Image Compression Using LZW Coding

COMP 204 Programming Studio

Report prepared by:

Meryem Rana TÜRKER 042001021

Date: 20 March 2023

Abstract

Image compression is a process used to reduce data size and make it more suitable for storage or transfer. With this project, I aimed to make a program that can compress images with the Lempel-Ziv-Welch (LZW) algorithm method using Python software language. The LZW algorithm is a lossless data compression technique that is used especially in files such as images, and works with the principle of replacing character strings with a code in text files. Thanks to this technique, depending on the levels the program will take an image file that we specify as input and produce a compressed file as output. It will also include a decompression module that can read the compressed file and recreate the original image file. The same LZW algorithm will be used to re-open the original image and reconstruct the data from the compressed file. This project provides a practical application of the LZW algorithm in image compression technique. The main purpose is to demonstrate the effectiveness of this algorithm in reducing file size without significant loss of image quality.

Keywords: image compression, Lempel-Ziv-Welch algorithm, data compression, lossless compression, file storage, decompression, repetitive sequences, image quality, file size reduction.

1.Definition of Problem

1.1. Overview about compression

Data compression is the compression of data on a computer so that it takes up less space. Lossless compression, which I use in my program, is the process of compressing a file without losing quality using various data compression algorithms and making it workable again when extracted. Thanks to this method, users can store much more files in their memory. The most important rule of lossless compression is that the processed data can be made the same again during the extraction phase.

LZW(Lempel–Ziv–Welch) algorithm is a dictionary-based algorithm published by Terry Welch in 1984 and by Abraham Lempel and Jacob Ziv in 1978. It has a system based on creating a dictionary for repetitive characters and converts the data into small codes during the data compression process.

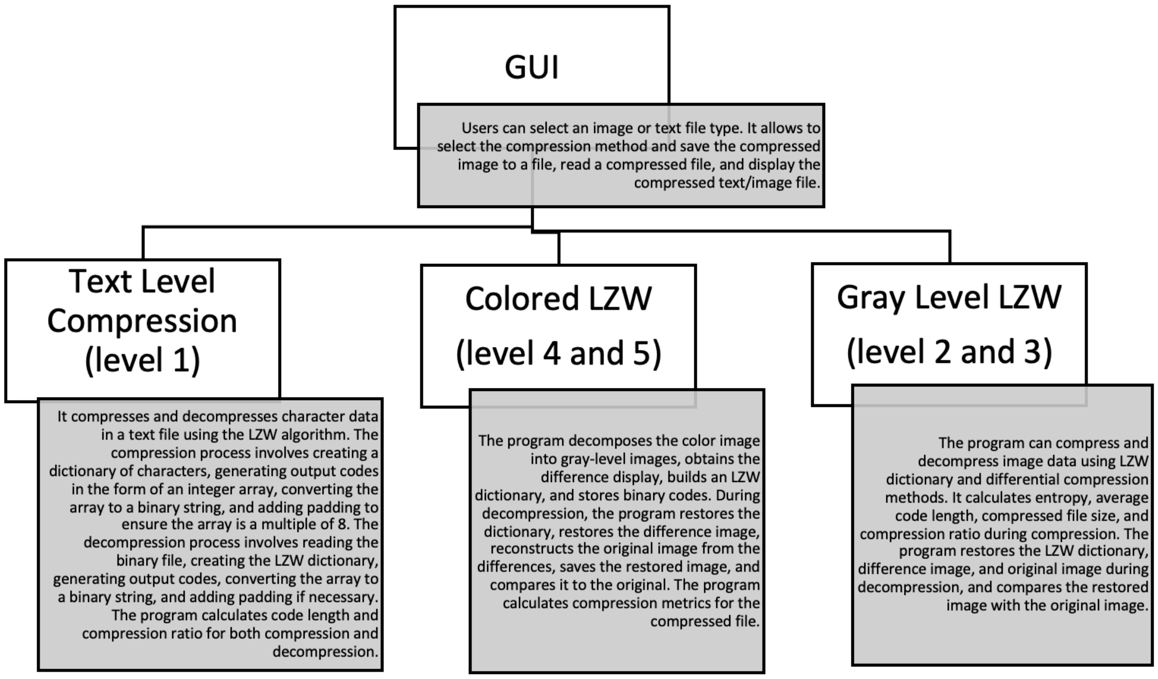
1.2. Description of Project

Compressing data becomes an important issue in our digital age, where everyone has large amounts of data. It is important for a computer engineer to understand and apply the algorithm of this method while developing programs that use a lot of data. In this project, the LZW algorithm is used to compress and decompress the data.

The memory disks in our computers we use today have the capacity to store large image data. However, if we want to store this data in its large size, it will be costly for us in terms of available memory usage and processing capability. Considering these reasons, it is necessary to use a program that can compress and decompress data with lossless compression method and save the new version. The aim of this project is to provide efficiency to reuse the same data by compressing the existing image or text file with the help of LZW algorithm and bringing the compressed file back to its original structure.

2. Solutions

I designed a program that can first compress and decompress a text file and then the recorded image file, using the python software language, depending on the instructions given respectively in the project content. I created a plan in my program by following the software development process that we saw in the lesson (Figure 1).


GUI
Text Level Compression (level 1)
Colored LZW (level 4 and 5)
Gray Level LZW (level 2 and 3)

Users can select an image or text file type. It allows to select the compression method and save the compressed image to a file, read a compressed file, and display the compressed text/image file.

It compresses and decompresses character data in a text file using the LZW algorithm. The compression process involves creating a dictionary of characters, generating output codes in the form of an integer array, converting the array to a binary string, and adding padding to ensure the array is a multiple of 8. The decompression process involves reading the binary file, creating the LZW dictionary, generating output codes, converting the array to a binary string, and adding padding if necessary. The program calculates code length and compression ratio for both compression and decompression.

The program decomposes the color image into gray-level images, obtains the difference display, builds an LZW dictionary, and stores binary codes. During decompression, the program restores the dictionary, restores the difference image, reconstructs the original image from the differences, saves the restored image, and compares it to the original. The program calculates compression metrics for the compressed file.

The program can compress and decompress image data using LZW dictionary and differential compression methods. It calculates entropy, average code length, compressed file size, and compression ratio during compression. The program restores the LZW dictionary, difference image, and original image during decompression, and compares the restored image with the original image.

Figure 1: Class Hierarchy Diagram of the Program

The program has the button with the option to open the desired image from the user, which is given with the last stage. At the first stage, the program can open the text file from the computer and then compress it. There are parts about the graphical user interface that I can not do. For example, after the picture you choose, it divides the picture into grayscale and processed version. But it does not ask you to select a text file and select buttons for levels.

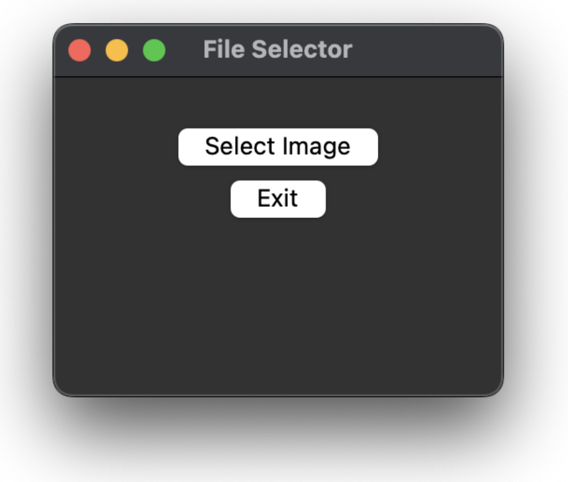


Figure 2: Graphical User Interface of the Program

2.1. Solutions For Levels

2.1.1. Solution of Level-1 : LZW Encoding and Decoding

The LZW algorithm is an algorithm for recognizing and encoding repetitive words. By adding a further alphabetic character, each word is defined to have a prefix equivalent to a pre-coded letter. The ascii character set serves as the alphabet for normal text files. The gray level of a pixel in a grayscale image with 256 gray levels is represented by an 8-bit integer. This algorithm forms the basis of the image compression program. Therefore, before doing the other steps, I wrote a program that can use the LZW method as we saw in the lesson. In the first stage of this, I used the flowchart of the LZW algorithm. (Figure 3)

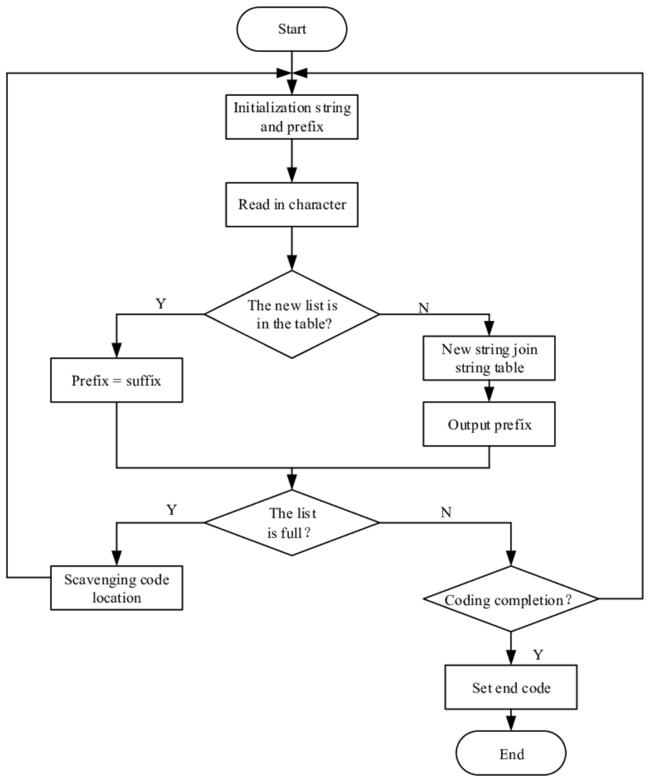


Figure 3: LZW Text Algorithm Flowchart

At Level 1, the program aims to compress and decompress character data by working on the text file. For compression, it first reads the characters from the text file we created on the computer. Creates the LZW dictionary of these characters. It generates output codes conforming to this algorithm in the form of an integer array and converts the array to a binary string. Checks if the array is a multiple of 8 and adds a zero if not. This process is padding. Adds 8 bits to the beginning of the array to indicate the number of zeros appended to the end of the array. The program also calculates the code length and compression ratio. To decompress, the program first reads the binary file. It creates an LZW dictionary for the file it reads. It generates output codes in the form of an integer array from the dictionary it creates. It then converts the array to binary string. If the array is not a multiple of 8, it adds zeroes and writes it to a binary output file. In addition, in the last stage, the program calculates the code length and compression ratio as well as the compression part.

I benefited from additional methods that encode, decode, save and read the document on the computer. I used a UML class diagram to develop the "Text Level Compression" coding for this level. (Figure 4)

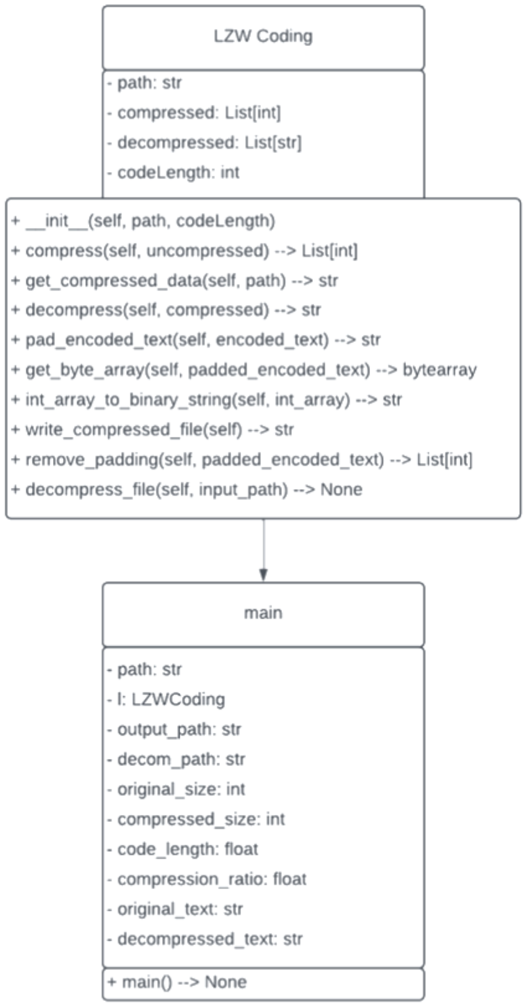
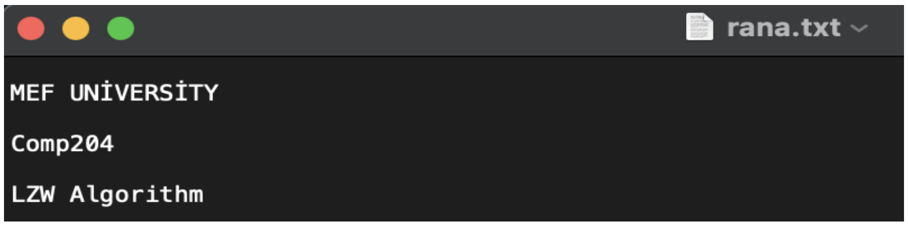

LZW Coding
- path: str
- compressed: List[int]
- decompressed: List[str]
- codeLength: int
+ __init__(self, path, codeLength)
+ compress(self, uncompressed) --> List[int]
+ get_compressed_data(self, path) --> str
+ decompress(self, compressed) --> str
+ pad_encoded_text(self, encoded_text) --> str
+ get_byte_array(self, padded_encoded_text) --> bytearray
+ int_array_to_binary_string(self, int_array) --> str
+ write_compressed_file(self) --> str
+ remove_padding(self, padded_encoded_text) --> List[int]
+ decompress_file(self, input_path) --> None
main
- path: str
- l: LZWCoding
- output_path: str
- decom_path: str
- original_size: int
- compressed_size: int
- code_length: float
- compression_ratio: float
- original_text: str
- decompressed_text: str
+ main() --> None

Figure 4: UML Class diagram of Level 1

First I created a usable txt file on my computer. I used object-oriented programming on the basis of the codes. I created a constructor in order to learn the file path to access the file I created. I created a function named "compress" after the class and variables I defined. The purpose of the function is to implement the LZW algorithm. With this algorithm, the characters in the text in the document I specify in the file path are based, and then a dictionary structure is started by defining the corresponding ASCII values for those words. According to the function, if the next letter in the text corresponds to an existing word in the dictionary, the letter continues to be taken from the text with the if structure. It prints the dictionary value matching up to that time as the result. If the text is not finished, it continues with the new letter that does not fit in step 2.

Using this algorithm, I have defined the task of performing compression to the function. Next, the goal is to make the compressed file binary. The first function I used was "int_array_to_binary_string". Its job is to transform an input integer string into a binary string representation. In the next step, I developed the "pad_encoded_text" functions to add padding to the compressed binary array whose length is a multiple of 8 and return the input "bytearray" object using "get byte array" in the process. The first of the file operations functions I set up was the "get_compressed_data" function, which initially learns the file path and returns compressed data as an array of 1s and 0s, was the first file operations function I created. I aimed to export compressed data to a binary file using the "write_compressed_file" function. IIn the decompression phase, I used the "decompress_file" function. With this function, I first read the compressed data from a file and then decompress it at the decompression stage. In the file named "main", there is the calculation part of the code in my "level1" document.

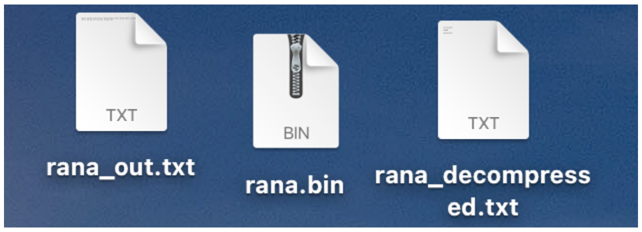
Figure 5-6 : Level 1 Input Images



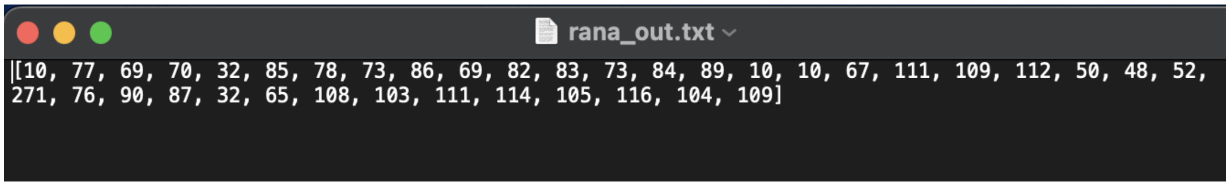
```
MEF UNİVERSİTY

Comp204

LZW Algorithm
```

(Figure 5) (Figure 6)

Figure 7-11: Level 1 Output Images

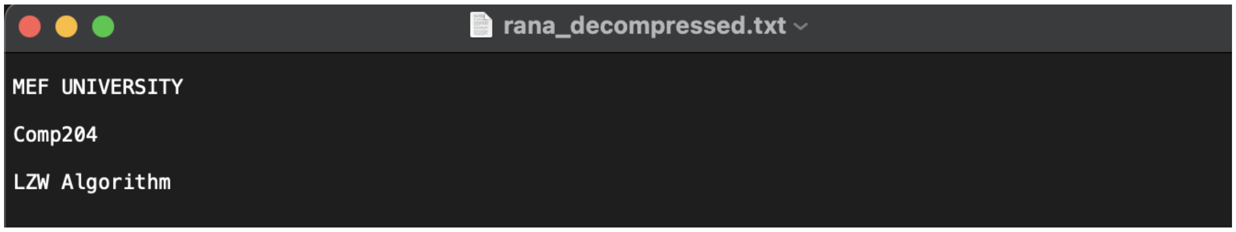


(Figure 7)



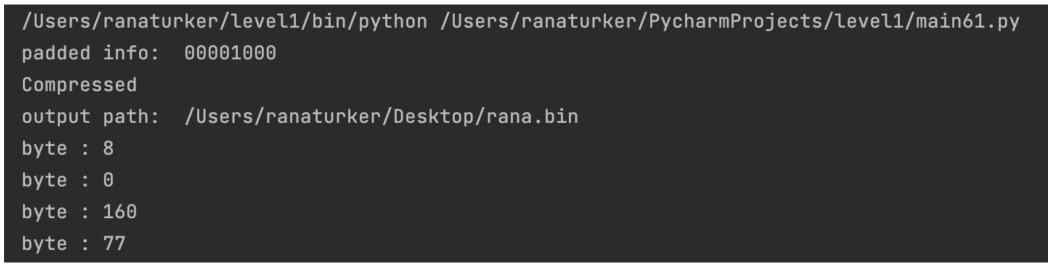
```
[10, 77, 69, 70, 32, 85, 78, 73, 86, 69, 82, 83, 73, 84, 89, 10, 10, 67, 111, 109, 112, 50, 48, 52,
271, 76, 90, 87, 32, 65, 108, 103, 111, 114, 105, 116, 104, 109]
```

(Figure 8)



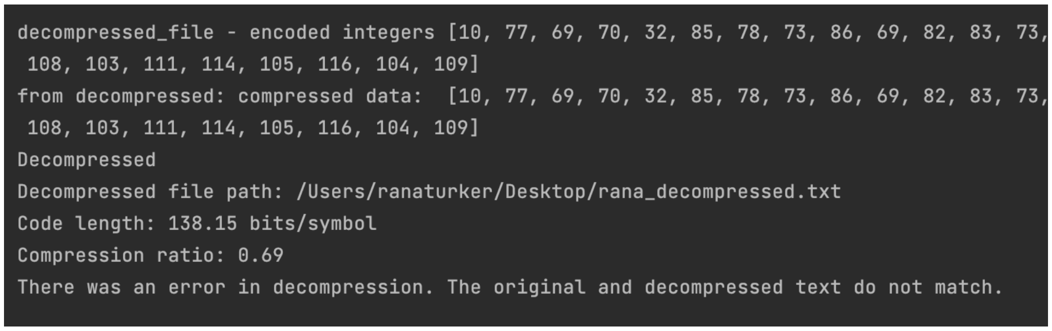
```
MEF UNIVERSITY

Comp204

LZW Algorithm
```

(Figure 9)



```
/Users/ranaturker/level1/bin/python /Users/ranaturker/PycharmProjects/level1/main61.py
padded info:  00001000
Compressed
output path:  /Users/ranaturker/Desktop/rana.bin
byte : 8
byte : 0
byte : 160
byte : 77
```

(Figure 10)



```
decompressed_file - encoded integers [10, 77, 69, 70, 32, 85, 78, 73, 86, 69, 82, 83, 73,
 108, 103, 111, 114, 105, 116, 104, 109]
from decompressed: compressed data:  [10, 77, 69, 70, 32, 85, 78, 73, 86, 69, 82, 83, 73,
 108, 103, 111, 114, 105, 116, 104, 109]
Decompressed
Decompressed file path: /Users/ranaturker/Desktop/rana_decompressed.txt
Code length: 138.15 bits/symbol
Compression ratio: 0.69
There was an error in decompression. The original and decompressed text do not match.
```

(Figure 11)

2.1.2.Solution of Level-2: Image Compression (Gray Level)

In level 2, the program can use compression and decompression features for images. Gray level image compression is at this level. For this, the program reads a black and white image file and scans the image. By establishing the compression and decompression functions like in the prior level, I began the coding portion. My script begins by reading a binary image file I called "small image.png" into a list of grayscale pixel values. I used a low byte image from the BlackBoard system so that the program could run fast. The "compressed_image.bin" file contains the compressed pixel data from the image after it has been compressed using the LZW compression algorithm. The program also calculates entropy, average code length, compressed file size and compression ratio. To compress, the codes read the stored data. It then restores the LZW dictionary, restores the image from the compressed data with the name "restored_image.png" and saves the restored image. Finally, it compares the restored images with the original image we initially loaded.

Figure 12: Level 2 Input Image

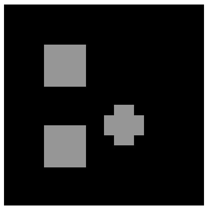
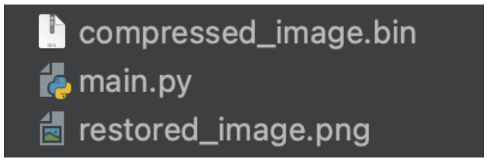
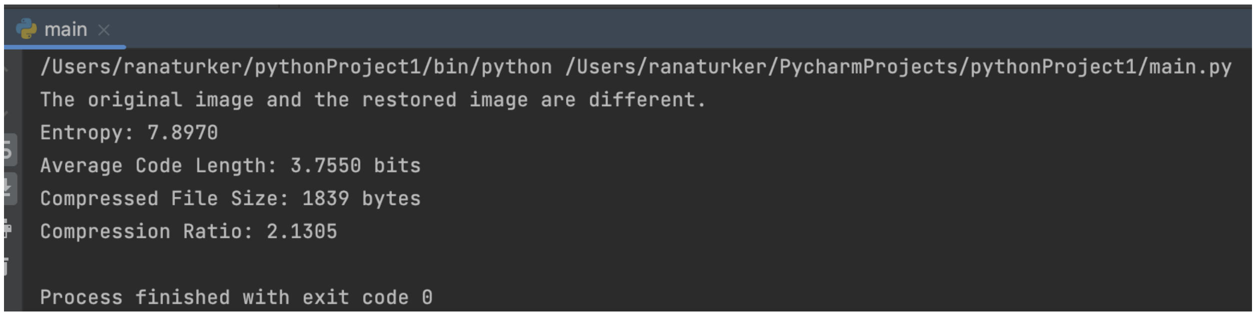


Figure 13-14: Level 2 Output Images



```
compressed_image.bin
main.py
restored_image.png
```

(Figure 13)



```
main
/Users/ranaturker/pythonProject1/bin/python /Users/ranaturker/PycharmProjects/pythonProject1/main.py
The original image and the restored image are different.
Entropy: 7.8970
Average Code Length: 3.7550 bits
Compressed File Size: 1839 bytes
Compression Ratio: 2.1305

Process finished with exit code 0
```

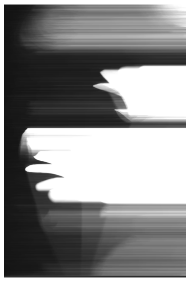
(Figure 14)

2.1.3. Solution for Level-3: Image Compression (Gray Level differences)

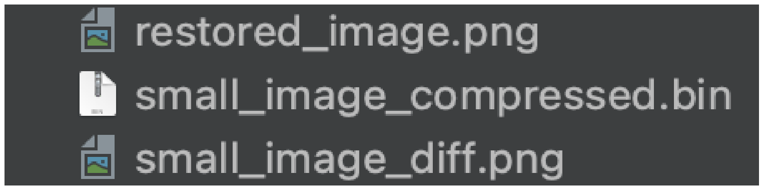
At level 3, the program again uses a compression and decompression method for the image. The difference from the previous level is that instead of focusing on the pixel size, it reduces the redundancy of the data in the picture by encoding the difference between the pixels. It basically handles the gray level differences. For compression, firstly, an image file that we saved on the computer is read into the program. The system obtains the difference image by taking the differences between consecutive gray levels on a row basis and column basis. Builds the LZW dictionary from the difference image it obtains. It then stores the binary code instead of the gray levels. It saves the stored code in a compressed file. As in the previous levels, the program calculates the entropy, average code length, size of the compressed file and compression ratio and prints it on the screen. In the decompression section, the program reads the stored data sequentially to decompress. Restores the LZW dictionary as a result of this data. Restores diff image from compressed data, restores original image from diffs, saves restored image. In the final stage, it compares the original and restored images. It informs the user on the screen with the text "Images are not identical".

Figure 12: Level 3 Input Image

Figure 13-15: Level 3 Output Images

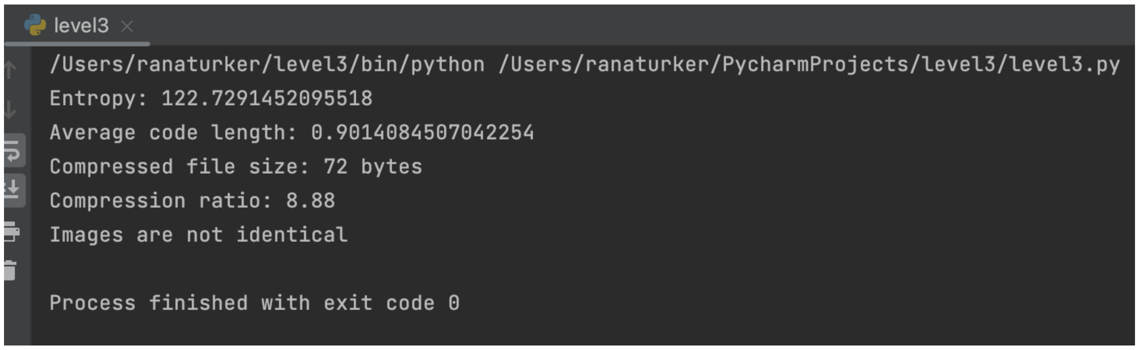


(Figure 13)



```
restored_image.png
small_image_compressed.bin
small_image_diff.png
```

(Figure 14)



```
/Users/ranaturker/level3/bin/python /Users/ranaturker/PycharmProjects/level3/level3.py
Entropy: 122.7291452095518
Average code length: 0.9014084507042254
Compressed file size: 72 bytes
Compression ratio: 8.88
Images are not identical

Process finished with exit code 0
```

(Figure 15)

2.1.4) Solution for Level-4: Image Compression (Color) & Level-5: Image Compression (Color differences)

At the 4th level, the program is in a form that can use colored images. I aimed to use the algorithm on a color picture. For compression, the program first reads a color image file from the computer. It obtains a difference image by calculating the differences on the basis of rows and columns. This image is shown using the Matplotlib library. Separates RGB components into three gray-level images. Scans the LZW dictionary to generate it, stores the binary code instead of gray levels, and saves the compressed file. The program calculates entropy, average code length, size of the compressed file and compression ratio and prints it to the screen. I couldn't do the decompression part. Here's what to do according to the slides: the program reads the stored data. Restores the LZW dictionary with the help of this data. In the next step, it restores the image from the compressed data, saves the restored image. Finally, it compares the original and restored images.

At level 5, the first step is again to read a color image file and split it into Red, Green and Blue components and decompose the image into three gray level images. The program then obtains the difference display by taking the row-column differences between successive gray levels in each row and column, respectively. An LZW dictionary is created through the data by scanning the difference image. Finally, the binary code is stored instead of gray levels and the compressed file is saved. The data stored in the decompression phase is read. The LZW dictionary is restored according to this data. The difference image is restored from the compressed data and the original image is rebuilt from the differences. The restored image is then saved and a comparison is made between the original and restored images to ensure the accuracy of the algorithm.

Figure 16: Level 4-5 Input Image

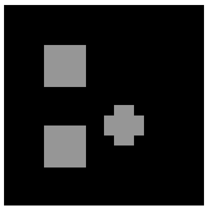
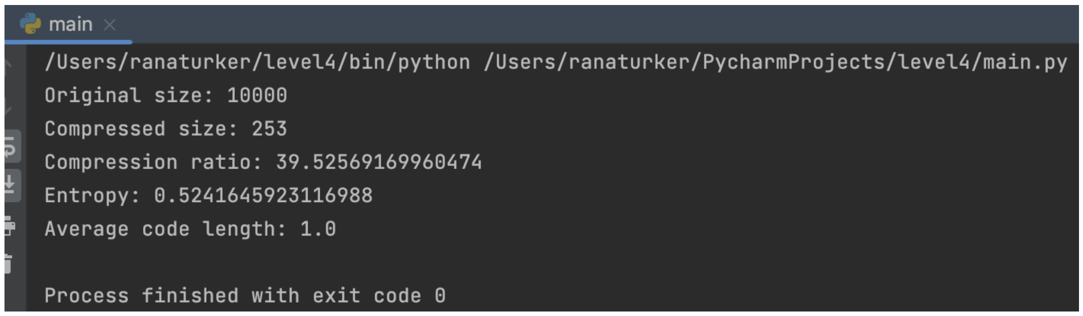
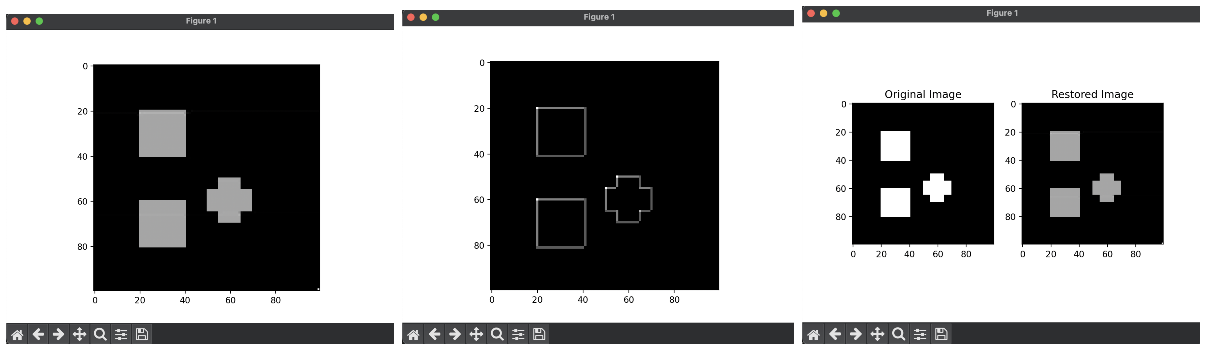


Figure 17-22: Level 4-5 Output Images



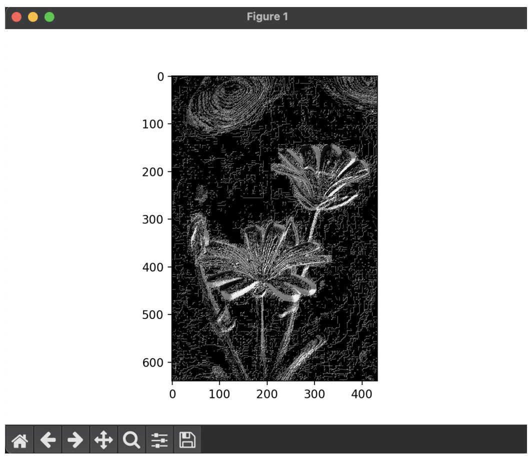
```
/Users/ranaturker/level4/bin/python /Users/ranaturker/PycharmProjects/level4/main.py
Original size: 10000
Compressed size: 253
Compression ratio: 39.52569169960474
Entropy: 0.5241645923116988
Average code length: 1.0

Process finished with exit code 0
```

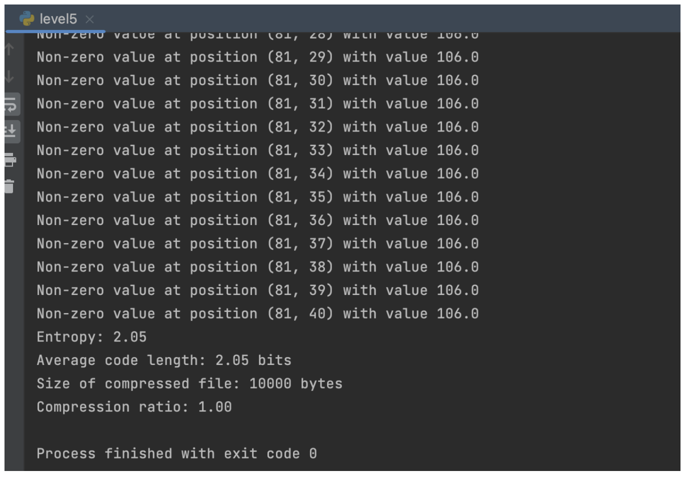
(Figure 17)



(Figure 18-19-20)

At this stage, I wanted to use color pictures. However, when I "run", it can not pass from the first stage to the next stage. The program continues to run in an infinite loop.





```
Non-zero value at position (81, 28) with value 106.0
Non-zero value at position (81, 29) with value 106.0
Non-zero value at position (81, 30) with value 106.0
Non-zero value at position (81, 31) with value 106.0
Non-zero value at position (81, 32) with value 106.0
Non-zero value at position (81, 33) with value 106.0
Non-zero value at position (81, 34) with value 106.0
Non-zero value at position (81, 35) with value 106.0
Non-zero value at position (81, 36) with value 106.0
Non-zero value at position (81, 37) with value 106.0
Non-zero value at position (81, 38) with value 106.0
Non-zero value at position (81, 39) with value 106.0
Non-zero value at position (81, 40) with value 106.0
Entropy: 2.05
Average code length: 2.05 bits
Size of compressed file: 10000 bytes
Compression ratio: 1.00

Process finished with exit code 0
```

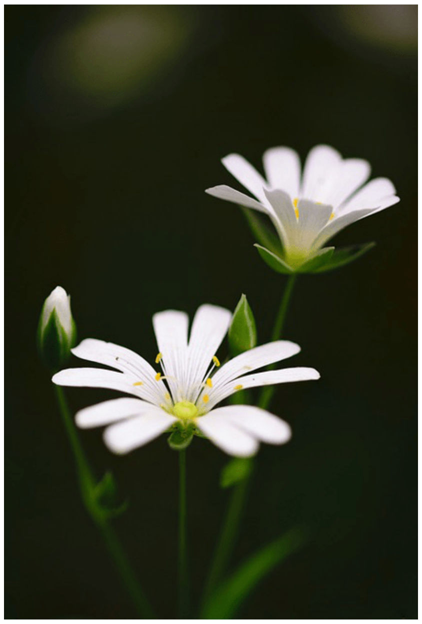
(Figure 21)

(Figure 22)

2.1.5. Solution for Level-6: Graphical User Interface

At the last step, it was intended to design a GUI that allows users to pick and upload files from a directory in order to make the application user-friendly. Graphical user interfaces are called GUIs. The Interface allows the user to select between text and image file types. The manner of compression is your choice. Users can also read compressed files, browse compressed text/image files, and save compressed images to files. For this, I utilized the Tkinter library. The "select" button selects a picture, which is then edited by the program using image manipulation methods in NumPy and the Python Imaging Library (PIL). The “convolve” function we use applies the matrix to the input image using NumPy multiply and add functions. The “threshold” function linearly scales the pixel density of an image. The “readPILimg” and ”color2gray” programs have the ability to read image files and convert them to grayscale. After the image processing stage is finished, the prepared program saves the processed images in the file directory. Only a button that lets the user choose a picture from their computer and divides the image into grayscale and processed version was anything I could add at this point.

Figure 23: Level 6 Input Image



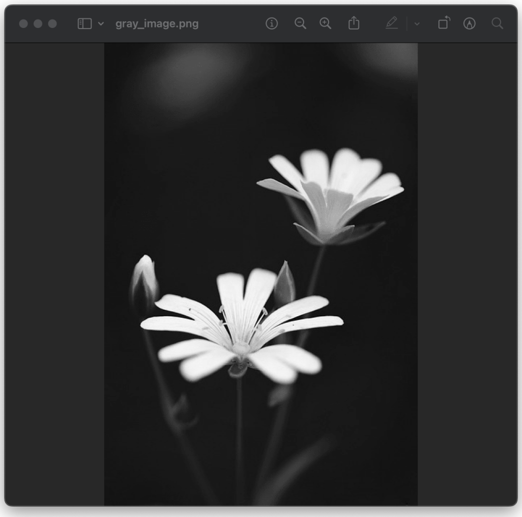


Figure 24: Level 6 Output Image

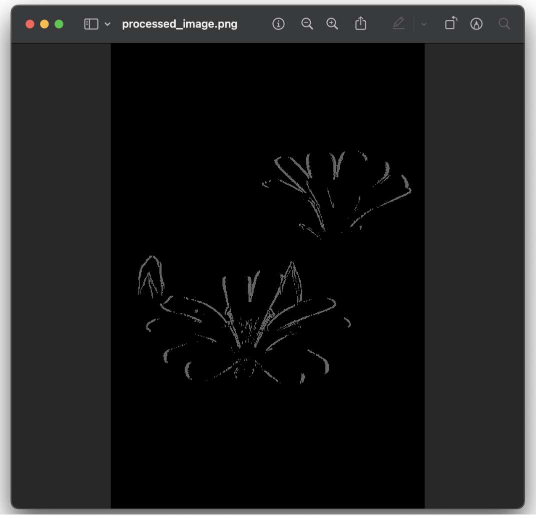


Figure 25: Level 6 Output Image

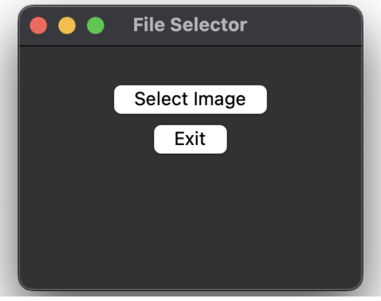
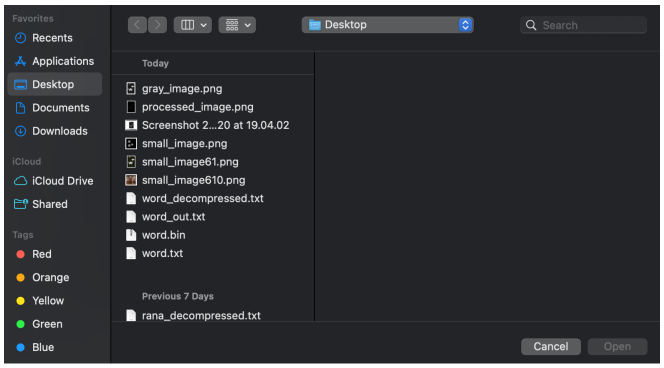


Figure 26: Level 6 GUI Image



Today
gray_image.png
processed_image.png
Screenshot 2...20 at 19.04.02
small_image.png
small_image61.png
small_image610.png
word_decompressed.txt
word_out.txt
word.bin
word.txt

Previous 7 Days
rana_decompressed.txt

Figure 27: Level 6 “Select Fıle” Image

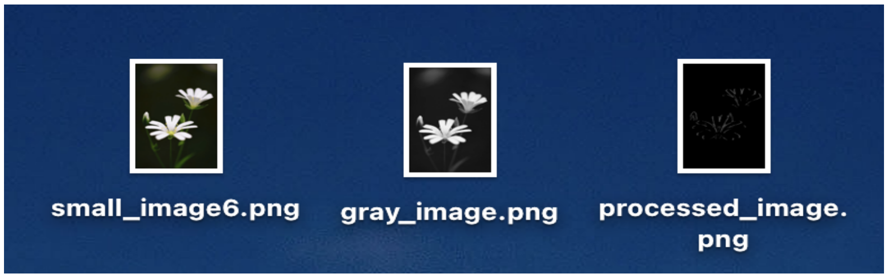


Figure 28: Level 6 Desktop Output Image

3) Achievements of the Project

As a result of the project, I learned a lot about algorithms and interfaces. I believe I've improved my python programming skills while trying to fix my mistakes throughout the process. By doing research on the LZW algorithm, I learned the main logic of the code and how it works. After working on this project for weeks, I understood the logic of image compression. Most importantly, I aimed to consciously implement the software development stages during the project. I gained experience and enthusiasm to be able to do larger projects in the future.

References

- A Design of an Assessment System for UML Class Diagram. (2007, August 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/4301193

- Garg G, Kumar R. Various Image Compression Techniques: A Review. IUP Journal of Telecommunications. 2022;14(1):24-34. Accessed March 20, 2023. https://ezproxy.mef.edu.tr:3093/linkprocessor/plink?id=bf685643-6214-3a44-b061-b3aabc64fbc9
- GeeksforGeeks. (2023, March 15). LZW Lempel Ziv Welch Compression technique. Retrieved March 20, 2023, from https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/

● Kaur, S. (2012, July 27). Design and Implementation of LZW Data Compression Algorithm.
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3916673#references-widget

● Ming-Bo Lin, Jang-Feng Lee, ALossless Data Compression and Decompression Algorithm and Its Hardware Architecture VLSI IEEE Transactions, volume 14, p. 925 - 936
Posted: 2006