PERFORMANCE COMPARISON OF DIFFERENT DATA COMPRESSION TECHNIQUES

Project Report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology

in

INFORMATION TECHNOLOGY

under the Supervision of

MR.AMIT KUMAR SINGH

By

GEETANJALI CHOUDHARY(111403)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

CERTIFICATE

This is to certify that the work titled "Performance Comparison of Different Data Compression Techniques**"** submitted by **Geetanjali Chaudhary** in the partial fulfillment for the award of degree of Bachelor of Technology in Information Technology from Jaypee University of Information Technology, Waknaghat 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.

Signature of Supervisor:

Name of Supervisor : Mr Amit Singh

Designation : Assistant Professor

Date : 14-May-2015

ACKNOWLEDGEMENT

I would like to express my gratitude to all those who gave us the possibility to complete this project. I want to thank the Department of CSE & IT in JUIT for giving us the permission to commence this project in the first instance, to do the necessary research work.

I am deeply indebted to my project guide Mr Amit Singh, whose help, stimulating suggestions and encouragement helped me in all the time of research on this project. I feel motivated and encouraged every time I get his encouragement. For his coherent guidance throughout the tenure of the project, I feel fortunate to be taught by him, who gave me his unwavering support.

Geetanjali Chaudhary

Contents

LIST OF FIGURES

ABSTRACT

The objective of image compression is to reduce irrelevance and redundancy of the image data in order to be able to store or transmit data in an efficient form.

Image compression may be lossy or lossless. Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clipart or comics. Lossy compression methods, especially when used at low bit rates, introduce compression artifacts. Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate. The lossy compression that produces imperceptible differences may be called visually lossless.

1.1NEED FOR IMAGE COMPRESSION:-

- Some files are large and consume lots of hard disk space. The files size makes it time-consuming to move them from place to place over networks or to distribute over the Internet.
- Compression shrinks files, making them smaller and more practical to store and share.
- Compression works by removing repetitious or redundant information, effectively summarising the contents of a file in a way that preserves as much of the original meaning as possible.

1.2Lossless compressions:-

- When a file that has been compressed can be decoded back into its original form with zero loss of information.
- All lossless compression uses techniques to break up a file into smaller segments, for storage or transmission, that get reassembled later.
- For example Medical images, text needed in legal purposes and computer executable files.

1.3 Lossy compressions:-

- After compression, the original file cannot be brought back again.
- Eliminates repeated or "unnecessary" pieces of data.
- For e.g. Videos and audios.

PROBLEM STATEMENT

The problem we intend to solve is based on commonly used lossless image compression algorithm which are presented and then compared in terms of their performance. One of the problem faced while evaluating the performance of image compression algorithms is that they rely on different image file formats and use different performance measure

MOTIVATION

Digital communication systems require large storage space and quite amount of bandwidth for transmission. Although the capacity of current technologies continues to grow, the requirements for data storage and transmission bandwidth grow as well. Still or motion pictures are getting to take more and more share from total bandwidth capacity of networks. Hence image and video compression are gaining more importance to address the capacity problem. Research on image coding and compression techniques is a highly active area. The main research target is to get higher compression ratios with minimum loss of quality.

CHAPTER -1

INTRODUCTION

1.1INTRODUCTION TO IMAGE COMPRESSION

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level. The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from Web pages.

Compressing an image is significantly different than compressing raw binary data. Compression programs can be used to compress images, but the result is less than optimal. This is because images have certain statistical properties which can be exploited by encoders specifically designed for them. Also, some of the finer details in the image can be sacrificed for the sake of saving a little more bandwidth or storage space. Image compression algorithms are designed to minimize image file size in order to speed up image data transmission. Any compression algorithm can be viewed as a function that maps sequences of units (normally octets) into other sequences of the same units. Compression is successful if the resulting sequence is shorter than the original sequence plus the map needed to decompress it. Decompression restores the original image without loss of fidelity.The following figure shows the example of image after and before image compression.

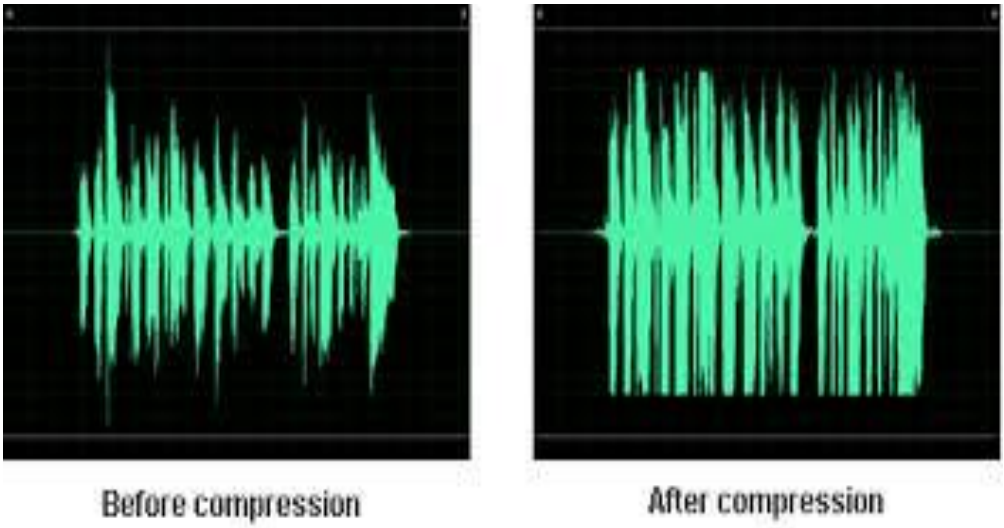


Fig1.1.Image compression before and after compression

1.3 CHARACTERSTICS:-

- In image compression, a small loss in quality is usually not noticeable.
- There is no "critical point" up to which compression works perfectly, but beyond which it becomes impossible.
- When there is some tolerance for loss, the compression factor can be greater than it can when there is no loss tolerance.
- Repeated data is important in compression. If a text has many repeated data, it can be compressed to a high ratio.
- Texts are always compressed with lossless compression techniques, because a loss in a text will change its originality.

1.3 PRINCIPLE:-

The principle of image compression algorithms are

(i) reducing the redundancy in the image data.

(ii) producing a reconstructed image from the original image with the introduction of error that is insignificant to the intended applications.

1.4 PROPERTIES:-

Scalability generally refers to a quality reduction achieved by manipulation of the bitstream or file (without decompression and re-compression).Despite its contrary nature, scalability also may be found in lossless codecs, usually in form of coarse-to-fine pixel scans. Scalability is especially useful for previewing images while downloading them (e.g., in a web browser) or for providing variable quality access to e.g., databases. There are several types of scalability:

- **Quality progressive** or layer progressive: The bitstream successively refines the reconstructed image.
- **Resolution progressive**: First encode a lower image resolution; then encode the difference to higher resolutions.
- **Component progressive**: First encode grey; then color.

Region of interest coding

. Certain parts of the image are encoded with higher quality than others. This may be combined with scalability (encode these parts first, others later).

Meta information

Compressed data may contain information about the image which may be used to categorize, search, or browse images. Such information may include color and texture statistics, small preview images, and author or copyright information.

Processing power

Compression algorithms require different amounts of processing power to encode and decode. Some high compression algorithms require high processing power.

1.4APPLICATIONS:-

- In health industry, where the constant scanning and/or storage of medical images and documents take place.

- In the security industry, image compression can greatly increase the efficiency of recording, processing and storage. For e.g. a video networking or closed-circuit television application, several images at different frame rates may be required.
- Museums and galleries consider the quality of reproductions to be of the utmost importance. Image compression, therefore, can be very effectively applied in cases where accurate representations of museum or gallery items are required, such as on a Web site.

1.5TYPES OF IMAGE COMPRESSION

Compression can be divided into two categories

Lossless and Lossy compression.The following figure shows the two types of compression techniques.

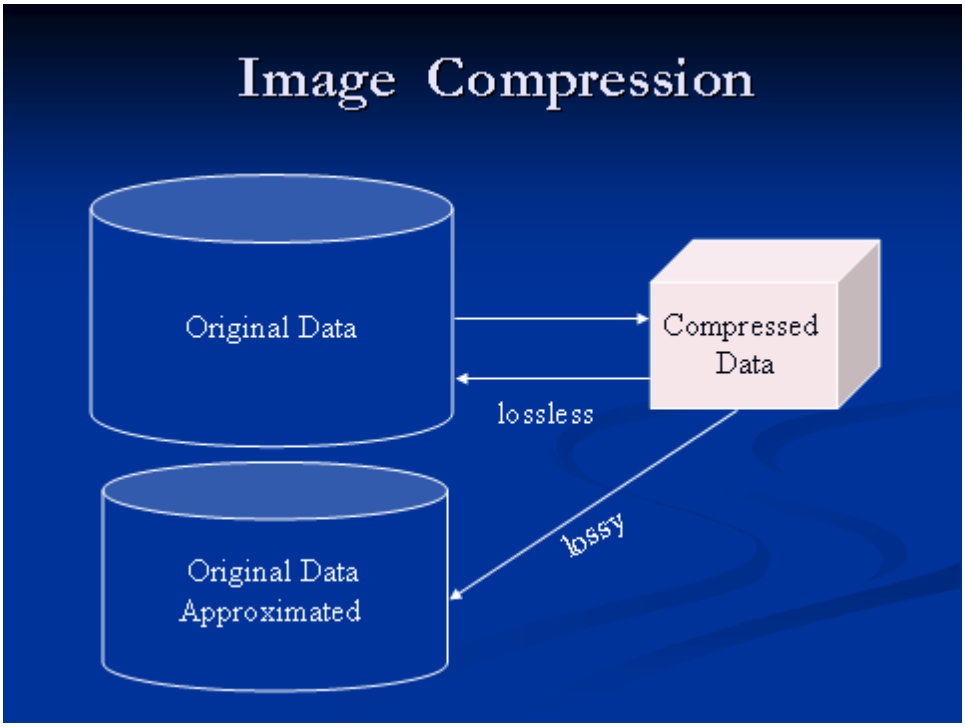


Fig 1.2Different types of image compression techniques

1.5.1LOSSLESS COMPRESSION:-

In lossless compression, the reconstructed image after compression is numerically identical to the original image. 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.

A text file or program can be compressed without the introduction of errors, but only up to a certain extent. This is called lossless *compression*. Beyond this point, errors are introduced. In text and program files, it is crucial that compression be lossless because a single error can seriously damage the meaning of a text file, or cause a program not to run. In image compression, a small loss in quality is usually not noticeable. There is no "critical point" up to which compression works perfectly, but beyond which it becomes impossible. When there is some tolerance for loss, the compression factor can be greater than it can when there is no loss tolerance. For this reason, graphic images can be compressed more than text files or programs

The biggest drawback for lossless image compression is that images can only be reduced to about one-third of their original image size . The goal of lossless image compression is to represent an image signal with the smallest possible number of bits without loss of *any* information, thereby speeding up transmission and minimizing storage requirements

Lossless compression is preferred for archival purposes and often for medical imaging, technical drawings, clip art or comics.

1.5.2LOSSY 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.

Lossy methods are especially suitable for natural images such as photographs in applications where minor (sometimes imperceptible) loss of fidelity is acceptable to achieve a substantial reduction in bit rate.

1.6LOSSY VERSUS LOSSLESS COMPRESSION:-

The advantage of lossy methods over lossless methods is that in some cases a lossy method can produce a much smaller compressed file than any lossless method, while still meeting the requirements of the application.

Lossy methods are most often used for compressing sound, images or videos. This is because these types of data are intended for human interpretation where the mind can easily "fill in the blanks" or see past very minor errors or inconsistencies – ideally lossy compression is transparent (imperceptible), which can be verified via an ABX test.

TRANSPARENCY

When a user acquires a lossily compressed file, (for example, to reduce download time) the retrieved file can be quite different from the original at the bit level while being

indistinguishable to the human ear or eye for most practical purposes. Many compression methods focus on the idiosyncrasies of human physiology, taking into account, for instance, that the human eye can see only certain wavelengths of light. The psychoacoustic model describes how sound can be highly compressed without degrading perceived quality. Flaws caused by lossy compression that are noticeable to the human eye or ear are known as compression artifacts

COMPRESSION RATIO

The compression ratio (that is, the size of the compressed file compared to that of the uncompressed file) of lossy video codecs is nearly always far superior to that of the audio and still-image equivalents.

- Video can be compressed immensely (e.g. 100:1) with little visible quality loss
- Audio can often be compressed at 10:1 with imperceptible loss of quality
- Still images are often lossily compressed at 10:1, as with audio, but the quality loss is more noticeable, especially on closer inspection.

1.7METHODS OF LOSSLESS COMPRESSION:-

- Run length coding
- Arithemetic coding
- Huffman coding

1.8METHODS OF LOSSY COMPRESSION:-

- Fractal compression
- Transform coding

LOSSLESS ALGORITHMS :-

ARITHMETIC CODING :-

Arithmetic coding assigns a sequence of bits to a message, a sting of symbols. Arithmetic coding can treat the whole symbols in a list or in a message as one unit [22]. Unlike Huffman

coding, arithmetic coding doesn´t use a discrete number of bits for each. The number of bits used to encode each symbol varies according to the probability assigned to that symbol. Low probability symbols use many bit, high probability symbols use fewer bits [23]. The main idea behind Arithmetic coding is to assign each symbol an interval. Starting with the interval [0...1), each interval is divided in several subinterval, which its sizes are proportional to the current probability of the corresponding symbols [24]. The subinterval from the coded symbol is then taken as the interval for the next symbol. The output is the interval of the last symbol .The following figure shows the example of arithmetic coding.

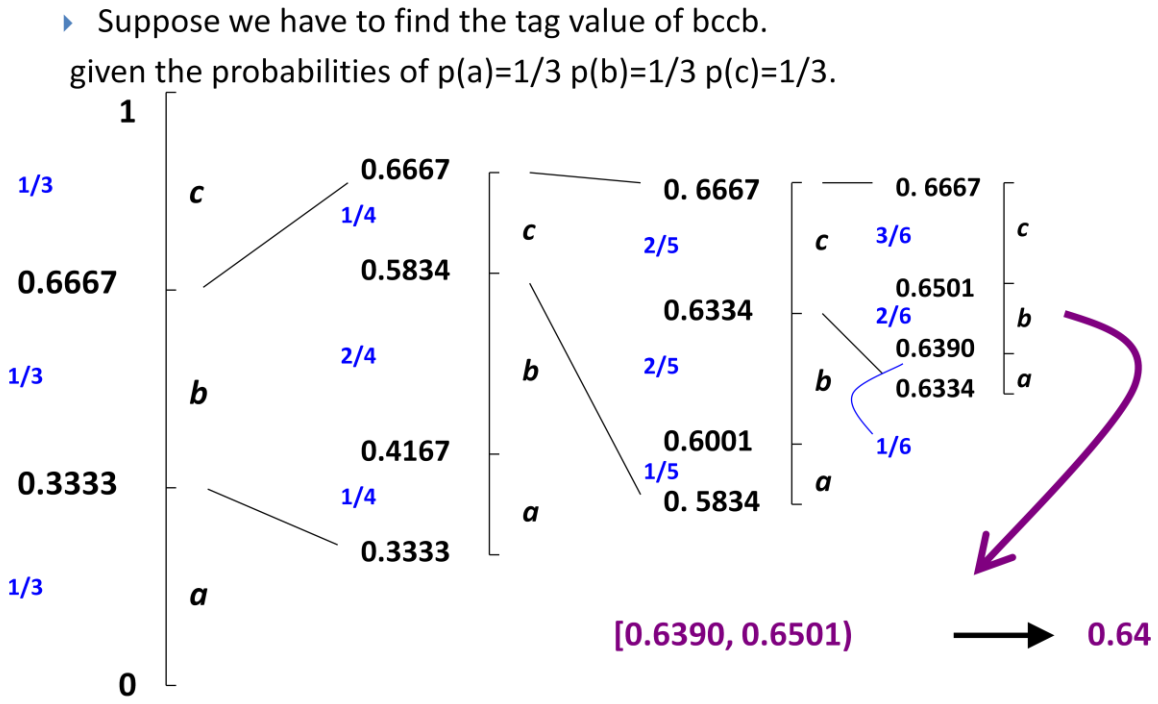


Fig 1.3 Example of Arithmetic coding

HUFFMAN CODING:-

Huffman coding is one of the most popular technique for removing coding redundancy. It has been used in various compression applications, including image compression. It is a simple, yet elegant, compression technique that can supplement other compression algorithms. The following figure shows the example of Huffman coding.

Table 15.1 Frequency of characters

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

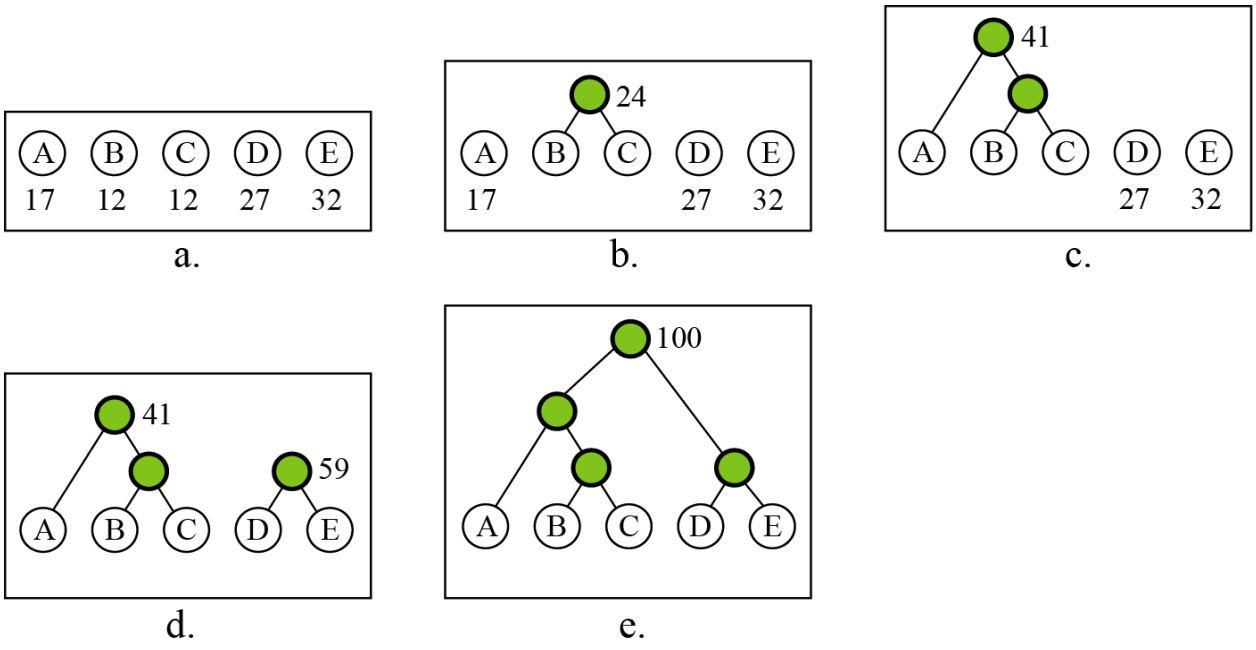


Fig 1.4 Example of Huffman coding

RUN LENGTH CODING:-

Run-length encoding is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

Run-length encoding performs lossless data compression and is well suited to palette-based bitmapped images such as computer icons. It does not work well at all on continuous-tone images such as photographs, although JPEG uses it quite effectively on the coefficients that remain after transforming and quantizing image blocks.The following figure shows the example of run length coding.

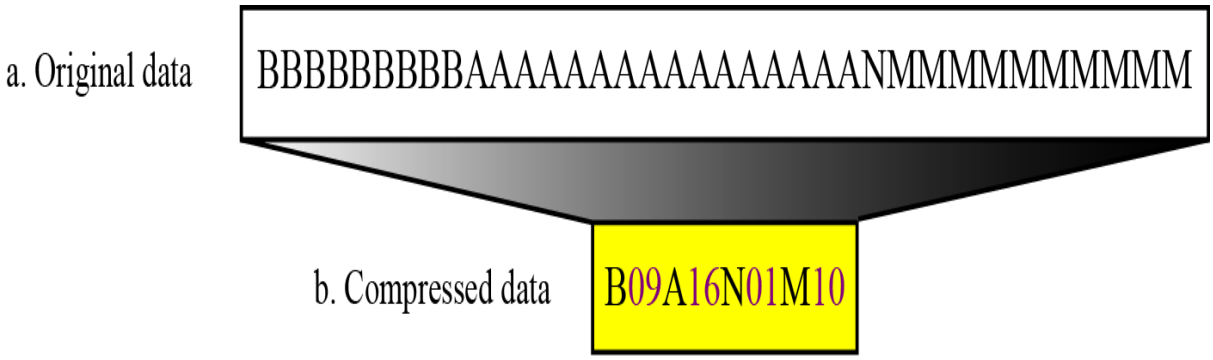


Fig 1.5 Example of run length coding

1.8LOSSY COMPRESSION TECHNIQUES:-

- TRANSFORM CODING

More generally, some forms of lossy compression can be thought of as an application of transform encoding – in the case of multimedia data, perpetual coding: it transforms the raw data to a domain that more accurately reflects the information content. For example, rather than expressing a sound file as the amplitude levels over time, one may express it as the frequency spectrum over time, which corresponds more accurately to human audio perception.

While data reduction (compression, be it lossy or lossless) is a main goal of transform coding, it also allows other goals: one may represent data more accurately for the original amount of space – for example, in principle, if one starts with an analog or high-resolution digital master, an MP3 file of a given size should provide a better representation than a raw uncompressed audio in WAV or AIFF file of the same size. This is because uncompressed audio can only reduce file size by lowering bit rate or depth, whereas compressing audio can reduce size while maintaining bit rate and depth.

METHODS

IMAGE:-

- Fractal compression
- JBIG2 (lossless or lossy compression)
- JPEG
- JPEG 2000, JPEG's successor format that uses wavelets (lossless or lossy compression)

- JPEG XR, another successor of JPEG with support for high dynamic range wide pixel formats (lossless or lossy compression)
- PGF , Progressive Graphics File (lossless or lossy compression)
- S3TC texture compression for 3D computer graphics hardware
- Wavelet compression
- Block Truncation Coding Absolute Moment BTC or simply BTC

FRACTAL COMPRESSION:-

Fractal compression is a lossy compression method for digital images, based on fractals. The method is best suited for textures and natural images, relying on the fact that parts of an image often resemble other parts of the same image.Fractal algorithms convert these parts into mathematical data called "fractal codes" which are used to recreate the encoded image.

With fractal compression, encoding is extremely computationally expensive because of the search used to find the self-similarities. Decoding, however, is quite fast. While this asymmetry has so far made it impractical for real time applications, when video is archived for distribution from disk storage or file downloads fractal compression becomes more competitive.

At common compression ratios, up to about 50:1, Fractal compression provides similar results to DCT-based algorithms such as JPEG. At high compression ratios fractal compression may offer superior quality. For satellite imagery, ratios of over 170:1 have been achieved with acceptable results. Fractal video compression ratios of 25:1-244:1 have been achieved in reasonable compression times (2.4 to 66 sec/frame).

Compression efficiency increases with higher image complexity and color depth, compared to simple grayscale images.

Resolution independence and fractal scaling

An inherent feature of fractal compression is that images become resolution independent after being converted to fractal code. This is because the iterated function

systems in the compressed file scale indefinitely. This indefinite scaling property of a fractal is known as "fractal scaling".

Fractal interpolation

The resolution independence of a fractal-encoded image can be used to increase the display resolution of an image. This process is also known as "fractal interpolation". In fractal interpolation, an image is encoded into fractal codes via fractal compression, and subsequently decompressed at a higher resolution. The result is an up-sampled image in which iterated function systems have been used as the interpolant. Fractal interpolation maintains geometric detail very well compared to traditional interpolation methods like bilinear interpolation and bi cubic interpolation. Since the interpolation cannot reverse Shannon entropy however, it ends up sharpening the image and adding random instead of meaningful detail. One cannot, for example, enlarge an image of a crowd where each person's face is one or two pixels and hope to identify them.

JPEG:-

In computing, **JPEG** (seen most often with the **.jpg** or **.jpeg** filename extension) is a commonly used method of lossy compression for digital images, particularly for those images produced by digital photography. The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

JPEG compression is used in a number of image file formats. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices; along with JPEG/JFIF, it is the most common format for storing and transmitting photographic images on the World Wide Web.These format variations are often not distinguished, and are simply called JPEG.

The term "JPEG" is an acronym for the Joint Photographic Experts Group, which created the standard. The MIME media type for JPEG is *image/jpeg* (defined in RFC 134, except in older Internet Explorer versions, which provides a MIME type of *image* when uploading JPEG images.

JPEG/JFIF supports a maximum image size of 65535×65535 pixels,hence up to 4 gigapixels (for an aspect ratio of 1:1).

The JPEG compression algorithm is at its best on photographs and paintings of realistic scenes with smooth variations of tone and color. For web usage, where the amount of data used for an image is important, JPEG is very popular. JPEG/Exif is also the most common format saved by digital cameras.

There are several different ways in which image files can be compressed. For Internet use, the two most common compressed graphic image formats are the JPEG format and the GIF format. The JPEG method is more often used for photographs, while the GIF method is commonly used for line art and other images in which geometric shapes are relatively simple.

Other techniques for image compression include the use of fractals and wavelets. These methods have not gained widespread acceptance for use on the Internet as of this writing. However, both methods offer promise because they offer higher compression ratios than the JPEG or GIF methods for some types of images. Another new method that may in time replace the GIF format is the PNG format.

On the other hand, JPEG may not be as well suited for line drawings and other textual or iconic graphics, where the sharp contrasts between adjacent pixels can cause noticeable artifacts. Such images may be better saved in a lossless graphics format such as TIFF, GIF, PNG, or a raw image format. The JPEG standard actually includes a lossless coding mode, but that mode is not supported in most products.

As the typical use of JPEG is a lossy compression method, which somewhat reduces the image fidelity, it should not be used in scenarios where the exact reproduction of the data is required (such as some scientific and medical imaging applications and certain technical image processing work).

JPEG is also not well suited to files that will undergo multiple edits, as some image quality will usually be lost each time the image is decompressed and recompressed, particularly if the image is cropped or shifted, or if encoding parameters are changed – see digital generation loss for details. To avoid this, an image that is being modified or may be modified in the future can be saved in a lossless format, with a copy exported as JPEG for distribution.

JPEG uses a lossy form of compression based on the discrete cosine transform (DCT). This mathematical operation converts each frame/field of the video source from the spatial (2D)

domain into the frequency domain (a.k.a. transform domain.) A perceptual model based loosely on the human psychovisual system discards high-frequency information, i.e. sharp transitions in intensity, and color hue. In the transform domain, the process of reducing information is called quantization. In simpler terms, quantization is a method for optimally reducing a large number scale (with different occurrences of each number) into a smaller one, and the transform-domain is a convenient representation of the image because the high-frequency coefficients, which contribute less to the overall picture than other coefficients, are characteristically small-values with high compressibility. The quantized coefficients are then sequenced and losslessly packed into the output bitstream. Nearly all software implementations of JPEG permit user control over the compression-ratio (as well as other optional parameters), allowing the user to trade off picture-quality for smaller file size. In embedded applications (such as miniDV, which uses a similar DCT-compression scheme), the parameters are pre-selected and fixed for the application.

There are also many medical imaging and traffic systems that create and process 12-bit JPEG images, normally grayscale images. The 12-bit JPEG format has been part of the JPEG specification for some time, but this format is not as widely supported.

PROGRESSIVE GRAPHICS FILE:-

PGF (Progressive Graphics File) is a wavelet-based bitmapped imageformat that employs lossless and lossy data compression. PGF was created to improve upon and replace the JPEG format. It was developed at the same time as JPEG 2000 but with a focus on speed over compression ratio.

PGF can operate at higher compression ratios without taking more encoding/decoding time and without generating the characteristic "blocky and blurry"artifacts of the original DCT-based JPEG standard. It also allows more sophisticated progressive downloads.

PGF supports a wide variety of color models:

Grayscale with 1, 8, 16, or 31 bits per pixel

Indexed color with palette size of 256

RGB color image with 12, 16 (red: 5 bits, green: 6 bits, blue: 5 bits), 24, or 48 bits per pixel

ARGB color image with 32 bits per pixel

L*a*b color image with 24 or 48 bits per pixel

CMYK color image with 32 or 64 bits per pixel

VIDEO:-

- Motion JPEG
- MPEG-1 Part 2
- MPEG-2 Part 2
- MPEG-4 Part 2
- H.264/MPEG-4 AVC (may also be lossless, even in certain video sections)

The following figures shows the video before and after compression.

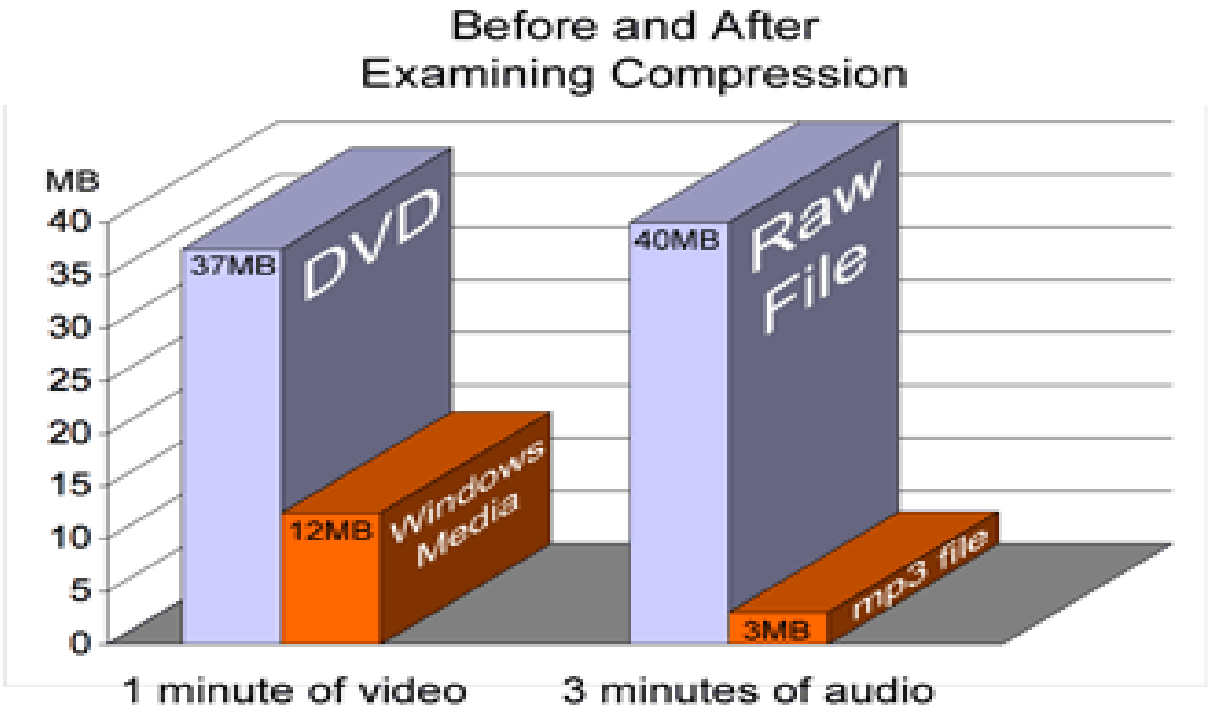


Fig 1.5 Video before and after examining compression

CHAPTER-2

LITERATURE WORK

2.1 ANALYSIS OF VARIOUS RESEARCH PAPERS

Aarti [1] proposed a method for lossless compression and decompression using a simple coding technique called Huffman coding which is used on text string as well as on different image file formats for performance analysis. The results reveals that the original image used for coding is almost close to the decoded output image.

The result shows that the higher Code redundancy helps to achieve more compression. The above presented Huffman coding and decoding is used for scan testing to reduce test data volume, test data compression and decompression time. Hence we conclude that Huffman coding is efficient technique for image compression and decompression to some extent. The results also reveal that the original image used for coding is almost close to the decoded output image. This study presented an analysis of Huffman compression using metrics. Image compression using the Huffman Coding depends on the no. of pixels in an image , size of an image and compression ratio. Huffman is a good coding technique for compressing the data and general types of images.

Dubey et Singh[2]This paper presents new JPEG2000 based lossy image compression method based on 2D discrete wavelet transform. In the proposed method , 3D image is divided into smaller non-overlapping tiles on which 2D DWT is applied .Thereafter , Hard Thresholding and Huffman coding are respectively applied on each of the tiles to get compressed image. The Performance of proposed compression method is measured over various images and found to be efficient method of image compression in terms of short coding ,less calculations. This compression method is simpler and has better performance compared to that of JPEG compression as we are applying 2D DWT.

Gulhane et sadique[3] initiated an enquiry XML compression in real-time database systems (RTDBS). Xml documents are identified as the major resources that need to be compressed and effectively partial decompression even at the compressed database level with support from underlying operating systems. The main criteria in assessing any XML compressor and decompressor, the success ratio in terms of the CR1 and CS (compression ratio and

compressed size). The proposed research aims at the investigation of efficient Partial query processing techniques on compressed xml dataset in database systems. Focusing on query proceeding time and memory space.

Mathur et Loonker et Saxena[4] converted an image into an array using Delphi image control tool. Image control can be used to display a graphical image - Icon (ICO), Bitmap (BMP), Metafile (WMF), GIF, JPEG, etc, then an algorithm is created in Delphi to implement Huffman coding method that removes redundant codes from the image and compresses a BMP image file (especially grayscale image) and it is successfully reconstructed. This reconstructed image is an exact representation of the original because it is lossless compression technique. This Program can also be applied on other kind of RGB images (BMP, JPEF, Gif, and tiff) but it gives some color quality loss after reconstruction .Compression ratio for grayscale image is better as compared to other standard methods.

CHAPTER-3

LOSSLESS COMPRESSION TECHNIQUES

3.1HUFFMAN CODING:-

Huffman code procedure is based on the two Observations.

a. More frequently occurred symbols will have shorter code words than symbol that occur less frequently.

b. The two symbols that occur least frequently will have the same length.

The Huffman code is designed by merging the lowest probable symbols and this process is repeated until only two probabilities of two compound symbols are left and thus a code tree is generated and Huffman codes are obtained from labeling of the code tree.

Huffman Data Compression

1. Initialize and encode the data model with a single, complete pass over the entire data file.
2. When more data is available to transmit, encode the next symbol based on the data model.

Decoding Process:

1. Receive the data model.
2. When more data is available to decode, decode the next symbols using the data model.

Huffman data dcompression:-

Generally speaking, the process of decompression is simply a matter of translating the stream of prefix codes to individual byte values, usually by traversing the Huffman tree node by node as each bit is read from the input stream (reaching a leaf node necessarily terminates the search for that particular byte value). Before this can take place, however, the Huffman tree must be somehow reconstructed. In the simplest case, where character frequencies are fairly predictable, the tree can be preconstructed (and even statistically adjusted on each compression cycle) and thus reused every time, at the expense of at least some measure of compression efficiency.

Huffman Code Construction

Huffman Coding Algorithm is a bottom-up approach.

3.1.2ALGORITHM

The steps of Huffman coding algorithm are given below:

1. Create a series of source reduction: combine the two lowest probability symbol into a single symbol; repeated until a reduced source with two symbols is reached.

2. Code each reduced symbol: start with the smallest source and working back to the original source;

(a) Creates the optimal code for a set of symbols; the symbol is coded one at a time;

(b) The code itself (or block code) –each code is mapped

into a fixed sequence of code symbols

(c) Create the optimal codes for a set of symbols and probabilities

(d) Coding and decoding is accomplished in a simple lookup table

(e) Block code (because source symbol is mapped into a sequence of codes

(f) Instantaneous, unique decodable

3.1.2STATIC HUFFMAN CODING:-

- In a static method the mapping from the set of messages to the set of codewords is fixed before transmission begins, so that a given message is represented by the same codeword every time it appears in the message being encoded.
- Static coding requires two passes: one pass to compute probabilities (or frequencies) and determine the mapping, and a second pass to encode.
- Static Huffman coding assigns variable length codes to symbols based on their frequency of occurrences in the given message. Low frequency symbols are encoded using many bits, and high frequency symbols are encoded using fewer bits.

  The message to be transmitted is first analyzed to find the relative frequencies of its constituent characters.

  coding process generates a binary tree, the Huffman code tree, with branches labeled with bits (0 and 1).

  The Huffman tree (or the character codeword pairs) must be sent with the compressed information to enable the receiver decode the message.

3.1.3ADAPTIVE HUFFMAN CODING

All of the adaptive methods are one-pass methods; only one scan of the message is required. Huffman's algorithm has been modified for use in adaptive coding, a variation of previous coding methods used in lossless data compression. What makes adaptive coding so useful is its adaptability to streaming media. It is the preferred choice of data compression when dealing with streaming media, where bits of information are gradually transmitted instead of being transferred as one single

file. A normal data compressor contains a simple encoder and decoder, since it is only required to scan over a single file of data in order to calculate a probability model. However, an Adaptive Huffman compressor is able to adapt to changes in characteristic of the media due to the complexity of its encoder, which allows a probability model to be calculated even without an initial scan over the entire media file.

Adaptive Huffman Method:
Encoding Process:
1. Initialize the data model per bit. (The encoder begins with a clean slate, creating the data model as bits of information are received)
2. When more data is available to transmit:
(a) Encode the next symbol based on the data model.
(b) Modify the data model using the last encoded symbol.
Decoding Process:
1. Initialize the data model per bit. (The decoder begins with a clean slate, creating the data model as bits of information are received)
2. When more data is available to transmit:
(a) Decode the next symbol based on the data model.
(b) Modify the data model using the decoded symbol.

The adaptive Huffman method is much more extensive than the static Huffman method as it allows for data to be decompressed while it is being transmitted, instead of relying on having an entire file to encode.

3.1.4 FLOWCHART DIAGRAM OF HUFFMAN CODING:-

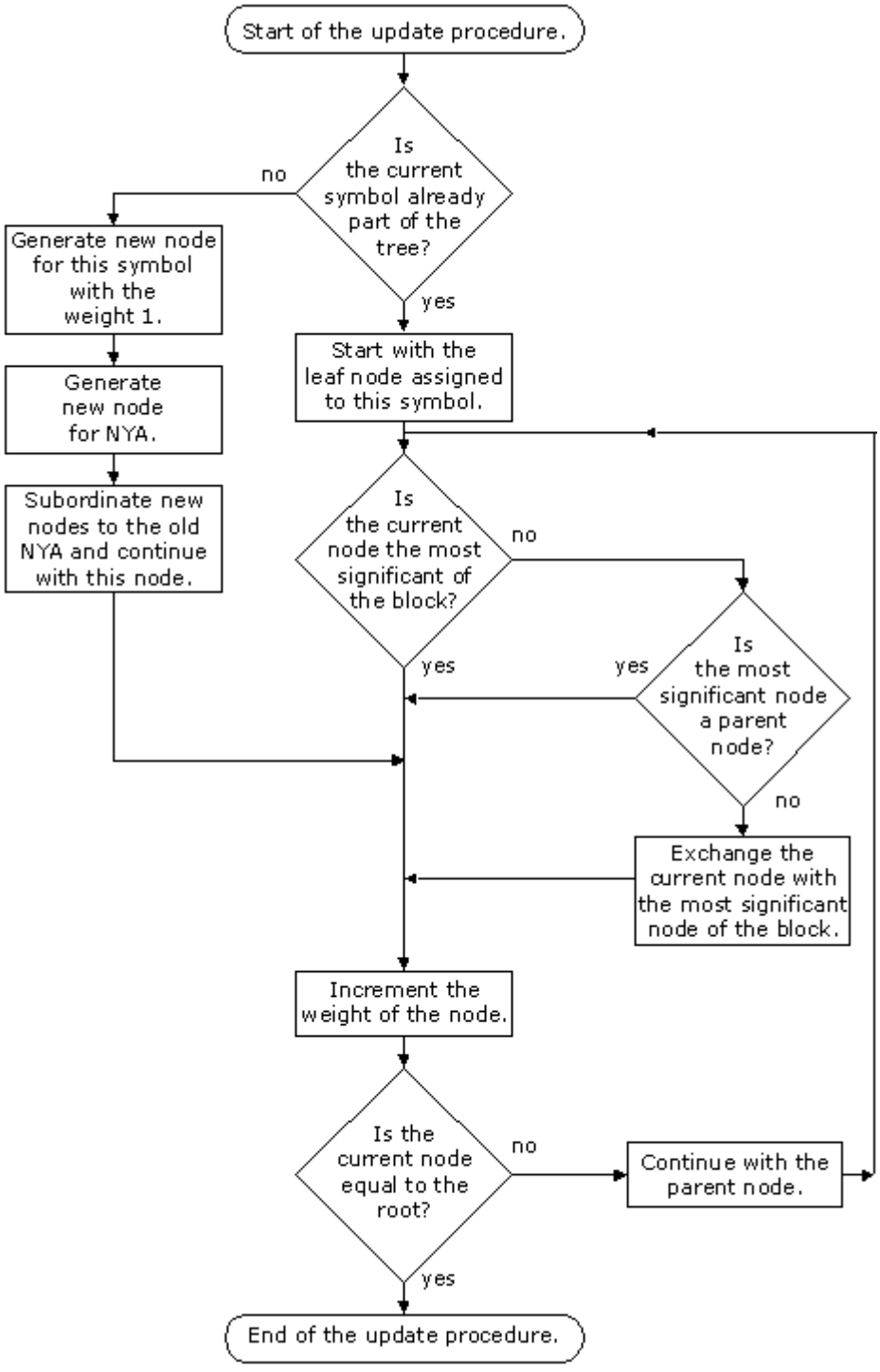


Fig3.1 Flowchart of adaptive Huffman Coding

3.2RUN LENGTH CODING:-

Run-length encoding (**RLE**) is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. Consider, for example, simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could greatly increase the file size.

RLE may also be used to refer to an early graphics file format supported by CompuServe for compressing black and white images, but was widely supplanted by their later Graphics interchange Format. RLE also refers to a little-used image format in Windows 3.x, with the extension **rle**, which is a Run Length Encoded Bitmap, used to compress the Windows 3.x startup screen.

Typical applications of this encoding are when the source information comprises long substrings of the same character or binary digit.

For example, consider a screen containing plain black text on a solid white background. There will be many long runs of white pixels in the blank space, and many short runs of black pixels within the text. A hypothetical scan line, with B representing a black pixel and W representing white, might read as follows:

```
WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWW
WWWWWWWWWWWWWWWWBWWWWWWWWWWWWWW
```

With a run-length encoding (RLE) data compression algorithm applied to the above hypothetical scan line, it can be rendered as follows:

```
12W1B12W3B24W1B14W
```

This can be interpreted as a sequence of twelve Ws, one B, twelve Ws, three Bs, etc.

3.2.1APPLICATIONS

Run-length encoding performs lossless data compression and is well suited to palette-based bitmapped images such as computer icons. It does not work well at all on continuous-tone images such as photographs, although JPEG uses it quite effectively on the coefficients that remain after transforming and quantizing image blocks.

Common formats for run-length encoded data include Truevision TGA, PackBits, PCX and ILBM. ITU also describes a standard to encode run-length-colour for fax machines, known as T.45.

Run-length encoding is used in fax machines (combined with other techniques into Modified Huffman coding). It is relatively efficient because most faxed documents are generally white space, with occasional interruptions of black.

3.2.1FLOWCHART:-

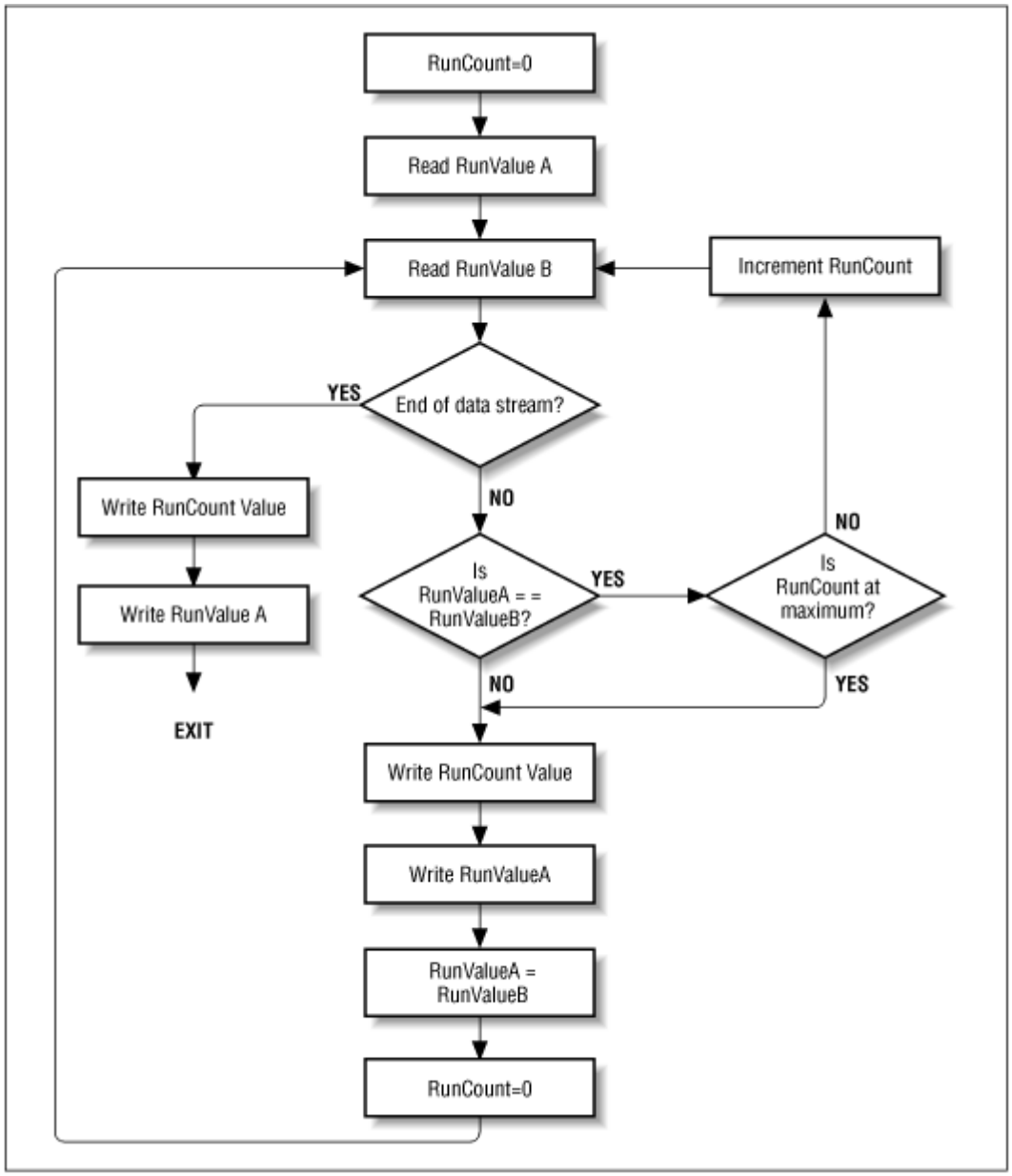


Fig3.2 Flowchart of Run length coding

3.3ARITHEMTIC ENCODING:-

Arithmetic coding is a form of entropy encoding used in lossless data compression. Normally, a string of characters such as the words "hello there" is represented using a fixed number of bits per character, as in the ASCII code. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding, such as Huffman coding, in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \leq n < 1.0$).

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

- Range, keeps track of where the next range should be.
- High and low, specify the output number.
- Loop. For all the symbols.
    - Range = high_range of the symbol - low_range of the symbol
    - Number = number - low_range of the symbol
    - Number = number / range

3.3.1PSEUDOCODE

And this is the pseudo code for the initialization:

- Get probabilities and scale them
- Save probabilities in the output file
- High = FFFFh (16 bits)

- Low = 0000h (16 bits)
- Underflow_bits = 0 (16 bits should be enough)

- High and low, they define where the output number falls.
- Underflow_bits, the bits which could have produced underflow and thus they were shifted.

And the routine to encode the symbol is:-

- Range = (high - low) + 1
- High = low + ((range * high_values [symbol]) / scale) - 1
- Low = low + (range * high_values [symbol - 1]) / scale
- Loop. (will exit when no more bits can be outputted or shifted)
- Msb of high = msb of low?
- Yes
    - Output msb of low
    - Loop. While underflow_bits > 0 Let's output underflow bits pending for output
        - Output Not (msb of low)
    - go to shift
- No
    - Second msb of low = 1 and Second msb of high = 0 ? Check for underflow
    - Yes
        - Underflow_bits += 1 Here we shift to avoid underflow
        - Low = low & 3FFFh
        - High = high | 4000h
        - go to shift
    - No
        - The routine for encoding a symbol ends here.

- Shift low to the left one time. Now we have to put in low and high new bits
- Shift high to the left one time, and or the lsb with the value 1
- Repeat to the first loop.

Decoding

The first thing to do when decoding is read the probabilities, because the encode did the scaling you just have to read them and to do the ranges. The process will be the following: see in what symbol our number falls, extract the code of this symbol from the code. Before starting we have to init "code" this value will hold the bits from the input, init it to the first 16 bits in the input. And this is how it's done:

- Range = (high - low) + 1 See where the number lands
- Temp = ((code - low) + 1) * scale) - 1) / range)
- See what symbols corresponds to temp.
- Range = (high - low) + 1 Extract the symbol code
- High = low + ((range * high_values [symbol]) / scale) - 1
- Low = low + (range * high_values [symbol - 1]) / scale Note that those formulae are the same that the encoder uses
- Loop.
- Msb of high = msb of low?
- Yes
    - Go to shift
- No
    - Second msb of low = 1 and Second msb of high = 0 ?
    - Yes
        - Code = code ^ 4000h
        - Low = low & 3FFFh
        - High = high | 4000h
        - go to shift
    - No
        - The routine for decoding a symbol ends here

- Shiftlow to the left one time. Now we have to put in low, high and code new bits
- Shift high to the left one time, and or the lsb with the value 1
- Shift code to the left one time, and or it the next bit in the input
- Repeat to the first loop.

3.3.3 FLOWCHART:-

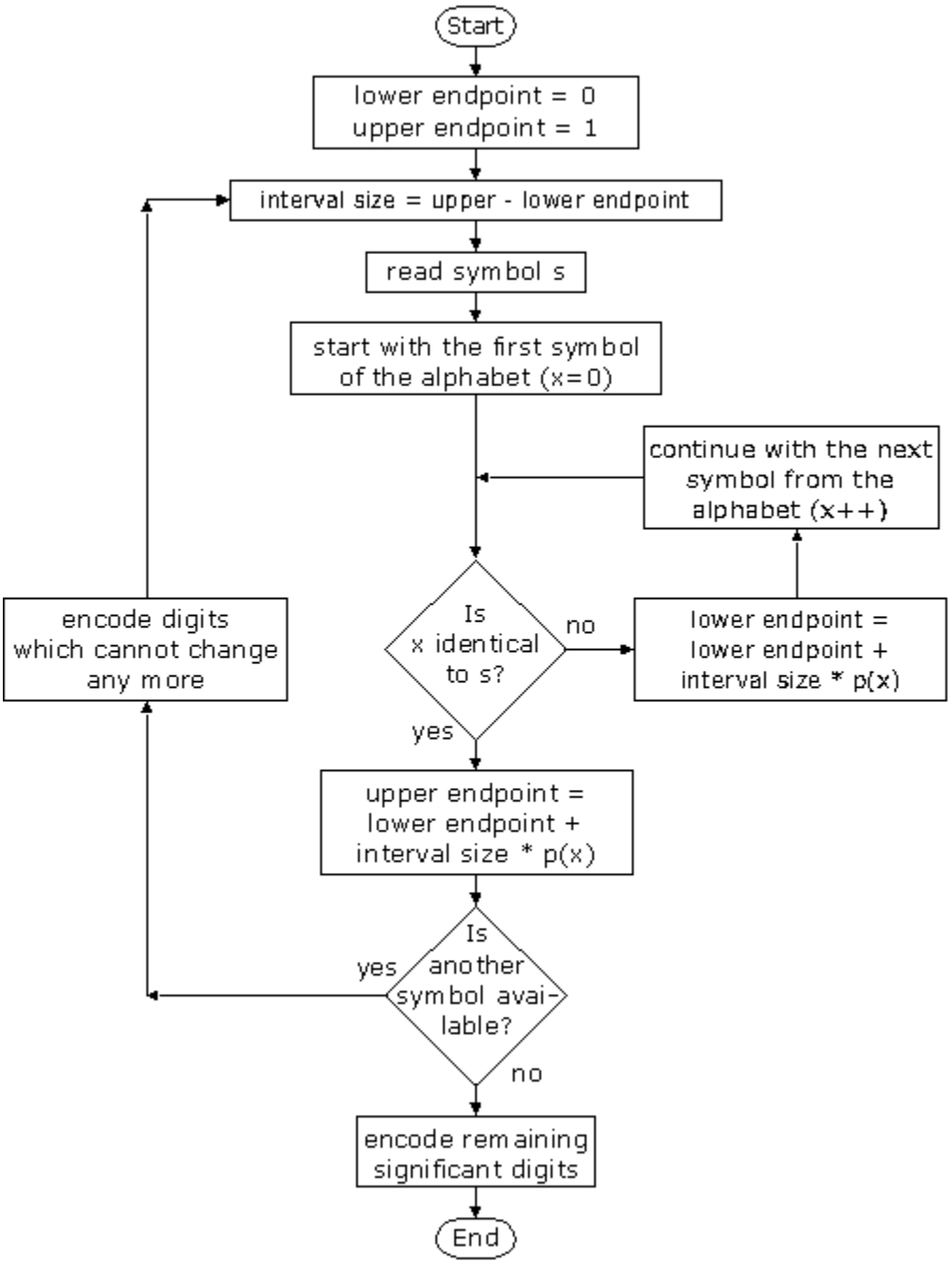


Fig 3.3 Flowchart of arithmetic coding

3.4 COMPARISON BETWEEN ARITHMETIC AND HUFFMAN ENCODING

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
Direct random Access	No	Yes

Table 3.1 Comparison between Arithmetic coding and Huffman coding

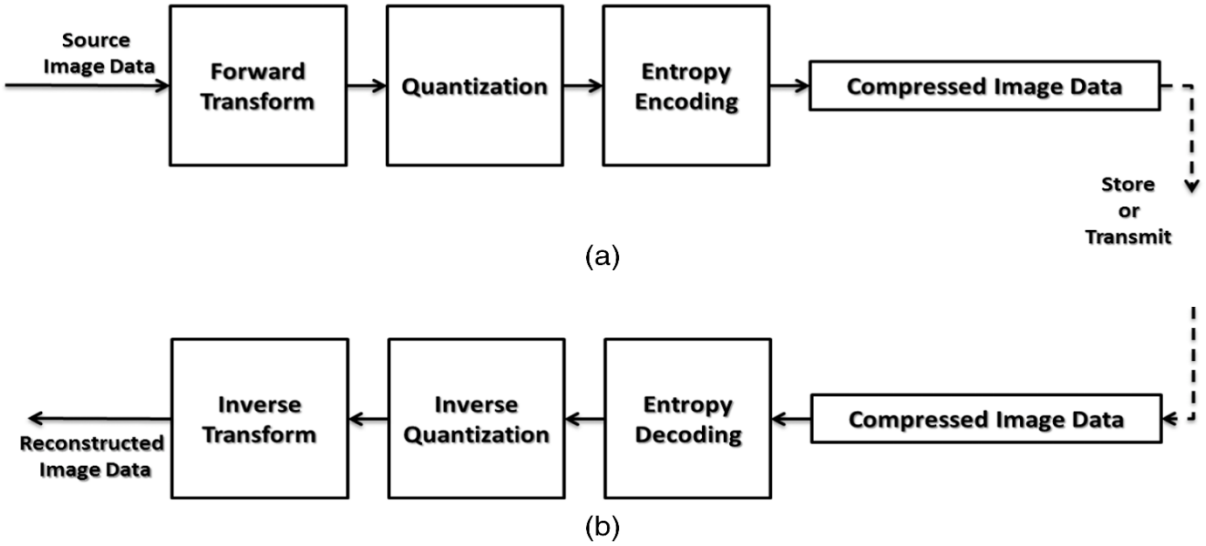
CHAPTER-4

LOSSY COMPRESSION TECHNIQUES

4.1PROPOSED MODEL

IMPLEMENATATION OF HUFFMAN CODING ALGORITHM IN 2D MEDICAL IMAGES.

MODEL:-



It presents new JPEG2000 based lossy image compression method based on 2D discrete wavelet transform. In the proposed method , DWT is applied .Thereafter , Hard Thresholding and Huffman coding are respectively applied on each of the tiles to get compressed image. The Performance of proposed compression method will be measured over various images and found to be efficient method of image compression in terms of short coding ,less calculations.

This compression method is simpler and has better performance compared to that of JPEG compression as we are applying 2D DWT.

4.2DISCRETE WAVELET TRANSFORM (DWT)

The discrete wavelet transform (DWT) refers to wavelet transforms for which the wavelets are discretely sampled. A transform which localizes a function both in space and scaling and

has some desirable properties compared to the Fourier transform. The transform is based on a wavelet matrix, which can be computed more quickly than the analogous Fourier matrix. Most notably, the discrete wavelet transform is used for signal coding, where the properties of the transform are exploited to represent a discrete signal in a more redundant form, often as a preconditioning for data compression. The discrete wavelet transform has a huge number of applications in Science, Engineering, Mathematics and Computer Science.

Quantization,

in mathematics and digital signal processing, is the process of mapping a large set of input values to a (countable) smaller set – such as rounding values to some unit of precision. A device or algorithmic function that performs quantization is called a quantizer. The round-off error introduced by quantization is referred to as quantization error.

In analog-to-digital conversion, the difference between the actual analog value and quantized digital value is called quantization error or quantization distortion. This error is either due to rounding or truncation. The error signal is sometimes modeled as an additional random signal called quantization noise because of its stochastic behaviour. Quantization is involved to some degree in nearly all digital signal processing, as the process of representing a signal in digital form ordinarily involves rounding. Quantization also forms the core of essentially all lossy compression algorithms.

Because quantization is a many-to-few mapping, it is an inherently non-linear and irreversible process (i.e., because the same output value is shared by multiple input values, it is impossible in general to recover the exact input value when given only the output value).

The set of possible input values may be infinitely large, and may possibly be continuous and therefore uncountable (such as the set of all real numbers, or all real numbers within some limited range). The set of possible output values may be finite or countably infinite. The input and output sets involved in quantization can be defined in a rather general way. For example, *vector quantization* is the application of quantization to multi-dimensional (vector-valued) input data.

There are two substantially different classes of applications where quantization is used:

- The first type, which may simply be called *rounding* quantization, is the one employed for many applications, to enable the use of a simple approximate representation for some quantity that is to be measured and used in other calculations. This category includes the simple rounding approximations used in everyday arithmetic. This category also

includes analog-to-digital conversion of a signal for a digital signal processing system (e.g., using a sound card of a personal computer to capture an audio signal) and the calculations performed within most digital filtering processes. Here the purpose is primarily to retain as much signal fidelity as possible while eliminating unnecessary precision and keeping the dynamic range of the signal within practical limits (to avoid signal clipping or arithmetic overflow. In such uses, substantial loss of signal fidelity is often unacceptable, and the design often centers around managing the approximation error to ensure that very little distortion is introduced.

- The second type, which can be called *rate–distortion optimized* quantization, is encountered in source coding for "lossy" data compression algorithms, where the purpose is to manage distortion within the limits of the bit rate supported by a communication channel or storage medium. In this second setting, the amount of introduced distortion may be managed carefully by sophisticated techniques, and introducing some significant amount of distortion may be unavoidable. A quantizer designed for this purpose may be quite different and more elaborate in design than an ordinary rounding operation. It is in this domain that substantial rate–distortion theory analysis is likely to be applied. However, the same concepts actually apply in both use cases.

The analysis of quantization involves studying the amount of data (typically measured in digits or bits or bit *rate*) that is used to represent the output of the quantizer, and studying the loss of precision that is introduced by the quantization process (which is referred to as the *distortion*). The general field of such study of rate and distortion is known as *rate–distortion theory*.

CHAPTER-5

TOOLS AND TECHNOLOGIES USED

5.1JAVA 1.6 VERSION:

5.1.1CHARACTERISTICS:

JAVA is a programming language, developed by Sun Microsystems and first released in 1995 (release 1.0). Since that time, it gained a large popularity mainly due to two characteristics:

- A JAVA programme is hardware and operating system independant. If well written (!), the same JAVA programme, compiled once, will run identically on a SUN/solaris workstation, a PC/windows computer or a Macintosh computer. Not mentioning other Unix flavors, including Linux, and every Web browser, with some restrictions described below.
  This universal executability is made possible because a JAVA programme is run through a JAVA Virtual Machine.
- it is an object oriented language. This feature is mainly of interest for software developers.

5.2 JAVA VIRTUAL MACHINE (JVM):

A JAVA programme is build by a JAVA compiler which generates its own binary code. This binary code is independant from any hardware and operating system. To be executed, it needs a virtual machine, which is a programme analyzing this binary code and executing the instructions.

Of course, this Java Virtual Machine (JVM) is hardware and operating system dependant. Two types of Virtual Machines exist: those included in every Web Browser, and those running as an independent programme, like the Java RunTime Environment (JRE) from Sun Microsystems. These programmes need to be downloaded for your particular platform. As seen in the next paragraph, these two types of Virtual Machines do not behave exactly the same.

Applet and Standalone Application:

A JVM in a web browser runs a JAVA programme as an Applet. The applet is embedded in a web page and downloaded from a web server like any other HTML page or image when requested. An independent JVM runs a JAVA programme as a Standalone Application.

5.3ECLIPSE GALILEO VERSION 3.5.1

Eclipse is an Integrated development environment(IDE). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications in Java.

Figure 5.1 Eclipse Galileo

CHAPTER -6

IMPLEMENTATION

6.1WORK DONE:-

Studied various research papers in accordance with various image compression schemes i.e

Lossy compression schemes and lossless compression schemes.Studied the use of these compression schemes in various applications like medical images etc.

.Then impelemented one research paper based model for implementing Huffman coding.

Implementation of the adaptive algorithm in this semester.There are two algorithms of adaptive algorithm i.e Vitter algorithm and FGK algorithm and out of these algorithms i have implemented one algorithm.

Adaptive Huffman coding is the improved version of the Huffman coding algorithm.

Steps followed :-

1.First of all we input one text file and read that file.

2.After that we caleculated the no. Of bits by that are currently present in that file.

3. After that we compress the file by using adaptive Huffman algorithm.

4.Then we calculated the no.of bits in that file after compression.

5.Then we calculate the various parameters like compression rate by using the formula below.

Compression ratio= bits in compressed file/bits in uncompressed file

.

6.2 IMPLEMENTED CODE:-

```
package com.adaptivehuffman;

import java.util.ArrayList;

import java.util.Collections;
```

```
public class AdaptiveHuffman {
	private Node nytNode;
	private Node root;
	private char[] codeStr;
	private ArrayList<Character> alreadyExist;
	ArrayList<Node> nodeList;
	private String tempCode = "";
	public AdaptiveHuffman(char[] codeStr){
		this.codeStr = codeStr;
		alreadyExist = new ArrayList<Character>();
		nodeList = new ArrayList<Node>();
		nytNode = new Node("NEW", 0);
		nytNode.parent = null;
		root = nytNode;
		nodeList.add(nytNode);
	}
public ArrayList<String> encode(){
		ArrayList<String> result = new ArrayList<String>();
		result.add("0");
		char temp = 0;
		for ( int i=0; i<codeStr.length; i++ ) {
			temp = codeStr[i];
			result.add(getCode(temp));
```

```
updateTree(temp);
                }
                return result;
}
public String decode(){
                String result = "";
                String symbol = null;
                char temp = 0;
                Node p = root;
                symbol = getByAsc(0);
                result += symbol;
                updateTree( symbol.charAt(0) );
                p = root;
                for ( int i=9; i<codeStr.length; i++ ) {
                        temp = codeStr[i];
                        if ( temp=='0' ){
                                p = p.left;
                        }else
                                p = p.right;
                               symbol = visit(p);

                        if ( symbol!=null ){
                                if ( symbol=="NEW" ){
```

```
symbol = getByAsc(i);
i+=8;}
result+=symbol;
updateTree( symbol.charAt(0) );
p = root;
                        }
                }


                return result;
        }



        private void updateTree(char c){
                Node toBeAdd = null;
                if ( !isAlreadyExist(c) ){
                        Node innerNode = new Node(null, 1);
                        Node newNode = new Node(String.valueOf(c), 1);
                        innerNode.left = nytNode;
                        innerNode.right = newNode;
                        innerNode.parent = nytNode.parent;
                        if ( nytNode.parent!=null )
                                nytNode.parent.left = innerNode;
                        else {
```

```
                    root = innerNode;
                }
                nytNode.parent = innerNode;
                newNode.parent = innerNode;
                nodeList.add(1, innerNode);
                nodeList.add(1, newNode);
                alreadyExist.add(c);
                toBeAdd = innerNode.parent;
            } else {
                toBeAdd = findNode(c);
            }
            while ( toBeAdd!=null ) {
                Node bigNode = findBigNode(toBeAdd.frequency);
                if ( toBeAdd!=bigNode && toBeAdd.parent!=bigNode &&
bigNode.parent!=toBeAdd)
                    swapNode(toBeAdd, bigNode );
                toBeAdd.frequency++;
                toBeAdd = toBeAdd.parent;
            }
        }
private boolean isAlreadyExist(char temp) {
            for ( int i=0; i<alreadyExist.size(); i++ ) {
                if ( temp==alreadyExist.get(i) ;
```

```
return true;
                }
                return false;
        }
        private String getByAsc(int index) {
                int asc = 0;
                int tempInt = 0;
                for ( int i=7; i>=0; i-- ) {
                        tempInt = codeStr[++index] - 48;
                        asc += tempInt * Math.pow(2, i);
                }
                char ret = (char) asc;
                return String.valueOf(ret);
        }
private String visit(Node p) {
                        if ( p.letter!=null ){
                        return p.letter;
                }
                return null;
        }
private String getCode(char c){
                tempCode = "";
```

```
        getCodeByTree(root, String.valueOf(c), "");
        String result = tempCode;
        if ( result=="" ) {
                getCodeByTree(root, "NEW", "");
                result = tempCode;
                result += toBinary( getAscii(c) );
        }
        return result;
}
private Node findNode(char c) {
        String temp = String.valueOf(c);
        Node tempNode = null;
        for ( int i=0; i<nodeList.size(); i++ ) {
                tempNode = nodeList.get(i);
                if ( tempNode.letter!=null && tempNode.letter.equals(temp) )
                        return tempNode;
        }
        return null;
}private void swapNode(Node n1, Node n2) {
        int i1 = nodeList.indexOf(n1);
        int i2 = nodeList.indexOf(n2);
        nodeList.remove(n1);
        nodeList.remove(n2);
```

```
            nodeList.add( i1, n2);
            nodeList.add( i2, n1);
            Node p1 = n1.parent;
            Node p2 = n2.parent;
                    if ( p1!=p2 ) {
                    if ( p1.left==n1 ) {
                            p1.left = n2;
                    } else {
                            p1.right = n2;
                    }
if ( p2.left==n2 ) {
                            p2.left = n1;
                    } else {
                            p2.right = n1;
                    }
            } else {
                    p1.left = n2;
                    p1.right = n1;
            }
            n1.parent = p2;
            n2.parent = p1;

}           private Node findBigNode(int frequency) {
```

```
Node temp = null;
        for ( int i=nodeList.size()-1; i>=0; i--) {
                temp = nodeList.get(i);
                if ( temp.frequency==frequency )
                        break;
        }
        return temp;
}
private void getCodeByTree(Node node, String letter, String code) {
                if ( node.left==null && node.right==null ) {
                if ( node.letter!=null && node.letter.equals(letter) )
                        tempCode = code;
        } else {
                if ( node.left!=null ) {
                        getCodeByTree(node.left, letter, code + "0");
                }
                if ( node.right!=null ) {
                        getCodeByTree(node.right, letter, code + "1");
                }
        }
}

public static int getAscii(char c){
```

```
return (int)c;
        }
        public static String toBinary(int decimal){
                String result = "";
                for ( int i=0; i<8; i++ ) {
                        if ( decimal%2==0 )
                                result = "0" + result;
                        else
                                result = "1" + result;
                        decimal /= 2;
                }
                return result;
        }
        public static double calCompRate(String text, ArrayList<String> code){
                double compRate = 0;
                double preNum = 8*text.length();
                double postNum = 0;-
                System.out.println("If using huffman coding, there are in total " +
(int)postNum + " bits.");
                System.out.println("The compress rate is: " + compRate);
                return compRate;
        }
        public static void displayList(ArrayList<String>  l){
```

```
            for ( int i=0; i<l.size(); i++ ) {
                System.out.println( l.get(i) );
            }
        }
        private static String catStr(ArrayList<String> l) {
         String result = "";
            for ( String s: l ){
                result += s;
            }
            return result;
        }.
        private void getStatistics() {
            ArrayList<Symbol> symbolList = new ArrayList<Symbol>();
            preOrder(root, symbolList);

            System.out.println("Symbol size is: " + symbolList.size());
            Collections.sort(symbolList);
            calRange(symbolList);
            FileHandler.writeSymbolToFile("data/symboltable.txt", symbolList);
        }
public static void preOrder(Node node, ArrayList<Symbol> symbolList){
            if( node!=null ){
                if ( node.letter!=null && (!node.letter.equals("NEW")) ) {
```

```
                    Symbol tempSymbol = new Symbol(node.letter,
node.frequency);
                    symbolList.add(  tempSymbol );
                }
                preOrder(node.left, symbolList);
                preOrder(node.right, symbolList);
            }
        }
        private void calRange(ArrayList<Symbol> symbolList) {
            // TODO Auto-generated method stub
            int total = codeStr.length;
            double low = 0;
            for ( Symbol tempSymbol: symbolList ){
                tempSymbol.probability = tempSymbol.frequency / (double)total;
                tempSymbol.low = low;
        tempSymbol.high = low + tempSymbol.probability;
                low += tempSymbol.probability;
            }
            System.out.println("low="+low);//It should be 1.
        }
        public static void main(String[] args) {
            String text = FileHandler.readFile("data/I have a dream.txt", true);
            AdaptiveHuffman ah = new AdaptiveHuffman( text.toCharArray() );
```

```
ArrayList<String> code = ah.encode();
FileHandler.writeFile("data/ihaveadreaminHuff.txt", catStr(code), true);
ah.getStatistics();
calCompRate(text, code);
String code = FileHandler.readFile("data/ihaveadreaminHuff.txt", false);
AdaptiveHuffman ah = new AdaptiveHuffman( code.toCharArray() );
String result = ah.decode();
FileHandler.writeFile("data/IhaveadreamFromHuff.txt", result, false);
}
```

6.3OUTPUT:-

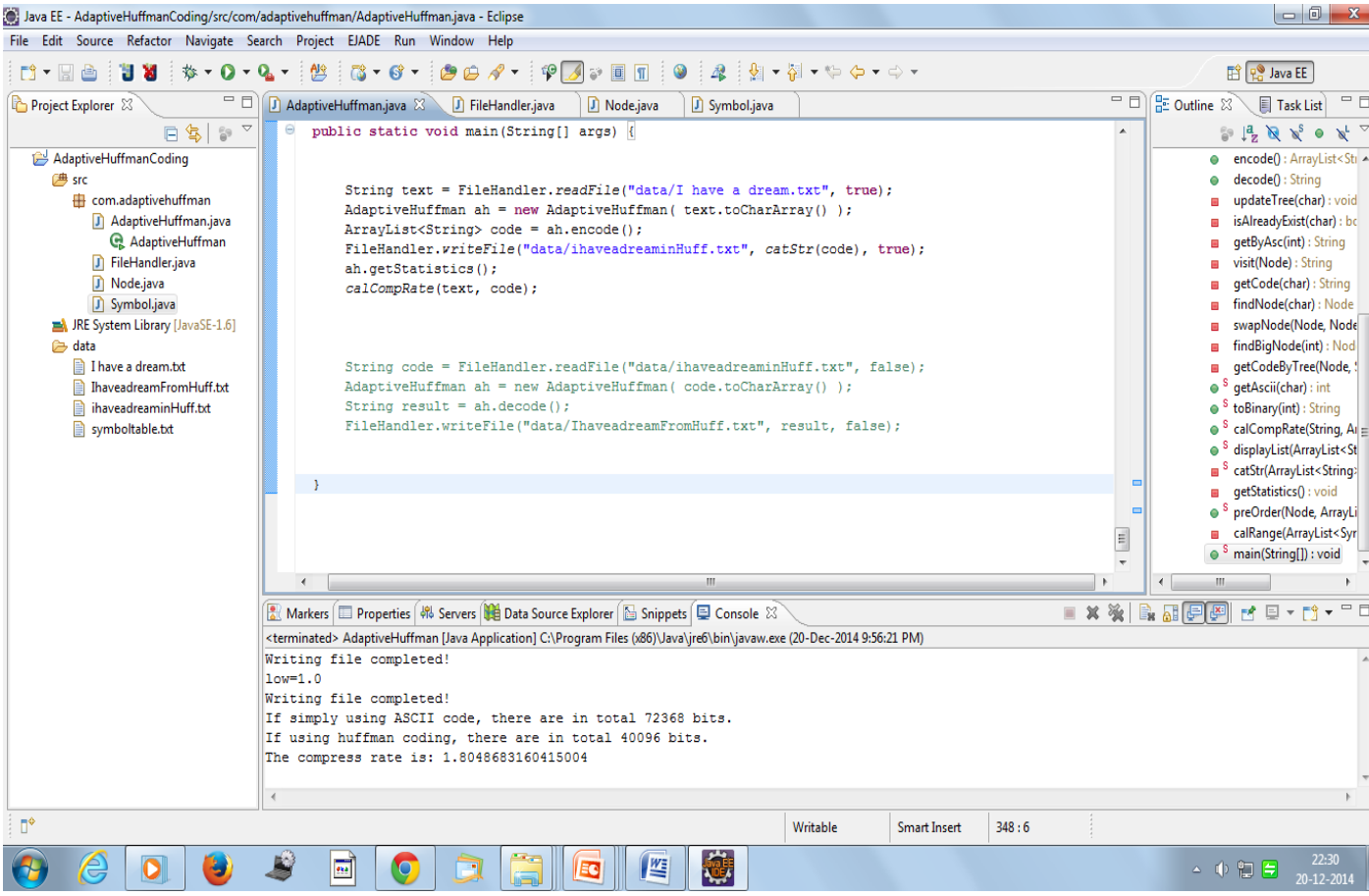


Fig 6.1 output of adaptive Huffman coding

IMPLEMENTATION OF RUN LENGTH CODING:-

```
import java.util.Scanner;

public class run{
  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);
    System.out.print("Enter string: ");
    String str = sc.nextLine();
    System.out.println("Input: " + str);

    String compressed = "";

    char ch=0;
    int count=1;
    for (int x = 0; x < str.length(); x++) {
      if (ch == str.charAt(x)){
        count = count + 1;
      } else {
        compressed = compressed + ch;
        if(count != 1){
          compressed = compressed + count;
        }
        ch = str.charAt(x);
        count = 1;
      }
    }
    compressed = compressed + ch;
    if(count != 1){
      compressed = compressed + count;
    }
    System.out.println("Compressed: " + compressed);
  }
```

}

OUTPUT:-

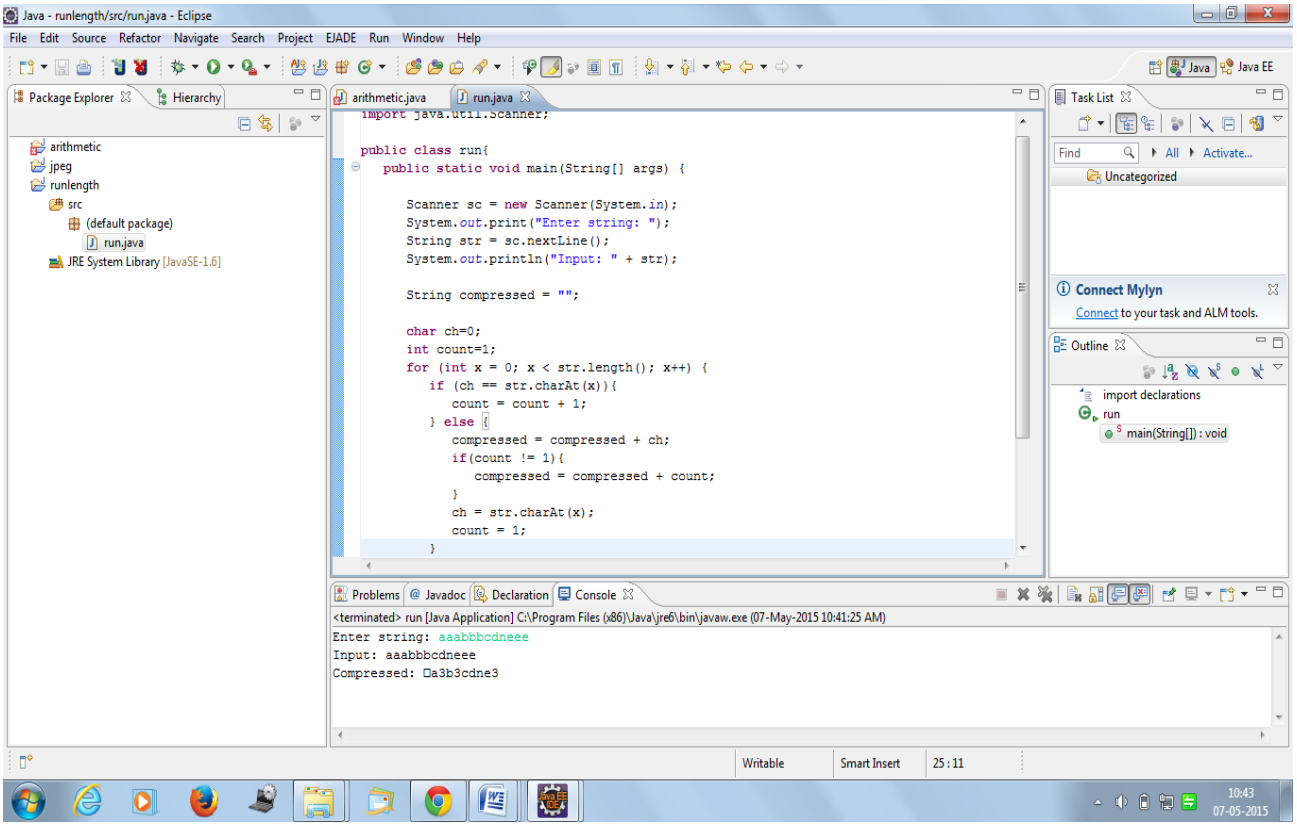


Fig 6.2 output of run length coding

IMPLEMNTATION OF ARITHMETIC ENCODING ALGORITHM:-

```
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

public class arith {

    public void encode(int[] sequence) {
        int n = sequence.length;

        List<Integer> alphabet = new ArrayList<Integer>();
        List<Integer> counts = new ArrayList<Integer>();
        for (int i = 0; i < n; ++i) {
            Integer letter = sequence[i];
            int index = alphabet.indexOf(letter);
            if (index == -1) {
                alphabet.add(letter);
                counts.add(1);
            } else {
                Integer value = counts.get(index);
                counts.set(index, value + 1);
            }
        }

        int alphabetSize = alphabet.size();
        double[] probabilities = new double[alphabetSize];
        double[] qumulative = new double[alphabetSize];

        probabilities[0] = alphabet.get(0) / (double) n;
        for (int i = 1; i < alphabetSize; ++i) {
            probabilities[i] = alphabet.get(i) / (double) n;
            qumulative[i] = qumulative[i - 1] + probabilities[i - 1];
```

```
        }


        alphabet = new ArrayList<Integer>();
        alphabet.add(1);
        alphabet.add(2);
        alphabet.add(3);

        probabilities = new double[] { 0.1, 0.6, 0.3 };
        qumulative = new double[] {0.0, 0.1, 0.7};

        sequence = new int[] { 2, 3, 2, 1, 2 };

        BigDecimal f = new BigDecimal(0);
        BigDecimal g = new BigDecimal(1);
        for (int i = 0; i < n; ++i) {
            int index = alphabet.indexOf(sequence[i]);
            double p = probabilities[index];
            double q = qumulative[index];

            BigDecimal pG = g.multiply(new BigDecimal(p));
            BigDecimal qG = g.multiply(new BigDecimal(q));

            f = f.add(qG);
            System.out.println("f: " + f.doubleValue());

            g = pG;
            System.out.println("g: " + g.doubleValue());
        }

        long codeLength = (long)(-(Math.log(g.doubleValue()) / Math.log(2.0)) + 1) + 1;
        System.out.println("length: " + codeLength);
    }
```

```
    public void decode(List<Integer> alphabet,
            double[] probabilities, int decodeLength, double code) {
        List<Integer> result = new ArrayList<Integer>();

        int alphabetSize = alphabet.size();
        double[] qumulative = new double[alphabetSize + 1];
        for (int i = 1; i < alphabetSize; ++i) {
            qumulative[i] = qumulative[i - 1] + probabilities[i - 1];
        }

        qumulative[alphabetSize] = 1;

        BigDecimal s = new BigDecimal(0);
        BigDecimal g = new BigDecimal(1);

        for (int i = 0; i < decodeLength; ++i) {
            int j = 0;
            while(s.add(g.multiply(new  BigDecimal(qumulative[j+1]))).doubleValue()  <
code) {
                j++;
            }

            s = s.add(g.multiply(new BigDecimal(qumulative[j])));
            g = g.multiply(new BigDecimal(probabilities[j]));

            result.add(alphabet.get(j));
        }
    }

    public static void main(String... args) {
        new arith().encode(new int[] { 2, 3, 2, 1, 2 });

        List<Integer> alphabet = new ArrayList<Integer>();
        alphabet.add(1);
```

```
            alphabet.add(2);
            alphabet.add(3);

            new arith().decode(alphabet, new double[]{0.1, 0.6, 0.3}, 5, 0.541);
        }
}
```

OUTPUT:-

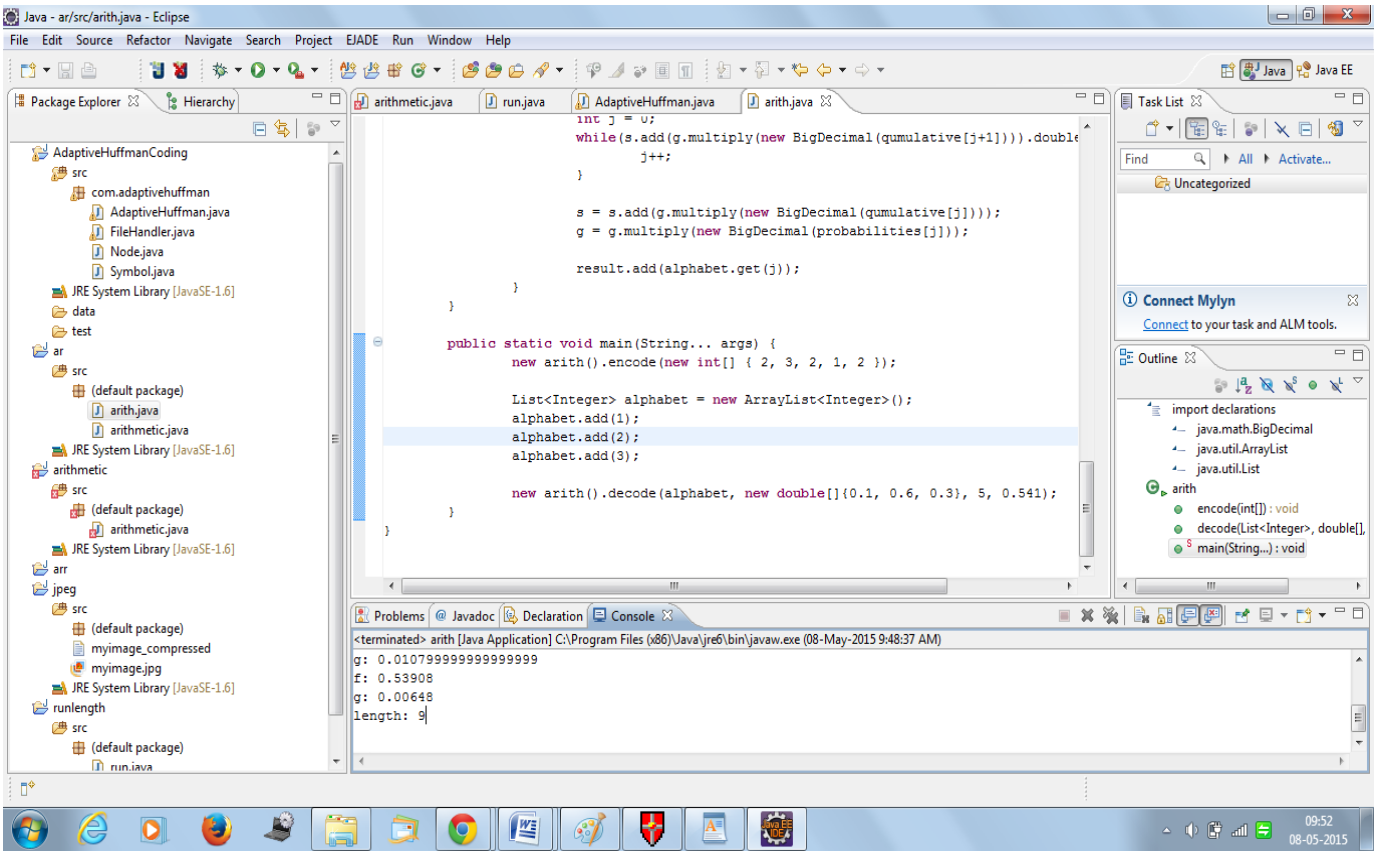


Fig 6.3 Output of Arithmetic coding

CHAPTER-7

CONCLUSION AND FUTURE WORK

The result shows that the higher Code redundancy helps to achieve more compression. The above presented Huffman coding and decoding is used for scan testing to reduce test data volume, test data compression and decompression time. Hence we conclude that Huffman coding is efficient technique for image compression and decompression to some extent. The results also reveal that the original image used for coding is almost close to the decoded output image. This study presented an analysis of Huffman compression using metrics. Image compression using the Huffman Coding depends on the no. of pixels in an image , size of an image and compression ratio. Huffman is a good coding technique for compressing the data and general types of images.

As the future work on compression of images for storing and transmitting can be done by other lossless methods of image compression because as it is concluded above, that the result of the decompressed image is almost same as that of the input image so it indicates that there is no loss of information during transmission. So other methods of image compression, any of the type i.e lossless or lossy can be carried out as namely JPEG method, LZW coding, etc. Use of different metrics can also take place to evaluate the performance of compression algorithms.

REFERENCES

[1]Aarti," Performance Analysis of Huffman Coding Algorithm,,"Department Of CSE,ACET,India, Volume 3,Issue5,May 2013.

3.[2]Vaishali G. Dubey, Jaspal Singh ," 3D Medical Image Compression Using Huffman Encoding Technique", International Journal of Scientific and Research Publications, Volume 2, Issue 9, September 2012.

4. [3] Vijay S Gulhane, Dr. Mir Sadique Ali,"Survey over Adaptive Compression Techniques," International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 1, January 2013 .

5. [4] Mridul Kumar Mathur, Seema Loonker, Dr. Dheeraj Saxena ,"Lossless huffman coding technique for image compression and reconstruction using binary trees", Assistant Professor, Department of Computer Science,Lachoo Memorial College of Science & Technology, Jodhpur,volume 3,2012.

6. Herbert Schildt, Complete Reference JAVA, 5th edition, Tata McGraw Hill.