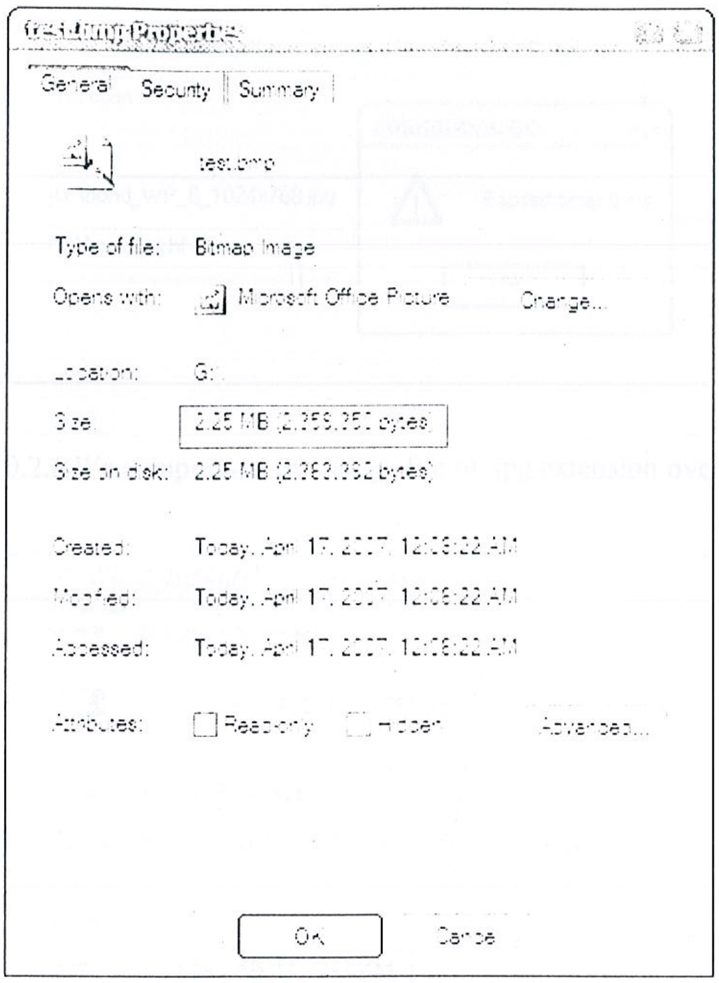
Compress

Compression algorithm: Huffman
Source file:
Compressed file: G:\test.bmp.hf
Decompressed file: G:\test.bmp

Close
Compress
Decompress

Compression
Elapsed time: 47 ms
OK

Fig.10.2.4 The same file was decompressed



test.bmp Properties

General | Security | Summary

test.bmp

Type of file: Bitmap Image

Opens with: Microsoft Office Picture Change...

Location: G:\

Size: 2.25 MB (2,359,350 bytes)

Size on disk: 2.25 MB (2,363,392 bytes)

Created: Today, April 17, 2007, 12:08:22 AM

Modified: Today, April 17, 2007, 12:08:22 AM

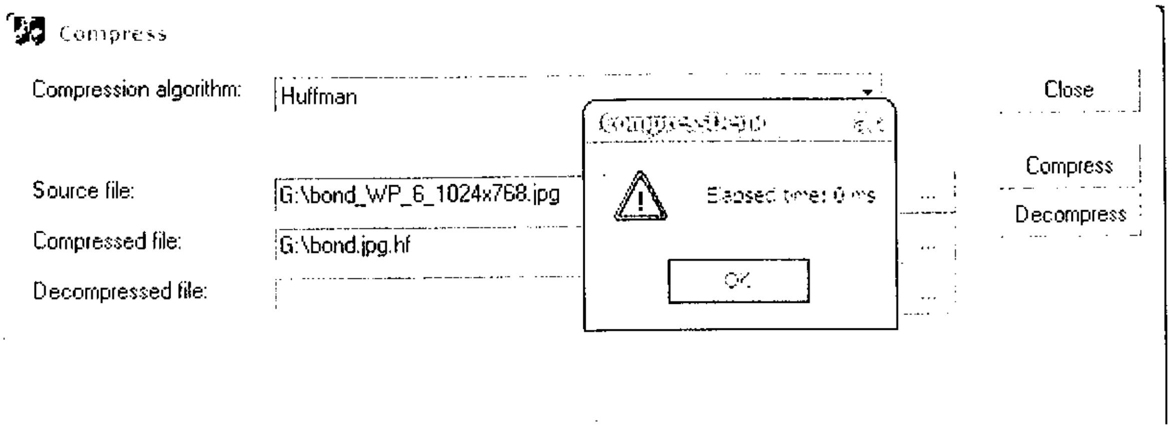
Accessed: Today, April 17, 2007, 12:08:22 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

OK Cancel

Fig10.2.5 we got the original file with the same size as the size was not decreased it showed Lossless compression.

JPG file (encoded)


Compress

Compression algorithm: Huffman

Source file: G:\bond_WP_6_1024x768.jpg

Compressed file: G:\bond.jpg.hf

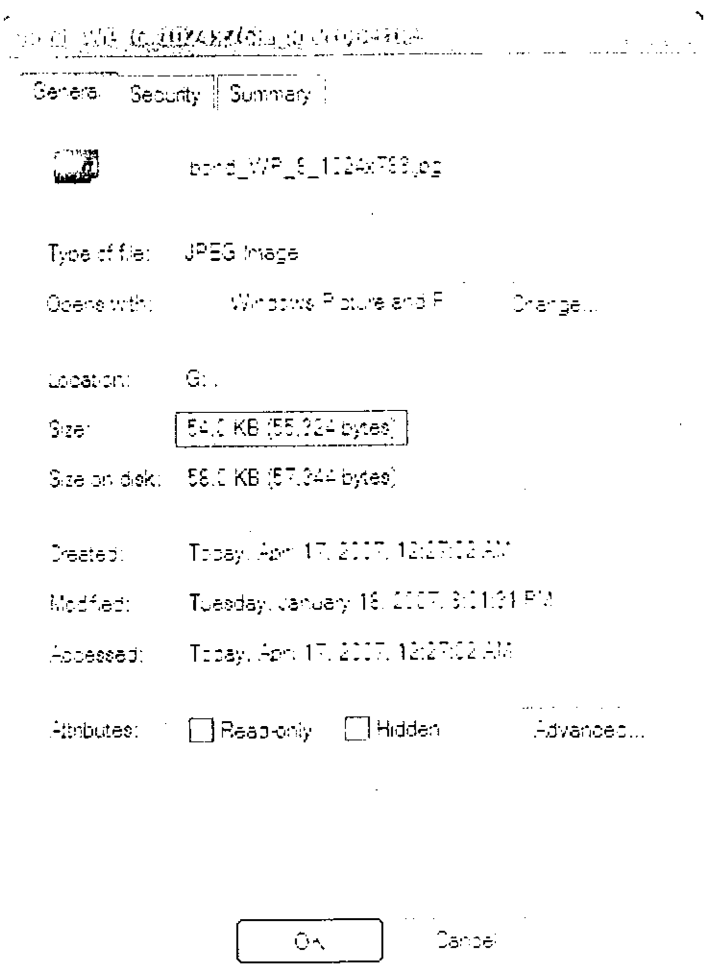
Decompressed file:

Compression

Elapsed time: 0 ms

OK

Fig 10.2.6 We compressed the image file of .jpg extension over here


bond_WP_6_1024x768.jpg Properties

General | Security | Summary

bond_WP_6_1024x768.jpg

Type of file: JPEG Image

Opens with: Windows Picture and F Change...

Location: G:\

Size: 54.0 KB (55,324 bytes)

Size on disk: 56.0 KB (57,344 bytes)

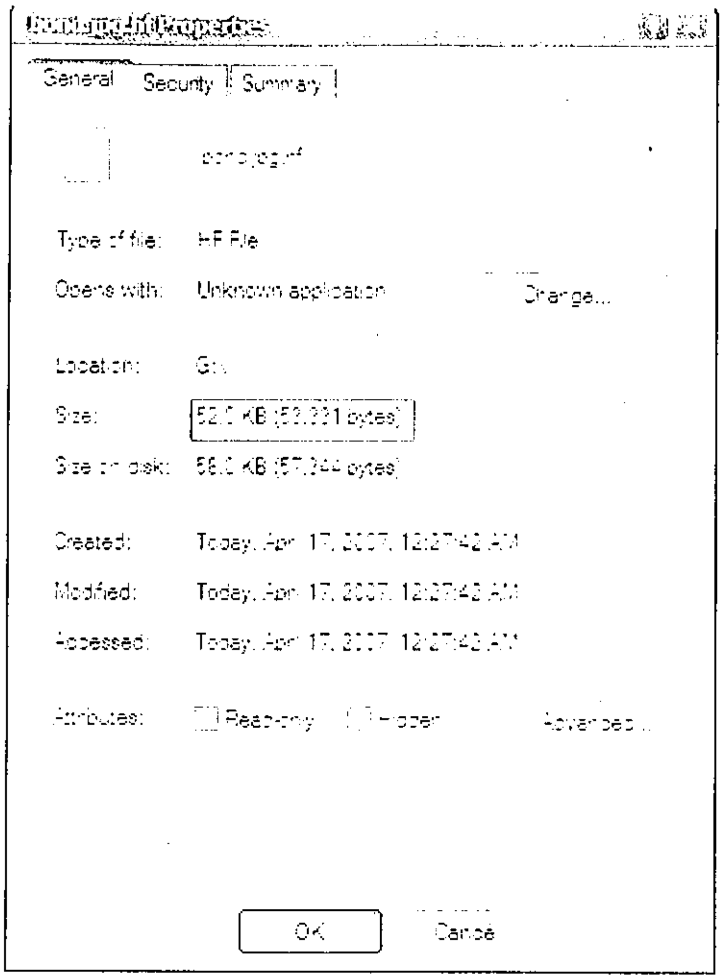
Created: Today, April 17, 2007, 12:27:02 AM

Modified: Tuesday, January 16, 2007, 9:01:31 PM

Accessed: Today, April 17, 2007, 12:27:02 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

Fig.10.2.7Highlighted text shows the original size of the image file



bond.jpg.hf Properties

General | Security | Summary

bond.jpg.hf

Type of file: HF File

Opens with: Unknown application

Location: G:\

Size: 52.0 KB (53,221 bytes)

Size on disk: 56.0 KB (57,344 bytes)

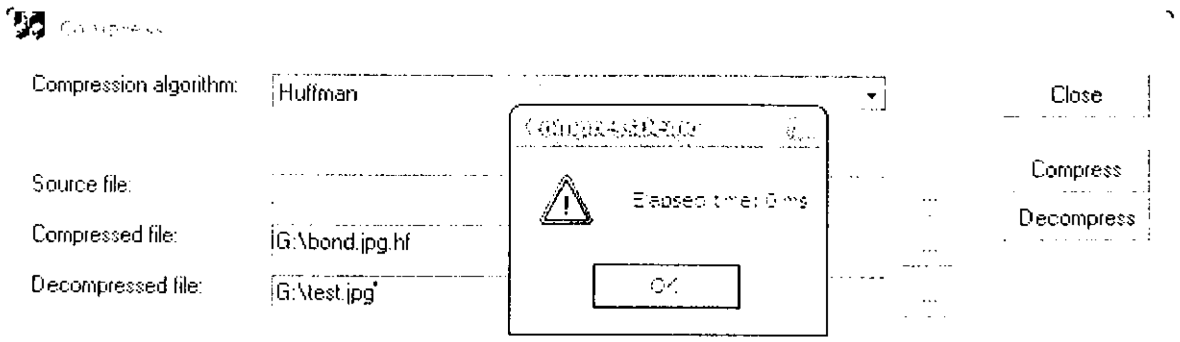
Created: Today, April 17, 2007, 12:27:42 AM

Modified: Today, April 17, 2007, 12:27:42 AM

Accessed: Today, April 17, 2007, 12:27:42 AM

Attributes: Read-only Hidden

Fig.10.2.8 Highlighted text shows the size after the compression of the jpg image file



Compress

Compression algorithm: Huffman

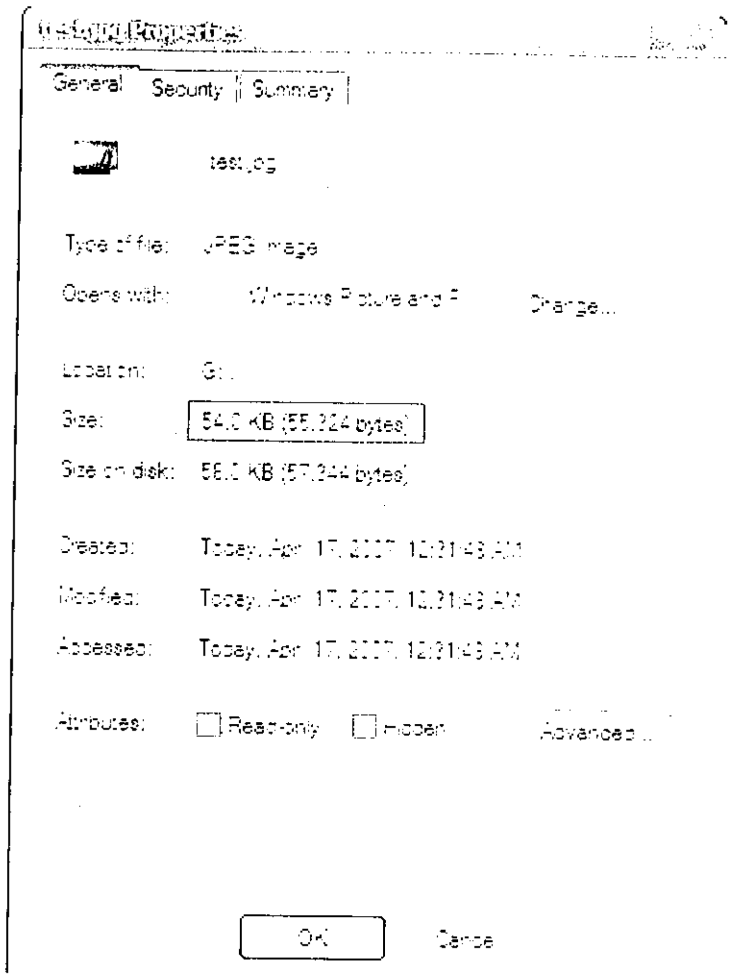
Source file:

Compressed file: G:\bond.jpg.hf

Decompressed file: G:\test.jpg

Elapsed time: 0 ms

Fig.10.2.9 The same file was decompressed



[illegible] Properties

General | Security | Summary

test.jpg

Type of file: JPEG image

Opens with: Windows Picture and F Change...

Location: G:\

Size: 54.0 KB (55,324 bytes)

Size on disk: 56.0 KB (57,344 bytes)

Created: Today, Apr 17, 2007, 12:31:43 AM

Modified: Today, Apr 17, 2007, 12:31:43 AM

Accessed: Today, Apr 17, 2007, 12:31:43 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

OK Cancel

Fig.10.2.10 we got the original file with the same size as the size was not decreased it showed Lossless compression.

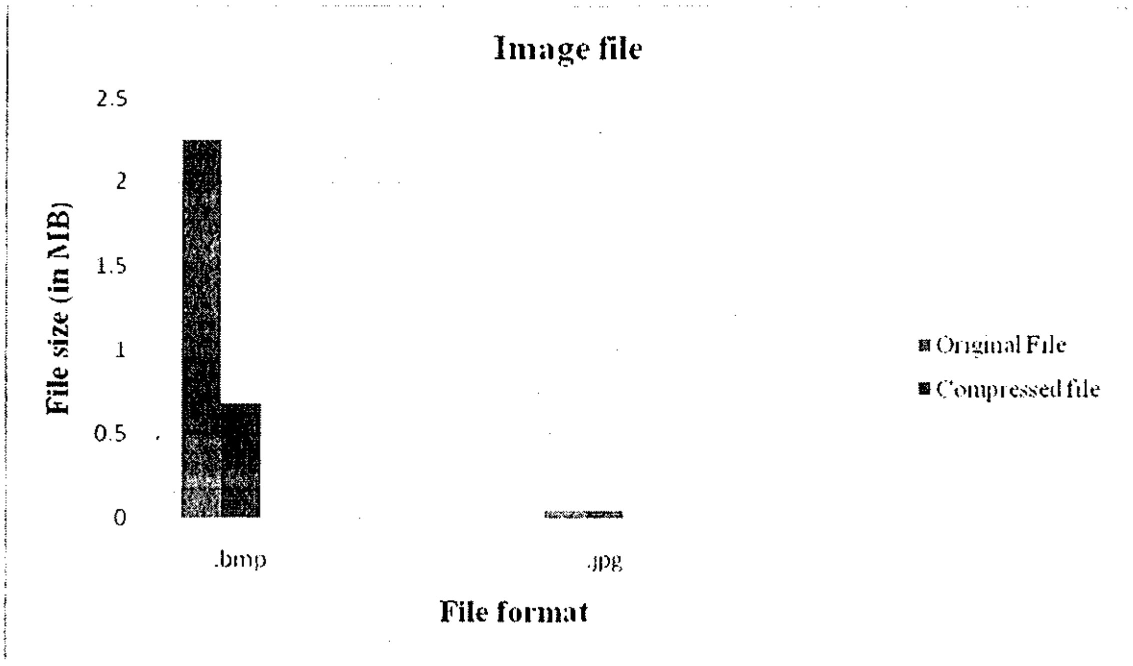
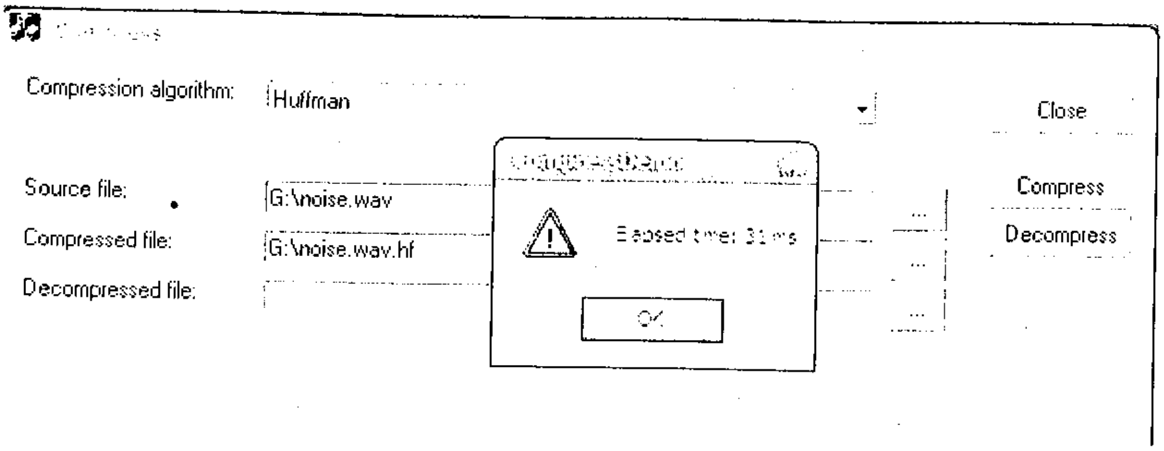


Fig.10.2.11 Comparison between original and compressed file.

10.3 Audio file of .wav format


Compression algorithm: Huffman

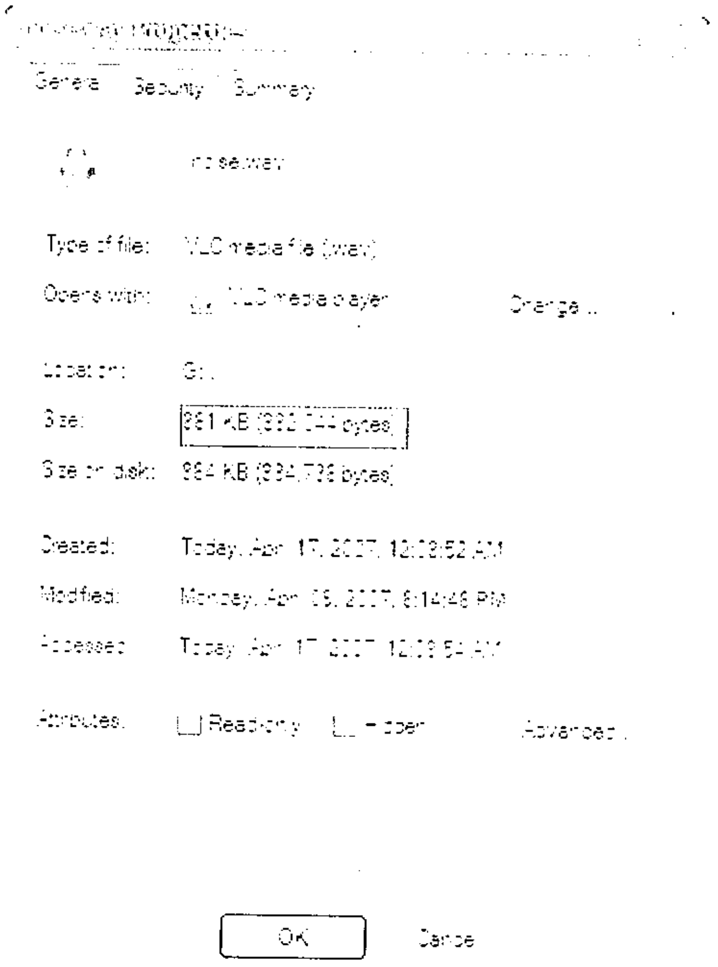
Source file: G:\noise.wav

Compressed file: G:\noise.wav.hf

Decompressed file:

Elapsed time: 31 ms

Fig.10.3.1 we compressed the audio file of .wav extension over here



noise.wav

Type of file: VLC media file (.wav)

Opens with: VLC media player

Location: G:\

Size: 981 KB ([illegible] bytes)

Size on disk: 984 KB ([illegible] bytes)

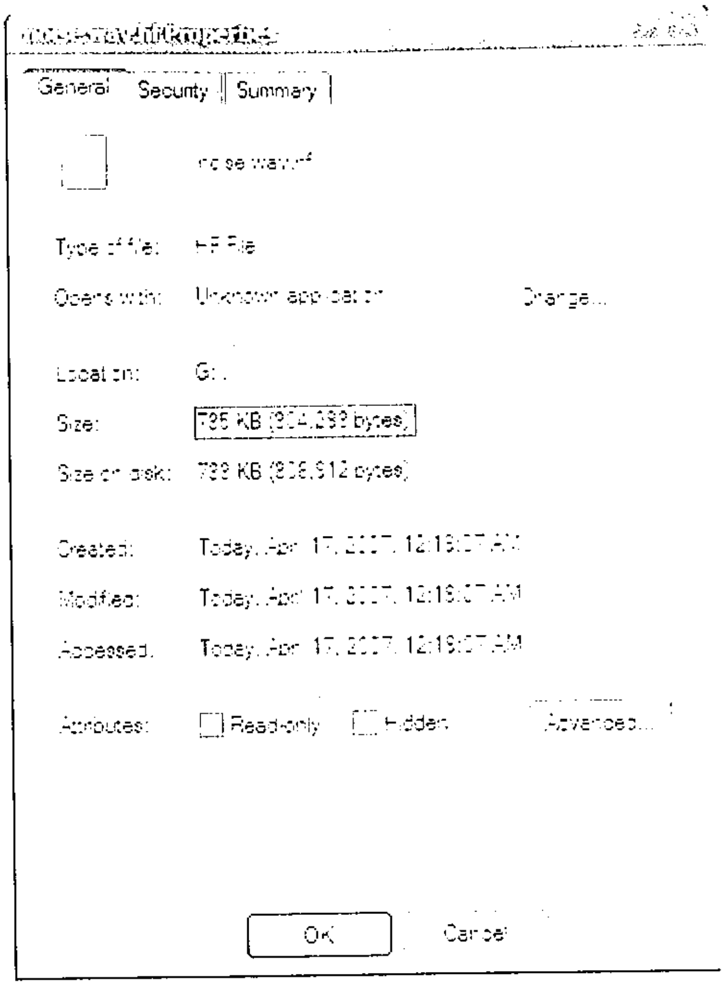
Created: Today, Apr 17, 2017, 12:09:52 AM

Modified: Monday, Apr 03, 2017, 8:14:46 PM

Accessed: Today, Apr 17, 2017, 12:09:54 AM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.3.2 Highlighted text shows the original size of the audio file



noise.wav.hf Properties

General | Security | Summary

noise.wav.hf

Type of file: HF File

Opens with: Unknown application Change...

Location: G:\

Size: 785 KB (804,293 bytes)

Size on disk: 788 KB (806,912 bytes)

Created: Today, April 17, 2007, 12:19:07 AM

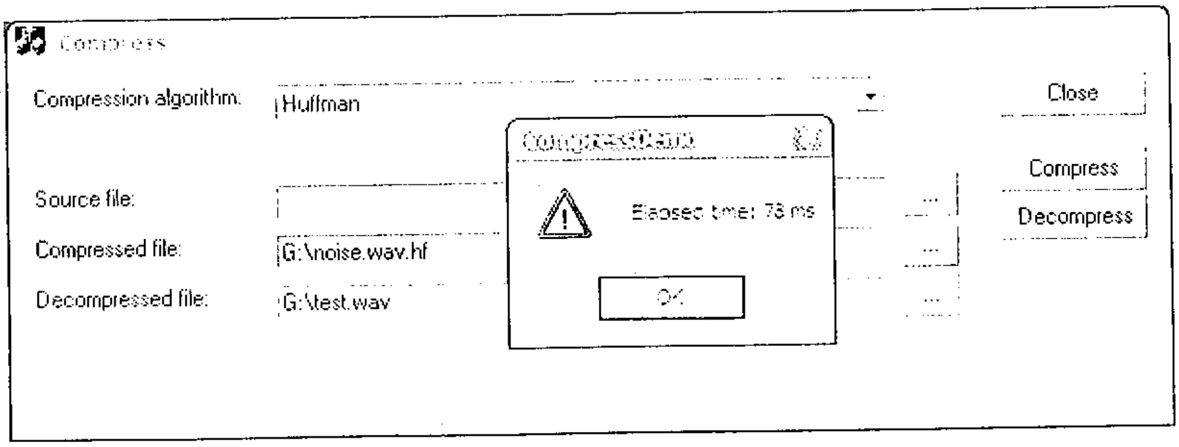
Modified: Today, April 17, 2007, 12:19:07 AM

Accessed: Today, April 17, 2007, 12:19:07 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

OK Cancel

Fig.10.3.3 Highlighted text shows the size after the compression of the .wav audio file



Compress

Compression algorithm: Huffman

Source file:

Compressed file: G:\noise.wav.hf

Decompressed file: G:\test.wav

Close

Compress

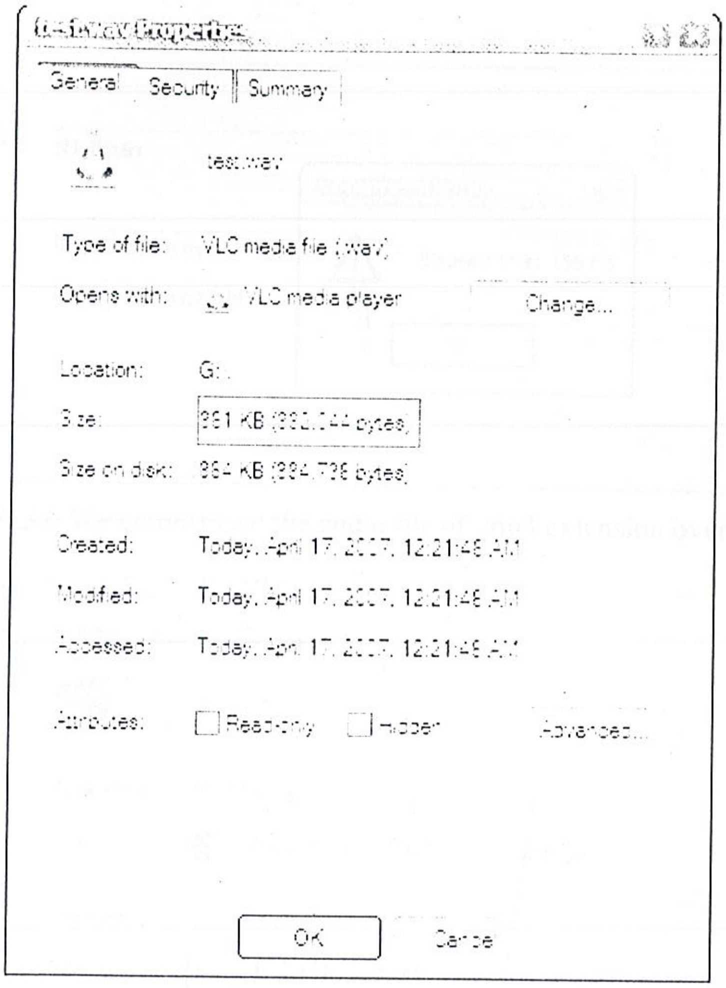
Decompress

Compression

Elapsed time: 78 ms

OK

Fig.10.3.4 The same file was decompressed



test.wav Properties

General | Security | Summary

test.wav

Type of file: VLC media file (.wav)

Opens with: VLC media player

Location: G:\

Size: 391 KB (400,044 bytes)

Size on disk: 364 KB (394,738 bytes)

Created: Today, April 17, 2007, 12:21:46 AM

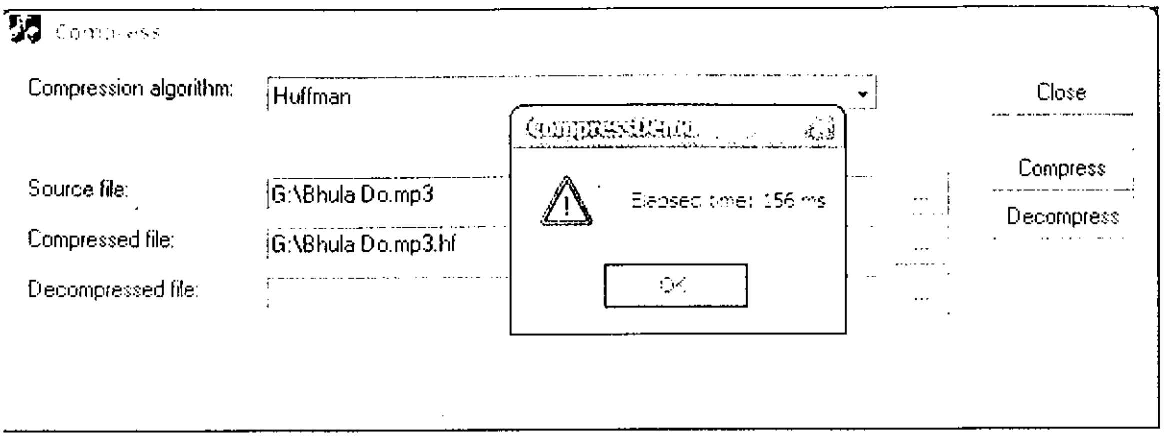
Modified: Today, April 17, 2007, 12:21:46 AM

Accessed: Today, April 17, 2007, 12:21:46 AM

Attributes: ☐ Read-only ☐ Hidden

Fig.10.3.5 we got the original file with the same size as the size was not decreased it showed Lossless compression.

MP3



Compress

Compression algorithm: Huffman

Source file: G:\Bhula Do.mp3

Compressed file: G:\Bhula Do.mp3.hf

Decompressed file:

Close

Compress

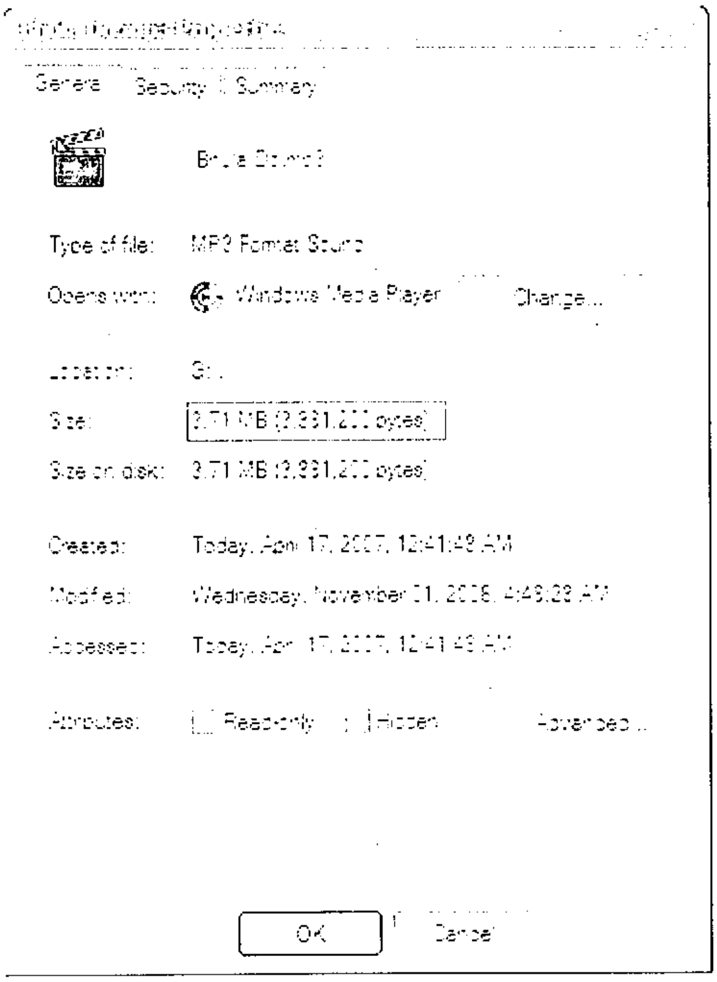
Decompress

CompressDemo

Elapsed time: 156 ms

OK

Fig10.3.6 We compressed the audio file of .mp3 extension over here



Bhula Do.mp3 Properties

General Security Summary

Bhula Do.mp3

Type of file: MP3 Format Sound

Opens with: Windows Media Player Change...

Location: G:\

Size: 3.71 MB (3,891,200 bytes)

Size on disk: 3.71 MB (3,891,200 bytes)

Created: Today, April 17, 2007, 12:41:49 AM

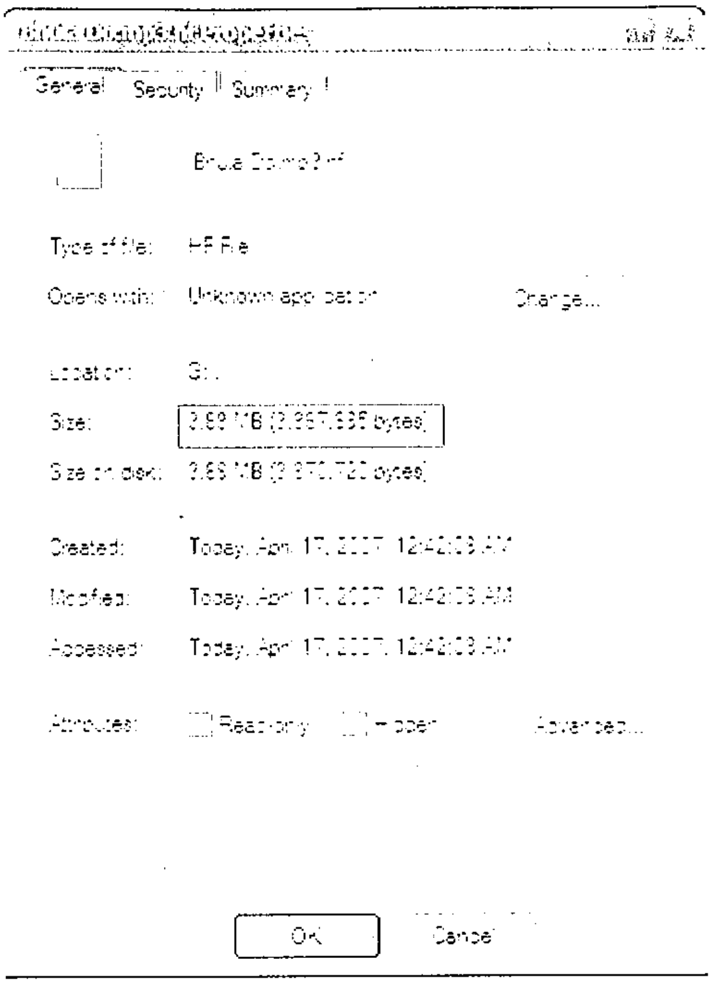
Modified: Wednesday, November 01, 2006, 4:49:23 AM

Accessed: Today, April 17, 2007, 12:41:49 AM

Attributes: Read-only Hidden Advanced...

OK Cancel

Fig.10.3.7 Highlighted text shows the original size of the audio file


[illegible]

General Security Summary

Bhula Do.mp3.hf

Type of file: HF File

Opens with: Unknown application Change...

Location: G:\

Size: 3.89 MB (3,957,535 bytes)

Size on disk: 3.89 MB (3,970,720 bytes)

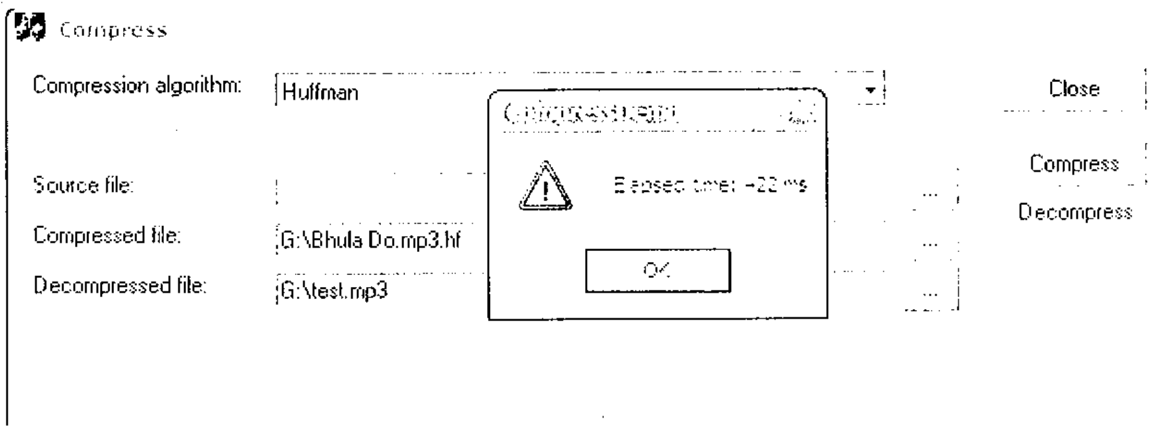
Created: Today, April 17, 2007, 12:42:03 AM

Modified: Today, April 17, 2007, 12:42:03 AM

Accessed: Today, April 17, 2007, 12:42:03 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

Fig.10.3.8 Highlighted text shows the size after the compression of the .mp3 audio file


Compress

Compression algorithm: Huffman

Source file:

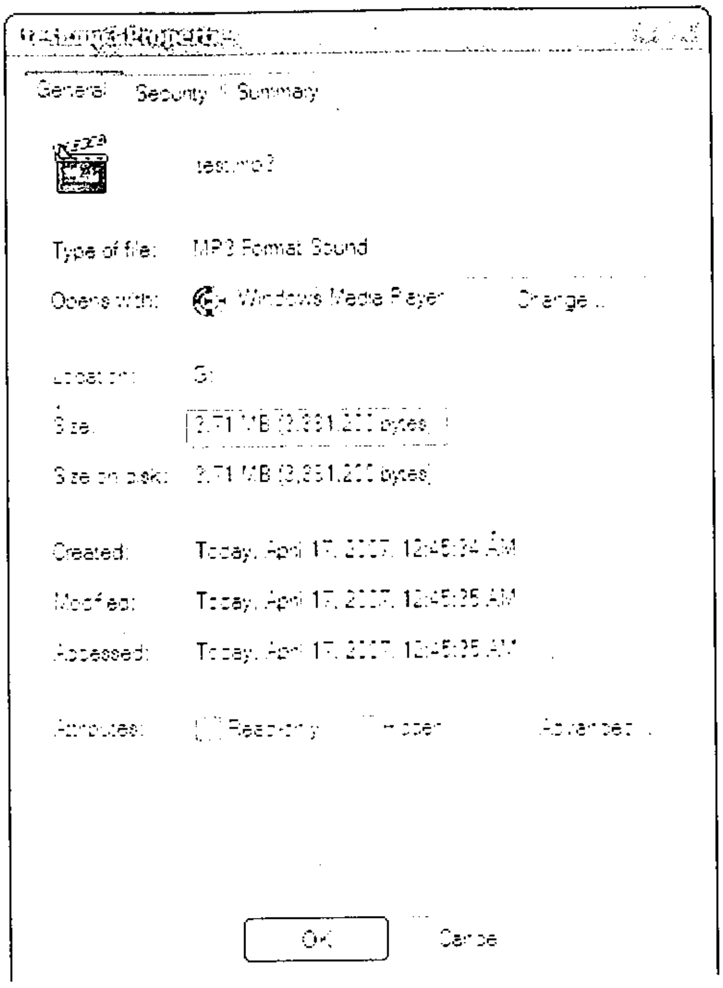
Compressed file: G:\Bhula Do.mp3.hf

Decompressed file: G:\test.mp3

[illegible]

Elapsed time: 422 ms

Fig.10.3.9 The same file was decompressed



test.mp3

Type of file:	MP3 Format Sound
Opens with:	Windows Media Player
Location:	G:\
Size:	3.71 MB (3,891,200 bytes)
Size on disk:	3.71 MB (3,891,200 bytes)
Created:	Today, April 17, 2007, 12:45:34 AM
Modified:	Today, April 17, 2007, 12:45:35 AM
Accessed:	Today, April 17, 2007, 12:45:35 AM
Attributes:	Read-only Hidden

Fig.10.3.10 we got the original file with the same size as the size was not decreased it showed Lossless compression.

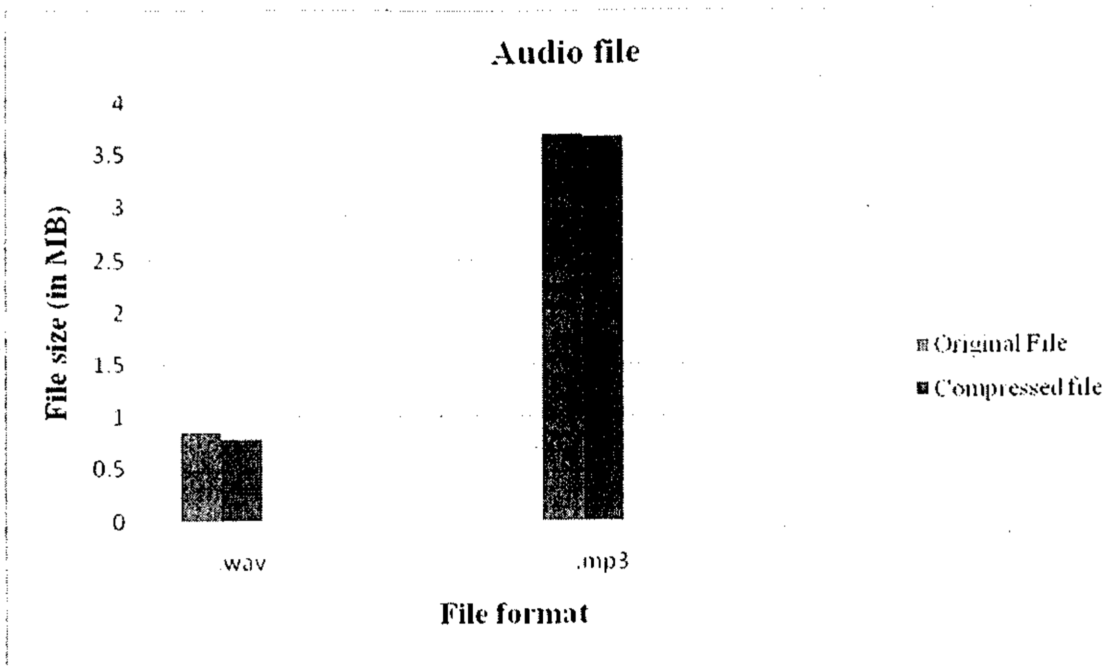
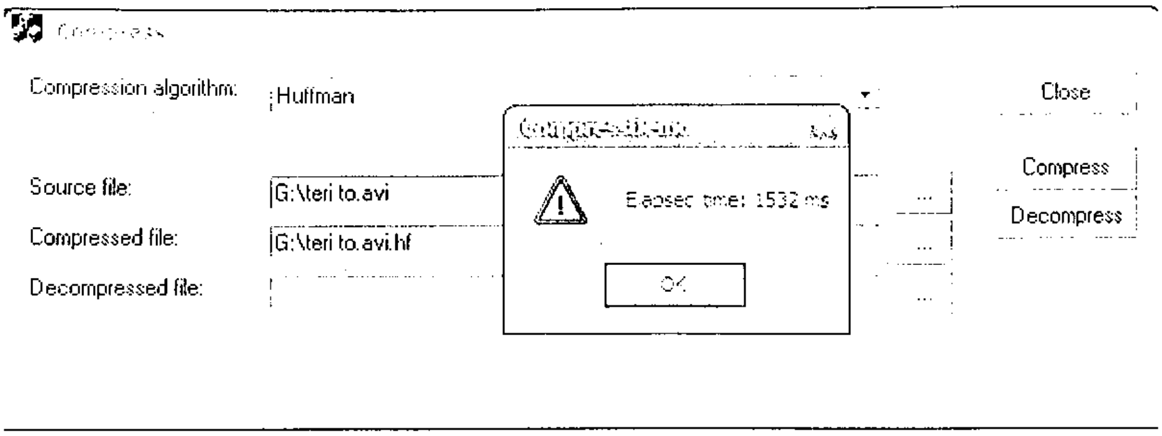


Fig.10.3.11 Comparison between original and compressed file.

10.4 Video file


Compress

Compression algorithm: Huffman

Source file: G:\teri to.avi

Compressed file: G:\teri to.avi.hf

Decompressed file:

Compress/Decompress

Elapsed time: 1532 ms

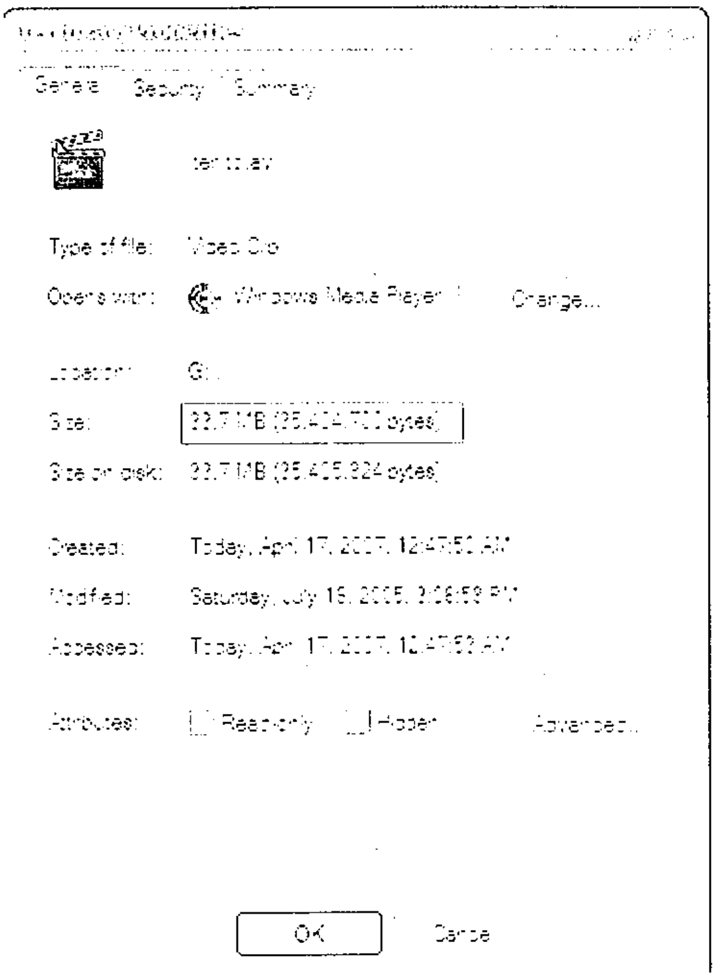
OK

Close

Compress

Decompress

Fig.10.4.1 we compressed the video file of .avi extension over here


[illegible]

General Security Summary

teri to.avi

Type of file: Video Clip

Opens with: Windows Media Player Change...

Location: G:\

Size: 33.7 MB (35,404,700 bytes)

Size on disk: 33.7 MB (35,405,824 bytes)

Created: Today, April 17, 2007, 12:47:50 AM

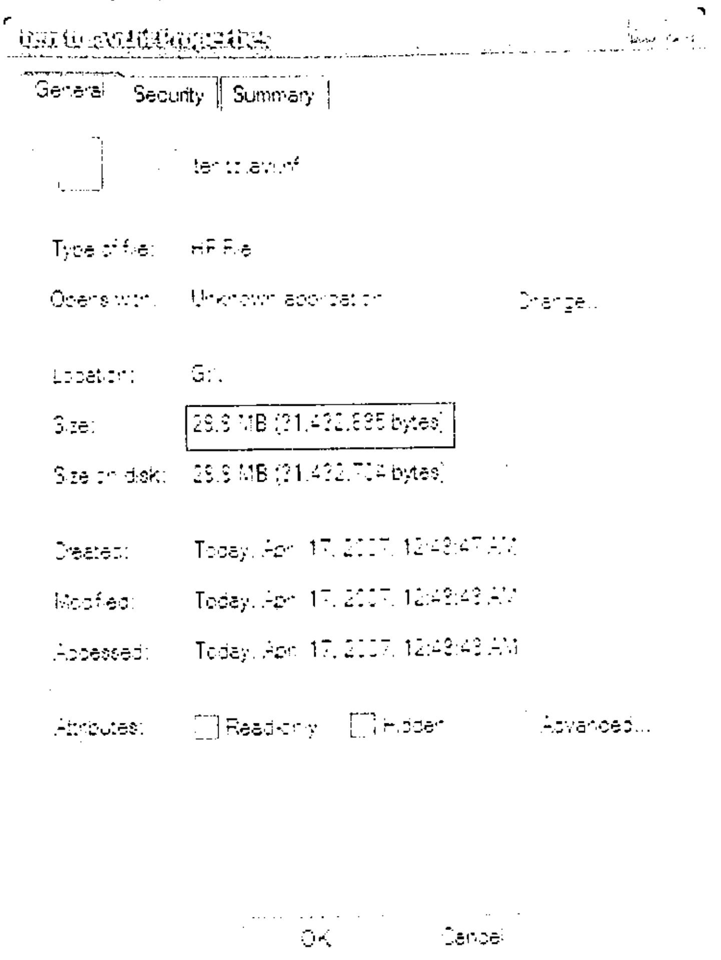
Modified: Saturday, July 16, 2005, 3:08:59 PM

Accessed: Today, April 17, 2007, 12:47:53 AM

Attributes: ☐ Read-only ☐ Hidden Advanced...

OK Cancel

Fig.10.4.2 Highlighted text shows the original size of the video file



teri to.avi.hf

Type of file: HF File

Opens with: Unknown application

Location: G:\

Size: 29.9 MB (31,432,695 bytes)

Size on disk: 29.9 MB (31,432,704 bytes)

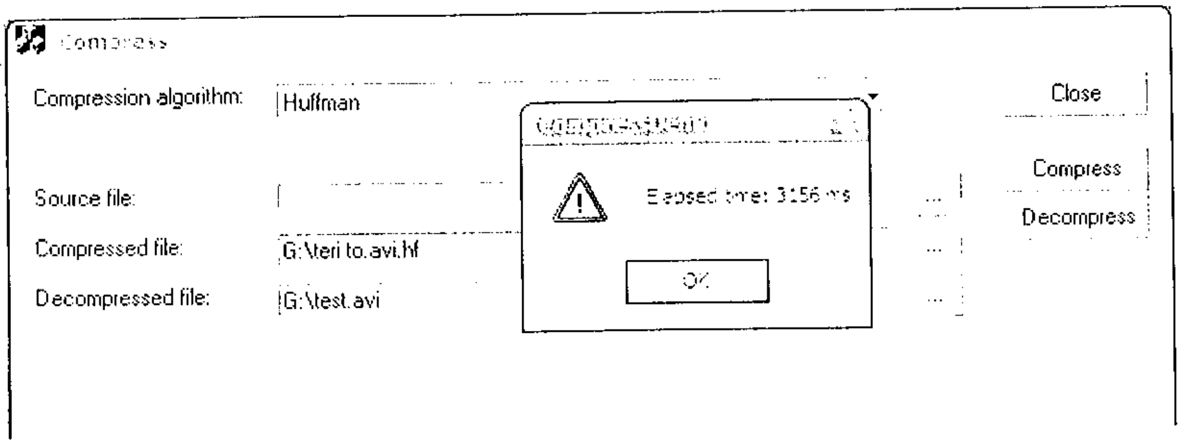
Created: Today, April 17, 2007, 12:49:47 AM

Modified: Today, April 17, 2007, 12:49:49 AM

Accessed: Today, April 17, 2007, 12:49:49 AM

Attributes: [ ] Read-only [ ] Hidden

Fig.10.4.3 Highlighted text shows the size after the compression of the .avi video file



Compress

Compression algorithm: Huffman

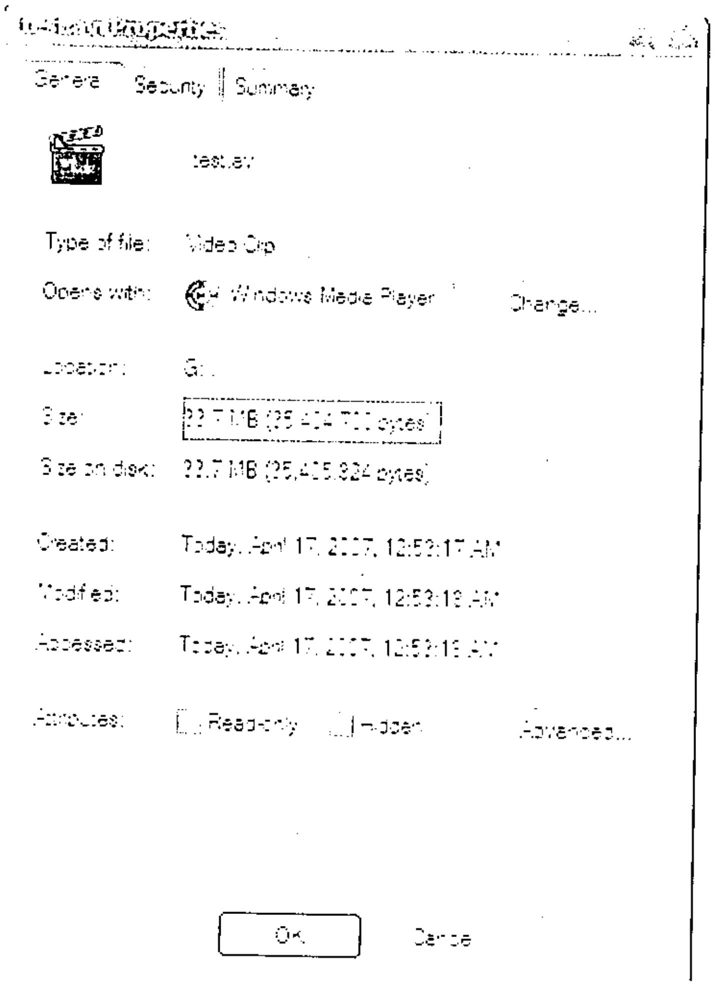
Source file:

Compressed file: G:\teri to.avi.hf

Decompressed file: G:\test.avi

Elapsed time: 3156 ms

Fig.10.4.4 The same file was decompressed



[illegible]

General | Security | Summary

test.avi

Type of file: Video Clip

Opens with: Windows Media Player Change...

Location: G:\

Size: 22.7 MB (25,404,700 bytes)

Size on disk: 22.7 MB (25,405,324 bytes)

Created: Today, April 17, 2007, 12:53:17 AM

Modified: Today, April 17, 2007, 12:53:19 AM

Accessed: Today, April 17, 2007, 12:53:19 AM

Attributes: [ ] Read-only [ ] Hidden Advanced...

OK Cancel

Fig.10.4.5 we got the original file with the same size as the size was not decreased it showed Lossless compression.

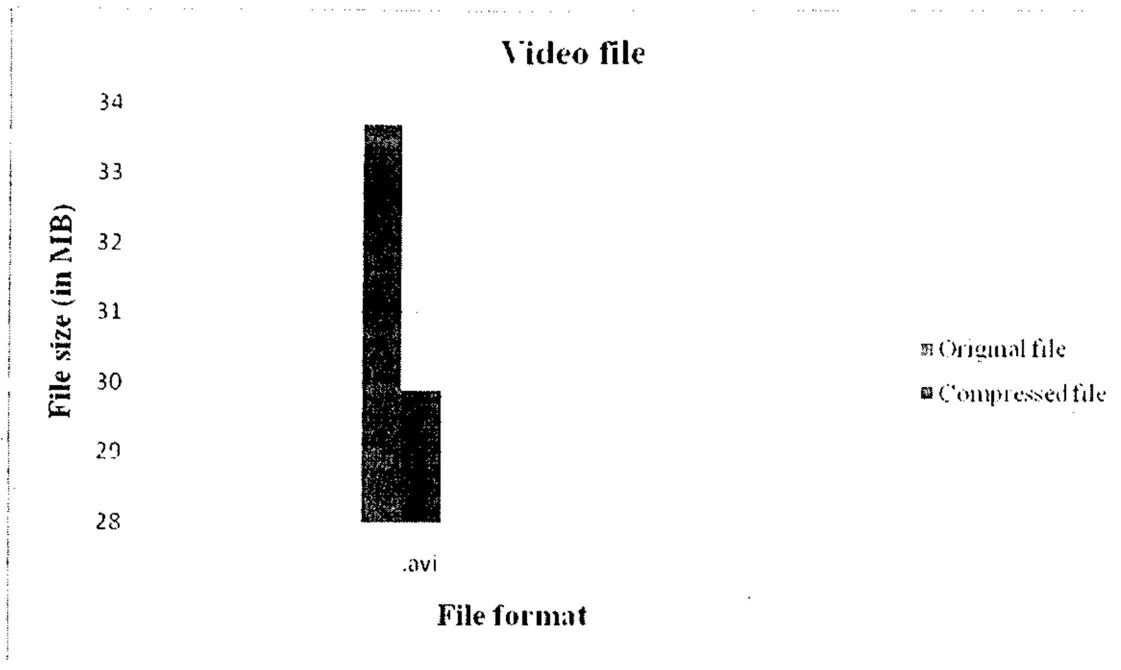


Fig.10.4.6 Comparison between original and compressed file.

11 Result of test cases and compression ratio

File format	Original File	Compressed file	Compression ratio (in %)
.docx	2.55 MB	1.78 MB	30.19 %
.bmp	2.25 MB	683 KB	69.73 %
.jpg	54 KB	52 KB	3.7 %
.wav	861 KB	785 KB	8.82 %
.mp3	3.71 MB	3.68 MB	0.806 %
.avi	33.7 MB	29.9 MB	11.27 %

Table 11.1: Compression ratio

Where, Compression ratio = ((original size – compressed size)÷(original size)) *100

File format	Original File	Compressed file	Compression ratio
.docx	2.55 MB	1.78 MB	1.468
.bmp	2.25 MB	683 KB	3.295
.jpg	54 KB	52 KB	0.019
.wav	861 KB	785 KB	1.096
.mp3	3.71 MB	3.68 MB	1.008
.avi	33.7 MB	29.9 MB	1.127

Table 11.2: Compression ratio

Where, Compression ratio = (original size ÷ compressed size)

12 Comparison of compression ratio with commercial applications

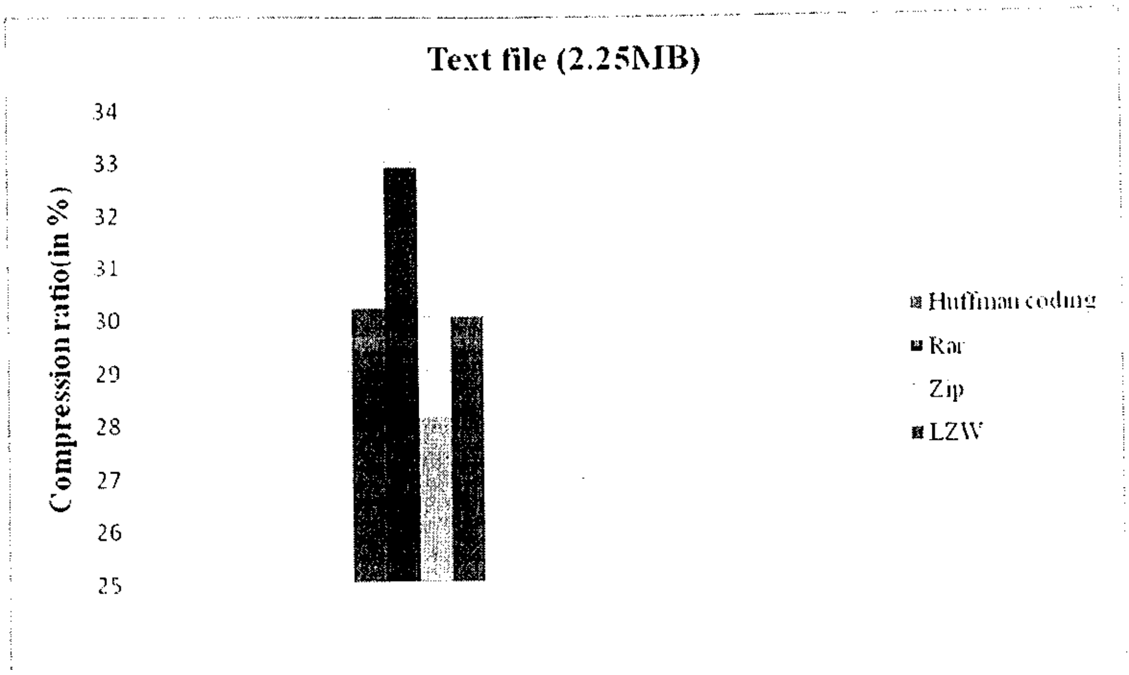


Fig.12.1

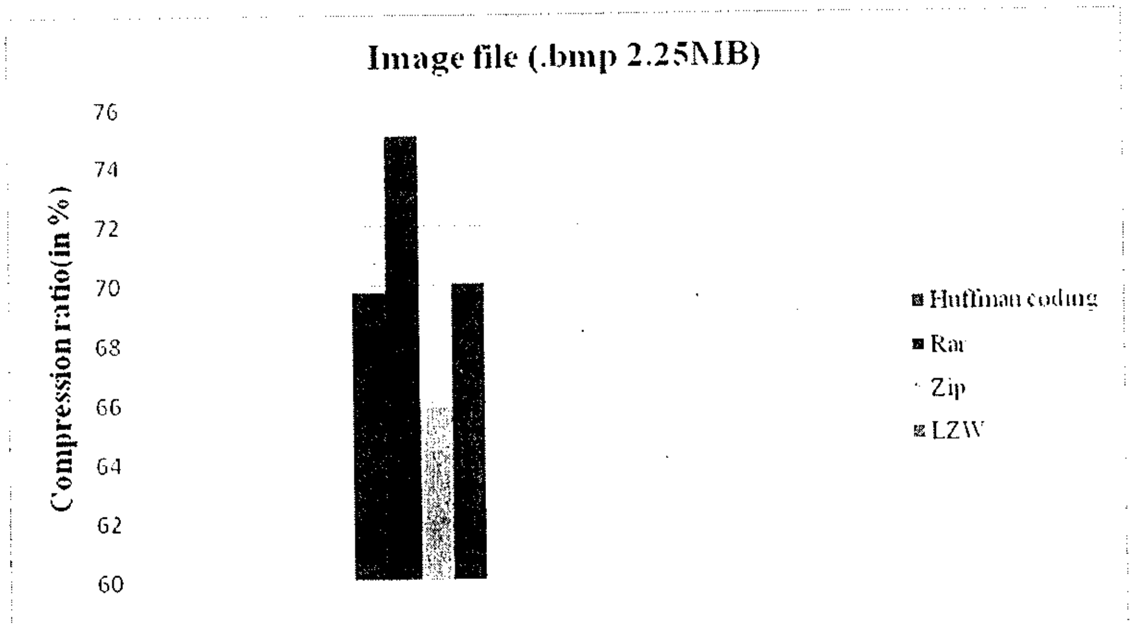


Fig.12.2

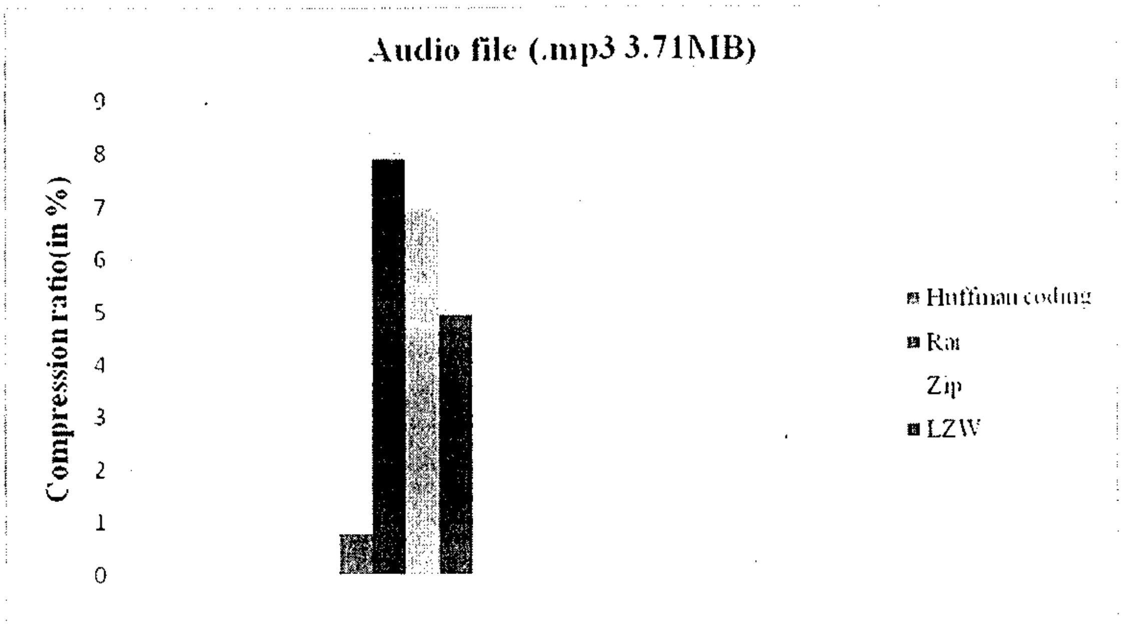


Fig12.3

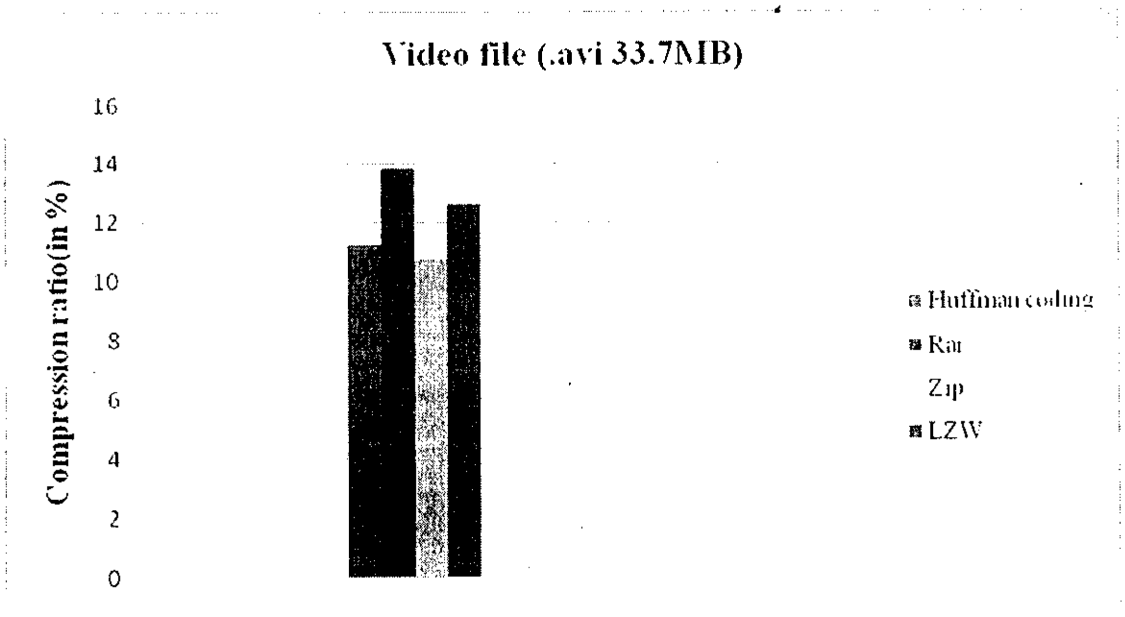


Fig12.4

13 Time comparison with commercial applications

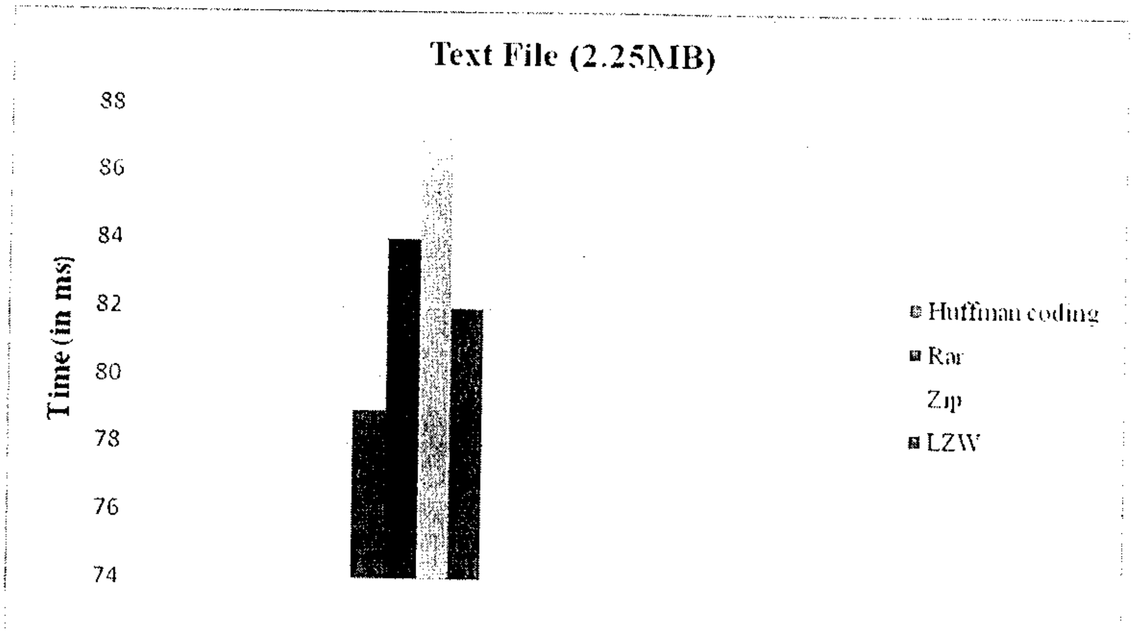


Fig.13.1

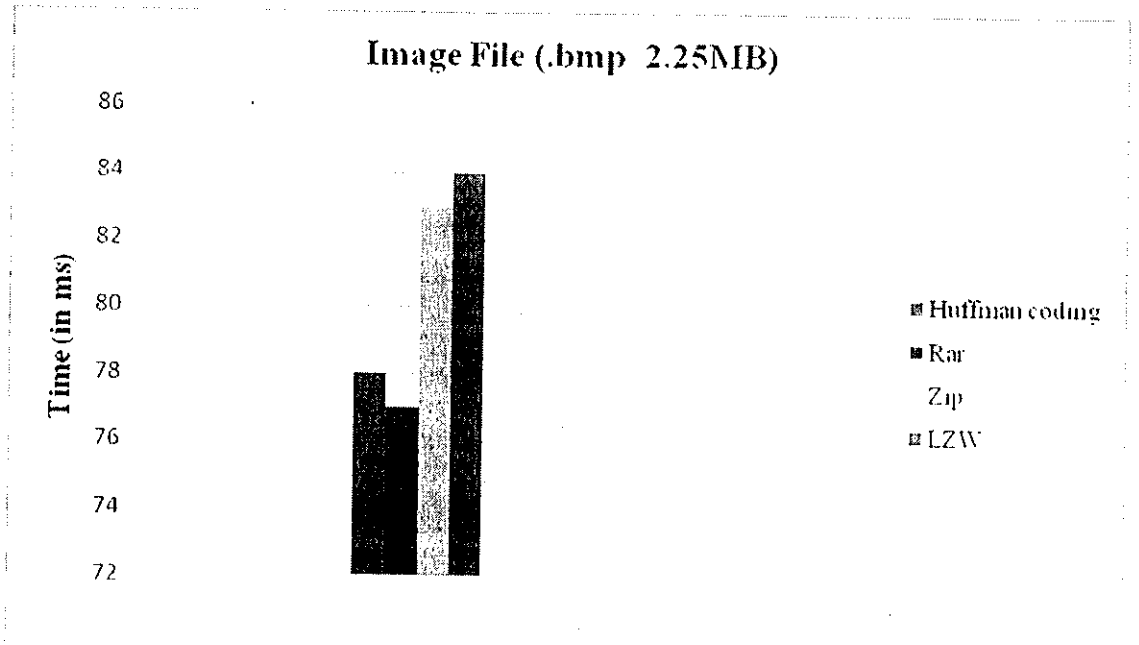


Fig.13.2

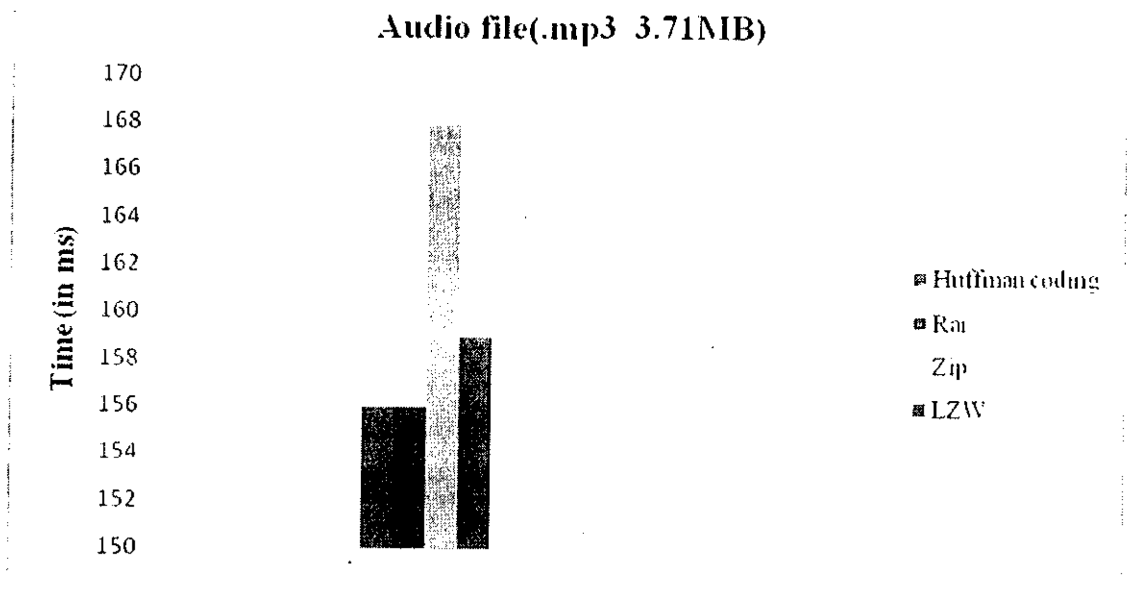


Fig.13.3

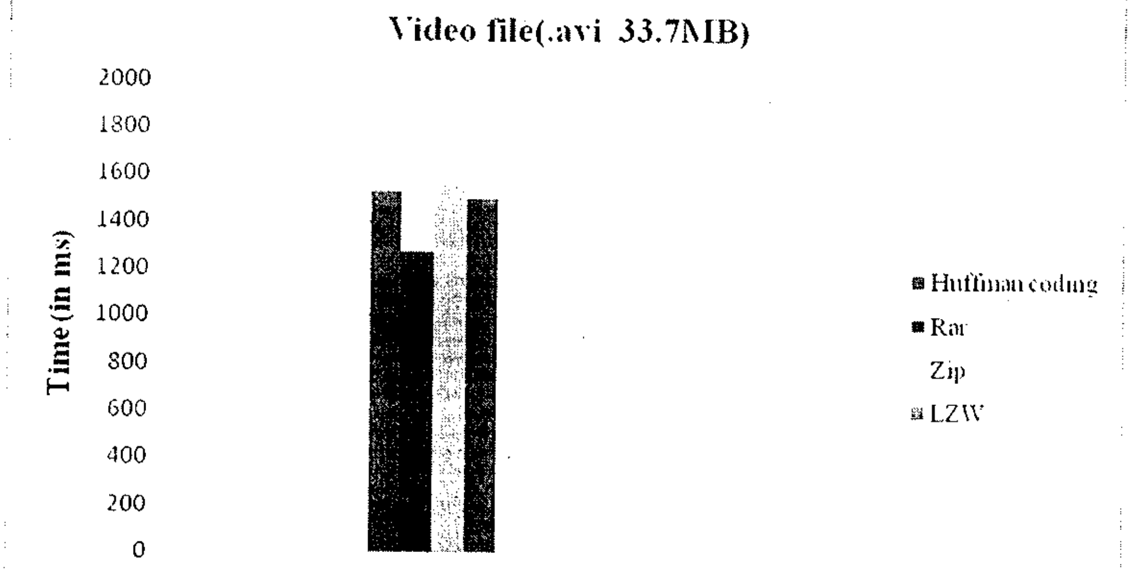


Fig.13.4

14 Limitations and issues

There are few limitations and issues associated with the developed application which are as follows.

14.1 Platform Issue

This issue remains in the mind that whether it will work on all the version of windows such as Windows 98/NT/2003 as the operating system used while designing we used Windows XP.

14.2 Algorithm Issue

As we will implemented only one algorithm for all type of file formats i.e. audio, video and text files. Compression ratio of audio, video files is much less as compared to text files because of the reason that audio and video files are digitally encoded. Also the time taken to compress and decompress audio and video files suffers due to the very same reasons.

14.3 Performance Issue

The designed application achieves the goal of both compression and decompression for various type of file formats as mentioned above in algorithm issue .But it may happen that designed application may be slower as compared to the other commercial applications available in the same field of Data compression toolkit.

14.4 Limitations and Miscellaneous Issues

The designed application does not have its own extension for compressed and decompressed files. We have to specify the name of the targeted file while performing compression and decompression.

15 Conclusion

The software performs the basic task of compression and decompression successfully but there is still scope for improvement in terms of additional features, various complexities etc. The scope and scale of this application can be extended by future developers. It can be used by both professional and home users according to their own wish.

16 Future scope of work

Storing data has become one of the most important aspects for growth of any organization belonging to any industry. Research is done constantly in the field of compression to get better compression ratio and time complexity. The future scopes of developed application are as follows:

1. The developed application can be use widely in offices and institutes, as the main data used for day-to-day work consist of text mainly. Since the developed application uses Huffman compression algorithm which is considered as one of the best algorithm for text compression. Thus developed application can be used for storing data in offices and institutes in compressed form.
2. The developed application can also be used in various biological researches, the data that is used to predict the various inventions can be stored in compressed form by the developed software. For instance storing of various DNA and RNA sequences in compressed files.
3. The developed application can be used by both home and professional users for both compression and decompression purposes.
4. By doing some modification(by adding various features like split file archieve, developing an extension for compressed files) the developed application can be made to compete with commercial software's like WinZip, WinRar and 7-zip etc.
5. The developed application can be used to build better and fast file sharing internet sites once the developed software has made its hold in the market.
6. By doing slight changes in the source code .the application can be made to run on other operating systems also.

17 Bibliography

Books

1. Sayood, Khalid. *Introduction to Data Compression.* Morgan Kaufmann Publishers, New York, 2000
2. Nelson, Mark. *The Data Compression.* IDG Books Worldwide, Inc., 1998
3. Bell, Timothy C., Cleary, John G., Witten, Ian H. *Text Compression*, Prentice Hall, Englewood Cliffs NJ, 1990
4. Mickey Williams and David Bennett, *Visual C++ 6 Unleashed.* Sams. Jul 24. 2000
5. Jesse Liberty, *Programming C#: Building .NET Applications with C#*, O'Reilly (February 22, 2005).

Research Paper

[1] M. G. Ross. *Measuring Image Quality.* IRE Trans. Image Quality, IQ-27 pp 18-22. 1996.

[2] Panrong, Xiao. *Image Compression By Wavelet Transform.*IRE Trans. Image Compression, IC-14 pp 16-19. 2001.

[3] Sebastian, Deorowicz. *Universal lossless data compression algorithms.*IRE Trans. Lossless data Compression Algorithms, LDCA- 10 pp 7-28. 2003

Internet Sources

1. http://www.ics.uci.edu/~dan/pubs/DC-Sec4.html.
2. http://www.cs.cf.ac.uk/Dave/Multimedia/node203.html.
3. http://www.breadfan.com/algorithms/ahuff.html.
4. http://www.rasip.fer.hr/research/compress/.