DATA COMPRESSION TECHINQUE

A Project Report

Submitted in Partial Fulfilment of the Requirement for the Award of the Degree of

BACHELOR OF TECHNOLOGY

(COMPUTER SCIENCE & ENGINEERING)

To

VEER BAHADUR SINGH PURVANCHAL UNIVERSITY, JAUNPUR

Under the Supervision of

Mr. Santosh Kumar Yadav

(Assistant Professor)

Submitted by:

Sushant Kumar (185527)

Mukul Gupta (185539)

Preeti Yadav (185556)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UNSIET VBSPU JAUNPUR

AUGUST, 2022

CANDIDATE'S DECLARATION

We, Sushant Kumar (185527), Mukul Gupta (185539), Preeti Yadav (185556) a student of B.Tech. (Computer Science & Engineering) at **Uma Nath Singh Institute of Engineering and Technology, VBS Purvanchal University, Jaunpur**, declare that the work presented in this project titled "Data Compression Technique", submitted to **Department of Computer Science & Engineering** for the award of Bachelor of Technology degree in Computer Science & Engineering. All the work done in this project is entirely our own except for the reference quoted. To the best of our knowledge, this work has not been submitted to any other university or Institution for award of any degree.

Date:

Place: UNSIET, Jaunpur

Student's Name & Roll No.

Sushant Kumar (185527)

Mukul Gupta (185539)

Preeti Yadav (185556)

CERTIFICATE

It is certified that, this project entitled "Data Compression Technique", submitted by (Sushant Kumar, Mukul Gupta, Preeti Yadav) in partial fulfilment of the requirement for the award of Bachelor of Technology in Computer Science & Engineering degree from VBS Purvanchal University, Jaunpur, is record of student's own study carried under my supervision. This Project report has not been submitted to any other university or institution for the award of any degree.

Project Guide **External Examiner**

Mr. Santosh Kumar Yadav

Assistant Professor

Dept. of CSE & IT

Date:

Place: UNSIET, Jaunpur

ABSTRACT

The spread of computing has led to an explosion in the volume of data to be stored on hard disks and sent over the Internet. This growth has led to a need for "data compression", that is, the ability to reduce the amount of storage or Internet bandwidth required to handle this data. The focus is on the most prominent data compression schemes, particularly popular .DOC, .TXT, .BMP, .TIF, .GIF, and .JPG files. By using different compression algorithms, we get some results and regarding to these results we suggest the efficient algorithm to be used with a certain type of file to be compressed taking into consideration both the compression ratio and compressed file size. Data compression is widely used by the community because through a compression we can save storage. Data compression can also speed up a transmission of data from one person to another. Data that can be compressed not only text data but can be images and video. Data compression technique is divided into 2 namely lossy compression and lossless compression. But which is often used to perform a compression that is lossless compression. A kind of lossless compressions such as Huffman, Shannon Fano, Tunstall, Lempel Ziv welch and run-length encoding. Each method has the ability to perform a different compression. The output generated in doing a can be known through the compression file size that becomes smaller than the original file.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to several individuals for supporting us in our project. First, we wish to express our sincere gratitude to our project guide, Mr. Santosh Kumar Yadav, for his patience, insightful comments, helpful information, practical advice and unceasing ideas that have helped us tremendously at all times in our studies related to this project. We also wish to express our sincere thanks towards Dr. Sanjeev Gangwar, Head of Department of Computer Science & Engineering. Also, we are grateful to the faculty of the Computer Science & Engineering department who helped us directly or indirectly during this course of work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

1. INTRODUCTION

The essential figure of merit for data compression is the "compression ratio", or ratio of the size of a compressed file to the original uncompressed file. For example, suppose a data file takes up 50 kilobytes (KB). Using data compression software, that file could be reduced in size to, say, 25 KB, making it easier to store on disk and faster to transmit over an Internet connection. In this specific case, the data compression software reduces the size of the data file by a factor of two, or results in a "compression ratio" of 2:1 [1, 2]. There are "lossless" and "lossy" forms of data compression. Lossless data compression is used when the data has to be uncompressed exactly as it was before compression. Text files are stored using lossless techniques, since losing a single character can be in the worst case make the text dangerously misleading. Archival storage of master sources for images, video data, and audio data generally needs to be lossless as well. However, there are strict limits to the amount of compression that can be obtained with lossless compression. Lossless compression ratios are generally in the range of 2:1 to 8:1 [2, 3]. Lossy compression, in contrast, works on the assumption that the data doesn't have to be stored perfectly. Much information can be simply thrown away from images, video data, and audio data, and the when uncompressed; the data will still be of acceptable quality. Compression ratios can be an order of magnitude greater than those available from lossless methods. The question of which are "better", lossless or lossy techniques is pointless. Each has its own uses, with lossless techniques better in some cases and lossy techniques better in others. In fact, as this paper will show, lossless and lossy techniques are often used together to obtain the highest compression ratios. Even given a specific type of file, the contents of the file, particularly the orderliness and redundancy of the data, can strongly influence the compression ratio. In some cases, using a particular data compression technique on a data file where there isn't a good match between the two can actually result in a bigger file.

1.1 CLASSIFICATION OF DATA COMPRESSION

Lossless data compression techniques can be classified further into Entropy based data compression technique and Dictionary based data compression technique. In Entropy based data compression technique, the frequency of repeating data is first found and the repeating sequences are encoded. In Dictionary based data compression, a dictionary is maintained and the encoder looks for any entry for every piece of data. If there is an entry the encoder replaces the piece of data with the mapped encoding. If the entry is not present a new entry is made in the dictionary.

Compression is the reduction of a file size from a large size to a smaller file size. A compression will be done to facilitate the transmission of a file with a large size and contains many characters. The workings of a compression are by looking for patterns of repetition in the data and replace it with a certain sign. The type of compression has two methods, lossless compression and lossy compression.

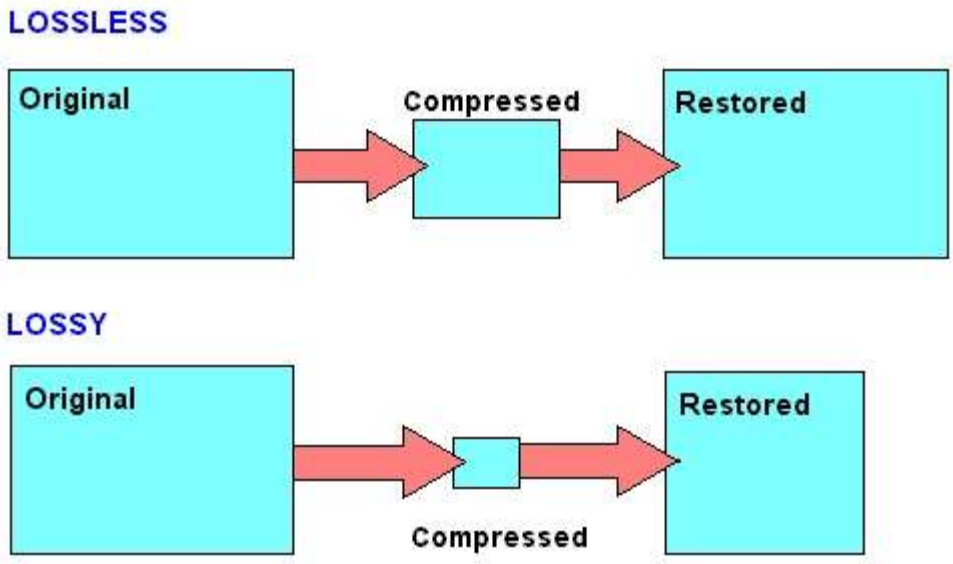

Fig. 1.1. Classification of Compression

Lossless compression is the process of converting the original data with compressed data becomes more concise without reducing the loss of information. Lossy compression is the prose converts the original data into the compression data there are different values, but the value of this difference is considered does not reduce the essential information from the original data.

1.2 APPLICABILITY THAT EXISTS IN A DATA COMPRESSION

1. Compression for audio
2. Compression for text
3. Compression for video
4. Compression for image

1.3 COMPRESSION FOR AUDIO

Whether you're mixing a music track or recording a podcast, audio compression is a useful tool for sound engineers who want to smooth out the sound of instruments, vocals, and more. From an acoustic guitar with too much pick noise to a piano that doesn't have any punch to it, an audio compressor has the power to decrease the dynamic range of an audio signal by making the loudest noises quieter and the quietest noises louder.

"You may set your levels so everything sounds good in one spot, but as the song goes on, some things may be hard to hear or get too loud and jump out of the mix," explains producer and mixing engineer Peter Rodocker. "The most common use of compression is to keep this from happening so you can get a more cohesive sound."

The fundamental controls of audio compressors.

Understanding the technical aspects and terminology of audio compression will allow you to work more comfortably with a wide range of compressors.

Threshold: The threshold is the level at which compression begins. Once a threshold level is set, any audio below the threshold will be unaffected, but any audio above the threshold will be compressed by the ratio set. "Imagine a guitarist that's strumming at a constant volume, then hits a few notes super hard in the middle of the song. You would want the compressor to be set where the threshold grabs only that loudest part of the track," says Rodocker.

Ratio: The ratio is the amount of attenuation — or downward compression — that will be applied once a signal reaches the threshold. Peter explains that "the volume of audio is measured in decibels (dB), so if you set a 3:1 compression ratio, every 3dB of input signal above the threshold will produce 1dB of output signal."

Attack time: The attack time is how long it takes the audio signal to be fully compressed to the ratio set after it has crossed the threshold. “It’s going to affect the first part of the sound you hear,” says Rodocker. “So, you can set a slower attack to create some punch from an instrument that’s softer, like a keyboard, or a fast attack to quickly compress the sound of a loud guitar pick.”

1.4 COMPRESSION FOR TEXT

These techniques are particularly intended for compressing natural language text and other data with a similar sequential structure such as program source code. However, these techniques can achieve some compression on almost any kind of (uncompressed) data.

These are the techniques used by general purpose compressors such as zip, gzip, bzip2, 7zip, etc.

Entropy encoding techniques such as Huffman coding and arithmetic coding (only) need a probability distribution as an input. The method for determining the probabilities is called a model. Finding the best possible model is the real art of data compression.

There are three types of models:

1. Static
2. Semi adaptive or semi static
3. Adaptive.

A **static model** is a fixed model that is known by both the compressor and the decompressor and does not depend on the data that is being compressed. For example, the frequencies of symbols in English language computed from a large corpus of English texts could be used as the model.

A **semi adaptive or semi static model** is a fixed model that is constructed from the data to be compressed. For example, the symbol frequencies computed from the text to be compressed can be used as the model. The model has to be included as a part of the compressed data. A semi adaptive model adapts to the data to be compressed. For example, a static model for English language would be in optimal to compressing other languages

or something completely different such as DNA sequencies. A semi adaptive model can always be the optimal one.

The drawback of a semi adaptive model is the need to include the model as a part of the compressed data. For a short data, this could completely negate any compression achieved. On the other hand, for a very long data, the space needed for the model would be negligible. Sometimes it may possible to adapt the complexity of the model to the size of the data in order to minimize the total size. (According to the Minimum Description Length (MDL) principle, such a model is the best model in the machine learning sense too.)

Every semi adaptive model can be seen as a static model with adaptable parameters. An extreme case would be to have two static models and a one-bit parameter to choose between them.

An **adaptive model** changes during the compression. At a given point in compression, the model is a function of the previously compressed part of the data. Since that part of the data is available to the decompressor at the corresponding point in decompression, there is no need to store the model. For example, we could start compressing using a uniform distribution of symbols but then adjust that distribution towards the symbol frequencies in the already processed part of the text.

Not having to store a model saves space, but this saving can be lost to a poor compression rate in the beginning of the compression. There is no clear advantage either way. As the compression progresses, the adaptive model improves and approaches optimal. In this way, the model automatically adapts to the size of the data.

The data is not always uniform. An optimal model for one part may be a poor model for another part. An adaptive model can adapt to such local differences by forgetting the far past.

1.5 COMPRESSION FOR VIDEO

Video compression has become a necessity these days with the growing number of video streaming sites and video downloads resulting in a shortage of storage space. So, when it comes to choosing the right compression format, a plethora of options are available.

Selecting the smallest video file format here becomes important to retain maximum files within minimum size. Some of the popular names in the list of smallest video format include AVI, WMV, MP4, H.264, and others. Analyse your requirements and choose from one of these compressed video formats. Best compressed Video Formats are:

Selecting the right compressed video depends on your requirements as there are a number of these formats available. Some of these formats use techniques and algorithm that results in file compression while maintaining high quality. Know about the top 5 formats that are used commonly for video compression.

H.264: Also known as MPEG-4 Part 10, Advanced Video Coding (MPEG-4 AVC) H.264 is one of the best video compression formats as it is capable of offering high-quality files at lower bit rates. This is a versatile format that is compatible with a number of players and devices. Moreover, the format is also suitable for Blu-ray disc playback as it offers high quality at high bitrate files. Flexibility is another reason that makes this format popular and widely in use.

DivX: Developed by DivX, LLC this is another format that is known for its compression capabilities. The format can shrink your video files into small size while maintaining high quality. This is an editable video format that is mainly used in a number of commercial settings.

MP4: MP4 is termed as a universal format as it is compatible with the majority of the players and devices available. This multimedia container format is capable of compressing videos into smaller file size while keeping decent quality. Almost all versions of MPEG-4 and H.264 are compatible with the format. Majority of the online streaming sites use MP4 as their format.

AVI: Standing for Audio Video Interleaved, AVI is a multimedia container format that offers high quality compressed files. The format is capable of streaming multiple audio and video files. Wide range of codecs is supported by the format thus facilitating an array of file settings as desired. Though the format is not very popular for video streaming or downloading, it can be a good choice for video storage on your PC.

WMV: WMV stands for Windows Media Video and this format developed by Microsoft is a part of the Windows Media Framework. If you are looking for file compression to send your large files over email and other sources that have a limitation to the file size WMV is an ideal format. Loss in quality is one of the drawbacks using this compression format.

1.6 COMPRESSION OF IMAGES

Photo compression, or image compression, is a process that reduces an image's file size so that it takes up less memory in your computer without downgrading the image's quality too much. Compressing photos is not an overly complicated process, and it's a good way to save computer memory.

Compressing photos saves space on your computer and makes them easier to email or post on the web, as the photos will take up less memory. Compression is a helpful tool, especially if you need to alter an image for your website or a social media marketing post. Since it's not too difficult to compress photos, there's no reason to shy away from the process due to initial lack of knowledge.

CHAPTER 2

2. PROBLEM FORMULATION AND PROPOSED WORK

2.1 PROBLEM DEFINITION

Data compression is a process of converting particular amount of data into another form in which data has taken a lesser amount of storage.

Data compression technique is used mainly for two reasons:

1. Increase the transfer speed of data.
2. Decrease the storage space.

In daily life, when we send the large amount of data to another person there is many problems occur to send data and there are many platforms to compress data but when we compress the data on that platform then amount of data is loss or data is scattered.

2.2 OBJECTIVE

Audio data compression reduces the transmission bandwidth and storage requirements of audio data. Audio compression algorithms are implemented in soft-ware as audio codecs. Lossy audio compression algorithms provide higher compression at the cost of fidelity, are used in numerous audio applications. These algorithms almost all rely on psychoacoustics to eliminate less audible or meaningful sounds, thereby reducing the space required to store or transmit them.

Video compression uses modern coding techniques to reduce redundancy in video data. Most video compression algorithms and codecs combine spatial image com-pression and temporal motion compensation. Video compression is a practical implementation of source coding in information theory. In practice most video codecs also use audio compression techniques in parallel to compress the separate, but combined data streams.

Grammar-Based Codes they can extremely compress highly-repetitive text, for instance, biological data collection of same or related species, huge versioned document collection, internet archives, etc. The basic task of grammar-based codes is constructing a context-free grammar deriving a single string. Sequitur and Repair are practical grammar compression algorithms which public codes are available.

2.3 PROPOSED WORK

We take an any form of data i.e., images, data, video and audio. We apply suitable algorithm on it and compress their sizes. Compression is performed by a program that uses a formula or algorithm to determine how to shrink the size of the data. For instance, an algorithm may represent a string of bits -- or 0s and 1s -- with a smaller string of 0s and 1s by using a dictionary for the conversion between them, or the formula may insert a reference or pointer to a string of 0s and 1s that the program has already seen.

Text compression can be as simple as removing all unneeded characters, inserting a single repeat character to indicate a string of repeated characters and substituting a smaller bit string for a frequently occurring bit string. Data compression can reduce a text file to 50% or a significantly higher percentage of its original size.

For data transmission, compression can be performed on the data content or on the entire transmission unit, including header data. When information is sent or received via the internet, larger files, either singly or with others as part of an archive file, may be transmitted in a ZIP, GZIP or other compressed format.

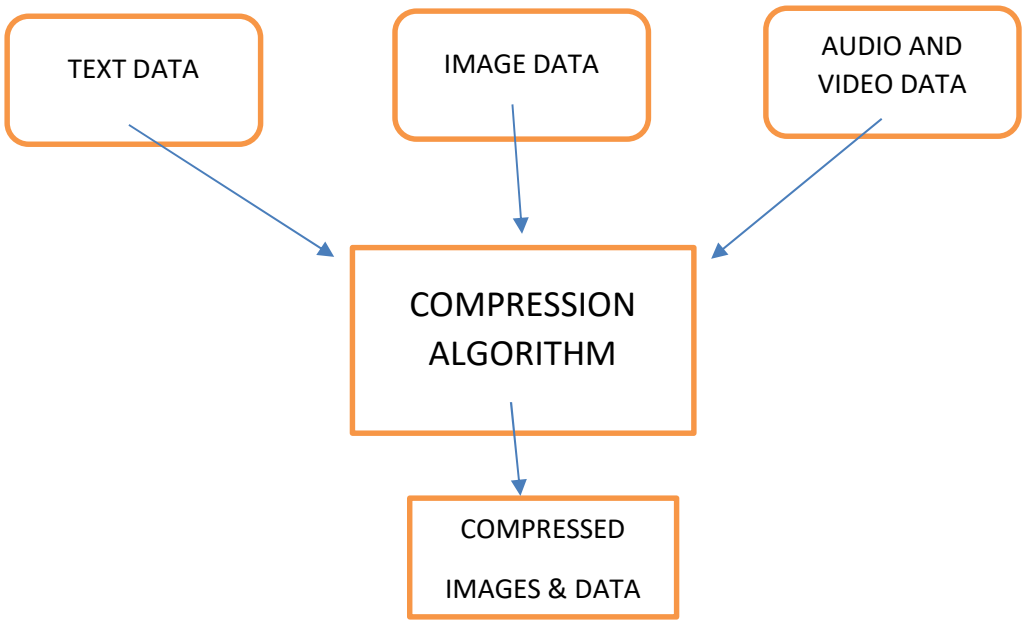

Fig. 2.1. Flowchart of Proposed System

CHAPTER 3

3. METHODOLOGY

3.1 DATA COMPRESSION

Compressing data can be a lossless or lossy process. Lossless compression enables the restoration of a file to its original state, without the loss of a single bit of data, when the file is uncompressed. Lossless compression is the typical approach with executables, as well as text and spreadsheet files, where the loss of words or numbers would change the information.

Lossy compression permanently eliminates bits of data that are redundant, unimportant or imperceptible. Lossy compression is useful with graphics, audio, video and images, where the removal of some data bits has little or no discernible effect on the representation of the content.

Graphics image compression can be lossy or lossless. Graphic image file formats are typically designed to compress information since the files tend to be large. JPEG is an image file format that supports lossy image compression. Formats such as GIF and PNG use lossless compression.

Compression is the process of converting a data set into a code to save the need for storage and transmission of data making it easier to transmit a data. With the compression of a can save in terms of time and storage that exist in memory (storage). Many compression algorithm techniques can be performed and function properly such as the Huffman, Lempel Ziv Welch, Run Length Encoding, Tunstall, And Shannon Fano methods. The data process of data compression.

How the data when not compressed then uncompressed data will be continued and processed by compression method that is lossless compression then the data has been compressed will produce a size smaller than the size of the file before it is compressed.

3.2 RUN-LENGTH ENCODING (RLE)

This is lossless data compression algorithm in which algorithm is applied on data in such way that large repeating pattern is caprice into small number of characters of string, algorithm produce two bytes of output for each repeating character in data, one byte says that the total number character present in repeating pattern, and another byte says that actual repeating character in string it's also called as run, (and its count called as run count.) If we consider an image which has a lot of colours in it. A hypothetical line representing the pixel colours is as follows –

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWWWWWRRRRR

Once the Run length data compression algorithm is applied to the above line it can be represented as –

6W3R5G5W6B8W5R

Hence the large string is compressed into a smaller string. As evident in the above example that the data was compressed efficiently. But this is not always the case. In RLE algorithm the worst-case situation can result into the output being double the size of the Input string.

For example: if **WRGB** is to be encoded using RLE than its output will be **1W1R1G1B** which means that the size of encoded string is more than the original string.

3.2.1 FLOW CHART OF RUN LENGTH ALGORITHM

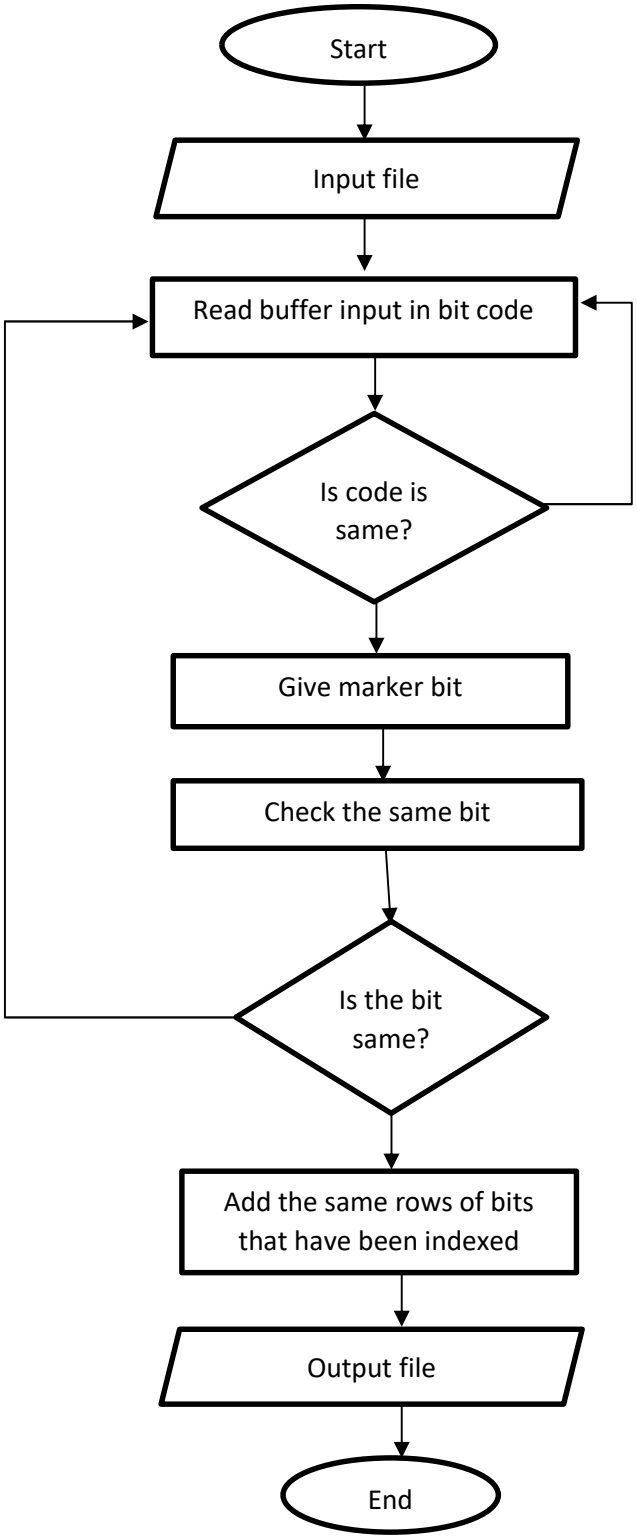

Fig. 3.1. Flowchart of Run-length algorithm

3.3 SHANNON-FANO ALGORITHM

This algorithm is mainly used to compress text files, it is similar to Huffman Coding (HC) algorithm; the only difference is that the SF algorithm uses the top-down approach while HC algorithm uses the bottom-up approach on the text. SF algorithm is rarely used as it is not useful for all the data formats as compared to Huffman Coding algorithm which is faster and better. In SF algorithm, the letter from string is converted to its binary form from their respective ASCII value; then the probability of the occurrence of each letter is calculated and then the tree is created based on the frequency.

Considering the same string that was used in run length algorithm:

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWW WWRRRRR

Calculating the total number of times each character has occurred and arranged it in descending order of their frequency. Then a tree can be created for the characters:

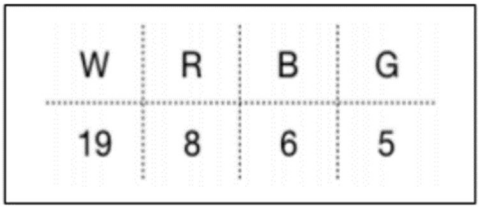

Fig. 3.2. Frequency of letters in the considered sequence

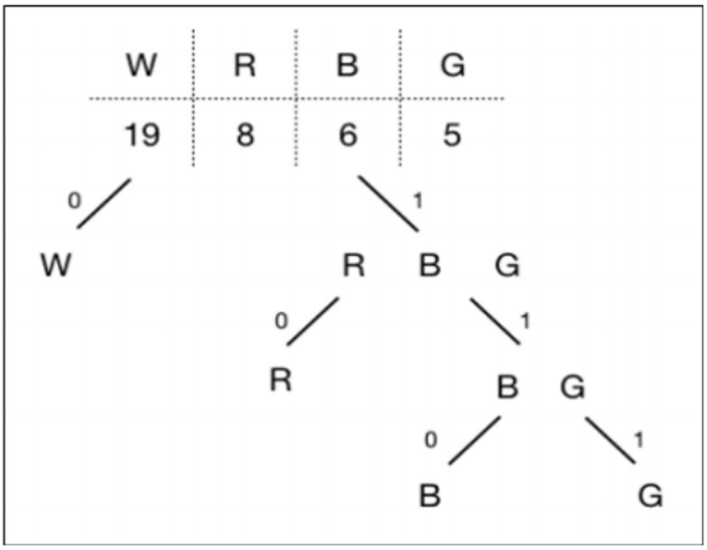

Fig. 3.3. Tree for Shannon-Fano Encoding

The value of the characters becomes as follows

W-0 R-10 B-110 G–111

Hence the encoded string becomes

000000101010111111111111111000001101101101101101100000000001010101010.

The size of this output string is 68 bits i.e., **8.5 bytes.**

3.4 HUFFMAN CODING

The idea is founded on assigning variable length of code to input character length of code is depends on size frequency of character the highest repeatable code gets smallest code and smallest repeatable code get largest code. It is working is same as Shannon-Fanon Algorithm.

In Huffman coding the most repeatable word gets least binary number and least repeatable word get highest binary number. In Huffman coding the more often a symbol occurs in the original data the shorter the binary string used to represent it in the compressed data. Huffman coding requires two passes one to build a statistical model of the data and a second to encode it so is a relatively slow process. It is similar to Shannon Fano but it follows bottom-up approach instead of top-down approach. Considering the same string that was used in run length algorithm:

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWWWWRRRRR

Firstly the total number of times each character has occurred should be counted and the frequencies should be arranged in descending order. Refer fig. 3.3. for the character count. We then create the tree for the character

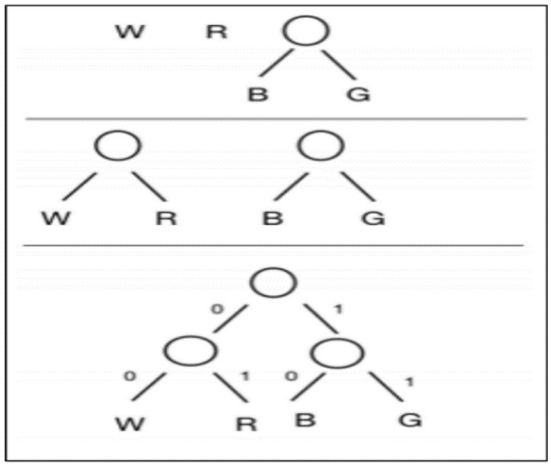

Fig. 3.4. Huffman Tree

The value of the characters becomes as follows

W- 00 R -01 B -10 G -11

Hence the encoded string becomes

000000000000010101111111111000000000010101010100000000000000000101010101

The size of this output string is 76 bits i.e., **9.5 bytes.**

3.4.1 Flowchart of Huffman Coding

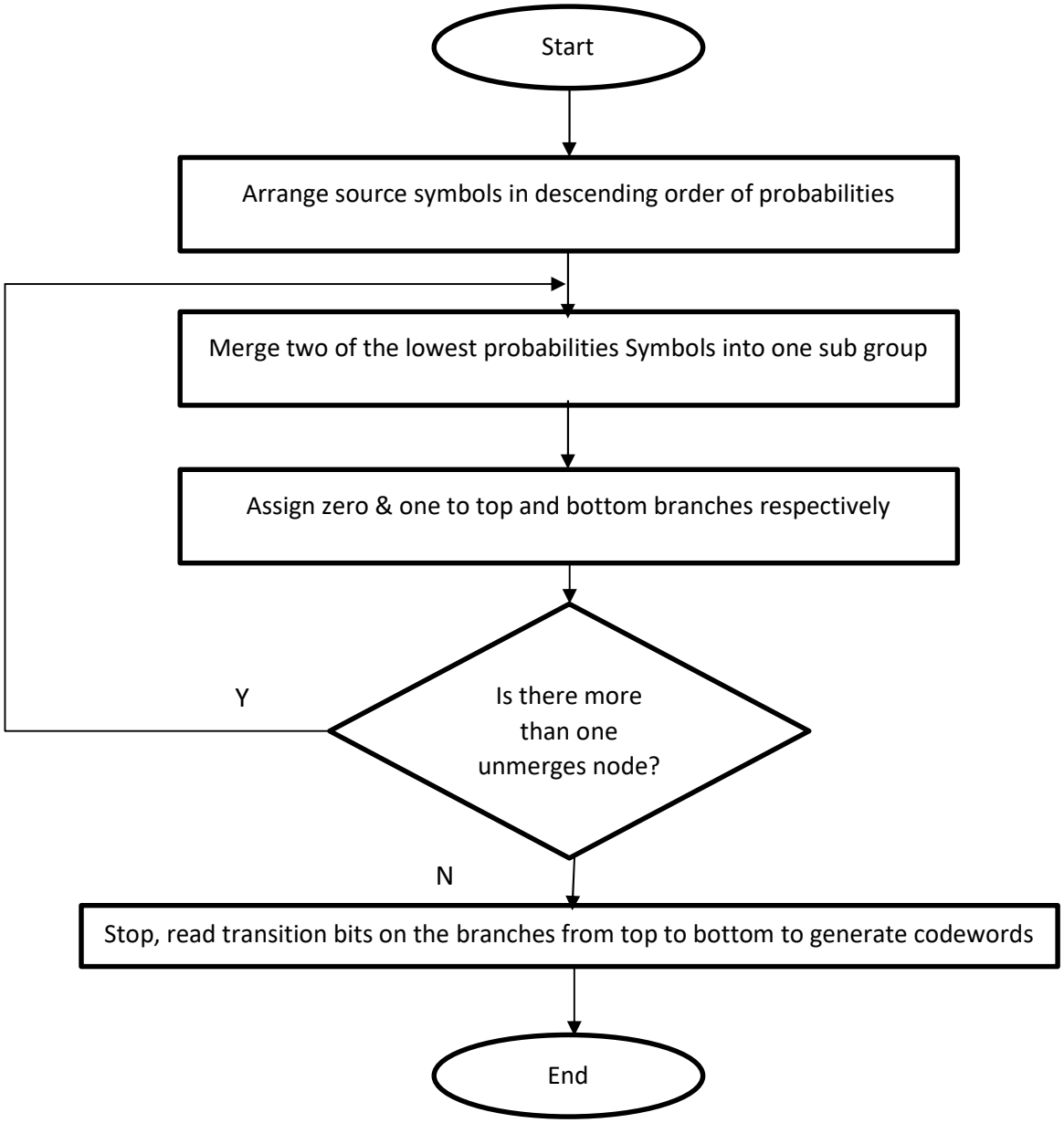

Fig. 3.5. Flowchart of Huffman coding

3.5 LEMPEL-ZIV-WELCH

This algorithm is used in PDF and TIFF (Tagged Image File Format), it's mostly used in UNIX file compression utility and is used in GIF Format. LZW reads a sequence of symbol and performs grouping of the symbols into string and then converts into code, so code takes less space than actual string. In this algorithm, as the input data being processed, a dictionary keeps track of all the corresponding words and where they are encountered and converts them into a code value. The words are replaced by their corresponding codes and so the input file is compressed and decompression creates the same string table and by analysing the input stream it decodes and translates the code back to the original string or text.

Considering the same string as example:

WWWWWWWRRRGGGGGWWWWWBBBBBBWWWWWWWWWRRRRR

Create Small Table/ Sample Table as follows. This table defines codes for each distinct character in the data. Small table is used as initializer of the process

Character	Code
W	1
R	2
G	3
B	4

Table. 3.1. Sample Table for Lempel-Ziv-Welch

Then, a complete dictionary is made as in the table 3.2.

Encoded Output	Index	Pattern
	1	W
	2	R
	3	G
	4	B
1	5	WW
5	6	WWW
6	7	WWWR
2	8	RR
8	9	RRG
3	10	GG
10	11	GGG
10	12	GGW
5	13	WWWW
5	14	WWB
4	15	BB
15	16	BBB
16	17	BBBW
13	18	WWWWW
13	19	WWWWR
8	20	RRR
8	-	-

Table. 3.2. Dictionary for Lempel-Ziv-Welch

Encoded Output is: **1,5,6,2,8,3,10,10,5,5,4,15,16,13,13,8.**

The binary encoding can be written as

00001,00101,00110,00010,01000,00011,01010,01010,00101,00101,00100,01111,010000,01101,01101,01000.

Thus, the compression converts the string into a sequence of 80 bits i.e., **10 bytes.**

3.5.1 Flowchart of Lempel-Ziv-Welch

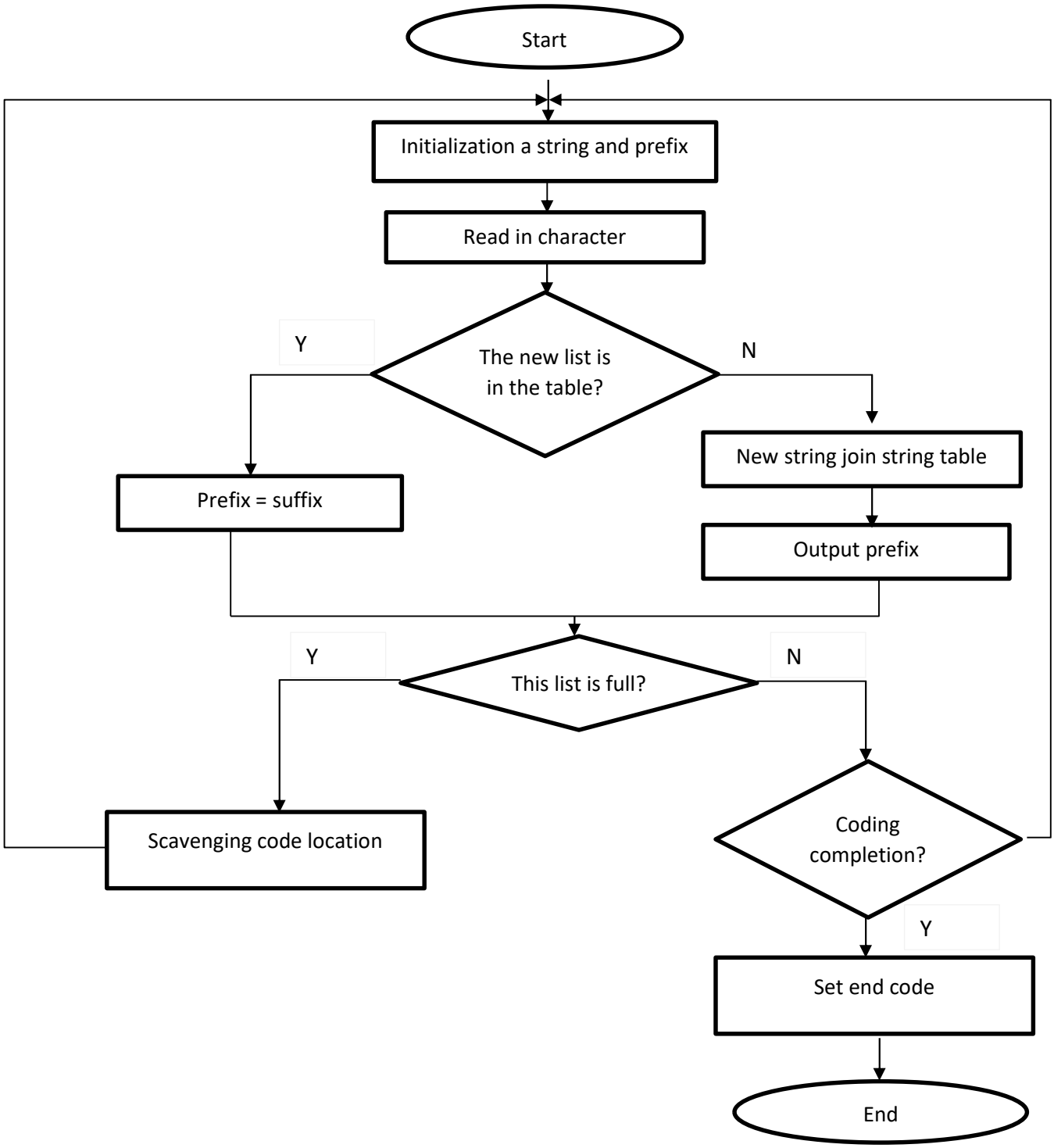

Fig. 3.6. Flow Chart of Lempel-Ziv-Welch

3.6 COMPRESSION PERFORMANCE

While measuring the compression performance many factors have to be taken into consideration but the main two factors are space and time efficiency. By finding the compression ratio, compression factor, data saving percentage, compression time and the code efficiency of each algorithm we can compare them and measure their performance.

$$\text{Compression Ratio} = \frac{\text{Size of Uncompressed Data}}{\text{Size of Compressed Data}}$$

$$\text{Compression factor} = \frac{1}{\text{Compression Ratio}}$$

$$\text{Saving Percentage} = \frac{\text{Uncompressed Data} - \text{Compressed Data}}{\text{Uncompressed Data}}\%$$

Size of Uncompressed Data is 38 bytes which is same for all the techniques since the string sequence used to study compression is same for all techniques.

Compression ratios, Compression factors and Saving percentage for different Compression techniques are as in the table 3.3.

Technique	Compressed Data Size (bytes)	Compression Ratio	Compression Factor	Saving Percentage
Run Length Encoding	14	2.7	0.37	63.1
Shannon Fano	8.5	4.5	0.22	77.6
Huffman Coding	9.5	4	0.25	75.0
Lempel Ziv Welch	10	3.8	0.26	73.6

Table. 3.3. Compression ratios, Compression factors and Saving percentage for different Compression technique

3.7 PYTHON

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development.

Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development.

Python supports multiple programming pattern, including object-oriented, imperative, and functional or procedural programming styles.

Python is not intended to work in a particular area, such as web programming. That is why it is known as multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc.

We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to assign an integer value in an integer variable.

Python makes the development and debugging fast because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

3.8 IDE (VS CODE)

Visual Studio Code is a lightweight source code editor. The Visual Studio Code is often called VS Code. The VS Code runs on your desktop. It’s available for Windows, macOS, and Linux.

VS Code comes with many features such as IntelliSense, code editing, and extensions that allow you to edit Python source code effectively. The best part is that the VS Code is open-source and free.

Besides the desktop version, VS Code also has a browser version that you can use directly in your web browser without installing it.

3.8.1 Setting up Visual Studio Code

To set up the VS Code, you follow these steps:

First, navigate to the VS Code official website and download the VS code based on your platform (Windows, macOS, or Linux).

Second, launch the setup wizard and follow the steps.

Once the installation completes, you can launch the VS code application:

Install Python Extension

To make the VS Code works with Python, you need to install the Python extension from the Visual Studio Marketplace.

The following illustrates the steps:

First, click the Extensions tab.

Second, type the python keyword on the search input.

Third, click the Python extension. It'll show detailed information on the right pane.

Finally, click the Install button to install the Python extension.

Now, you're ready to develop the first program in Python.

3.9 ZLIB

The zlib is a Python library that supports the zlib C library, a higher-level generalization for deflating lossless compression algorithms. The zlib library is used to lossless compress, which means there is no data loss between compression and decompression).

It also provides portability advantages across the different platforms and doesn't expand the data. This library plays a very significant role in terms of security. Many applications require the compression and decompression of arbitrary data such as strings, files, or structured in-memory content.

This library is well-suited with the gzip file format/tool, the most popular and useful compression application on UNIX systems.

compression method-The compression () method

The zlib library facilitates us to compress () method, which is used to compress a data string. Below is the syntax of the function.

compress (data, level=-1)

Parameter -The data parameter specifies the bytes to be compressed, and the level represents an integer value between -1 to 9. The level parameter is used to define the level of the compression. Level 9 signifies the slowest; however, it brings the highest compression level. The value -1 is a default that is level 6. Level 0 yields no compression.

3.10 PIP

PIP is a package manager for Python packages, or modules if you like.

Python pip is the package manager for Python packages. We can use pip to install packages that do not come with Python. The basic syntax of pip commands in command prompt is:

pip 'arguments'

How to install pip?

Python pip comes pre-installed on 3.4 or older versions of Python. To check whether pip is installed or not type the below command in the terminal.

pip --version

This command will tell the version of the pip if pip is already installed in the system.

If you do not have pip installed on your system refer to the below articles.

How to install Package with Pip?

We can install additional packages by using the Python pip install command.

3.11 Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps −

Import the Tkinter module.

Create the GUI application main window.

Add one or more of the above-mentioned widgets to the GUI application.

Enter the main event loop to take action against each event triggered by the user.

3.12 PILLOW

Python Imaging Library (expansion of PIL) is the de facto image processing package for Python language. It incorporates lightweight image processing tools that aids in editing, creating and saving images. Support for Python Imaging Library got discontinued in 2011, but a project named pillow forked the original PIL project and added Python3.x support to it. Pillow was announced as a replacement for PIL for future usage. Pillow supports a large

number of image file formats including BMP, PNG, JPEG, and TIFF. The library encourages adding support for newer formats in the library by creating new file decoders. This module is not preloaded with Python. So, to install it execute the following command in the command-line:

pip install pillow

1. Opening an image using open (): The PIL.Image.Image class represents the image object. This class provides the open () method that is used to open the image.

2. Displaying the image using show (): This method is used to display the image. For displaying the image Pillow first converts the image to a .png format (on Windows OS) and stores it in a temporary buffer and then displays it. Therefore, due to the conversion of the image format to .png some properties of the original image file format might be lost (like animation). Therefore, it is advised to use this method only for test purposes.

3. Obtaining information about the opened image

A) Getting the mode (colour mode) of the image: The mode attribute of the image tells the type and depth of the pixel in the image. A 1-bit pixel has a range of 0-1, and an 8-bit pixel has a range of 0-255.

B) Getting the size of the image: This attribute provides the size of the image. It returns a tuple that contains width and height.

C) Getting the format of the image: This method returns the format of the image file.

4. Rotating an image using rotate (): After rotating the image, the sections of the image having no pixel values are filled with black (for non-alpha images) and with completely transparent pixels (for images supporting transparency)

5. Resizing an image using resize (): Interpolation happens during the resize process, due to which the quality of image changes whether it is being upscaled (resized to a higher dimension than original) or downscaled (resized to a lower Image then original). Therefore resize () should be used cautiously and while providing suitable value for resampling argument.

6. Saving an image using save (): While using the save () method Destination path must have the image filename and extension as well. The extension could be omitted in Destination path if the extension is specified in the format argument.

3.13 OS Module

It is possible to automatically perform many operating system tasks. The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc.

You first need to import the os module to interact with the underlying operating system. So, import it using the import os statement before using its functions.

1. Getting Current Working Directory

The **getcwd()** function confirms returns the current working directory.

2. Creating a Directory

We can create a new directory using the **os.mkdir()** function.

3. Changing the Current Working Directory

We must first change the current working directory to a newly created one before doing any operations in it. This is done using the **chdir()** function.

4. Removing a Directory

The **rmdir()** function in the OS module removes the specified directory either with an absolute or relative path.

5. List Files and Sub-directories

The **listdir()** function returns the list of all files and directories in the specified directory.

3.14 TOOLS/ HARDWARE/ SOFTWARE

3.14.1 Hardware

RAM 8GB

Hard Disk 2GB

Processor: Pentium 4 (1.5GHZ)

CPU: 2 x 64-bit, 2.8 GHz, 8.00 GT/s CPUs or better

3.14.2 Software

Python 3.10.

VS Code

CHAPTER 4

4. PROJECT RESULT

4.1 PROJECT CODE

4.1.1 For image compression

```
# run this in any directory
# add -v for verbose
# get Pillow (fork of PIL) from
# pip before running -->pip
# pip install Pillow
# import required libraries
import os
import sys
from tkinter import filedialog
from PIL import Image
from tkinter.filedialog import*
# define a function for
# compressing an image
def compressMe(file, verbose = False):
# Get the path of the file
# filepath = os.path.join(os.getcwd(),file)
    filepath = askopenfilename()
    # open the image
    picture = Image.open(filepath)
    print('compressing', filepath[37:])
    # Save the picture with desired quality
    # To change the quality of image,
    # set the quality variable at
    # your desired level, The more
    # the value of quality variable
    # and lesser the compression
```

```
    picture.save("Compressed_"+filepath[37:],
                 "JPEG",
                 optimize = True,
                 quality = 10)
    return
# Define a main function
def main():
    verbose = False
    # checks for verbose flag
    if (len(sys.argv)>1):
        if (sys.argv[1].lower()=="-v"):
            verbose = True
       # finds current working dir
    # change working directory
    cwd = os.chdir("Newfolder")
    formats = ('.jpg', '.jpeg')
    # looping through all the files
    # in a current directory
    for file in os.listdir(cwd):
        # If the file format is JPG or JPEG
        if os.path.splitext(file)[1].lower() in formats:
            # print('compressing', file)
            compressMe(file, verbose)
            break
    print("Done")
# Driver code
if __name__ == "__main__":
    main()
```

4.1.2 For Data compression

```
import io
import dataCompression
def user_choice():
    print("Welcome in LZW Compression:\n")
    return 1
option = user_choice()
if option == 1:
    # for data:
    dataCompression.CompressDecompressData()
elif option == 2:
    #For Image
    list = [99,99,126,126,99,99,100,100,99,99,100,100,100,126,126,126]
    print("Image Values: ",list)
    convertedString = []
    realString = ""
    for f in list:
        convertedString.append(chr(f))
    for s in convertedString:
        realString +=str(s)
    print("Converted String: ",realString)
    newFile = open("compressed.txt", "w+")
    newFile.write(realString)
    imageCompression.CompressDecompressData(realString)
else:
    print("Select a valid option: ")
```

2nd file:

```
import io
def compress(uncompressed):
    """Compress a string to a list of output symbols."""
    dict_size = 256
    dictionary = dict((chr(i), i) for i in range(dict_size))
    w = ""
    result = []
    for c in uncompressed:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            dictionary[wc] = dict_size
            dict_size += 1
            w = c
    #Output the new Dictionary.
    newDictionary = list(dictionary.items())
    print("****New Dictionary****")
    print(" ITEM  INDEX ")
    for d in newDictionary[256:]:
        print(d,"\n")
    #Output the code for w.
    if w:
        result.append(dictionary[w])
    file1 = open("compressed.txt","w")
    file1.writelines(str(result))
    file1.close()
    print('compressed file saved at compressed.txt')
    return result
```

3rd File:

```
import io
def select_your_choice():
        print("Select what to do:\n")
         string = str(input("1. Type '1' if you want to Enter data.\n2. Type '2' if you want to Enter a file.\n3.Type exit for close.\n=> "))
        if string is '1':
            return 1
        elif string is '2':
            return 2
        else:
            return 3
def user_input_file():
    filename = input("Enter the file name with path.\n")
    myfile = open(filename)
    info = myfile.readlines()
    return info
def compressing_and_decompressing(string):
    import data_compression
    import decompression
    print("Compressing Data >>>")
    compressed = data_compression.compress(string)
    print ("Compressed output: ",compressed)
    print("\n")
    print("Decompressing Data >>>")
    decompressed = decompression.decompress(compressed)
    print ("Decompressed output: ",decompressed)
    print("\n")
    file1 = open("decompressed.txt","w")
    file1.writelines(str(decompressed))
    file1.close()
```

```
    print ("COMPARE:")
    if string == decompressed:
        print("Successfully Done")
    else:
        print("Not done!")
        print("\n")
def CompressDecompressData():
    print("**********************")
    print(" DATA COMPRESSION ")
    print("**********************")
    choice = select_your_choice()
    if choice is 1:
        string = str(input("Enter the string:\n"))
        print("Real Data: ",string)
        print("\n")
        compressing_and_decompressing(string)
    elif choice is 2:
        string = str(user_input_file())
        print("Real Data: ",string)
        print("\n")
        compressing_and_decompressing(string)
```

4.2 OUTPUT

4.2.1 Image Compression

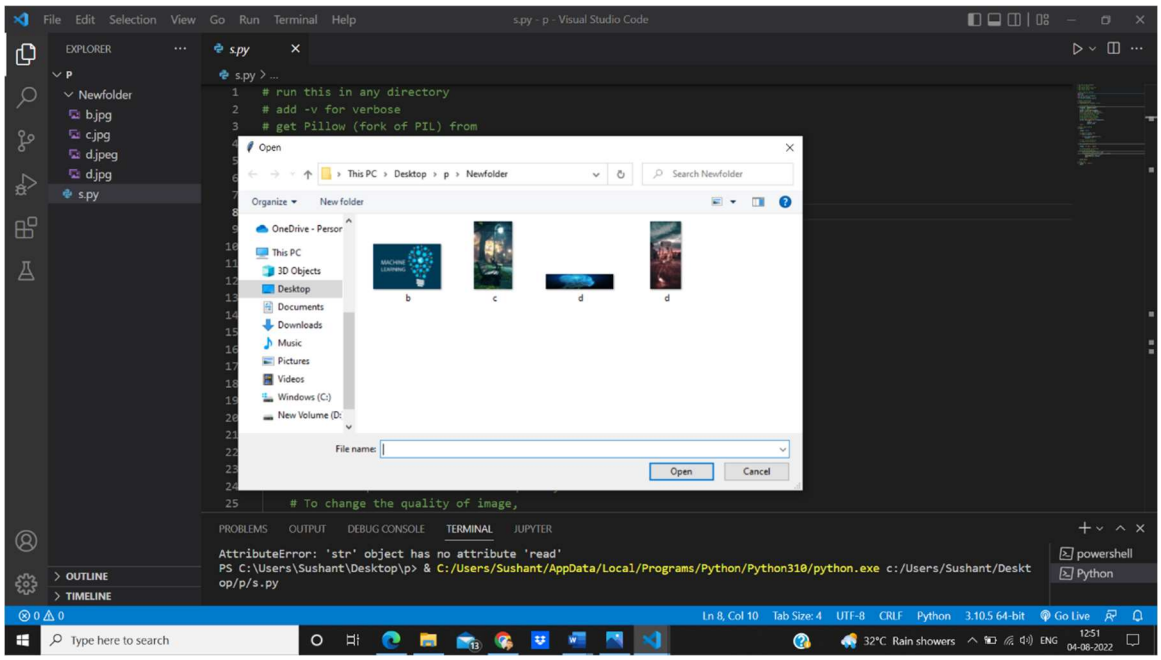

Fig. 4.1. Open the image folder

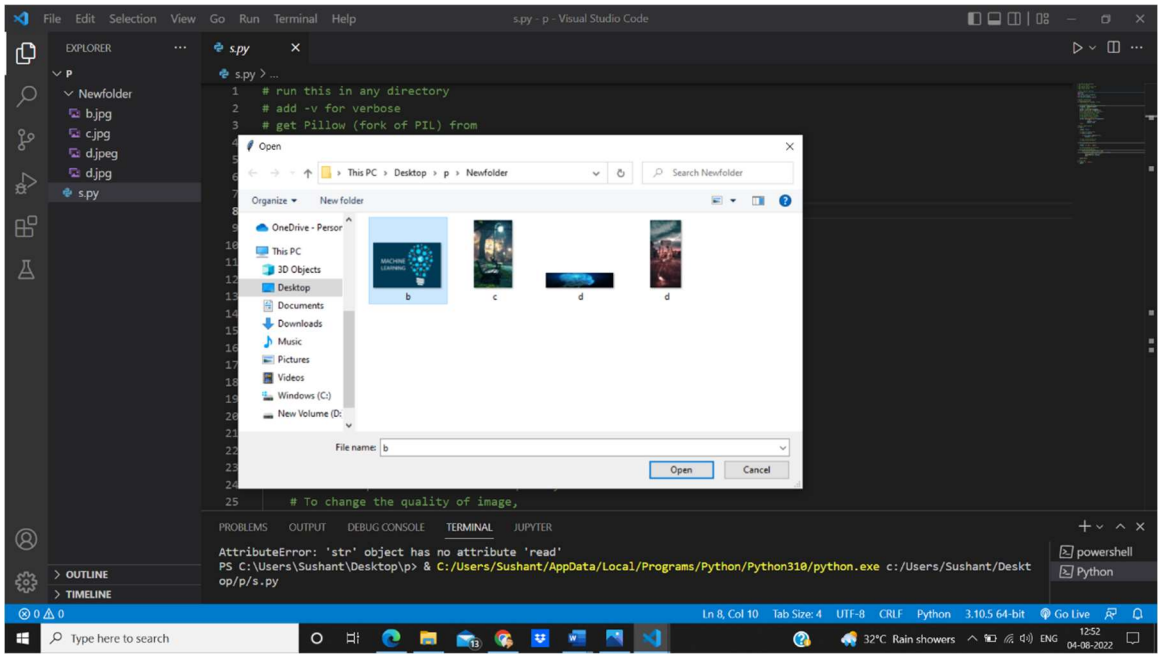

Fig. 4.2. Select the image

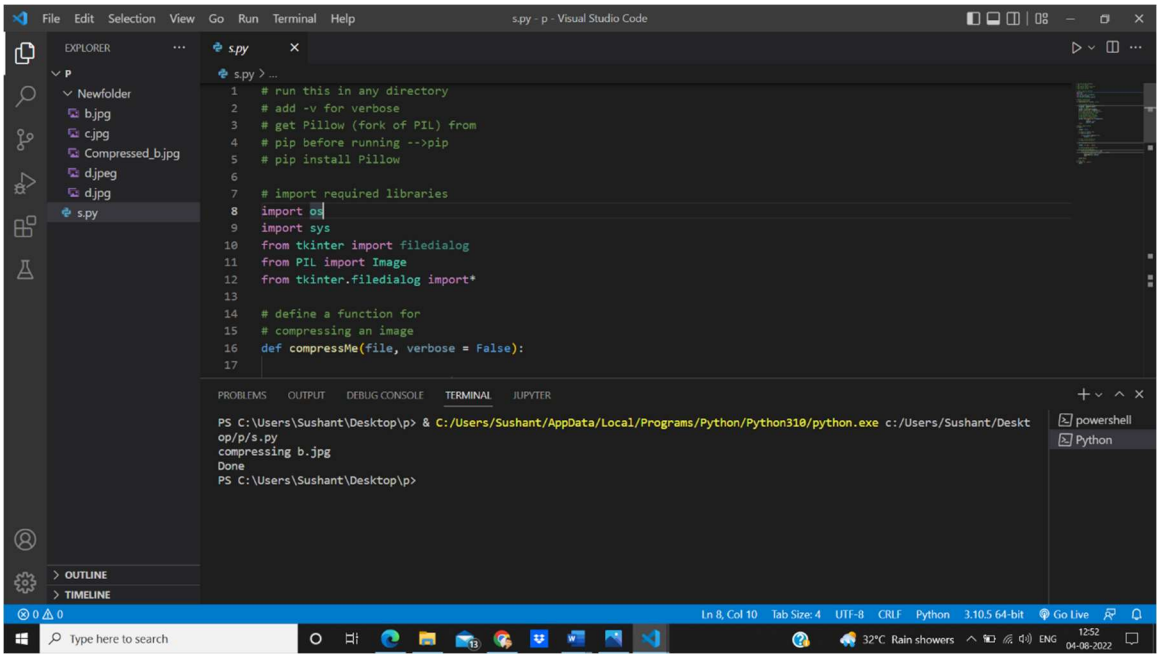

Fig. 4.3. Compressed image

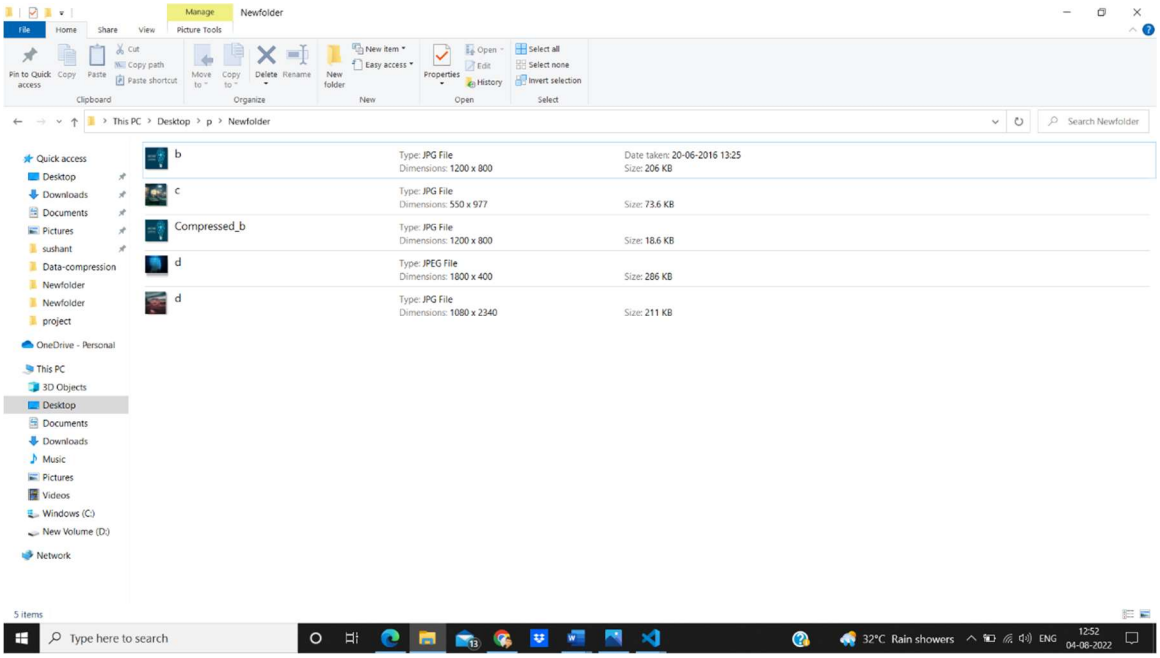

Fig. 4.4. Compressed Image size is 18.6kb

4.2.2 Data Compression

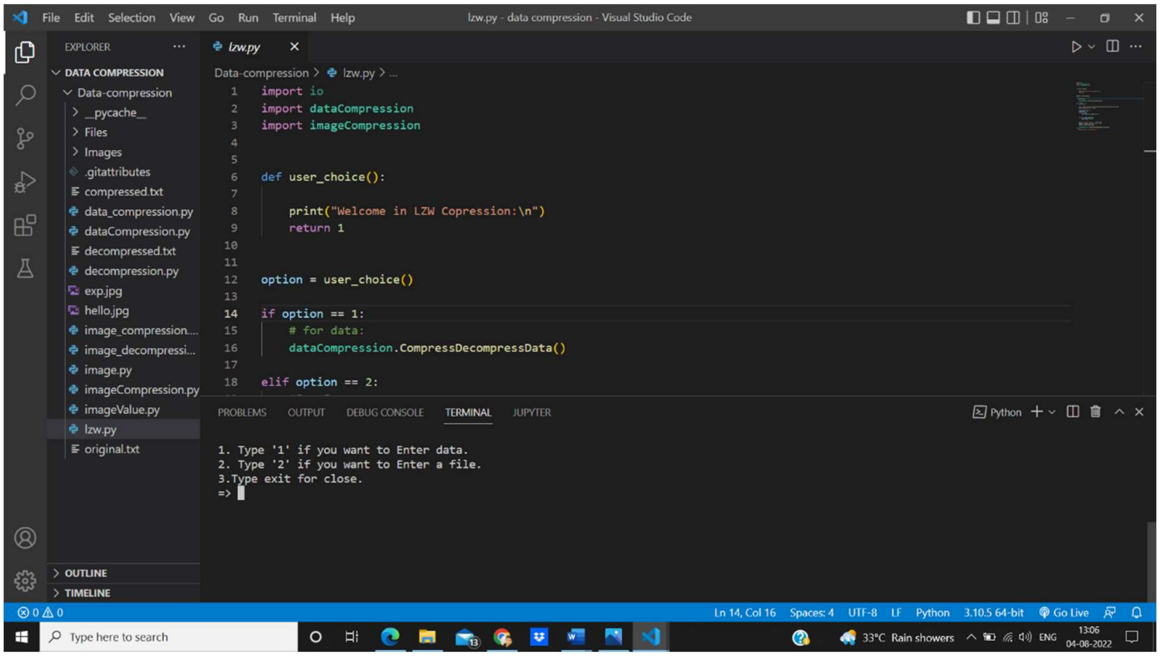

Fig. 4.5. Select data or file

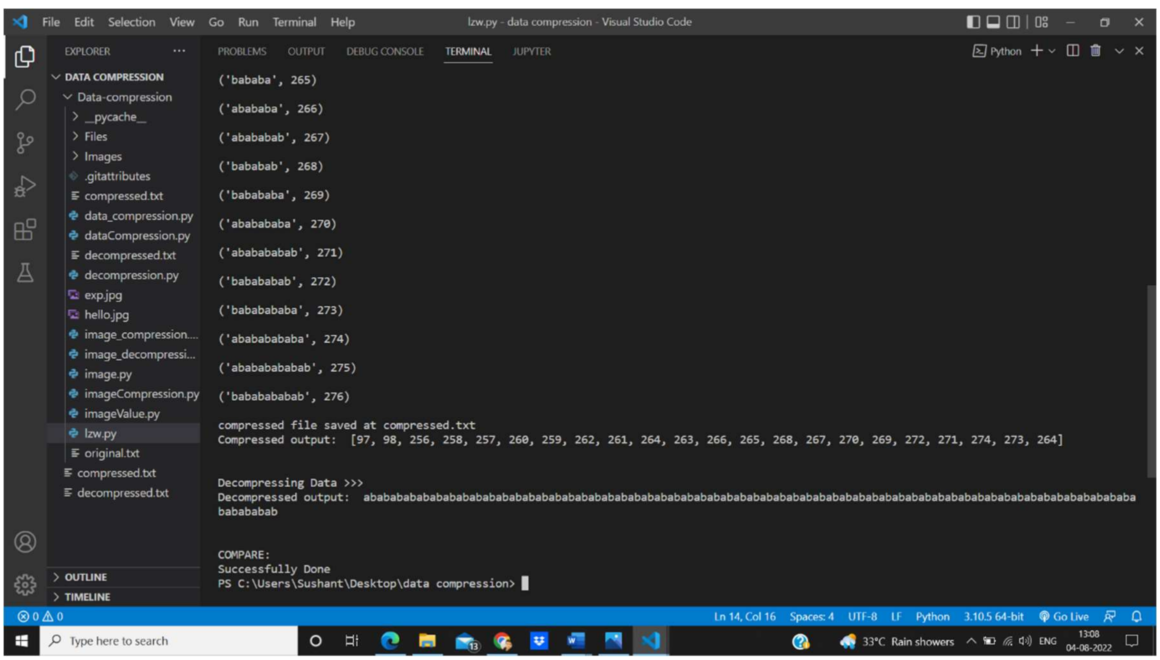

Fig. 4.6. Compressed string and Decompressed string

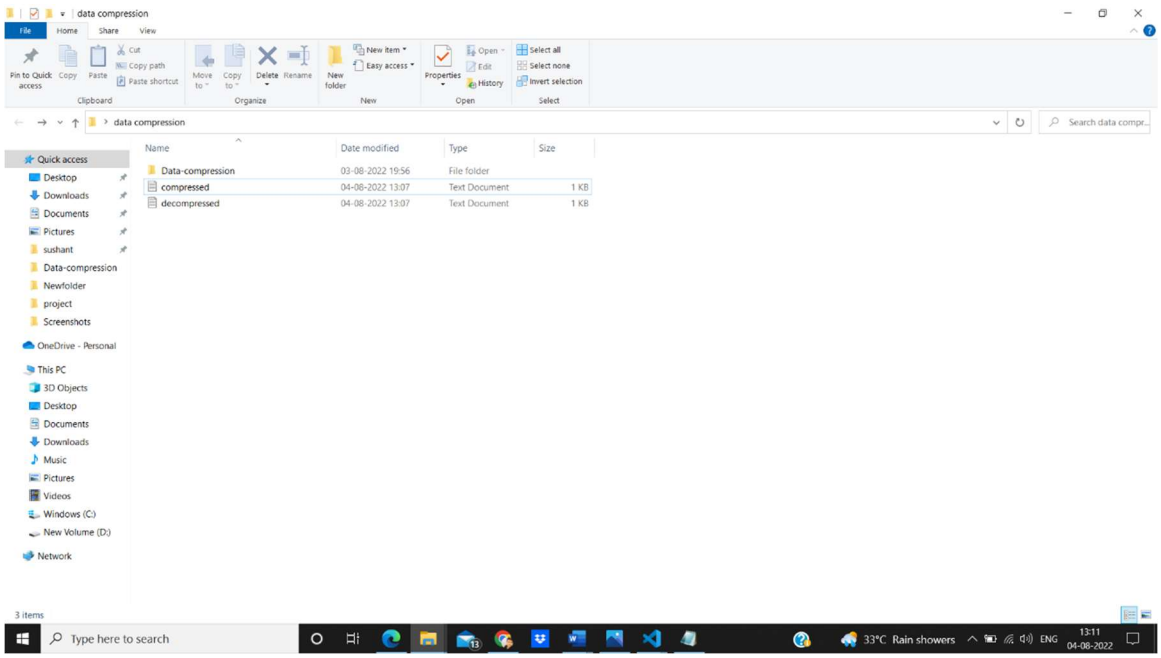

Fig. 4.7. Data Compressed

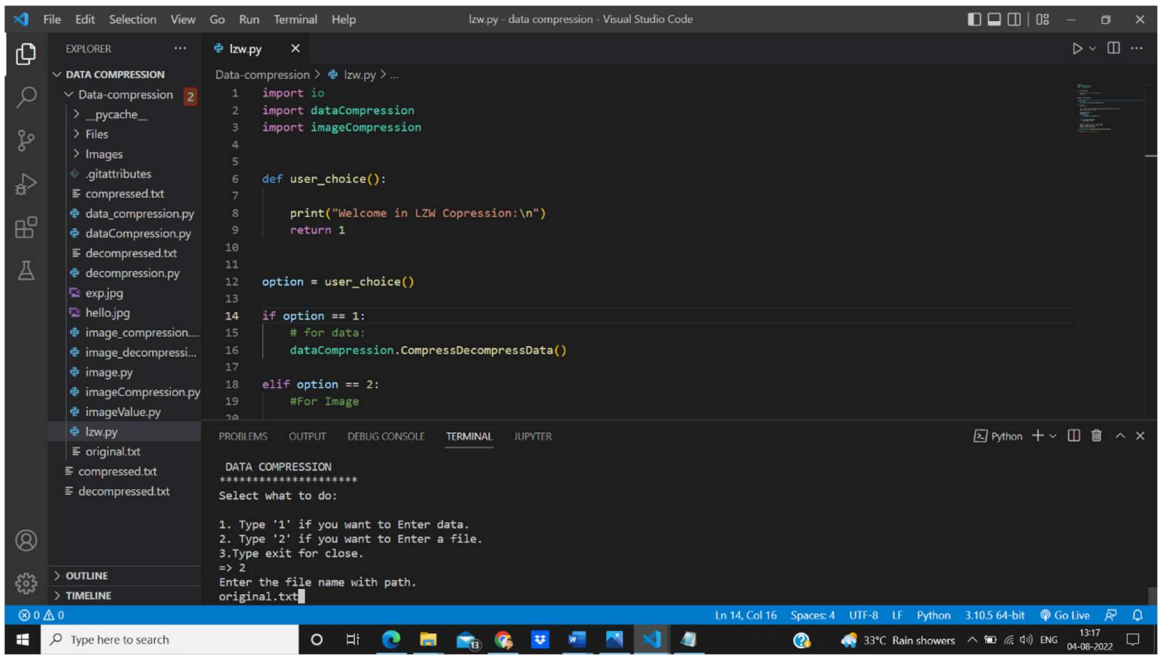

Fig. 4.8. Enter a file path

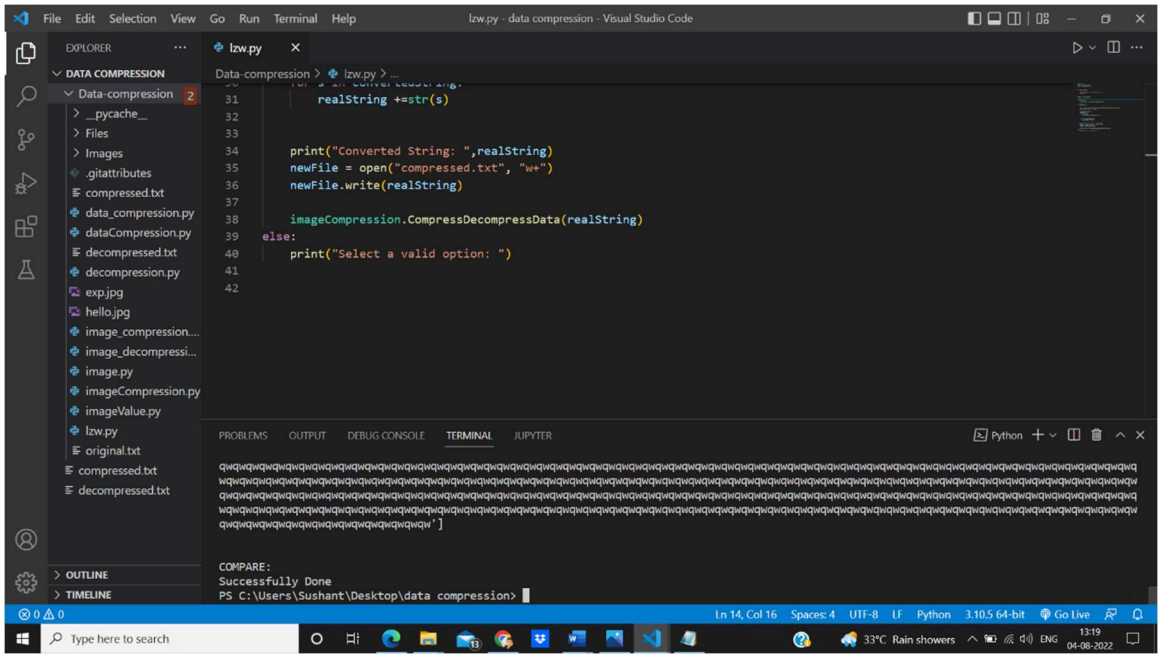

Fig. 4.9. Compressed Txt file

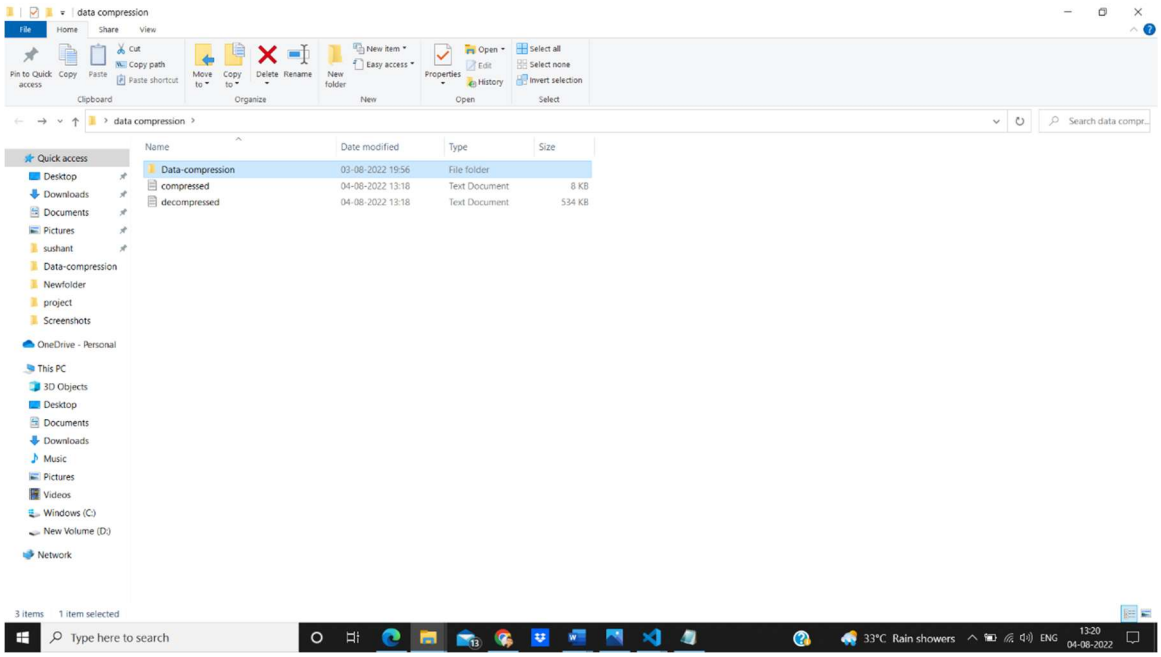

Fig. 4.10. Compressed file 534kb in 8kb

CHAPTER 5

5. CONCLUSION AND FUTURE SCOPE

Various lossless data compression techniques studied in depth from various research papers and authors thus resulting that text data can be compressed more easily and efficiently by various techniques. For sequences with different levels of repetition and for different lengths of repeating sequences, different algorithms work differently. For instance, Shannon Fano algorithm's compression and decompression time for text files is less as compared to Huffman coding. Run-length encoding works around consecutively repeating bits of data but does not consider all the redundancy in the data. Also, Shannon-Fano algorithm is better in terms of performance as compared to RLE but reliability is low since it can generate two different encoding sequence for same data. Hence, output is inconsistent. Huffman encoding requires two passes and hence is slow as compared to LZW which can do the encoding in a single pass. Also, one more fact can be inferred that is the data compression algorithms work on redundancy reduction by encoding redundant data. Employing better algorithms for pattern deduction or redundancy deduction using machine learning can help improve compression ratios even better.

6. REFERENCES

[1] Md. Jayedul Haque Department of Computer Science & Engineering United International University Bangladesh Dhaka https://www.researchgate.net/publication/313779883_Study_on_Data_Compression_Technique

[2] Pooja Raundale, PhD M.C.A. Department (Head of Department) Sardar Patel Institute of Technology Andheri, Mumbai, India https://www.researchgate.net/publication/333862056_Comparative_Study_of_Data_Compression_Techniques?enrichId=rgreq-57037be9c4fd3e651fe0fe7cbfac63ef-XXX&enrichSource=Y292ZXJQYWdlOzMzMzg2MjA1NjtBUzoxMDMwOTQ4MTgzMzQ3MjI0QDE2MjI4MDg2MDkyMjU%3D&el=1_x_2&_esc=publicationCoverPdf

[3] Luluk Anjar Fitriya College Student, Faculty of Electrical Engineering, Telkom University, Bandung, Indonesia https://www.researchgate.net/publication/322557949_A_review_of_data_compression_techniques