LOSSLESS IMAGE COMPRESSION USING HUFFMAN CODING

November 5, 2023

MOHD YUMAN (2022MCB1270) ,
KESHAV BANSAL (2022MCB1268) ,
ADITYA CHANDANSHIVE (2022CSB1063)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Law Kumar

Summary: The project "Lossless Image Compression using Huffman Coding" aims to develop a system that can efficiently compress images while preserving all the original image data without any loss. This is achieved through the implementation of Huffman coding, a popular technique in data compression.This is a very efficient method because it compresses the image without losing any of its key data. Our project takes a sample image(BMP format), converts it into a custom format-COMP.

1. Introduction

This is a basic stepwise introduction of what our project does:

1.1. Huffman Coding for Image Compression

The provided code is a C++ implementation of Huffman coding, a data compression algorithm. It is tailored for use with BMP image files and follows a series of steps to compress the image data while preserving its integrity.

1.2. Reading the BMP Image

The program starts by reading a BMP image file, `"Test2.bmp"`. It extracts essential information about the image, including its dimensions and the RGB values of its individual pixels. This step is crucial for understanding the data to be compressed.

1.3. Frequency Analysis

The code proceeds to calculate the frequency of occurrence for each unique pixel value (in hexadecimal format) in the image. This frequency analysis is a fundamental component of Huffman coding, as it determines how the binary codes will be assigned to each pixel value.

1.4. Huffman Coding

The core of the program is the implementation of the Huffman coding algorithm. It constructs a Huffman tree based on the frequency distribution of pixel values. The tree's structure ensures that more frequent pixel values are assigned shorter binary codes, optimizing compression. These codes are stored in the `arr` array for reference.

1.5. Header Creation

To maintain data integrity and facilitate decompression, the code creates a header that includes information about the number of unique pixel values and their associated frequencies. This header is written to a compressed file named `"Test2.comp"`.

1.6. Bitstream Generation

The program generates a bitstream for the entire image, where each pixel's binary Huffman code is appended to the bitstream. This bitstream represents the compressed image data.

1.7. Writing to Compressed File

The generated bitstream is written to the compressed file. Special attention is given to ensuring that any remaining bits are properly flushed and saved in the file, guaranteeing data consistency.

1.8. Displaying Huffman Codes

For verification and reference, the program displays the Huffman codes assigned to each pixel value in the console output. This step allows users to understand the compression process and verify the accuracy of the generated codes.
This C++ code showcases how Huffman coding can be applied to image compression and provides a practical example of how data can be efficiently stored and transmitted while preserving its original content.

2. Equations

In this section, we present the key equations and mathematical concepts related to Huffman coding, as implemented in the provided code.

2.1. Huffman Tree Probability

The probability of a node in the Huffman tree is calculated as the frequency of occurrence of a pixel value divided by the total number of pixels in the image:

$$P(\text{pixel\_value}) = \frac{\text{Frequency of pixel\_value}}{\text{Total number of pixels}} \tag{1}$$

2.2. Average Code Length

The average code length for a Huffman tree is a measure of the efficiency of the compression and can be calculated as the sum of the probabilities of each pixel value multiplied by the length of its corresponding Huffman code:

$$\text{Average Code Length} = \sum_i P(\text{pixel\_value}_i) \cdot \text{Length of Huffman code for pixel\_value}_i \tag{2}$$

2.3. Huffman Coding Efficiency

The efficiency of Huffman coding can be measured using the entropy of the source image. Entropy is a measure of the amount of information in the image and is calculated as:

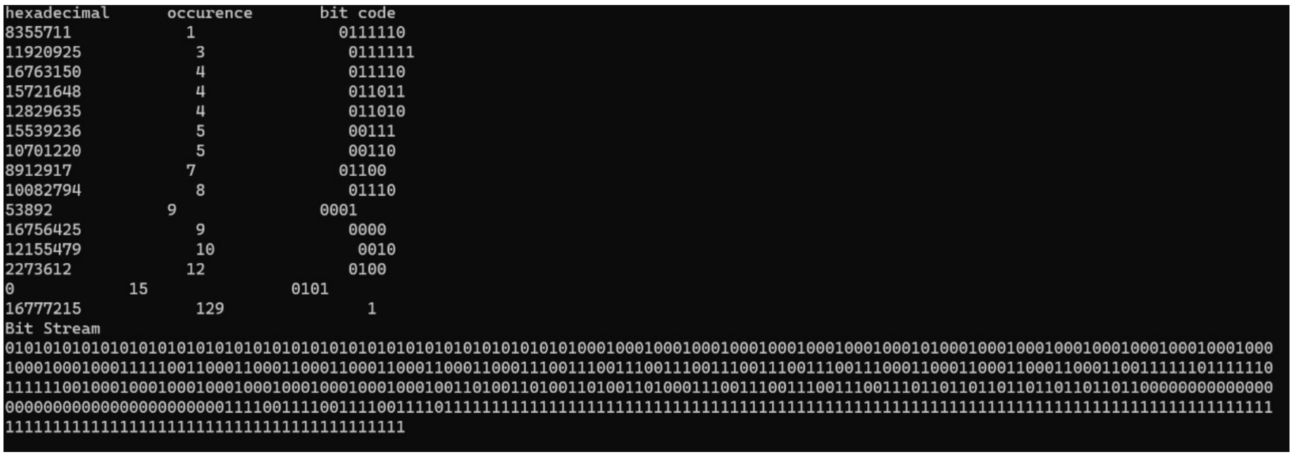
$$\text{Entropy} = -\sum_i P(\text{pixel\_value}_i) \cdot \log_2(P(\text{pixel\_value}_i)) \tag{3}$$

The efficiency of Huffman coding can be evaluated by comparing the average code length to the entropy. A more efficient Huffman tree results in a smaller average code length compared to the entropy.

3. Figures, Tables and Algorithms

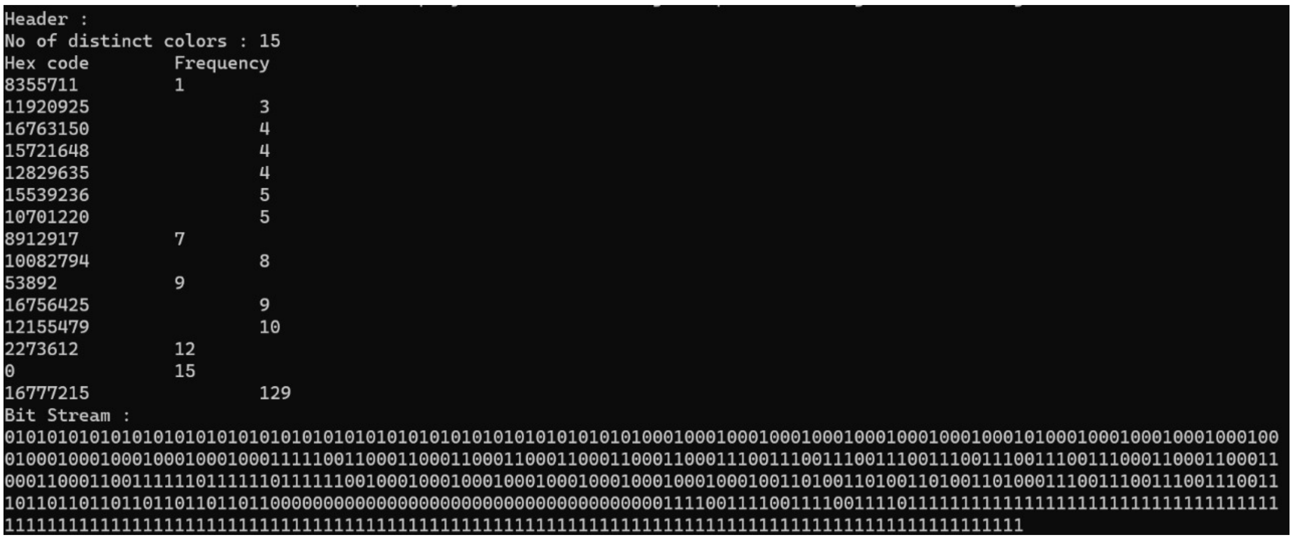
Figures, Tables and Algorithms have to contain a Caption that describes their content, and have to be properly referred in the text.

3.1. Figures



```
hexadecimal      occurence       bit code
8355711            1               0111110
11920925            3               0111111
16763150            4               011110
15721648            4               011011
12829635            4               011010
15539236            5               00111
10701220            5               00110
8912917            7               01100
10082794            8               01110
53892          9               0001
16756425            9               0000
12155479            10               0010
2273612            12              0100
0            15               0101
16777215            129               1
Bit Stream
0101010101010101010101010101010101010101010101010101010101010101000100010001000100010001000100010001000101000100010001000100010001000100010001000
1000100010001111100110001100011000110001100011000110001100011100111001110011100111001110011100111000110001100011000110001100111111101111110
1111110010001000100010001000100010001000100010001001101001101001101001101001101000111001110011100111001110110110110110110110110110000000000000000
0000000000000000000000001111001111001111001111011111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111
```

Figure 1: Input image file.



```
Header :
No of distinct colors : 15
Hex code        Frequency
8355711         1
11920925                3
16763150                4
15721648                4
12829635                4
15539236                5
10701220                5
8912917         7
10082794                8
53892           9
16756425                9
12155479                10
2273612         12
0               15
16777215                129
Bit Stream :
01010101010101010101010101010101010101010101010101010101010101010001000100010001000100010001000100010001010001000100010001000100
0100010001000100010001000111110011000110001100011000110001100011000111001110011100111001110011100111001110001100011000110001100011
0001100011001111110111111011111100100010001000100010001000100010001000100110100110100110100110100011100111001110011100111
10110110110110110110110110000000000000000000000000000000000000001111001111001111001111011111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
```

Figure 2: Output image file.

3.2. Tables

This is a simple table to show pixel values and their codes. The values given are only for example purposes and true values depend upon the image.

Table 1: Huffman Code Table

Pixel Value	Huffman Code
0x00	110
0x01	1010

.

3.3. Algorithms

This is a simple algorithm showing the flow in our code.

Algorithm 1 BMP Image Compression Algorithm

```
Read the BMP image and extract its width, height, and pixel data
Calculate the distinct colors and their frequencies
Create a Huffman tree and generate bit codes for colors
Write the compressed BMP file
ReadBMP
Open the BMP file filename in binary mode
Read BMP header information
Read pixel data
return bmp, which is an instance of the data_image struct
CalculateColorsAndFrequencies(pixelData)
Initialize a dictionary colorCount
for each pixel value in pixelData do
    Increment the corresponding color's count in colorCount
end for
Create colors list from colorCount
return colors
CreateHuffmanTree(colors)
Create a priority queue
for each color in colors do
    Add color to the priority queue with its frequency
end for
while priority queue has more than one element do
    Pop two colors with the lowest frequencies
    Merge them into a new color with the sum of frequencies
    Add the merged color back to the priority queue
end while
return root color of the Huffman tree
AssignHuffmanCodes(color, code)
if color has a pixel value then
    Set color.bit to code
else
    Recursively call AssignHuffmanCodes for left and right children
end if
WriteCompressedBMP(filename, width, height, colors, pixelData)
Open the output BMP file filename in binary write mode
Write the BMP header
Write the color palette
Encode pixel data with Huffman coding
Write the encoded bytes to the file
```

4. Conclusions

n our project on lossless image compression using Huffman coding, we've successfully achieved our goal of reducing image sizes without compromising quality. Huffman coding proved its efficiency, allowing us to store and transmit images more effectively.
This project has emphasized the importance of optimizing data storage and preserving details in applications like medical imaging. It's been a valuable learning experience, highlighting the significance of smart coding and data compression techniques.

As we conclude, we've proven that Huffman coding is a practical solution for lossless image compression. Moving forward, we'll continue to explore innovative approaches to meet the growing demand for efficient image compression in the digital age.

5. Bibliography and citations

Sources

1. Gupta, S. K. (2016). An Algorithm for Image Compression Using Huffman Coding Techniques. `http://www.ijarse.com/images/fullpdf/1467705501_8_Research_Paper.pdf`
2. Geeks for Geeks. . *Geeks for Geeks.* Retrieved from: `https://www.geeksforgeeks.org/`

Acknowledgements

I wish to express my gratitude to my teacher, Anil Kumar, for entrusting me with this project. His assignment provided the opportunity for me to learn and grow. I also extend my thanks to my mentor, Law Kumar, for their guidance during the project's development. Their expertise and support enriched my understanding of the subject matter. With their assistance and feedback, I successfully completed the project. Law Kumar sir provided valuable resources and maintained a conducive learning environment. My fellow students' collaboration and insights were equally vital to the project's success, which, in part, was due to Law Kumar's mentorship and Anil Kumar's assignment.