DATA COMPRESSION

Report for Mini Project, 5th Semester

Indian Institute of Information Technology,Allahabad

1st December, 2016

by

Arun kumar reddy	- IRM2014005
Sannihith K	- IHM2014004
Pranay Kumar	- IIT2014136
Pruthvi Anvesh	- ITM2014012

Under the supervision of

Dr. Satish K Singh

Assistant Professor, Indian Institute of Information Technology, Allahabad

Candidate's Declaration

We hereby declare that the work presented in this project report entitled "**Data Compression**", submitted report of 5th Semester Mini Project at Indian Institute of Information Technology, Allahabad, is an authenticated record of our original work carried out under the guidance of **Dr. Satish K Singh**. Due acknowledgements has been made in the text to all other material used. The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Date : 1st December, 2016

Place: Indian Institute of Information Technology, Allahabad.

Certificate

I do hereby recommend that the thesis work prepared under my supervision titled "**DATA COMPRESSION**" be accepted in the partial fulfillment of the requirements of the 5th semester mini project.

Dr. Satish K Singh

Assistant Professor, IIIT Allahabad

Date : 1st December, 2016

Place : Indian Institute of Information Technology, Allahabad.

Table of Contents

Introduction

“The Amount of data produced every two days is greater than the entire data produced from the dawn of time till 2003”

- Eric Schmidt, Google

In this glorious era of communication the amount of data that we produce and consume every day increases exponentially. With each passing second the amount of data increases in Gigabytes and there will come a time when the hardware that we use to store the data will not be enough. So we have to find new techniques to store data using lesser space than it would normally take. This is why Data compression is absolutely essential.

Data Compression is used in day to day life without actual knowledge such as the media we share on social networking sites are heavily compressed so that they can be transmitted very easily. For example, In the latest era we all use expensive devices that give us high quality photos and media files but obviously take up a lot of memory. So when we attempt to share these data on Social Networking applications, The applications internally applies a compression algorithm to reduce the file size of the media files which are being shared. Now having a smaller size, These media files can be transmitted very quickly and efficiently. So Data compression is very important because it directly affects communication. There are two types of Compression

1) Lossless Data Compression

2) Lossy Data Compression

In case of Lossless Data Compression, No Data is lost or discarded and the entire amount of data is retained even after compression. In case of Lossy Data Compression, some of the data is lost forever and so we always make sure that the data that is lost during Lossy data compression has the least effect on the overall data. In the project we would be focusing on Lossless Data Compression for text files and Lossy compression for bitmap images.

Our motivation for studying Data Compression is that since there is a growing need for reducing the data that is produced .we wanted to study the different existing techniques which are currently in use and if possible try to make them more efficient. We wanted to study these different techniques and analyze their performances in different situations and form a conclusion on which technique would be better suited for a particular type of input. We were curious as to how the file compression which we use every day for a variety of purposes actually works. Another major factor was to study how social networking applications exchange media files so quickly and efficiently and implement these techniques to understand how image compression works.

Problem Description

Lossless Data Compression for Text

Objectives

- To study the different data compression techniques applied in Huffman coding.
- To implement these techniques efficiently without any data loss (Lossless Compression).
- To successfully compress the text files without any data loss and achieving a good compression ratio.
- To analyze the various implementations of Huffman Coding Algorithms and Find out the time complexities and draw conclusions.
- After performing the analysis, To Find ways to optimize the algorithm and make it more efficient in space and time.

Lossy Data Compression for Images

Objectives

- To implement the JPEG compression algorithm.
- To successfully reduce the image size without much loss in the resolution/quality of the image.
- To ensure that the data loss which occurs in the compression process does not affect the image on a large scale.
- To analyze the JPEG compression algorithm and find ways to optimize it and make it more space efficient and hence reduce the size needed to store the image.
- After performing the analysis, To optimize the program and make it more efficient while retaining the same image quality as before.

Literature Survey

Ever since Shannon published his famous research paper on information theory in 1952, a lot of research began in the field of communication and this in turn inspired people to compress the data for effective data transmission.

Most of the attempts made to compress the data involved fixed length codes used to represent the characters in the data. Huffman while still a student at MIT designed a new technique using variable length codes, this new technique result in better results because the more frequently occurring characters would have smaller length codes. Hence smaller length codes would result in lesser data being used and therefore giving a better compression ratio. However the only drawback of Huffman coding was that the frequencies of the characters had to be known in advance. In a real time data transmission it's impossible to have such data, this problem can be solved by using Adaptive Huffman coding technique which keeps updating according to the real time data. Since the adaptive Huffman coding technique is real time, it has to be implemented very efficiently enough to run quickly. A Lot of Research has been done to find better ways to implement the Adaptive Huffman Coding Technique and make it more optimal. Currently some of the popular implementations are FGK Code and Vitter Algorithm which is a slightly more advanced implementation.

Even Now there is a lot research going on in finding better ways to implement the Huffman Algorithms and therefore get a more efficient result.

Images have always been a treasured commodities right from the days of the first camera. The advancement of technologies has ensured that everyone can capture high quality images from their devices. Hence, these heavy images need to be compressed for fast and efficient transmission. JPEG is one of the most popular methods used to compress images to be transmitted easily over the internet. JPEG stands for "Joint Photographic Experts Group" who created the JPEG standard. The JPEG image compression was first introduced in 1991. There have been many improvements over the years keeping up with the updating technologies and science discoveries. This technique employs the discrete cosine transform which converts the image data from spatial domain to a frequency domain. A perceptual model based loosely on the human psychovisual system discards high-frequency information, i.e. sharp transitions in intensity, and color hue.

The human eye is more sensitive to intensity rather than the colour and we use this property to separate the image data into luminance component and the chrominance component. The luminance component stores most of the information that is sensitive to the human eye and hence encoded separately. The usage of discrete cosine transform changes the data to frequency domain which is more suitable for huffman coding. Therefore, we can considerably reduce the size of the image without actually losing out any information. Therefore, JPG is a very popular and widely used technique of image compression.

Proposed Approach

The goal is to study and implement the different Huffman Coding Techniques and then analyse them for performance before testing on different set of test inputs. Huffman Coding is a simple coding technique that makes use variable length codes depending on the probability of occurrence of the respective character. So if a character has a higher probability then it will have a smaller code length compared to the character with a lower probability. We achieve this by Constructing a Huffman tree based on the frequencies of each of the characters and the path from the root to the character node becomes the codeword of that character. In the Minimum Variance Huffman technique, we implement algorithm such that the variance in the code lengths of each of the characters is as low as possible. Extended Huffman coding Technique is a slightly modified version of Huffman Coding and it can result in a better compression ratio. After studying all these Algorithms, We will perform an analysis on each of these algorithms and test it under different conditions from which some conclusions can be drawn such compression ratio, compression time and the situations to which each of these algorithms are more suited. We implement the JPEG compression algorithm and analyse its effect on the compressed image. The input image is first converted from the RGB color space to a Luminance and chrominance color space because the eye is more sensitive to intensity compared to colour. First we divide the image into 8x8 blocks and then apply the DCT to each of these blocks to convert them from spatial to frequency domain. Then the quantization step is applied on each of the blocks. Here is where the actual data loss takes place. After the quantization step, the blocks are then separated into AC and DC components and encoded separately. Huffman coding is the technique used here to encode the DC and AC coefficients and stored as a zip file on the hard disk. In the decoding process, the AC and DC components are decoded separately and then combined after which, the inverse quantization is applied and then followed by inverse DCT to each of the blocks to obtain the original image blocks. These blocks are then combined to give the compressed image which is much smaller compared to original image. Hence, we get a compressed image of a much lower size with almost the same quality.

Hardware and Software Requirements

- GCC compiler (version > 4.8)

  The GCC compiler is used to compile and run the compression programs.

- Imagemagick - An Image processing Library

  Imagemagick is an image processing library which provides pixel level to access to the pixels in the image. This library is used to read and write into images very efficiently.

Activity Time Chart

Starting date : August 22nd 2016

Week 1 – Successfully implemented General Huffman and Minimum Variance Huffman Coding Techniques (Both encoding and decoding).

Week 2 – Successfully implemented Extended Huffman Coding Technique (Both encoding and decoding)

Week 3 – Studied and Successfully implemented Adaptive Huffman Coding Algorithm encoding technique.

Week 4 – Implemented Adaptive Huffman Coding Algorithm decoding technique.

After Midsem

Starting date : October 27th 2016

Week 1 –Improved Adaptive Huffman Technique for better time complexity & fruitful results.

Week 2 – Started working on JPEG compression and studied the Discrete Cosine Transform technique..

Week 3 – Successfully compressed and encoded the bitmap image to a zip file.

Week 4 – Successfully extracted data from the zip file and reconstructed the original image.

Week 5 - Successfully debugged all the errors and end cases and designed the final version of the project.

Analysis

<u>General Huffman Algorithm Analysis:</u> Running time O(L+ n*log n)

L is Length of input file taken

Collection of n distinct characters present in file

```
priority queue Q
Q:=n
for i=1 to n-1
        z:=new treenode
        left(z):=ExtractMin(Q)     //O(log n)
        right(z):=ExtractMax(Q)   //O(log n)
        frequency(z):=frequency(left(z))+frequency(right(z))
        Insert(Q,z)         //O(log n)
return ExtractMin(Q)
```

Hence The run time of the program will be of the order L + n*logn.

n*logn - Building Huffman tree complexity.

L - Encoding the input file.

<u>Adaptive Huffman Algorithm Analysis:</u> Running time O(L * nlog n)

L is Length of input file taken

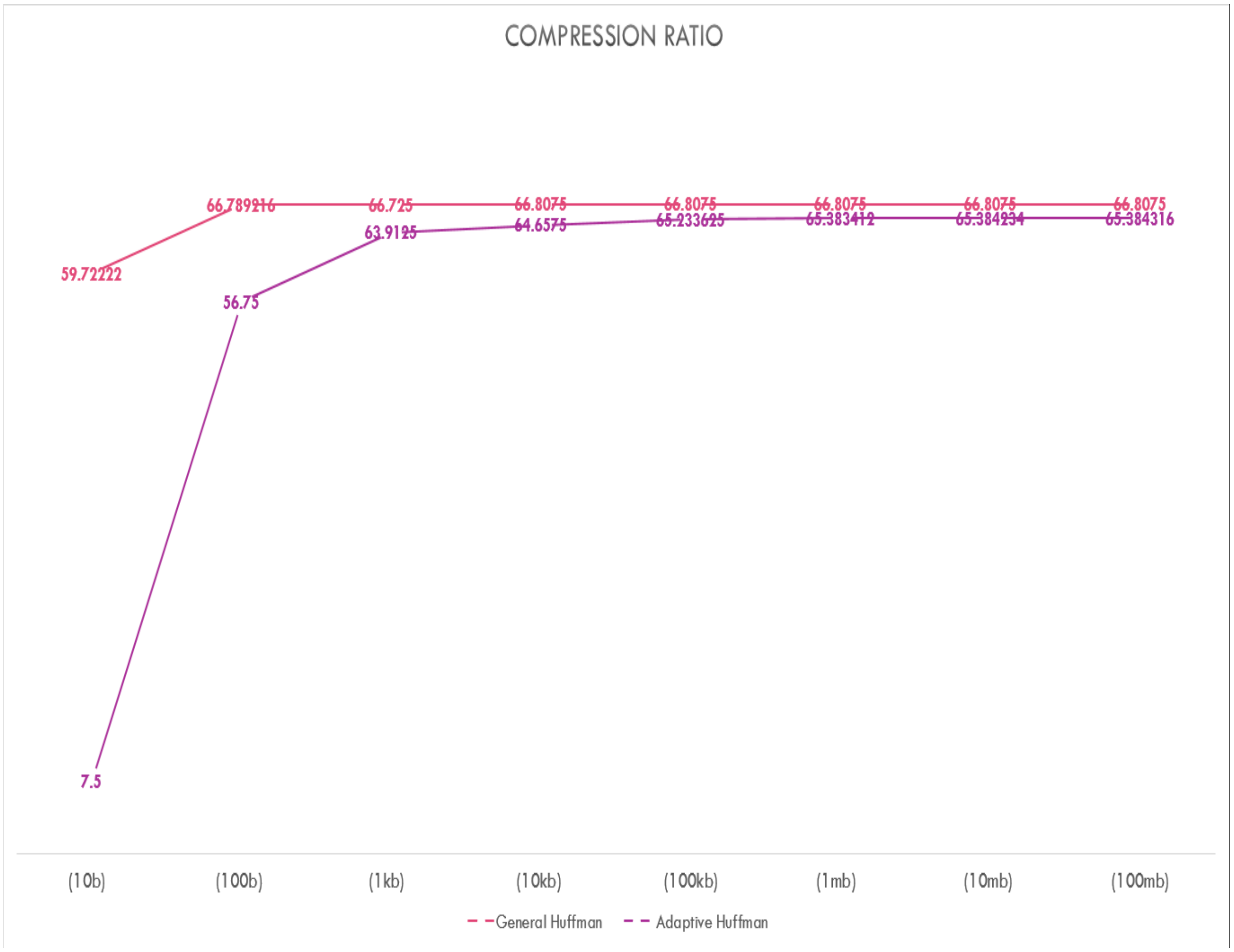
Collection of n distinct characters present in file

```
Map prefix  //prefix ascii codes for symbols occurring 1st time
Map bind   // to get symbol of node if already present in tree
setprefixes()       //set prefixes of all distinct characters //O(n*d)
```

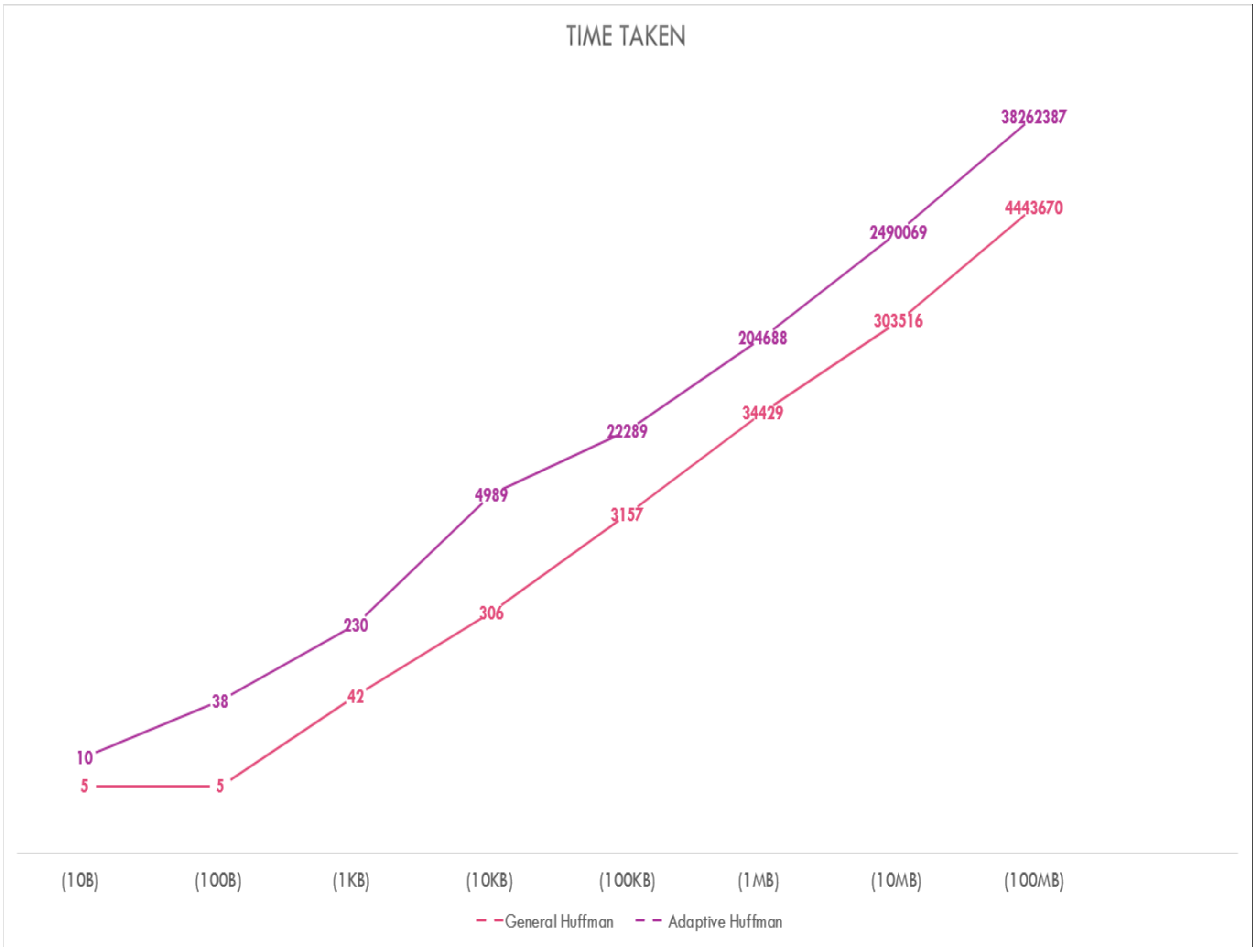
```
        prefix.insert(char,string)
add_character(new_char)
        merge is new tree node
        nyt is replaced by merge & children of merge are nyt and new_char
update(node)    //O(log n) by Master method.
        if(most significant node) then
                if(most significant node := parent.node) then
                        increment  nodeweight & update(parent.node)
                else
                        swap(most significant node & node)
                        increment node weight & update(parent.node)
        increment node weight & update(parent.node)
for  i=1 to L
        c is present character
        if  ‘c’ is present in tree then
                temp:=find ‘c’ in bind & then update temp
        else
                temp:=find ‘c’ in bind
                get the code of temp &  then update temp
```

Results

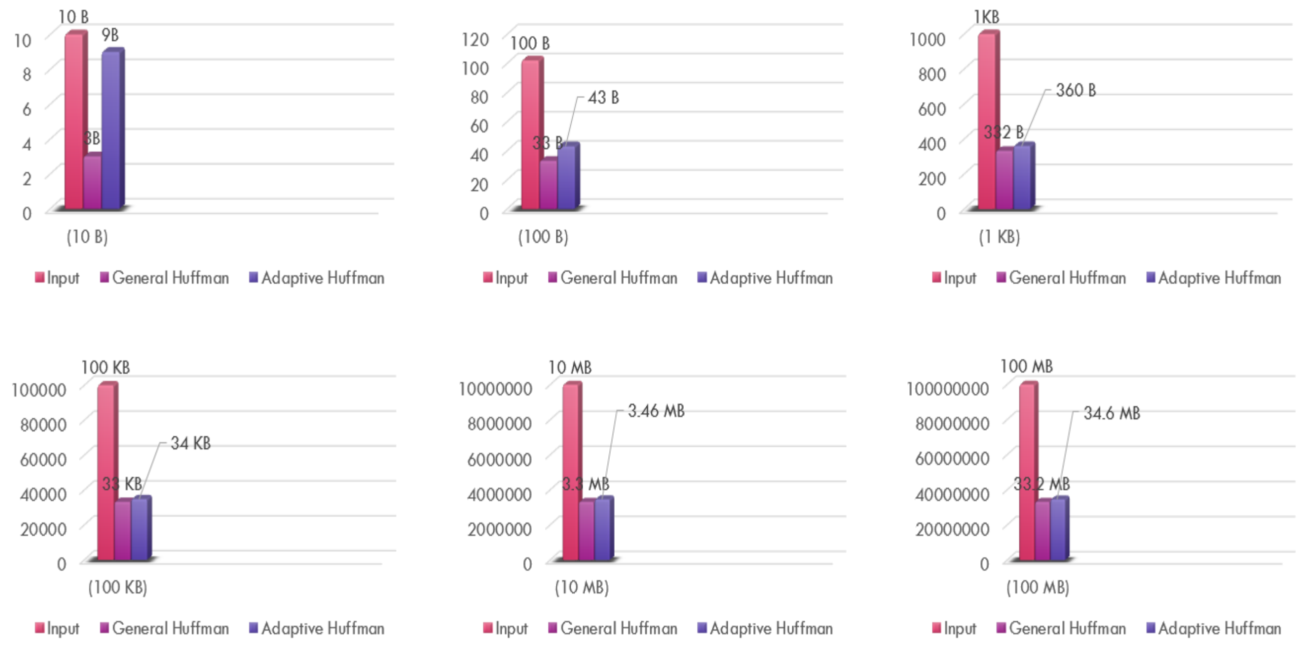
Compression ratio of General Huffman & Adaptive Huffman



Time Taken by General Huffman & Adaptive Huffman Coding



File Size Comparision before & after applying Huffman technique



JPEG Image Compression Algorithm Analysis:

DCT transformation of N*N matrix

```
for (i:=0 to N-1)
    for (j:=0 to N-1)
        temp = 0.0
        for (x:=0 to N-1)
           for (y :=0 to N-1)
                temp += Cosines[x][i] *Cosines[y][j] * Pixel[x][y]
        temp *= sqrt(2 * N) * Coefficient[i][j]
        DCT[i][j] = INT_ROUND(temp)
```

More efficient form of DCT can be calculated by matrix operations rather than above transformation because of higher complexity.

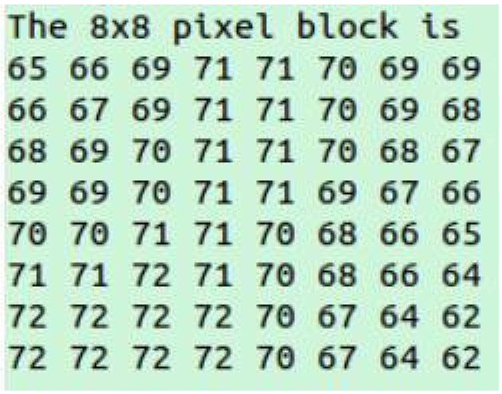
- Divide image into 8x8 blocks and level it by subtracting 128 from each entry.
- M is pixel matrix after levelling and T is DCT matrix. (Here T is orthogonal matrix).
- D := T * M * T' where D is DCT coefficient matrix which is to be quantized.
- Top left elements of D are DC components(low frequencies) .
- Select a quantization level (Matrix Q) which renders both high compression and excellent image quality.
- Quantization is achieved by dividing each element of D by corresponding element in Q.
- Quantized matrix C $C_{ij}=\text{round}(D_{ij}/Q_{ij})$
- Encode Quantized coefficients (Cij) in zig-zag sequence using better encoding technique like Huffman.
- Now the Decompression begins by multiplying each element of C with corresponding element in Q.

$$R_{ij}=C_{ij}*Q_{ij}$$

- Inverse DCT is applied to matrix R which is rounded to nearest integer which gives us decompressed JPEG version.

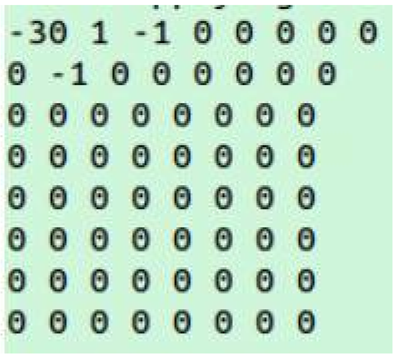
$$N=\text{round}(T' * R * T) +128$$

Pixel matrix of original image



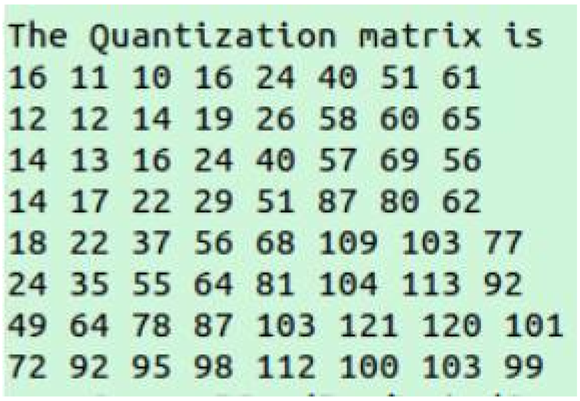
```
The 8x8 pixel block is
65 66 69 71 71 70 69 69
66 67 69 71 71 70 69 68
68 69 70 71 71 70 68 67
69 69 70 71 71 69 67 66
70 70 71 71 70 68 66 65
71 71 72 71 70 68 66 64
72 72 72 72 70 67 64 62
72 72 72 72 70 67 64 62
```

DCT coefficient Matrix after Quantization



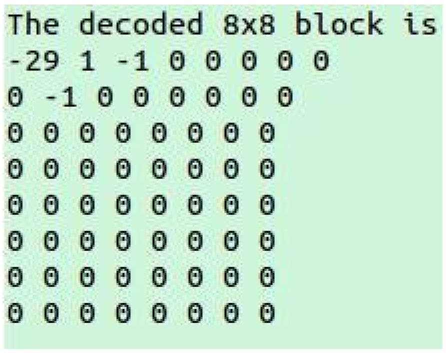
```
-30 1 -1 0 0 0 0 0
0 -1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

Quantization Matrix Q$_{50}$



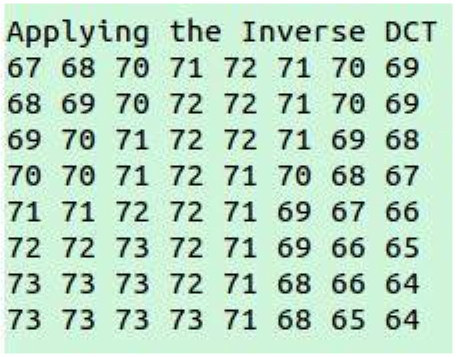
```
The Quantization matrix is
16 11 10 16 24 40 51 61
12 12 14 19 26 58 60 65
14 13 16 24 40 57 69 56
14 17 22 29 51 87 80 62
18 22 37 56 68 109 103 77
24 35 55 64 81 104 113 92
49 64 78 87 103 121 120 101
72 92 95 98 112 100 103 99
```

Decoded Matrix



```
The decoded 8x8 block is
-29 1 -1 0 0 0 0 0
0 -1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

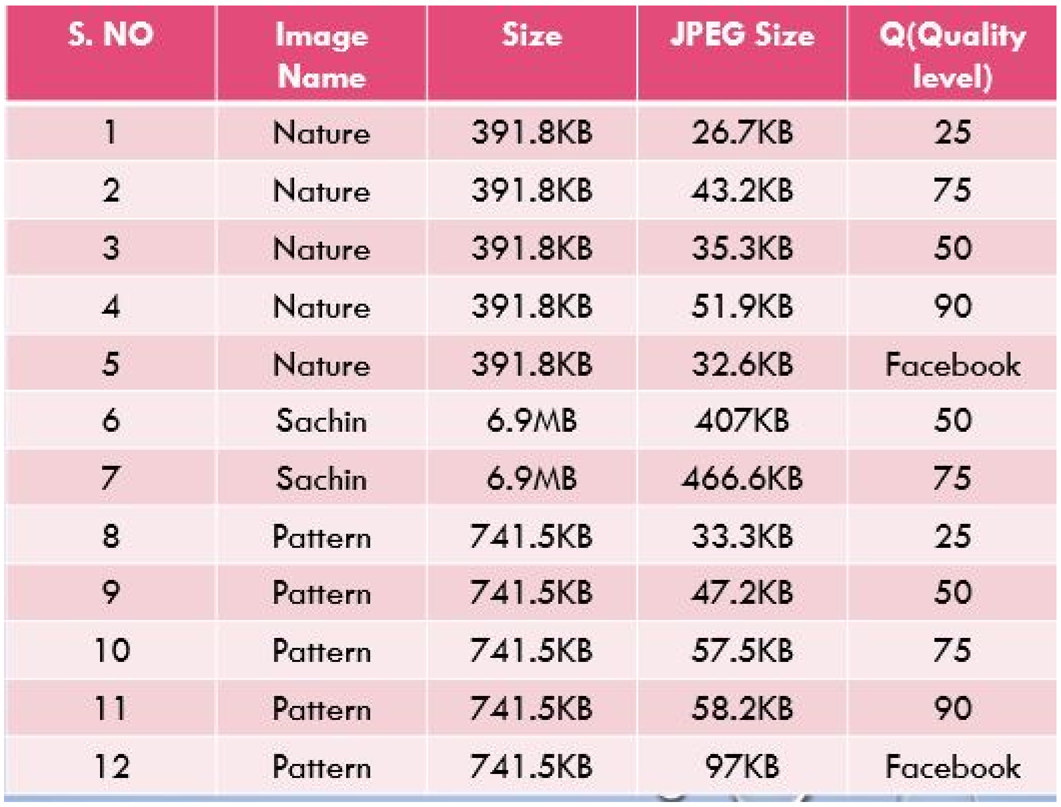
After applying inverse DCT



```
Applying the Inverse DCT
67 68 70 71 72 71 70 69
68 69 70 72 72 71 70 69
69 70 71 72 72 71 69 68
70 70 71 72 71 70 68 67
71 71 72 72 71 69 67 66
72 72 73 72 71 69 66 65
73 73 73 72 71 68 66 64
73 73 73 73 71 68 65 64
```

The matrix after applying inverse DCT is similar to original matrix

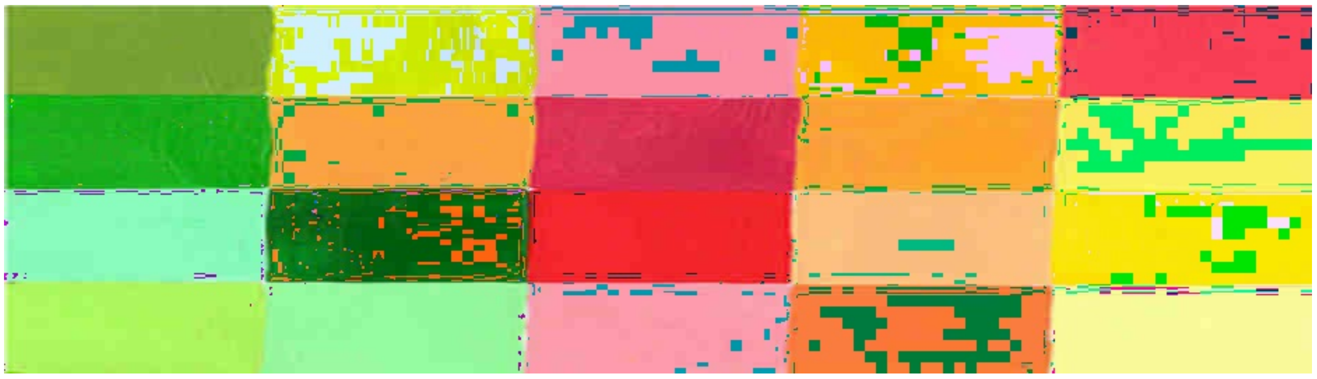
Comparison of Images with different quality levels



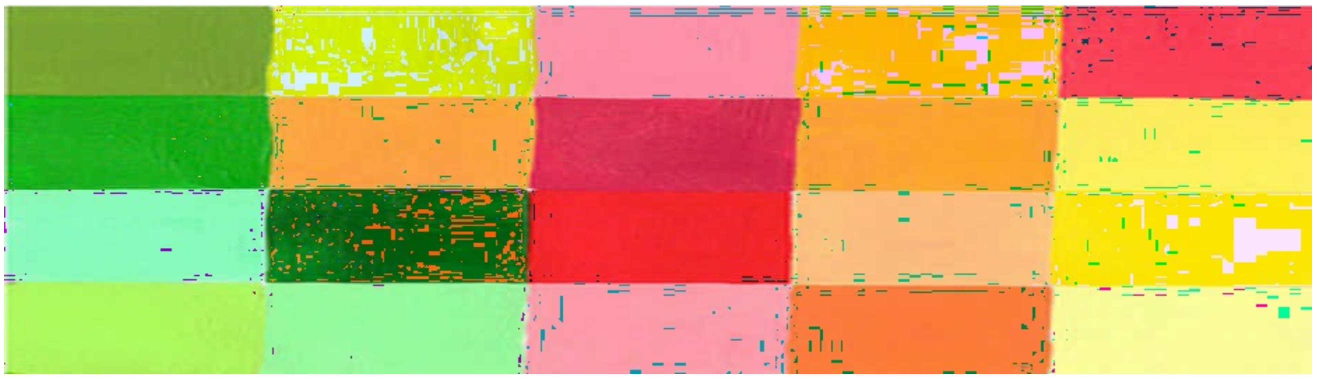
S. NO	Image Name	Size	JPEG Size	Q(Quality level)
1	Nature	391.8KB	26.7KB	25
2	Nature	391.8KB	43.2KB	75
3	Nature	391.8KB	35.3KB	50
4	Nature	391.8KB	51.9KB	90
5	Nature	391.8KB	32.6KB	Facebook
6	Sachin	6.9MB	407KB	50
7	Sachin	6.9MB	466.6KB	75
8	Pattern	741.5KB	33.3KB	25
9	Pattern	741.5KB	47.2KB	50
10	Pattern	741.5KB	57.5KB	75
11	Pattern	741.5KB	58.2KB	90
12	Pattern	741.5KB	97KB	Facebook

Progression of Quality of image with different quantization levels

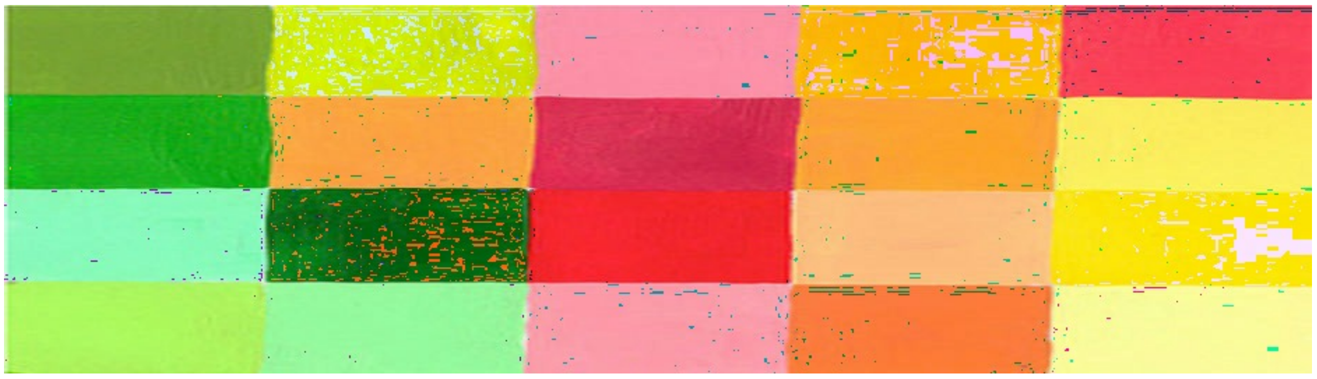
Quality level 25



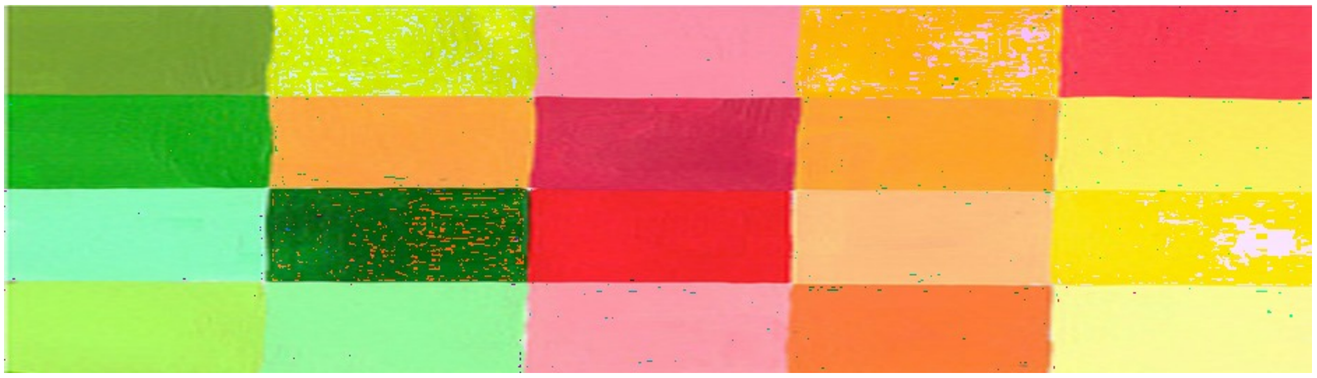
Quality level 50



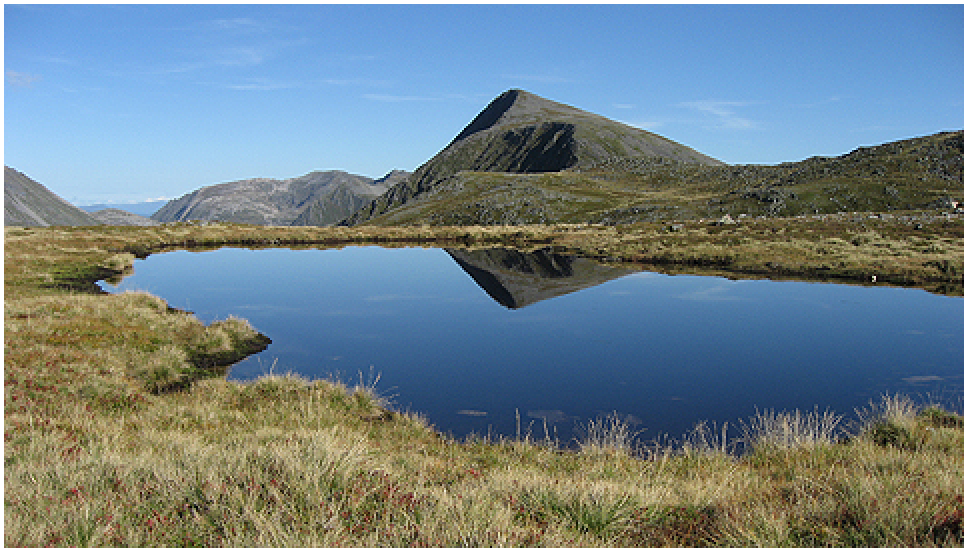
Quality level 75



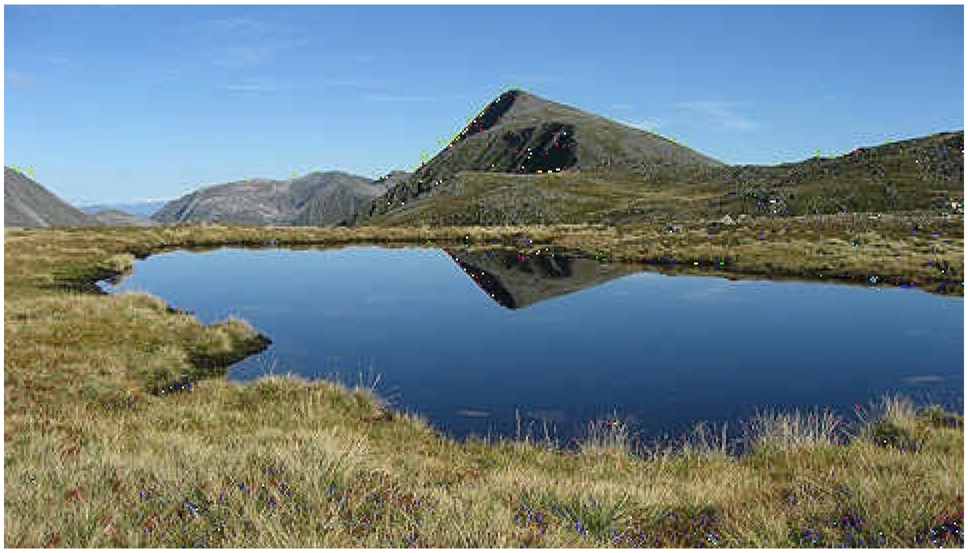
Quality level 90



Original Image



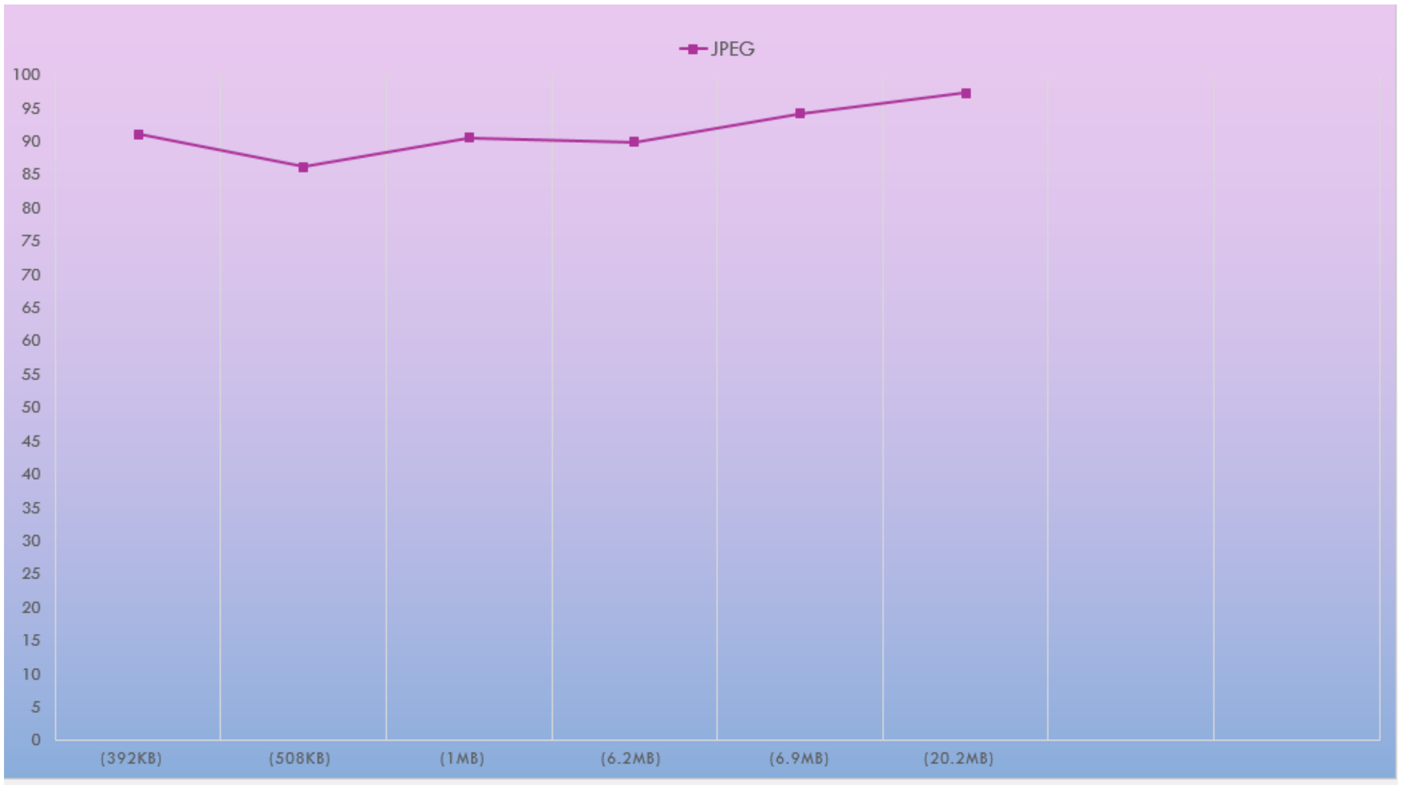
Compressed Image



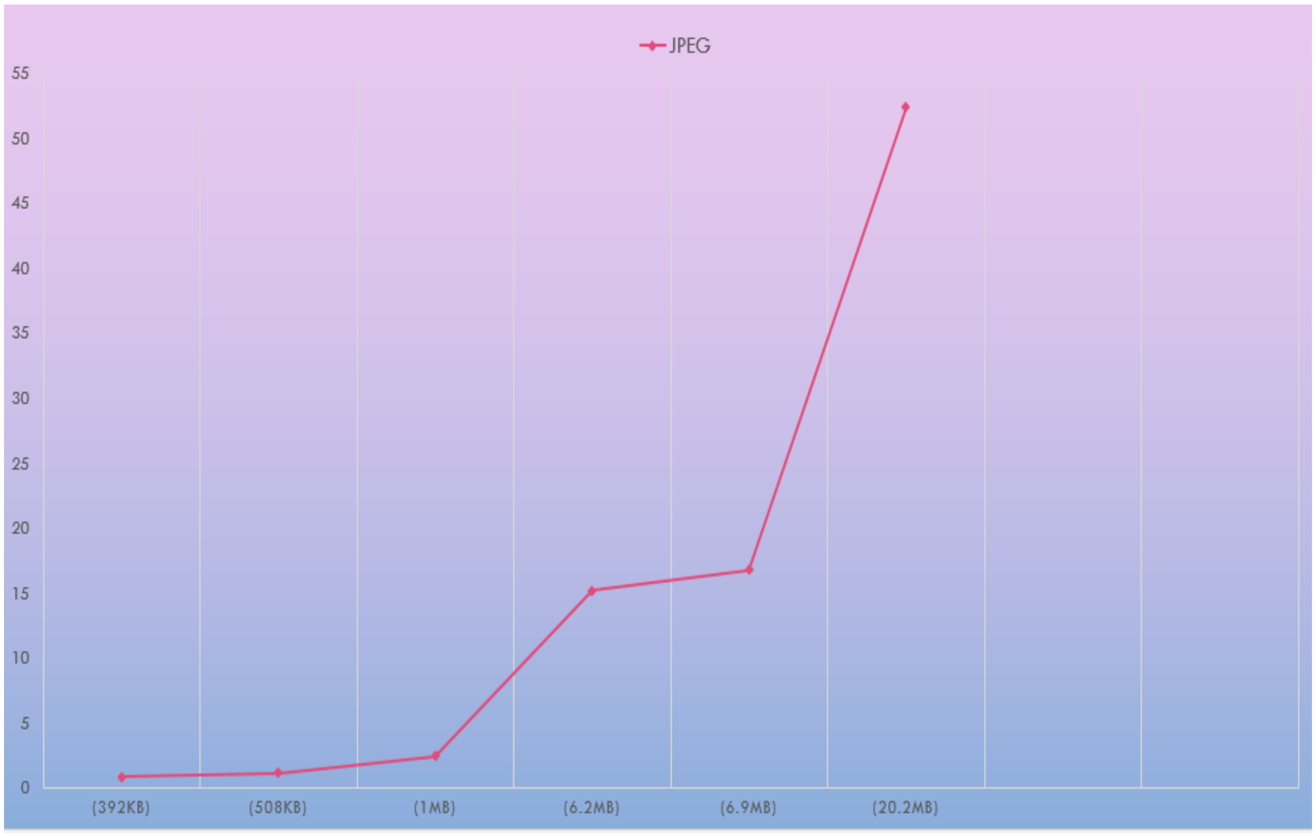
Comparison of Various File Sizes with Compression ratio & Running Time(Q=50)

Original file size	Compressed File size	Standard Compression	Compression Ratio	Running Time
20.2MB	556KB	672KB	97.31%	53seconds
6.9MB	403KB	512KB	94.20%	16.8sec
6.2MB	427KB	532KB	89.60%	15.2sec
1MB	95KB	170KB	92%	2.536sec
508KB	70.6KB	90KB	85.70%	1.2sec
392KB	35KB	61KB	91.30%	0.945sec

Compression Ratios of Various images



Running time for compressing various images of different sizes.



Conclusion

Successfully implemented Lossless compression of data using various types of Huffman codes, particularly Adaptive Huffman technique which is used in real time application of data in text format.

Compressed bitmap images to jpeg images by JPEG compression technique using DCT and quantization. Tested for different images of various pixels and also achieved around 93% compression of size with retaining picture quality almost as it is. Optimized the standard JPEG technique by selective encoding in a zig-zag fashion and hence giving us a considerably better results. The image quality was as good as the original with only slight data loss. Upon sending these same images using Facebook, the image quality had diminished very badly despite having nearly the same size as the images generated by applying the JPEG compression. Hence, The implementation of JPEG was performing better than the compression algorithm used by Facebook as a much better quality image was being generated despite having nearly the same size.

References

- Introduction to Data Compression – Khalid Sayood
- Complete Reference to Data Compression -- David Salomon
- MIT OPENCOURSEWARE and NPTEL
- https://courses.cs.washington.edu/courses/csep590a/07au/lectures/lecture02small.pdf
- http://www.binaryessence.com/dct/en000111.htm
- Wikipedia - Quantization