# IMAGE COMPRESSION USING HUFFMAN CODING

A

Mini Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

By

**AKASH.S.VORA**

**1602-19-733-126**

**KANCHARLA SRINAYAN**

**1602-19-733-171**

**Department of Computer Science & Engineering**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2021**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

## DECLARATION BY THE CANDIDATE

I, **AKASH S VORA,** bearing hall ticket number, **1602-19-733-126**, hereby declare that the project report entitled **"IMAGE COMPRESSION USING HUFFMAN CODING"** Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

**Akash S Vora,**

**1602-19-733-126.**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**Department of Computer Science & Engineering**

**BONAFIDE CERTIFICATE**

This is to certify that the project entitled **“IMAGE COMPRESSION USING HUFFMAN CODING”** being submitted by **AKASH S VORA,** bearing **1602-19-733-126,** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

**Ms. M. Sunitha Reddy**

**Assistant Professor**

**Dept. of CSE**

# ACKNOWLEDGEMENT

With immense pleasure, we record our deep sense of gratitude to our guide Ms. M. Sunitha Reddy, Assistant Professor, Vasavi College of Engineering, Hyderabad, for the valuable guidance and suggestions, keen interest and thorough encouragement extended throughout the period of the project work. I consider myself lucky enough to be part of this project. This project would add as an asset to my academic profile.

We express our thanks to all those who contributed for the successful completion of our project work.

# ABSTRACT

Images play an indispensable role in representing vital information. Thus, it needs to be saved for further use or must be transmitted over a medium. In order to have efficient utilization of disk space and transmission rate, images need to be compressed. Image compression is the technique of reducing the file size of a image without compromising with the image quality at acceptable level and is one of the most important steps in image transmission and storage.

Huffman coding is regarded as one of the most successful lossless compression techniques. It is based on the frequency of occurrence of a data item (pixel in images) and provides the least amount of information bits per source symbol. In this project, an image (in .bmp format) is taken as an input and using Huffman Coding, each pixel (in binary format) is compressed and then the percentage of compression is calculated. In addition to this, the encoded text is written in a text file.

# TABLE OF CONTENTS

**Page No.**

# LIST OF FIGURES

# INTRODUCTION

Multimedia images have become a vital component of everyday life. The amount of information encoded in an image is quite large. Even with the advances in bandwidth and storage capabilities, if images were not compressed, many applications would be too costly.

Thus, image compression is done by removing all redundant information. In this project, we have used the Huffman Coding technique to compress multiple bitmap images and finally calculate the average percentage of data saved over multiple images.

## 2.1 OVERVIEW

Information that can be viewed is very important for us to identify, recognize and understand the surrounding world. Basically, an image is a two dimensional array of dots, called pixels. The size of the image is the number of pixels. Every pixel in an image is a certain color. The shade of the image whether gray or color displayed for a given image (pixel) solely depends on the number that is stored in the array for the pixel. An image that takes large amount of data requires more memory to store, takes longer time to be transfered, and is difficult to process. Image compression becomes important due to the limit in the communication bandwidth, CPU speed, time taken for transmission and size required to store. The main aim of Image compression is to minimize the size of the image in bytes of a graphics file by maintaining a good quality of the image. A common characteristic of most images is that the neighboring pixels are correlated and therefore contain redundant information. The foremost task then is to find less correlated representation of the image. Using the compression algorithms, redundant bits are removed from the image so that image size is reduced and the image is compressed.

Image compression methods are classified into lossy and lossless compression. In lossless compression there is no information loss; the reconstructed image is exactly the same as the original, which is preferred for high value content, such as medical imagery or image scans made for archival purposes, artificial images such as technical drawings, icons or comics. Lossless compression methods include run length encoding, Huffman encoding, etc., In lossy compression, the reconstructed image contains degradation relative to the original because the compression scheme completely discards redundant information. However, lossy schemes are capable of achieving much higher compression. Under normal viewing conditions, no visible loss is perceived. Lossy compression is most commonly used to compress multimedia data like audio, video, and still images, especially in applications such as streaming media where minor loss of fidelity is acceptable to achieve a substantial reduction in bit rate, especially when used at low bit rates introduce compression artifacts.

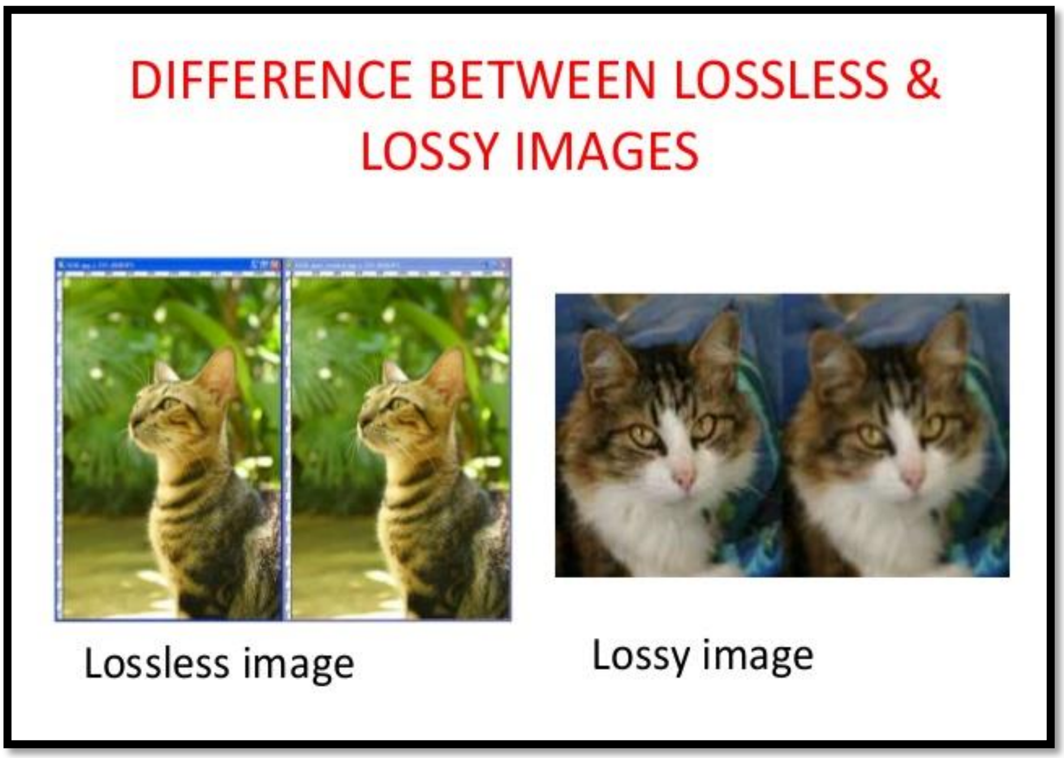



Figure 2.1.1

Lossless compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. Huffman Coding is a lossless compression technique.

**LOSSLESS IMAGE REPRESENTATION FORMAT : (BMP FORMAT)**

A bitmap (.bmp) is a type of memory organization or image file format used to store digital images. The term bitmap comes from the computer programming terminology, meaning just a map of bits, a spatially mapped array of bits. It is an uncompressed format.

Figure 2.1.2

# 2.2 PROCEDURE

- ❖ The first step of Huffman coding technique is to read the Bitmap image (which is in .bmp format) into a 2D array.
- ❖ The second step is to reduce the input image to a ordered histogram, where the probability of occurrence of pixel intensity values present in the image are stored.
- ❖ Find the number of pixel intensity values having non-zero probability of occurrence and calculate the maximum length of Huffman code words.
- ❖ Build the Huffman Tree.
- ❖ Backtrack from the root to the leaf nodes to assign code words.

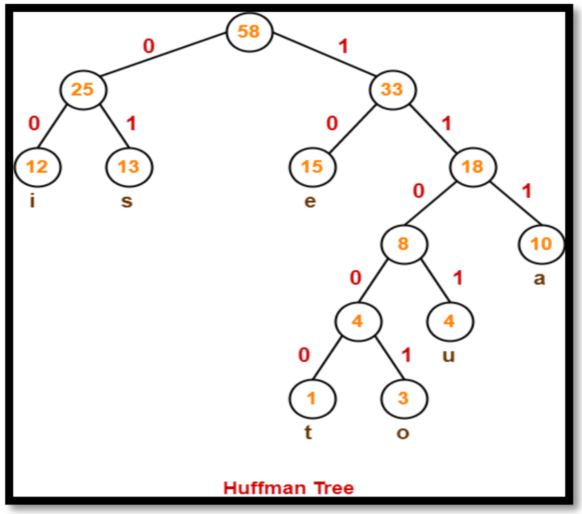



Figure 2.2.1

- ❖ Encode the image and write the Huffman encoded image to a text file.
- ❖ Print the Huffman codes.
- ❖ Calculate the percentage of data saved for each image and display the average percentage of the data saved.

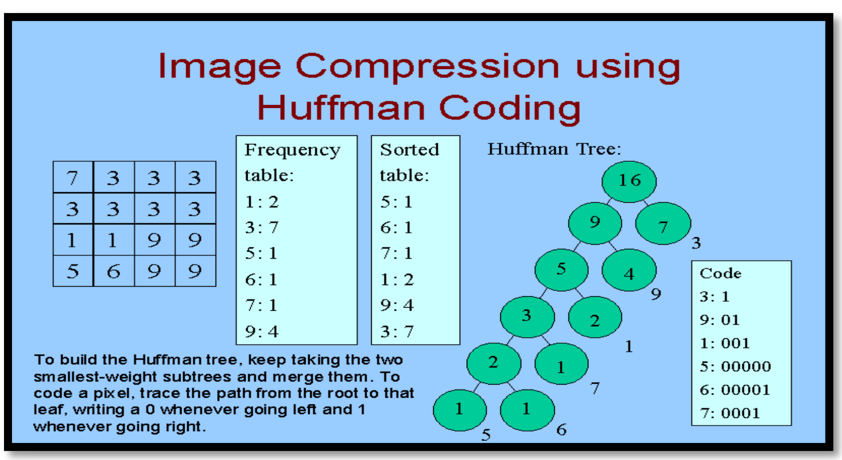



Figure 2.2.2

## 2.3 APPLICATIONS OF IMAGE COMPRESSION

Image compression has increased the efficiency of sharing and viewing personal images, it offers the same benefits to just about every industry in existence. Image compression was most commonly used in the data storage, printing and telecommunication industry. The digital form of image compression is also being put to work in industries such as fax transmission, satellite remote sensing, and high definition television.

In certain industries, the archiving of large numbers of images is required. A good example is the health industry, where the constant scanning and/or storage of medical images and documents take place. Image compression offers many benefits here, as information can be stored without placing large loads on system servers. Depending on the type of compression applied, images can be compressed to save storage space, or to send to multiple physicians for examination. And conveniently, these images can uncompress when they are ready to be viewed, retaining the original high quality and detail that medical imagery demands.

It is also useful to any organization that requires the viewing and storing of images to be standardized, such as a chain of retail stores or a federal government agency. In the security industry, image compression can greatly increase the efficiency of recording, processing and storage.

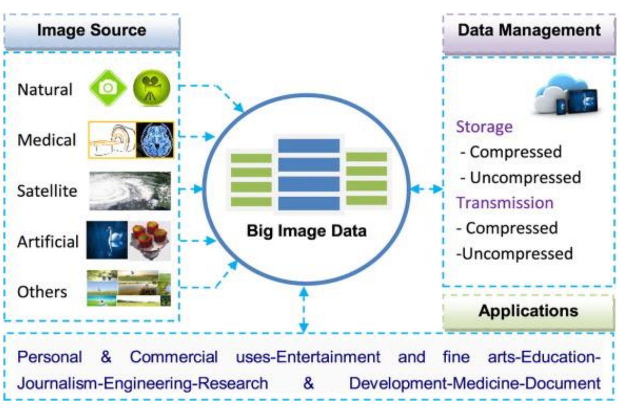



Figure 2.3.1

## 2.4 MOTIVATION

An image, 1024 pixel x 1024 pixel x 24 bit, without compression, would require around 3 MB of storage and several minutes for transmission. If the image is compressed at a certain compression ratio, the storage requirement is reduced and the transmission time drops significantly.

In a distributed environment large image files remain a major bottleneck within systems. Compression is an important component of the solutions available for creating file sizes of manageable and transmittable dimensions. Increasing the bandwidth is another method, but the cost sometimes makes this a less attractive solution. The easiest way to reduce the size of the image file is to reduce the size of the image itself. By shrinking the size of the image, fewer pixels need to be stored and consequently the file will take less time to load.

## 2.5 OBJECTIVE

To compress multiple bitmap (.bmp) images and calculate the average percentage of data saved.

# SYSTEM REQUIREMENTS

## Hardware:

- Minimum RAM required: 512 MB
- Input devices: Mouse, Keyboard
- Output devices: Monitor

## Software:

- Visual Studio Code
- Windows 8.1 or above

# IMPLEMENTATION

```
//This is a project for representing Image compression using Huffman Coding

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

// Function to concatenate two words based on condition like adding '0' or '1' to the
encoded text

void stringconcat(char* str, char* parentcode, char add);

// This Function would be useful for calculating the maximal codeword length in
Huffmann Codes

int fibo(int n);


void delay(float number_of_seconds)

{

   int ms = 1000 * number_of_seconds;

   clock_t start_time = clock();

   while (clock() < start_time + ms);

}


int main()

{
```

```
    int menu=0;

    int counter=0;

    int avg_perc=0;

    while(menu==0)

    {

        printf("\n*******************************************\n");

        printf("\tWELCOME TO THE IMAGE COMPRESSOR\n");

        printf("*******************************************\n");

        printf("THIS TOOL USES HUFFMAN CODING TO COMPRESS THE 
IMAGE GIVEN IN .BMP FORMAT.\n\n");

        int i, j;

        char filename[50];

        printf("Please enter the image you want to compress : ");

        scanf("%s",filename);

        int data = 0, offset, bpp = 0, width, height;

        long bmpsize = 0, bmpdataoff = 0;

        int** image;

        int temp = 0;

        clock_t  start,end;

        double time_taken;


        //Reading the BMP File

        FILE* image_file;
```

```
    image_file = fopen(filename, "rb");
    if (image_file == NULL)
    {
        printf("Error Opening File!!");
        exit(1);
    }
    else
    {
        start = clock();
        // Set file position of the stream to the beginning which contains file signature
"BM" for bitmap images
        printf("Processing BMP Header");
        for(int j=0;j<6;j++)
        {
            printf(".");
            delay(1);
        }
        printf("\n");

        //Set file position to the beginning which contains the ID  of the image "BM"
        offset = 0;
        fseek(image_file, offset, SEEK_SET);
        //Printing the ID of the file "BM" (if it is a .bmp file)
```

```
for(i = 0 ; i < 2 ; i++)
{
    fread(&data, 1, 1, image_file);
    printf("%c",data);
}
printf("\n");

// Set file position/offset to 2, which contains size of BMP File
offset = 2;
fseek(image_file, offset, SEEK_SET);
// Getting size of BMP File
fread(&bmpsize, 4, 1, image_file);

// Getting offset where the pixel array starts, since the information is at offset 10 from the start, as given in BMP Header
offset = 10;
fseek(image_file, offset, SEEK_SET);
// Bitmap data offset
fread(&bmpdataoff, 4, 1, image_file);

// Getting height and width of the image
// Width is stored at offset 18 and
// Height at offset 22, each of 4 bytes
```

```
fseek(image_file, 18, SEEK_SET);
fread(&width, 4, 1, image_file);
fread(&height, 4, 1, image_file);

// Number of bits per pixel
fseek(image_file, 2, SEEK_CUR);
fread(&bpp, 2, 1, image_file);

long long int no_of_bits = width*height*bpp;
printf("Number of bits in the original BMP image is %d bits.\n",no_of_bits);
printf("Number of bits per pixel is : %d bits.\n",bpp);

// Setting offset to the start of pixel data
fseek(image_file, bmpdataoff, SEEK_SET);

// Creating Image array
image = (int**)malloc(height * sizeof(int*));
for (i = 0; i < height; i++)
{
   image[i] = (int*)malloc(width * sizeof(int));
}

//Number of bytes in the image pixel array
```

```
int numbytes = (bmpsize - bmpdataoff) / 3;


// Reading the BMP File into image[] array
for (i = 0; i < height; i++)
{
    for (j = 0; j < width; j++)
    {
        fread(&temp, 3, 1, image_file);
        // the Image is a 32-bit BMP Image
        // 24 bits per pixel - Color
        // 8 bits - Transparency
        // 0x0000FF - 255 , since there are 256 pixel intensities - Thus, each pixel
intensity value is between 0 and 256
        temp = temp & 0x0000FF;
        image[i][j] = temp;
    }
}
end = clock();
}
time_taken = ((double)(end-start))/CLOCKS_PER_SEC;
printf("Total time taken for processing BMP Image : %f seconds\n",time_taken-
6); //Subtracted 6 due to the delay(1) given above.
```

```
// Finding the probability of occurrence

int hist[256];

for (i = 0; i < 256; i++)

{

   hist[i] = 0;

}


for (i = 0; i < height; i++)

{

   for (j = 0; j < width; j++)

   {

      hist[image[i][j]] += 1;

   }

}


//Finding number of non-zero occurrences, since all 256 intensities might not be
present in the BMP image

int nodes = 0;

for (i = 0; i < 256; i++)

{

   if (hist[i] != 0)

      nodes++;

}
```

```
// Calculating minimum probability among all probabilities of pixel intensities

float p = 1.0, ptemp;

for (i = 0; i < 256; i++)

{

   ptemp = (hist[i] / (float)(height * width));

   if (ptemp > 0 && ptemp <= p)

       p = ptemp;

}


//Calculating max length of Huffmann code word

i = 0;

while ((1 / p) > fibo(i))

   i++;

int maxcodelen = i - 3;


//Declaring these 2 structs so that it contains the information of all leaf nodes in the Huffman Tree

struct pixfreq

{

   int pix;

   float freq;
```

```
    struct pixfreq *left, *right;
    char code[maxcodelen];
};

struct sorted_pixfreq
{
    int pix, arrloc;
    float freq;
};

struct pixfreq* pix_freq;
struct sorted_pixfreq* huffcodes;
//If there are n leaf nodes, there are 2*n-1 nodes in the Huffman Tree
int totalnodes = 2 * nodes - 1;
pix_freq = (struct pixfreq*)malloc(sizeof(struct pixfreq) * totalnodes);
huffcodes = (struct sorted_pixfreq*)malloc(sizeof(struct sorted_pixfreq) *
nodes);

j = 0;
int totpix = height * width; //Total number of pixels
float probability;
for (i = 0; i < 256; i++)
{
```

```
    if (hist[i] != 0)
    {

        //Pixel intensity value
        huffcodes[j].pix = i;
        pix_freq[j].pix = i;

        //Location of the node in the pix_freq array
        huffcodes[j].arrloc = j;

        //Probability of occurrence of each pixel intensity value
        probability = (float)hist[i] / (float)totpix;
        pix_freq[j].freq = probability;
        huffcodes[j].freq = probability;

        //Declaring the child of leaf node as NULL pointer
        pix_freq[j].left = NULL;
        pix_freq[j].right = NULL;

        pix_freq[j].code[0] = '\0';
        j++;
    }
}
```

```
//Sorting the struct sorted_pixfreq using a temporary variable
struct sorted_pixfreq temphuff;

//Sorting w.r.t probability of occurrence in descending order
for (i = 0; i < nodes; i++)
{
    for (j = i + 1; j < nodes; j++)
    {
        if (huffcodes[i].freq < huffcodes[j].freq)
        {
            temphuff = huffcodes[i];
            huffcodes[i] = huffcodes[j];
            huffcodes[j] = temphuff;
        }
    }
}

//Building Huffman Tree
float combined_prob;
int combined_pix;
int n = 0, k = 0;
int nextnode = nodes; //Used for appending values to the struct pixfreq
```

```
    while (n < nodes - 1)
    {

        //Adding the lowest two probabilities
        combined_prob = huffcodes[nodes - n - 1].freq + huffcodes[nodes - n - 2].freq;
        combined_pix = huffcodes[nodes - n - 1].pix + huffcodes[nodes - n - 2].pix;

        //Appending to the pix_freq array
        pix_freq[nextnode].pix = combined_pix;
        pix_freq[nextnode].freq = combined_prob;
        pix_freq[nextnode].left = &pix_freq[huffcodes[nodes - n - 2].arrloc]; //Left child
        pix_freq[nextnode].right = &pix_freq[huffcodes[nodes - n - 1].arrloc]; //Right child
        pix_freq[nextnode].code[0] = '\0';
        i = 0;

        while (combined_prob <= huffcodes[i].freq)
            i++;

        // Inserting the new node in the huffcodes array
```

```
    for (k = nodes; k >= 0; k--)
    {
        if (k == i)
        {
            huffcodes[k].pix = combined_pix;
            huffcodes[k].freq = combined_prob;
            huffcodes[k].arrloc = nextnode;
        }
        //Else shifting the nodes to the right in huffcodes array
        else if (k > i)
            huffcodes[k] = huffcodes[k - 1];

    }
    n += 1;
    nextnode += 1;
}

//Assigning Huffman Codes through backtracking
char left = '0';
char right = '1';
int index;
for (i = totalnodes - 1; i >= nodes; i--)
{
```

```
    if (pix_freq[i].left != NULL)
        stringconcat(pix_freq[i].left->code, pix_freq[i].code, left);
    if (pix_freq[i].right != NULL)
        stringconcat(pix_freq[i].right->code, pix_freq[i].code, right);
}

// Encode the Image
int pix_val;
int l;

// Writing the Huffman encoded image into a text file "encoded_image.txt"
FILE* imagehuff = fopen("encoded_image.txt", "wb");
int res_len = 0;
for (i = 0; i < height; i++)
    for (j = 0; j < width; j++)
    {
        pix_val = image[i][j];
        for (l = 0; l < nodes; l++)
            if (pix_val == pix_freq[l].pix)
                fprintf(imagehuff, "%s", pix_freq[l].code);
                res_len+= strlen(pix_freq[l].code);
    }
```

```
printf("\nHUFFMAN CODES : \n\n");
printf("PIXEL VALUES ==>   CODE\n\n");
for (i = 0; i < nodes; i++)
{
   if (snprintf(NULL, 0, "%d", pix_freq[i].pix) == 2)
   {
      printf("     %d      ==>  %s\n", pix_freq[i].pix, pix_freq[i].code);
   }
   else
   {
      printf("    %d      ==>  %s\n", pix_freq[i].pix, pix_freq[i].code);
   }
   delay(0.10);

}

float avgbitnum = 0;
for (i = 0; i < nodes; i++)
   avgbitnum += pix_freq[i].freq * strlen(pix_freq[i].code);
printf("Average number of bits per pixel : %f", avgbitnum);
printf("\nNumber of bits in the encoded text is %d.\n",res_len);
int ind_perc = ((width*height*32) - res_len)*100/(width*height*32);
```

```
        printf("Percentage of data saved : %d \n",((width*height*32) -
res_len)*100/(width*height*32));

        avg_perc+=ind_perc;

        //Number of iterations

        counter++;

        printf("ENTER 0 TO RUN THE TOOL AGAIN OR 1 TO EXIT - ");

        scanf("%d",&menu);

        if(menu==1)

        {

            printf("AVERAGE PERCENTAGE SAVED OVER ALL IMAGES IS : %d
",avg_perc/counter);

            printf("\n\n");

            return 0;

        }


    }


}


void stringconcat(char* str, char* parentcode, char add)

{

    int i = 0;

    while (*(parentcode + i) != '\0')
```

```
    {
        *(str + i) = *(parentcode + i);
        i++;
    }
    if (add != '2')
    {
        str[i] = add;
        str[i + 1] = '\0';
    }
    else
        str[i] = '\0';
}

int fibo(int n)
{
    if (n <= 1)
        return n;
    return fibo(n - 1) + fibo(n - 2);
}
```

# OUTPUT OF THE PROGRAM

```
**********************************************

       WELCOME TO THE IMAGE COMPRESSOR
**********************************************

THIS TOOL USES HUFFMAN CODING TO COMPRESS THE IMAGE GIVEN IN .BMP FORMAT.

Please enter the image you want to compress : square_bmp.bmp
Processing BMP Header......
BM
Number of bits in the original BMP image is 8388608 bits.
Number of bits per pixel is : 32 bits.
Total time taken for processing BMP Image : 0.043000 seconds

HUFFMAN CODES :

PIXEL VALUES ==>   CODE

     41     ==>  0101101011111
     42     ==>  0101101011110
     43     ==>  010110101110
     44     ==>  01011010110
     45     ==>  00010101
     46     ==>  0011010
     47     ==>  011011
     48     ==>  010000
     49     ==>  0010011
     50     ==>  000010110
     51     ==>  000000111
     52     ==>  000001000
     53     ==>  01111000
     54     ==>  01110010
     55     ==>  01110110
     56     ==>  000010111
     57     ==>  01100010
```

```
58      ==>  01111011
59      ==>  000000000
60      ==>  000101000
61      ==>  000011010
62      ==>  01101001
63      ==>  01001010
64      ==>  01001111
65      ==>  01001101
66      ==>  01001100
67      ==>  01011001
68      ==>  01101010
69      ==>  000000010
70      ==>  000111011
71      ==>  001000110
72      ==>  000011110
73      ==>  01101011
74      ==>  01111101
75      ==>  000000001
76      ==>  000010100
77      ==>  000000110
78      ==>  01100101
79      ==>  01111110
80      ==>  000001001
81      ==>  01110100
82      ==>  000010000
83      ==>  000010101
84      ==>  000011000
85      ==>  000101101
86      ==>  000111010
87      ==>  001100000
88      ==>  010001010
89      ==>  011000110
90      ==>  011000011
91      ==>  011001111
92      ==>  011100110
93      ==>  010111110
94      ==>  011100001
95      ==>  011000010
96      ==>  010111100
97      ==>  011000001
98      ==>  011110010
```

```
 99      ==>  010110110
100      ==>  011110011
101      ==>  011001000
102      ==>  011101111
103      ==>  011101110
104      ==>  0000001011
105      ==>  0000010101
106      ==>  011100000
107      ==>  011101011
108      ==>  0000000110
109      ==>  0000000111
110      ==>  011111110
111      ==>  011111111
112      ==>  011001110
113      ==>  011001101
114      ==>  011111001
115      ==>  0000110010
116      ==>  0001001001
117      ==>  0001011101
118      ==>  0001110010
119      ==>  0010101010
120      ==>  0010101011
121      ==>  0101101010
122      ==>  0001110011
123      ==>  0010001111
124      ==>  0010001110
125      ==>  0001100001
126      ==>  0001100000
127      ==>  0001000010
128      ==>  0001001000
129      ==>  0001011100
130      ==>  0001000011
131      ==>  0001000101
132      ==>  0000100110
133      ==>  0001000100
134      ==>  0000010100
135      ==>  011111000
136      ==>  0000001000
137      ==>  0000110011
138      ==>  011010001
139      ==>  011010000
```

```
140       ==>  0000100100
141       ==>  011100011
142       ==>  0000001001
143       ==>  011100010
144       ==>  0000100101
145       ==>  0000100111
146       ==>  011101010
147       ==>  011001001
148       ==>  011100111
149       ==>  010110111
150       ==>  011001100
151       ==>  010001011
152       ==>  010010110
153       ==>  010101110
154       ==>  001101110
155       ==>  001011001
156       ==>  001011100
157       ==>  001010011
158       ==>  001011000
159       ==>  001100011
160       ==>  001001001
161       ==>  001101111
162       ==>  001100101
163       ==>  001100110
164       ==>  001010111
165       ==>  001011101
166       ==>  001101101
167       ==>  001110011
168       ==>  001110111
169       ==>  010010111
170       ==>  001110110
171       ==>  010101001
172       ==>  010011100
173       ==>  010110001
174       ==>  010010010
175       ==>  010101111
176       ==>  010101100
177       ==>  010101010
178       ==>  010011101
179       ==>  010110000
180       ==>  0000001010
```

```
181        ==>  010010011
182        ==>  010010001
183        ==>  010101011
184        ==>  010111111
185        ==>  010010000
186        ==>  010001000
187        ==>  001111011
188        ==>  001111010
189        ==>  001100111
190        ==>  001100001
191        ==>  001110010
192        ==>  001110001
193        ==>  001011011
194        ==>  001100100
195        ==>  001100010
196        ==>  001010001
197        ==>  000110111
198        ==>  001001010
199        ==>  001000100
200        ==>  000111101
201        ==>  000101100
202        ==>  000110001
203        ==>  000110100
204        ==>  000011011
205        ==>  000100101
206        ==>  000011100
207        ==>  000100111
208        ==>  001000010
209        ==>  001101100
210        ==>  010001001
211        ==>  001110000
212        ==>  010110100
213        ==>  011000000
214        ==>  010101000
215        ==>  011000111
216        ==>  010101101
217        ==>  010111101
218        ==>  01111010
219        ==>  001010010
220        ==>  001010000
221        ==>  001000001
```

```
    222      ==>  000110110
    223      ==>  001000000
    224      ==>  000110011
    225      ==>  000110010
    226      ==>  001000101
    227      ==>  001010110
    228      ==>  000111100
    229      ==>  001001000
    230      ==>  000001011
    231      ==>  00101111
    232      ==>  00011111
    233      ==>  00111010
    234      ==>  001010100
    235      ==>  001000011
    236      ==>  000101111
    237      ==>  000011111
    238      ==>  000100000
    239      ==>  000010001
    240      ==>  000100011
    241      ==>  000111000
    242      ==>  001011010
    243      ==>  001001011
    244      ==>  000110101
    245      ==>  000101001
    246      ==>  000100110
    247      ==>  000011101
    248      ==>  00111100
    249      ==>  0101110
    250      ==>  0101001
    251      ==>  0101000
    252      ==>  0100011
    253      ==>  0000011
    254      ==>  0011111
    255      ==>  1
Average number of bits per pixel : 5.604790
Number of bits in the encoded text is 3145728.
Percentage of data saved : 62
ENTER 0 TO RUN THE TOOL AGAIN OR 1 TO EXIT - 0
```

```
********************************************
       WELCOME TO THE IMAGE COMPRESSOR
********************************************
THIS TOOL USES HUFFMAN CODING TO COMPRESS THE IMAGE GIVEN IN .BMP FORMAT.

Please enter the image you want to compress : square_img_2.bmp
Processing BMP Header......
BM
Number of bits in the original BMP image is 8388608 bits.
Number of bits per pixel is : 32 bits.
Total time taken for processing BMP Image : 0.049000 seconds

HUFFMAN CODES :

PIXEL VALUES ==>   CODE


    0      ==>  0
    1      ==>  11001
    2      ==>  11101111
    3      ==>  11101110
    4      ==>  111011011
    6      ==>  111010011110
    7      ==>  1110100100010
    8      ==>  11101101011
    9      ==>  1110100101001
     10      ==>  111010011100
     11      ==>  1110100101010
     12      ==>  111010110000
     13      ==>  11011
     15      ==>  1110100100011
     26      ==>  1110110100100
     27      ==>  1110101011010
     29      ==>  11101100110
     30      ==>  111010010010
     41      ==>  111010010011
     67      ==>  11101100111
     68      ==>  1110100100000
     73      ==>  111010110111
     77      ==>  1110101110010
     78      ==>  1110101110000
     79      ==>  1110101110110
```

```
80      ==>  111010011011
81      ==>  1110101011111
84      ==>  1110101010010
87      ==>  1110101010011
89      ==>  1110101010000
91      ==>  1110101010001
104     ==>  1110101010111
106     ==>  11101100100
114     ==>  1110100101011
115     ==>  1110100101000
128     ==>  111010011111
130     ==>  1110100101100
133     ==>  1110100101101
134     ==>  11101100101
135     ==>  111010011101
138     ==>  11101101010
140     ==>  1110100100001
142     ==>  111011010011
165     ==>  11010
166     ==>  1110110100101
170     ==>  111010111100
173     ==>  1110101110011
175     ==>  111010111101
176     ==>  1110101110001
182     ==>  111010110010
191     ==>  1110101110111
194     ==>  1110101110100
195     ==>  1110101110101
196     ==>  111010110011
197     ==>  1110101011011
200     ==>  1110101011000
201     ==>  1110101011001
205     ==>  1110101011110
206     ==>  11101101000
211     ==>  1110101011100
212     ==>  1110101011101
218     ==>  111010110001
221     ==>  111010110110
228     ==>  11101011111
229     ==>  111010110100
230     ==>  1110101010110
```

```
    232      ==>  111010110101
    234      ==>  1110101010100
    235      ==>  1110101010101
    238      ==>  111010011010
    241      ==>  11100
    245      ==>  111010011000
    246      ==>  111010011001
    248      ==>  1110100101110
    249      ==>  1110100101111
    250      ==>  111011000
    251      ==>  11000
    252      ==>  111010100
    253      ==>  11101000
    254      ==>  1111
    255      ==>  10
Average number of bits per pixel : 1.075756
Number of bits in the encoded text is 3145728.
Percentage of data saved : 62
ENTER 0 TO RUN THE TOOL AGAIN OR 1 TO EXIT - 0

***********************************************
        WELCOME TO THE IMAGE COMPRESSOR
***********************************************
THIS TOOL USES HUFFMAN CODING TO COMPRESS THE IMAGE GIVEN IN .BMP FORMAT.

Please enter the image you want to compress : football.bmp
Processing BMP Header......
BM
Number of bits in the original BMP image is 8388608 bits.
Number of bits per pixel is : 32 bits.
Total time taken for processing BMP Image : 0.057000 seconds

HUFFMAN CODES :

PIXEL VALUES ==>   CODE

    0      ==>  000001001
    1      ==>  10111111111
    2      ==>  10100111101
    3      ==>  10111111110
    4      ==>  00001011011
```

```
5      ==>  00001011010
6      ==>  10100111100
7      ==>  00000001110
8      ==>  1110100111
9      ==>  1110000100
 10      ==>  00000001111
 11      ==>  1110100110
 12      ==>  1101101010
 13      ==>  1100010101
 14      ==>  1001111111
 15      ==>  1001010111
 16      ==>  0010010111
 17      ==>  0010010110
 18      ==>  0010111000
 19      ==>  111110000
 20      ==>  0000000101
 21      ==>  110011000
 22      ==>  110101000
 23      ==>  110110100
 24      ==>  101010110
 25      ==>  101001010
 26      ==>  101000111
 27      ==>  100011101
 28      ==>  100010110
 29      ==>  100000110
 30      ==>  001001111
 31      ==>  000011100
 32      ==>  000110100
 33      ==>  000010001
 34      ==>  000010111
 35      ==>  11111100
 36      ==>  11011101
 37      ==>  11110111
 38      ==>  11100100
 39      ==>  11011011
 40      ==>  11101110
 41      ==>  11010011
 42      ==>  11010111
 43      ==>  11001011
 44      ==>  11001101
 45      ==>  11100000
```

```
46       ==>  11011100
47       ==>  11011111
48       ==>  11111110
49       ==>  11111111
50       ==>  11001110
51       ==>  11110011
52       ==>  11001111
53       ==>  10111010
54       ==>  11010110
55       ==>  11000011
56       ==>  11000100
57       ==>  10110101
58       ==>  10100100
59       ==>  10110010
60       ==>  10101010
61       ==>  10100001
62       ==>  10010001
63       ==>  10011101
64       ==>  00111101
65       ==>  10000010
66       ==>  00110011
67       ==>  00100110
68       ==>  00100100
69       ==>  00001001
70       ==>  00001010
71       ==>  1111101
72       ==>  1110110
73       ==>  1101100
74       ==>  1100000
75       ==>  1100100
76       ==>  1011011
77       ==>  1001101
78       ==>  1001011
79       ==>  1001001
80       ==>  1000000
81       ==>  1000010
82       ==>  0011000
83       ==>  0010101
84       ==>  0001111
85       ==>  0001100
86       ==>  0001011
```

```
 87      ==>  0000110
 88      ==>  0001001
 89      ==>  0001000
 90      ==>  0000011
 91      ==>  0001010
 92      ==>  0001110
 93      ==>  0010000
 94      ==>  0010110
 95      ==>  0010100
 96      ==>  0011010
 97      ==>  0011100
 98      ==>  1000100
 99      ==>  1000110
100      ==>  1001100
101      ==>  1010111
102      ==>  1011110
103      ==>  1101000
104      ==>  1110001
105      ==>  1111000
106      ==>  00000011
107      ==>  00000010
108      ==>  00001111
109      ==>  00111010
110      ==>  00100010
111      ==>  00110010
112      ==>  00111100
113      ==>  10001111
114      ==>  10101000
115      ==>  10111000
116      ==>  11000111
117      ==>  11010010
118      ==>  11101010
119      ==>  11110110
120      ==>  000110110
121      ==>  000010000
122      ==>  001000110
123      ==>  001011110
124      ==>  001111111
125      ==>  001111101
126      ==>  100101000
127      ==>  001110110
```

```
128        ==>  001101100
129        ==>  001111100
130        ==>  001110111
131        ==>  100011100
132        ==>  001101111
133        ==>  100111001
134        ==>  100010111
135        ==>  100001111
136        ==>  100100000
137        ==>  100010101
138        ==>  100111000
139        ==>  101000100
140        ==>  100111110
141        ==>  100101001
142        ==>  101001011
143        ==>  101111110
144        ==>  101010111
145        ==>  111001100
146        ==>  111001111
147        ==>  111000011
148        ==>  0000000010
149        ==>  110101001
150        ==>  111001110
151        ==>  0000101100
152        ==>  0001101011
153        ==>  0000010101
154        ==>  0000010001
155        ==>  111110001
156        ==>  0000000011
157        ==>  0001101010
158        ==>  0000111010
159        ==>  0001101110
160        ==>  0010111110
161        ==>  0010111001
162        ==>  1001000010
163        ==>  0011111101
164        ==>  0011011100
165        ==>  1001010100
166        ==>  1001010101
167        ==>  1010011111
168        ==>  1001111110
```

```
169       ==>  1011101111
170       ==>  1011111110
171       ==>  1001000011
172       ==>  1011010000
173       ==>  1011000010
174       ==>  1011010001
175       ==>  1010001101
176       ==>  1010110111
177       ==>  1011000101
178       ==>  1010110110
179       ==>  1011000011
180       ==>  1011101100
181       ==>  1011010011
182       ==>  1101101011
183       ==>  1100010100
184       ==>  1011000110
185       ==>  1110000101
186       ==>  1011000111
187       ==>  1100110011
188       ==>  1100010110
189       ==>  1011101101
190       ==>  1011000100
191       ==>  1100001000
192       ==>  1011101110
193       ==>  1100001001
194       ==>  1100110010
195       ==>  1100010111
196       ==>  1000001110
197       ==>  1010001100
198       ==>  1011010010
199       ==>  1000001111
200       ==>  0011011101
201       ==>  0010011100
202       ==>  1001010110
203       ==>  0010011101
204       ==>  0011111100
205       ==>  0001101111
206       ==>  0010111111
207       ==>  0000111011
208       ==>  0000010111
209       ==>  0000010100
```

```
210        ==>  0000010000
211        ==>  111001010
212        ==>  111111011
213        ==>  0000000100
214        ==>  111111010
215        ==>  111011110
216        ==>  111010010
217        ==>  111001011
218        ==>  111001101
219        ==>  111010110
220        ==>  111110010
221        ==>  111011111
222        ==>  0000000110
223        ==>  111110011
224        ==>  111010111
225        ==>  0000010110
226        ==>  110000101
227        ==>  101011010
228        ==>  101001110
229        ==>  101100000
230        ==>  101000101
231        ==>  100001101
232        ==>  100010100
233        ==>  100001110
234        ==>  100001100
235        ==>  001011101
236        ==>  001001010
237        ==>  001101101
238        ==>  001000111
239        ==>  000000000
240        ==>  11110010
241        ==>  11101000
242        ==>  11010101
243        ==>  11011110
244        ==>  11001010
245        ==>  10111110
246        ==>  10100110
247        ==>  10101100
248        ==>  10111001
249        ==>  11000110
250        ==>  10110011
```

```
    251      ==>  10101001
    252      ==>  10011110
    253      ==>  10100000
    254      ==>  1111010
    255      ==>  01
Average number of bits per pixel : 6.418423
Number of bits in the encoded text is 2621440.
Percentage of data saved : 68
ENTER 0 TO RUN THE TOOL AGAIN OR 1 TO EXIT - 0

**********************************************
        WELCOME TO THE IMAGE COMPRESSOR
**********************************************
THIS TOOL USES HUFFMAN CODING TO COMPRESS THE IMAGE GIVEN IN .BMP FORMAT.

Please enter the image you want to compress : img_2.bmp
Processing BMP Header......
BM
Number of bits in the original BMP image is 6291456 bits.
Number of bits per pixel is : 24 bits.
Total time taken for processing BMP Image : 0.047000 seconds

HUFFMAN CODES :

PIXEL VALUES ==>   CODE

    0      ==>  1
    1      ==>  01
    2      ==>  001
    3      ==>  00001
    4      ==>  000001
    5      ==>  000110
    6      ==>  0000000
    7      ==>  0001001
    8      ==>  0001011
    9      ==>  00000011
     10      ==>  00010001
     11      ==>  00011101
     12      ==>  000000101
     13      ==>  000101001
     14      ==>  000101011
```

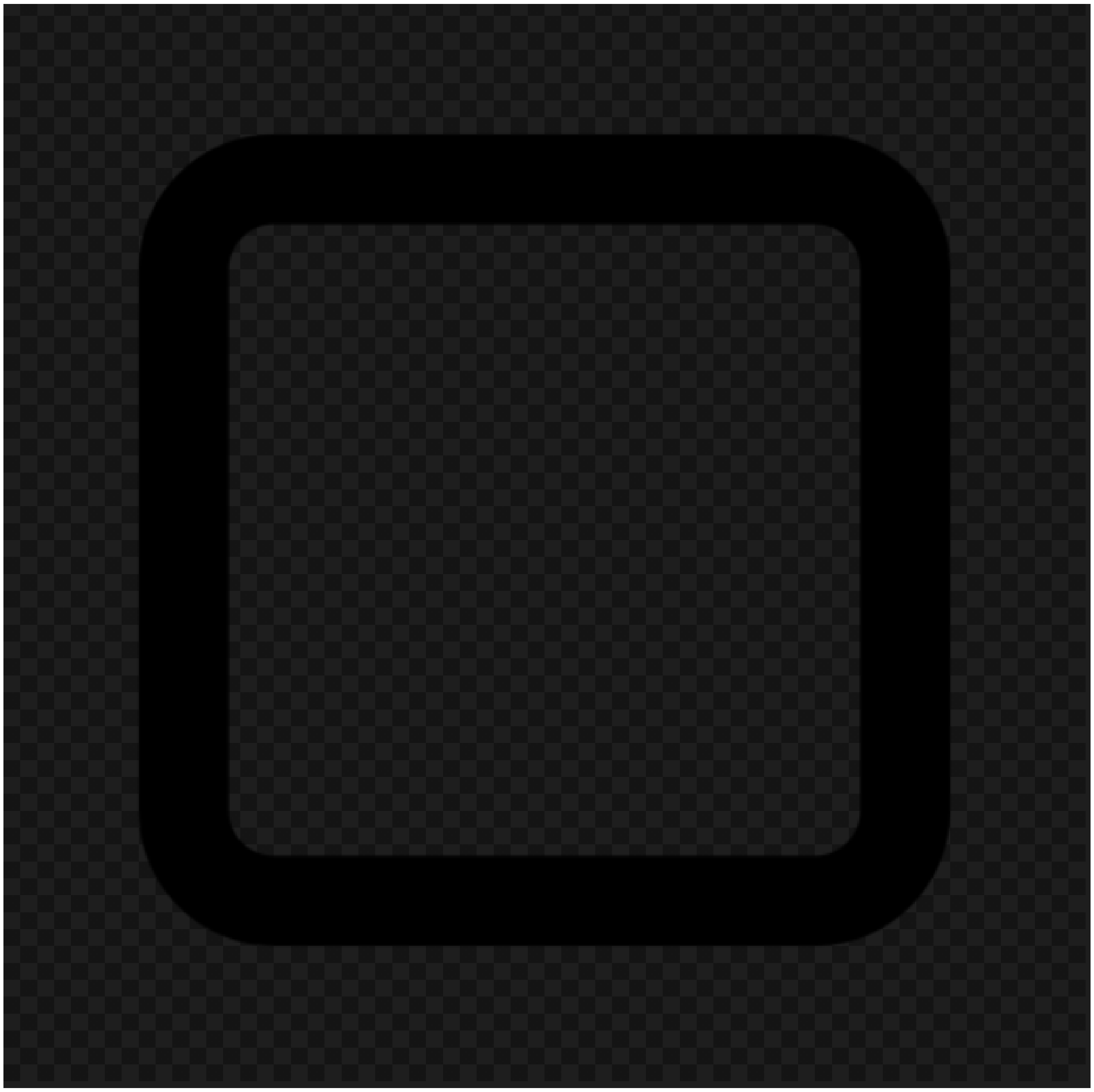
```
15        ==>  000111101
16        ==>  000111111
17        ==>  0001000001
18        ==>  0001000010
19        ==>  0001010101
20        ==>  0001111001
21        ==>  0001110010
22        ==>  00000010001
23        ==>  00000010010
24        ==>  00010000111
25        ==>  00010100010
26        ==>  00010000110
27        ==>  00010100001
28        ==>  00011100000
29        ==>  00011100011
30        ==>  00011100010
31        ==>  00011100111
32        ==>  00011100110
33        ==>  00011111000
34        ==>  000000100000
35        ==>  00011110001
36        ==>  00011111011
37        ==>  000100000000
38        ==>  000000100111
39        ==>  00011111010
40        ==>  000101000111
41        ==>  000101000110
42        ==>  000101010000
43        ==>  000101000000
44        ==>  000101010010
45        ==>  000111000010
46        ==>  0000001000011
47        ==>  000111110010
48        ==>  000111110011
49        ==>  0001000000010
50        ==>  0001000000100
51        ==>  0001010100111
52        ==>  0001010100010
53        ==>  0001010000011
54        ==>  0001000000011
55        ==>  0001010100011
```

```
56        ==>  0001110000111
57        ==>  0001000000111
58        ==>  0001000000110
59        ==>  0001111000000
60        ==>  00000010011000
61        ==>  0001111000010
62        ==>  00000010000100
63        ==>  00010000001010
64        ==>  00000010011010
65        ==>  00000010011001
66        ==>  00011100001101
67        ==>  00010100000101
68        ==>  00011110000111
69        ==>  000100000010111
70        ==>  000101010011001
71        ==>  000101000001001
72        ==>  000000100110110
73        ==>  000111000011000
74        ==>  000111100001100
75        ==>  00010101001101
76        ==>  000000100110111
77        ==>  000111100000100
78        ==>  0001010000010001
79        ==>  0000001000010110
80        ==>  0001111000001100
81        ==>  0000001000010111
82        ==>  0000001000010100
83        ==>  00011110000010100
84        ==>  0000001000010101
85        ==>  0001111000001111
86        ==>  0001010000010000
87        ==>  0001111000001101
88        ==>  0001111000001011
89        ==>  0001111000001110
90        ==>  0001000000101100
91        ==>  00011110000010101
92        ==>  00011100001100110
93        ==>  00011100001100111
94        ==>  0001111000011011001
95        ==>  0001010100110001
96        ==>  00010000001011010
```

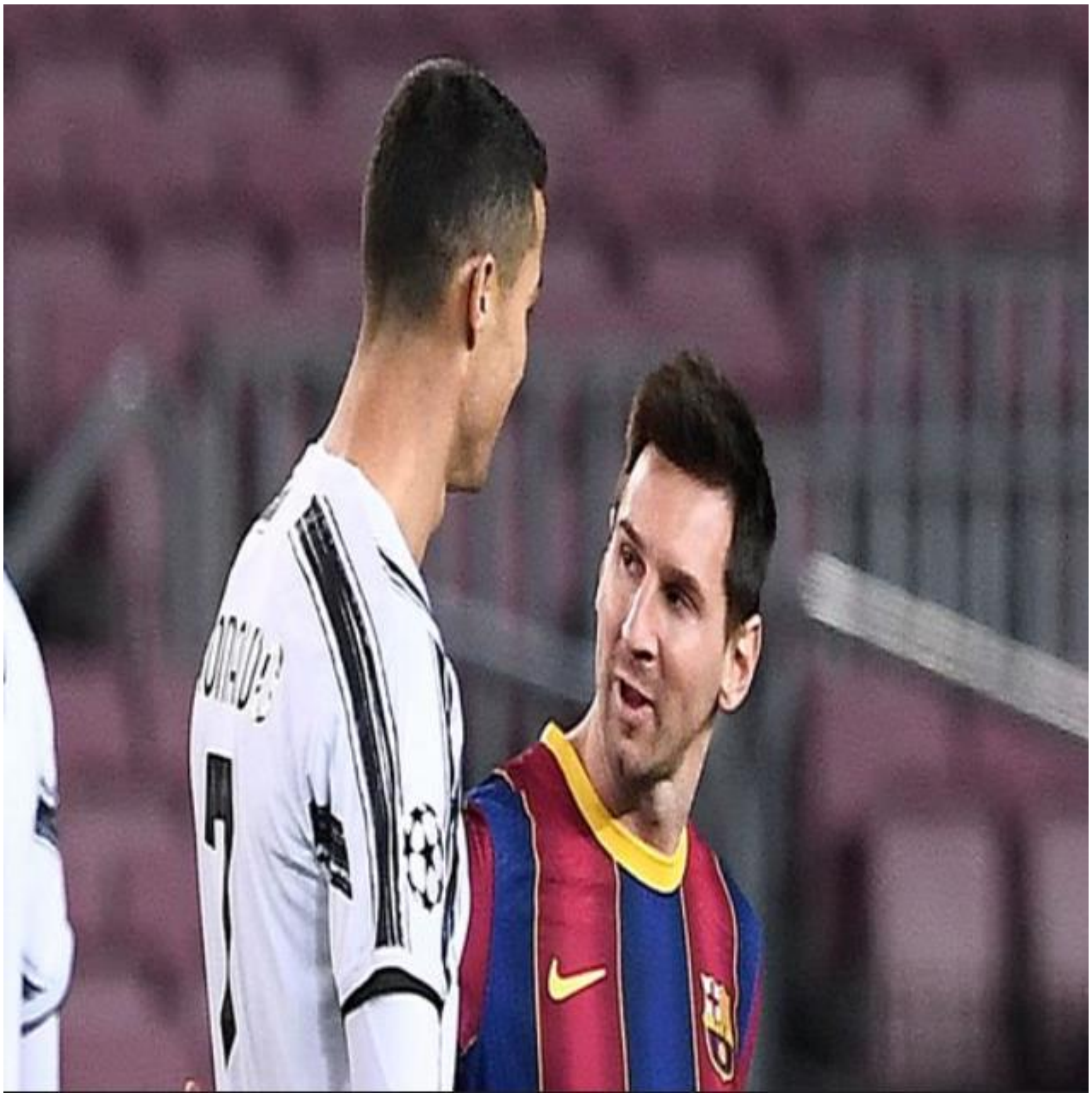
```
     97      ==>  0001000000101101110
     98      ==>  0001010100110000
     99      ==>  0001111000011010
    100      ==>  00011100001100100
    101      ==>  00011100001100101
    103      ==>  000111100001101101
    104      ==>  0001111000011011000
    108      ==>  00011110000110111
    110      ==>  000100000010110110
    116      ==>  0001000000101101111
Average number of bits per pixel : 2.801697
Number of bits in the encoded text is 4718592.
Percentage of data saved : 43
ENTER 0 TO RUN THE TOOL AGAIN OR 1 TO EXIT - 1
AVERAGE PERCENTAGE SAVED OVER ALL IMAGES IS : 58
```

**square_bmp.bmp**

**square_img_2.bmp**

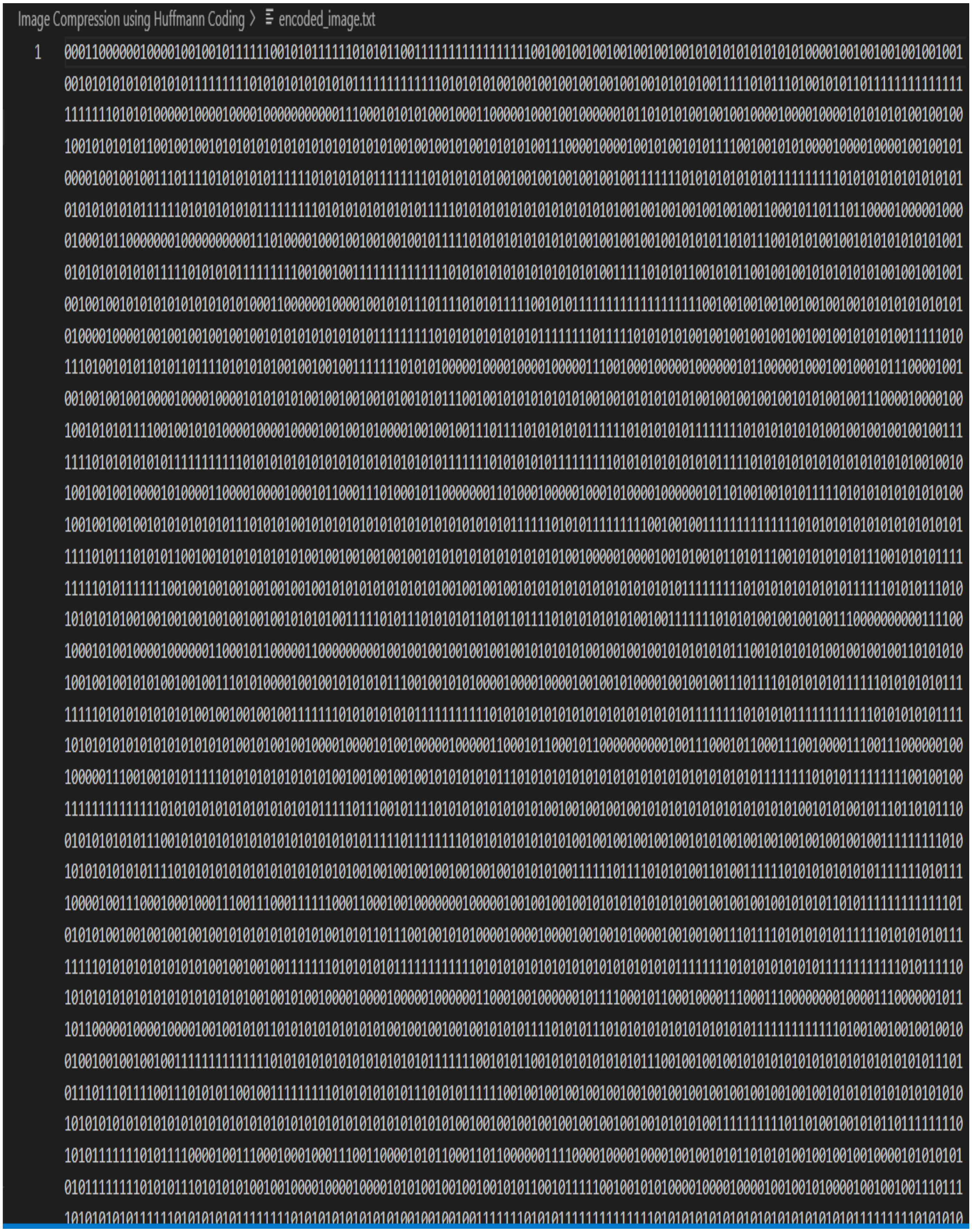


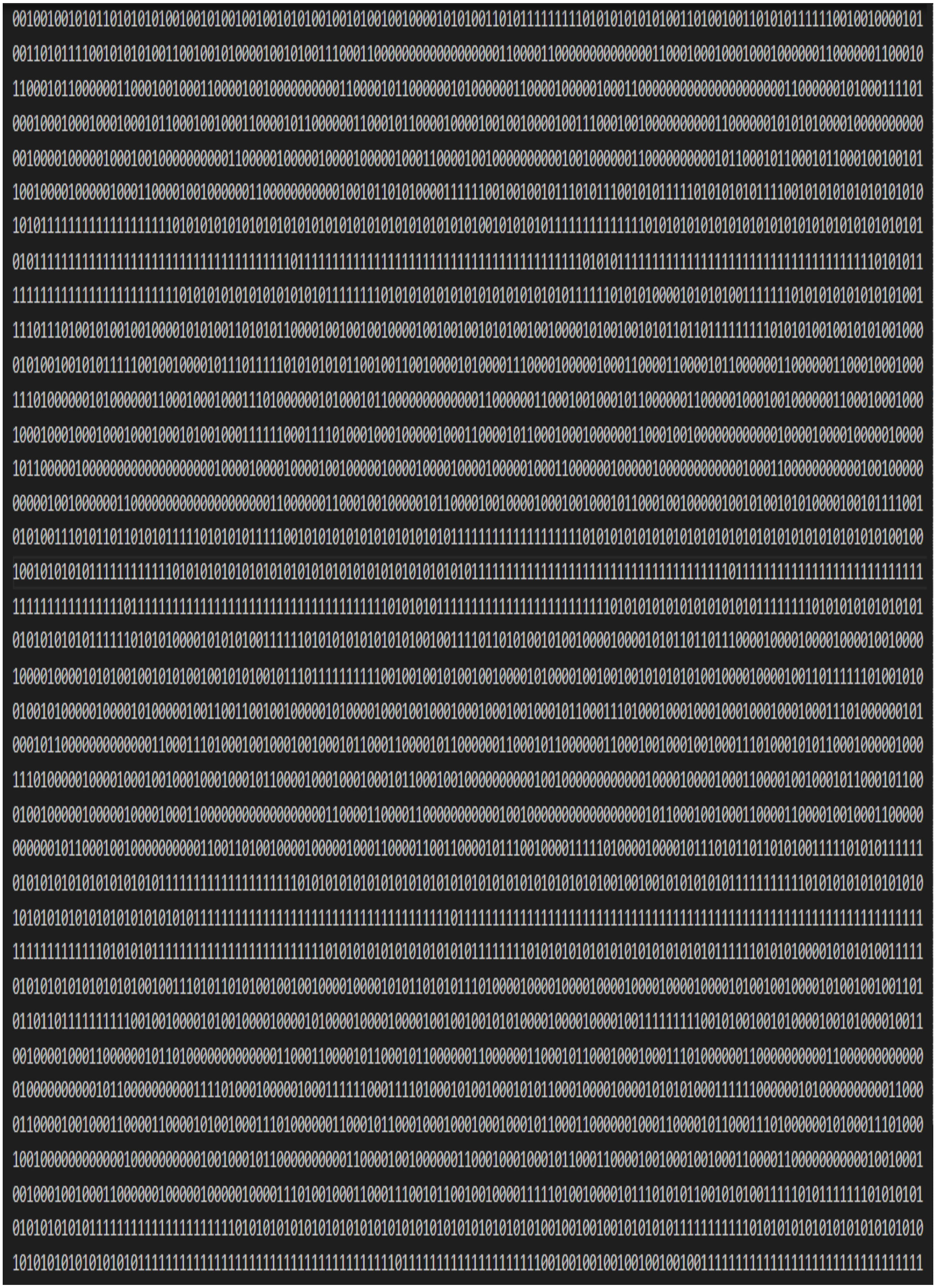
**football.bmp**



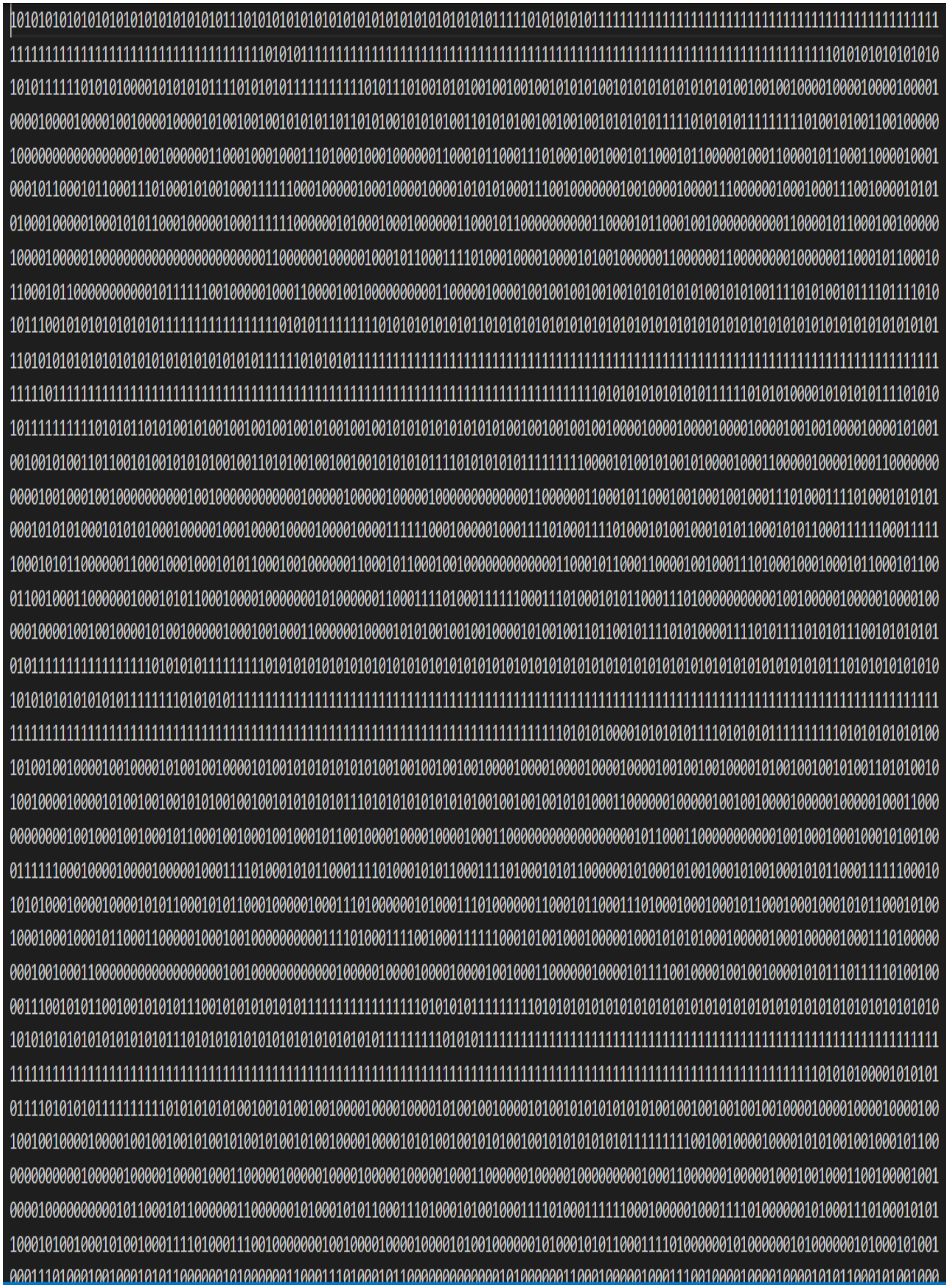
**img_2.bmp**

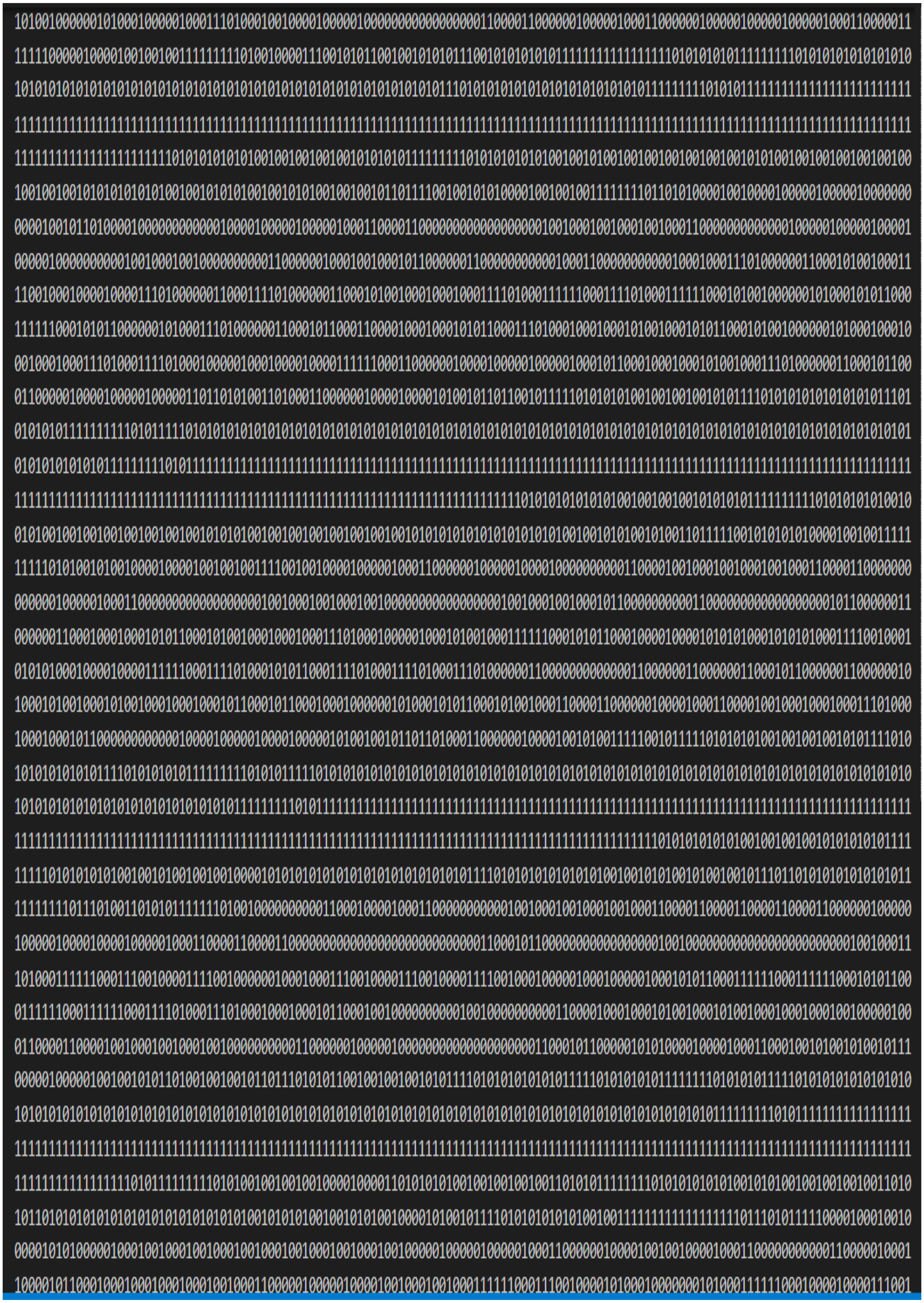
# encoded_image.txt



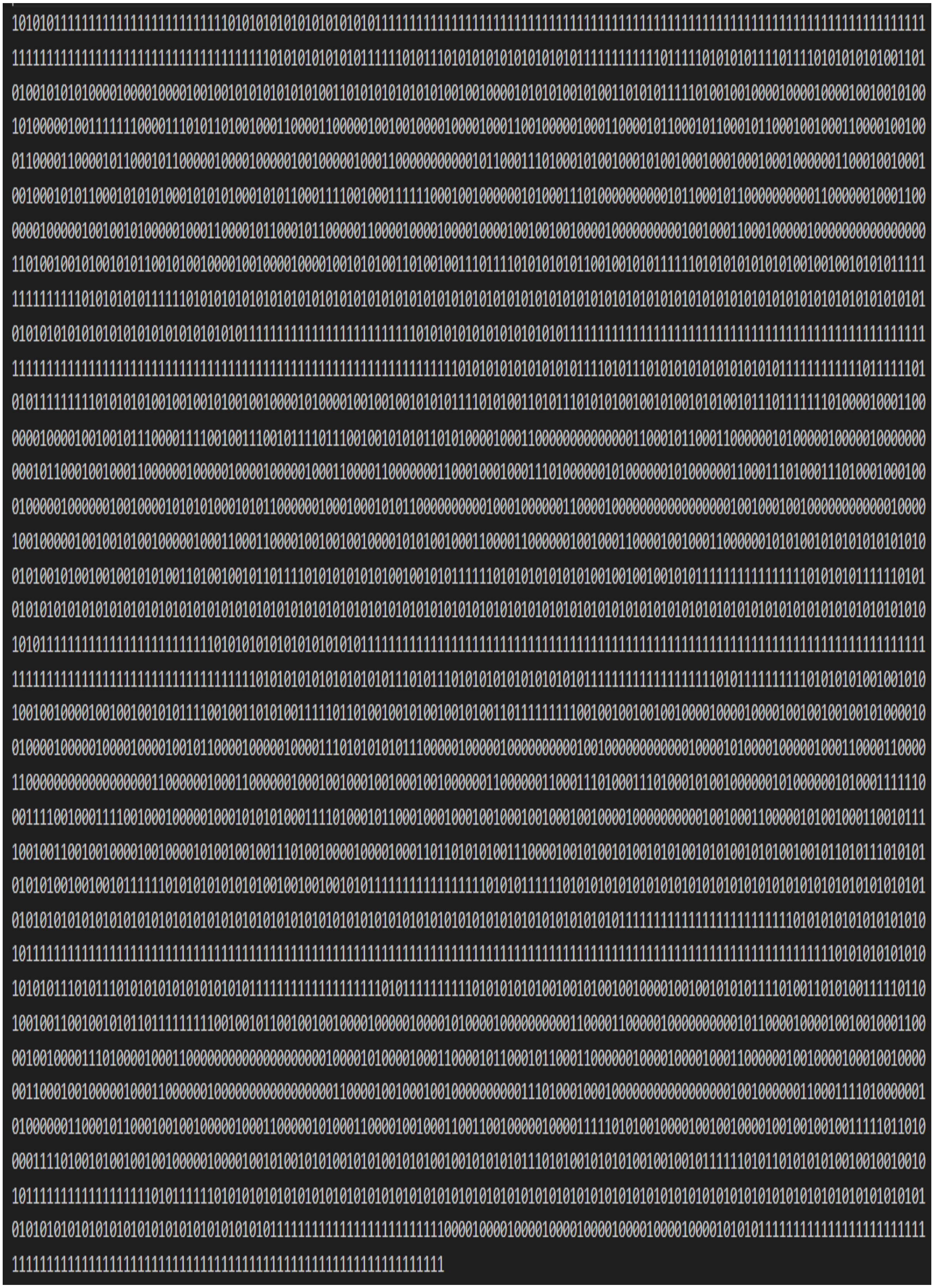


```
1111111111111111111110101010111111111111111111111111110101010101010101111111110101010101010101010101010101011111101010100001010101
0111101010101111111111111111101110101010111101110101010101010010010101010101010010010101001101010101010110110101011111011111001001001
0010010010010010010010010010010010010000100101110101010010100000100000100001111000001000000000011001010000100000000000001010001010101
1000111101000111111000000101000101100010001000101100000000000000110000001100000101001000110000000110001010010000001010000010000000000
0000110001010110001111110001111010001010110001010010001010101000100000100000011000000000000001100000010100010110000100011000000000000
1100000010000100000000001100000010011001000100100010110000001100010110010001001000100100000000000010000010001100001100000100000001000
0000000000110000001100000011000100010001110100011110100000011000000000001001100001000000000010110010000100000100011000000000000000000
0000100001111101010010000100000000000010010111111111111101010111001010101010101010101011111111111111111111111110101011111101010101010
1010101010101010101010101010101010101010101010101010101011101010101010101010101010101010101010101011100100101010101111111111111111111
1111111101011111111111111111111111111111111111111111111111101010101010101111111111111111111111111111111111111111111111111111111101
0101010101010111111111010101010101010101010101010101010111111010101000010101010111101010101111111111111111011010101010101111011101010100
1010100100101010101010101001010100100110101010101101101010111111111001001001001001001001001001001001001001000010000110111010101101
0010001100001100000111000001000100100010110001001000110000110000100100000011000101001000111101000100010001110100011101000100010001
1010001000100010110000001100000010100010001000101100010110000001100011101000101011000111111000100010001110100010100100010101100011
1010001111110001000001000100000010001000010000111111000000101000111010001011000110000110000100100011000010110001001000000100011000010
1100010010000010000100100000010001011000101100011000000110010010001100001011000100010000001100010110001001000100100010010001011000010
0100000000000100100000011000101011000000110000000000101010000100000100000000101000010000010000100001010000111100101000010000100001 0001
1000000101011110011111111010101110010101010101010101010101111111111111111111111111111010101011110101010101010101010101010101010101010101
0101010101010101010101010111010101010101010101010101010101010101010101111001010101010111111111111111111111111111110111111111111111111111
1111111111111111111111111111111010101010101111111111111111111111111111111111111111111111111111111111111111111111111101010101010101010
1010101010111111010101000010101010111101010101111111111111011110110101001010110110111010101010101001001010101010101010101010101011010
1010101101101010101010111101110010010010010010010100100100100100100001000010000111110101110100001000001000001001000010000000000100
0100011000000100011000000100010010001011000100010001110100010110001011000100010001110100010001000000110000001100000000000101001000010
1001000101001000000101000000110001110100010100100010000100001111110001111110001111110001000001000100000100010000100001000010000100001000
0100001111110001111010001111010000001010001001000010000100010110001011000000110001011000000000001100001011000100100000100000100100000
0000001011000000000001100000011000001000110000100010001111110001111110000000101000101100010110000000000001011000000110001001000110000011
0000100100010110000000000000100101100100000100011010010010010000100100101111010010010010010000010000100111101001111111101010111001 0
1010101010101010101111111111111111111111111101010101111101010101010101010101010101010101010101010101010101010101010101010101011101010101 01
0101010101010101010101010111110010101010111111111111111111111111111111111111111111111111111111111111111111111111111111111111101010101
0111111111111111111111111111111111111111111111111111111111111111111101010101010101010101010111111010101000010101010101111010101
0111111111110101110101101001001010011010100111010101010101001001010101010101010100001010100100110101010110110101010101011101010101001
0010010010010101011111010101011111111110100100001001001000010001001000101100011000010010001001000000110001000100011101000111010001 00
0100010001000100100010001000111010000000000000000000100100001000111010001110100011101000100010001110100000010100011110100010000010001
0101100011111100011111100010000010001000000100011111100010101100000010100011110100010000100001111111000000110001001000100100000000001
1000010010000000110001011000100100011000010010000000000000101010000010000000000110000000000001100010001110100010101100010000100001 0101
0100011111100011101000000000001000000000010110001011000000110001001000000100111111010000100011000010010010010101001010101011011101 01
0010000101011110100101111101110101011100101010101010101010111111111111111111011111111110101010101011110101010101010101010101010101010
```





01010111111111010111010101010101010101011111111111010111010101010100110101010101010100100100100101011010100110100001000010010010101
01011101010011010101001001001001000010000010010000101001010011010101011110010000100011000001000010000010000010010010011010110100000
10000010000100100000111011110001001000010000010001100001100000010010111000010001100001000100011101000000101000111010001000101000110
00010010000000000010010000001100000011000000000000100000100000100011000000000001001000100100000000001001000000100001110001011000111 11
10001111110001111110001001000111010001000001000000101000110000110000100100011110100010000010001010101000111101000000110001100001001
00000011000000000011000011000010010001011000101100000000000100011000010010000000001100000000000110000000000000000011100001000000
00000010001100000000000010000100001010111111010100100001000001000000000101010110100101111111110101110010101010111101010101010110101
01010101111111010101010101010101010101010101110101010101010101010101010101010101010101010101010101010101010101010101010101010101010
10101010101010101010101011111111111111111111111111111111111111111111111111111010101111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111010101010111111101011101010101010101010111111111110111101010101010
01110101010101010010010010010101001101001001100100100100101010111101010011010101010010010010000100001001000010100100001011010101011
11010000010000100100001001000010010010000100000110010010000100001000001000010000010001100110000111010001011000110000100100011000010
01000100100000000001001000100100001000001000101100010001000111010001000100000011000100100011000000000000000001100000000000100100001
01100010010000010000010000010000000000000000010110001011000101100010110000001100010110101000100100010101100010000010001000010000100
01000111101000111100100011110100000001010001010110001110100010100100010101100010101100000010100010010000100011000000000001100001100 0
01100001001000000110001000100010010000010000000000000000110000110000110000001000000000000000001110010000010000010000010000010010 00
01000010010011111101101001000000100011001010101101001011101111011010111001010101111110101010101010101010101011111101010101010101010
10101010101101011101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010111111
11111111111111111111111111111111111111111111010101111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111110101010101011111101011101010101010101010111111111110111010101010011101010101010101001001001 01
01011010100100100100100101011110010011010100111010100101001000010000100100100001010010100110101011110100100000101101000010010010000
10000010010000001000110000010110000100000100000000000010100001001100100010010000010000010000010000010001100001100001011000100100001 0
00001000101100010001000100010000000110001001000000000000011000100100011000011000011000011000000000010110000010000010001100000000000
10110000000110001000100011101000111010001000001000101001000000000000000000100010001010110000001000100010100100011101000101011000111 11
10001110100010100100011110100010001000000110001000100000011000101100011000000100101000001000001000100100000011000100010001011000000 00
00010000001000001000001000001000001000110000001000001000001000001011010001100000101000010000100100101011110101001000010000010000100000
10101010011010010111011110110101011001010101111110101010101010101010101011111111111110101010101110111010101010101010101010101010 1
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101111111111111111111111111111111 1
11111101010111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111010
10101010111111101011101010101010101010101111111111101111101010101110101110101010100100110101001101001001001010101111101010011010 10
01001010100101001001000010100100100001011010011011111001000010000110100100100001001000010000001001000010000010011110100000000110010
00011001000110000010010010101000010000000000000010010001100001011000100010001000100000001100010110001000100000000001100000000000000 0
00001000001000100100000100000100011000010010000001100000010100010100100000010100000010100011110100010101100010110001001000111010001
00000100011100100000000101000111101000111101000000101000111010001010010001110100010110001001000101100011000000100001001011001000110 0
00000000101100010001000101100000100100001000010010000010000010000010000010000010001100001100000110010001100000001010100101010101011110
11010010000100000100000101010100110100101110111101010101011001010101111110101010101010100101010101011111111111111101010101011111101
01010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010



# CONCLUSION AND FUTURE WORK

The main aim of this project was to compress multiple .bmp images and calculate the average percentage of data saved for multiple images, which was achieved.

We would like to extend this project by compressing and decompressing the image using Huffman coding in MATLAB software.

# REFERENCES


- Y.S. Abu-Mostafa, R.J. McEliece, Maximal Codeword Lengths in Huffman Codes, Computers & Mathematics with Applications, Volume 39, Issue 11, 2000
- Fundamentals of Computer Algorithms by Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, 2nd Edition
- https://en.wikipedia.org/wiki/BMP_file_format
- https://www.geeksforgeeks.org/
- https://tutorialspoint.dev/
- https://youtu.be/0KmimFoalTI
- https://youtu.be/acEaM2W-Mfw
- https://stackoverflow.com/