西北工業大學

Data Compression Technology and its Application

Development History of Data Compression Coding

Student Name : ABID ALI

Student ID : 2023280099

Student Email : abiduu354@gmail.com

Professor Name: Prof. MEI Shaohui

Professor's Email: meish@nwpu.edu.cn

Date of Submission : 2024-4-10

Development History of Data Compression Coding

Introduction

Data compression is valuable for saving storage space and speeding up data transfer between locations. Having a compression tool ready is crucial for compressing data efficiently between individuals. This method effectively reduces the size of various data types, including text, images, and videos. Two main compression techniques exist: lossy and lossless compression. It's especially important to use lossless compression methods like Huffman and Shannon Fano. Other examples of compression methods include Tunstall, Lempel Ziv Welch, and run-length encoding.

This essay delves into the mechanics of compression strategies, focusing on the predominant approaches utilized in data compression and the historical evolution of this field. One prominent form of compression explored is text compression, where the outcome often results in a compressed file size smaller than the original. Various data compression techniques are discussed, including those pioneered by Shannon, Fano, and Huffman. The primary goal of data compression is to enhance active data density by reducing redundant information in stored or transmitted data. Particularly, in storage and distributed systems, data compression plays a vital role. Furthermore, the essay thoroughly examines information theory concepts concerning the objectives and evaluation of data compression techniques.

Historical Evolution of Data Compression Techniques

1. Beginning: Morse code

Morse code, invented in 1838 for use in telegraphy, is an early example of data compression based on using shorter codewords for letters such as "e" and "t" that are more common in English. Modern work on data compression began in the late 1940s with the development of information theory.

MORSE CODE
was introduced in telegraphy.
Shorter CODE WORDS were used for common letters like 'e' and 't'.
1838

Fig 2-1: Morse code

2. Shannon-Fano

In 1949, Claude Shannon and Robert Fano invented Shannon-Fano coding. Their algorithm assigns codes to symbols in a given block of data based on the probability of the symbol occuring. The probability is of a symbol occuring is inversely proportional to the length of the code, resulting in a shorter way to represent the data.

SHANNON-FANO
The Shannon-Fano coding algorithm was introduced by CLAUDE SHANNON and ROBERT FANO.
1949
It assigns codes to symbols in a given block of data based on the PROBABILITY of the symbol occurring.

Fig 2-2: Shannon-Fano

3. Early Day of Data compression: Huffman Coding

Two years later, David Huffman was studying information theory at MIT and had a class with Robert Fano. Fano gave the class the choice of writing a term paper or taking a final exam. Huffman chose the term paper, which was to be on finding the most efficient method of binary coding. After working for months and failing to come up with anything, Huffman was about to throw away all his work and start studying for the final exam in lieu of the paper. It was at that point that he had an epiphany, figuring out a very similar yet more efficient technique to Shannon-Fano coding. The key difference between Shannon-Fano coding and Huffman coding is that in the former the probability tree is built bottom-up, creating a suboptimal result, and in the latter, it is built top-down.

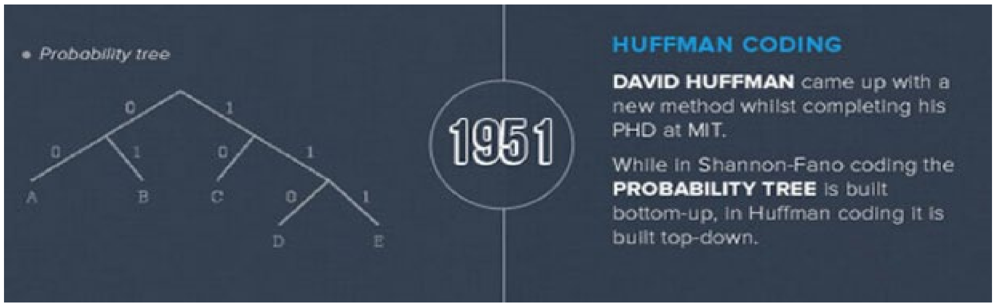
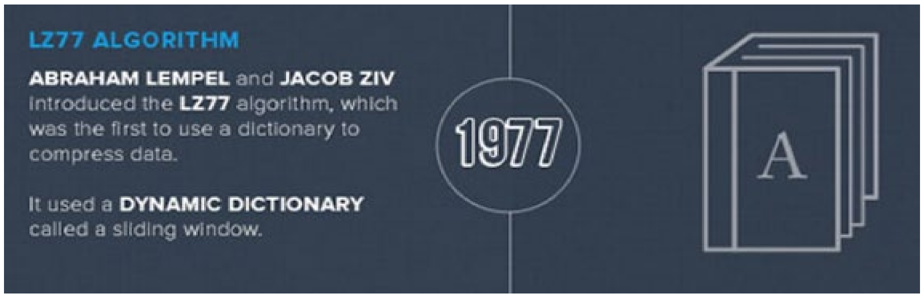

Probability tree
0
1
0
1
0
1
0
1
A
B
C
D
E
1951
HUFFMAN CODING
DAVID HUFFMAN came up with a new method whilst completing his PHD at MIT.
While in Shannon-Fano coding the PROBABILITY TREE is built bottom-up, in Huffman coding it is built top-down.

Fig 2-3: Shannon-Fano

4. LZ77 Algorithm

In 1977, Abraham Lempel and Jacob Ziv published their groundbreaking LZ77 algorithm, the first algorithm to use a dictionary to compress data. More specifically, LZ77 used a dynamic dictionary oftentimes called a sliding window.



LZ77 ALGORITHM

ABRAHAM LEMPEL and **JACOB ZIV** introduced the **LZ77** algorithm, which was the first to use a dictionary to compress data.

It used a **DYNAMIC DICTIONARY** called a sliding window.

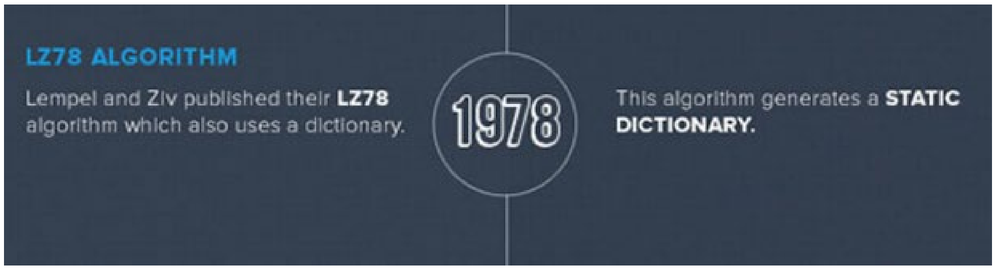
1977

A

Fig 2-4: LZ77 Algorithm

5. LZ78 Algorithm

In 1978, the same duo published their LZ78 algorithm which also uses a dictionary; unlike LZ77, this algorithm parses the input data and generates a static dictionary rather than generating it dynamically.



LZ78 ALGORITHM

Lempel and Ziv published their **LZ78** algorithm which also uses a dictionary.

1978

This algorithm generates a **STATIC DICTIONARY.**

Fig 2-5: LZ77 Algorithm

6. LZW Algorithm

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement and has the potential for very high throughput in hardware implementations.It is the algorithm of the Unix file compression utility compress and is used in the GIF image format.

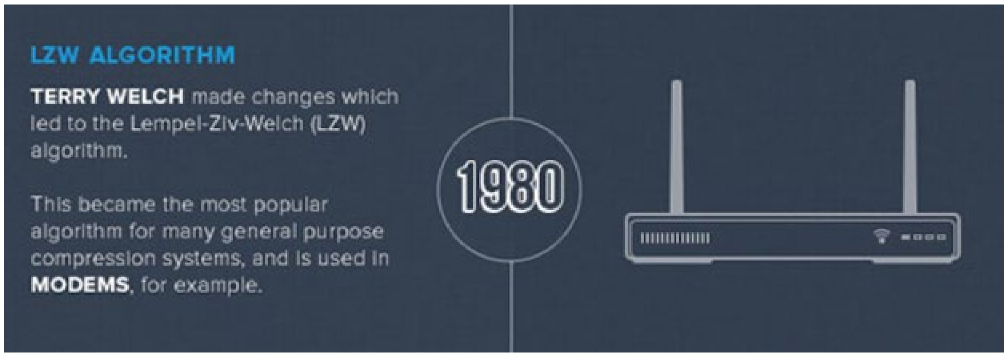

LZW ALGORITHM
TERRY WELCH made changes which led to the Lempel-Ziv-Welch (LZW) algorithm.
This became the most popular algorithm for many general purpose compression systems, and is used in MODEMS, for example.
1980

Fig 2-6: LZW Algorithm

7. MP3

MP3 employs information theory to compress audio files efficiently, utilizing perceptual coding and psychoacoustic models to discard less perceptible data. It employs lossy compression algorithms, reducing file size while maintaining perceptual quality through dynamic bit allocation. The format optimizes bit rate allocation based on human auditory perception and entropy coding techniques like Huffman and arithmetic coding. MP3's design hinges on principles of information theory, understanding human auditory perception to prioritize data allocation. By leveraging these principles, MP3 achieves high compression ratios while preserving perceptually relevant information, making it a widely used audio format.

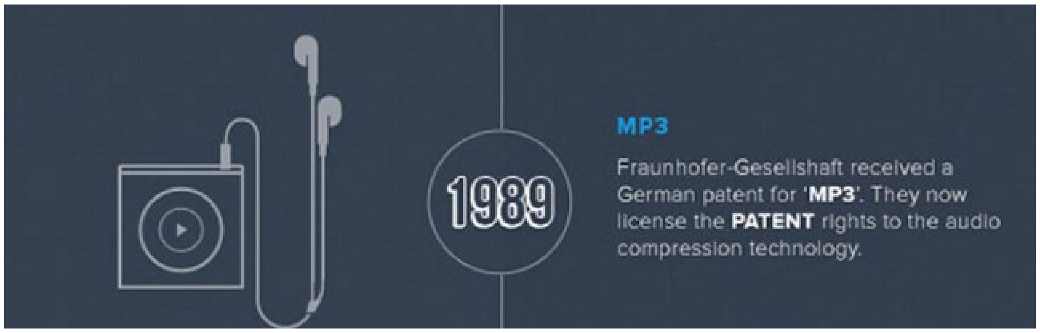

1989
MP3
Fraunhofer-Gesellshaft received a German patent for 'MP3'. They now license the PATENT rights to the audio compression technology.

Fig 2-7: MP3

8. JPEG

JPEG utilizes entropy coding, such as Huffman coding, to efficiently represent image data statistically. It employs quantization to reduce color and frequency information, discarding less perceptually significant details. The Discrete Cosine Transform (DCT) is used to concentrate image information into fewer coefficients for compression. Psycho-visual models remove imperceptible details, optimizing bit allocation. Variable rate coding strategies allocate bits effectively across image components. Employing lossy compression, JPEG balances compression ratio and visual fidelity based on information theory. Efficiency optimization in JPEG exploits redundancies and statistical properties for optimal compression performance.

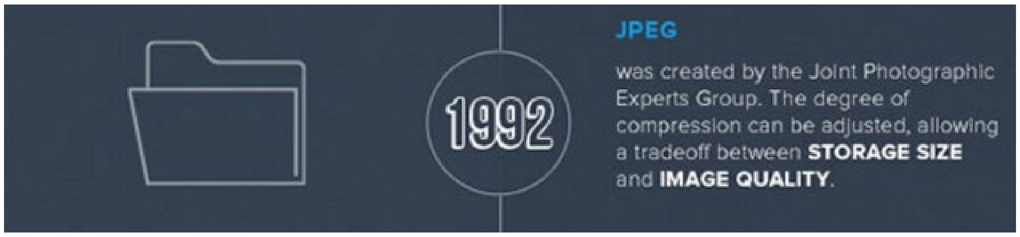

1992
JPEG
was created by the Joint Photographic Experts Group. The degree of compression can be adjusted, allowing a tradeoff between STORAGE SIZE and IMAGE QUALITY.

Fig 1-8: JPEG

9. Compressed Folders

Compressed folders, like zip files, use entropy coding, such as Deflate, to represent symbols with variable-length codes, reducing redundancy. They employ dictionary-based compression, such as LZ77 and LZ78, to find repetitive patterns and replace them with shorter codes. The compression aims to be lossless, preserving all original data during compression and decompression. Zip archives optimize file structure to minimize redundancy and maximize compression efficiency. Variable length encoding schemes are used to represent symbols efficiently. Statistical analysis identifies patterns and redundancies for effective compression. Compressed folders reduce entropy by removing redundancy and exploiting patterns, aligning with information theory principles.

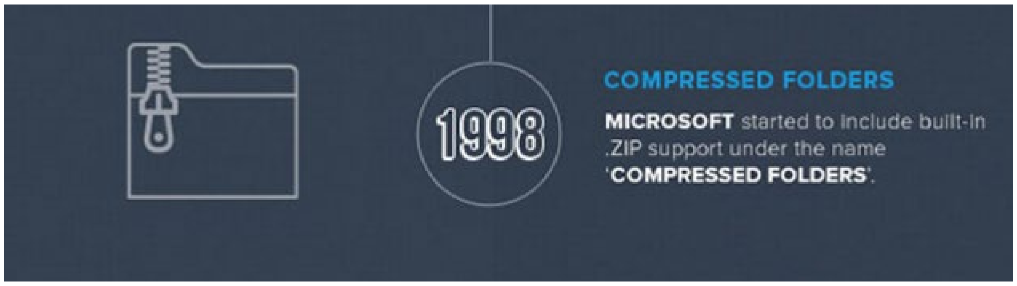

1998
COMPRESSED FOLDERS
MICROSOFT started to include built-in .ZIP support under the name 'COMPRESSED FOLDERS'.

Fig 2-9: Compressed Folders

Analysis and Comparison

Compression is the process of turning a set of data into a code so that less space is needed to store and send the data. By compressing data, you can save both time and space in your memory. There are many effective ways to use compression algorithms, such as the Huffman, Lempel, Ziv Welch, Run Length Encoding, Tunstall, and Shannon Fano methods. Figure 1 shows the way in which data is compressed.

When the data is not compressed, the uncompressed data is continued and processed using a lossless compression algorithm, and the compressed data has a smaller size than the file before it is compressed. Compression is the process of shrinking a file from its original size to a smaller size. A compression test will be done to facilitate the procedure. Transmission of a large file with several attachments in characters. The workings of compression are identified by looking for recurring patterns in data and replacing them with a certain pattern sign.

Here is an explanation of the data compression application.

1) Compression of audio
2) Text Compression
3) Video Compression
4) Image Compression

1. Compression of Audio

Audio compression is a type of data compression that can be used to reduce the size of an audio or video file.
1) MP3 and Vorbis are both overflow formats.
2) FLAC is a format for music that can hold an unlimited number of files.
Compression happens

Some problems with audio compression:
1) Sound recording technology changes quickly and in many ways.
2) A sound sample's value changes very quickly.
There are a lot of high-quality audio codecs. The following uses may be given more weight:
1) Rates of packing and unpacking
2) The amount of pressure
3) Help with software and hardware.

2. Text Compression

When a method for compressing text without losing information is discovered, the original text can be restored after an operation is performed on it. Sayood (2001) demonstrates how to extract the correct data from a decompressed file. Arithmetic encoding is a method of compression in which no data is lost.

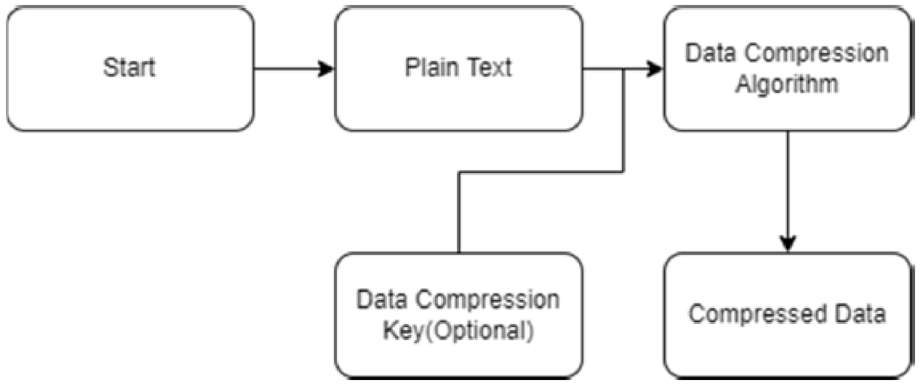


Fig 3-1: Data Compression Process Flow

3. Video Compression

A method for making a video file smaller is called "video compression. "Video compression is a way to cut down on the amount of data needed to do a job. Image compression space and image compression time motion depict a digital video picture.

$$Compression\ ratio = \frac{x}{original\ file} * 100$$

Fig 3-2: Compression Ratio

Run Length Encoding

The Run Length Encoding (RLE) algorithm is one way to compress data so that the size of the output is smaller than the size of the original data. As an example, this time shows the cost and benefit of data. RLE (Run Length Encoding) is the easiest type of data encoding that doesn't lose any information. It is a method for compressing a group of files that all have the same format.

Each number in the series will be written down as a piece of data. The name of this algorithm is very good when there are a lot of data points with the same value. Sequences are made up of things like icon files, line drawings, and animations. This Normal data doesn't work well with the method. The Run Length Encoding Algorithm is shown in Figure

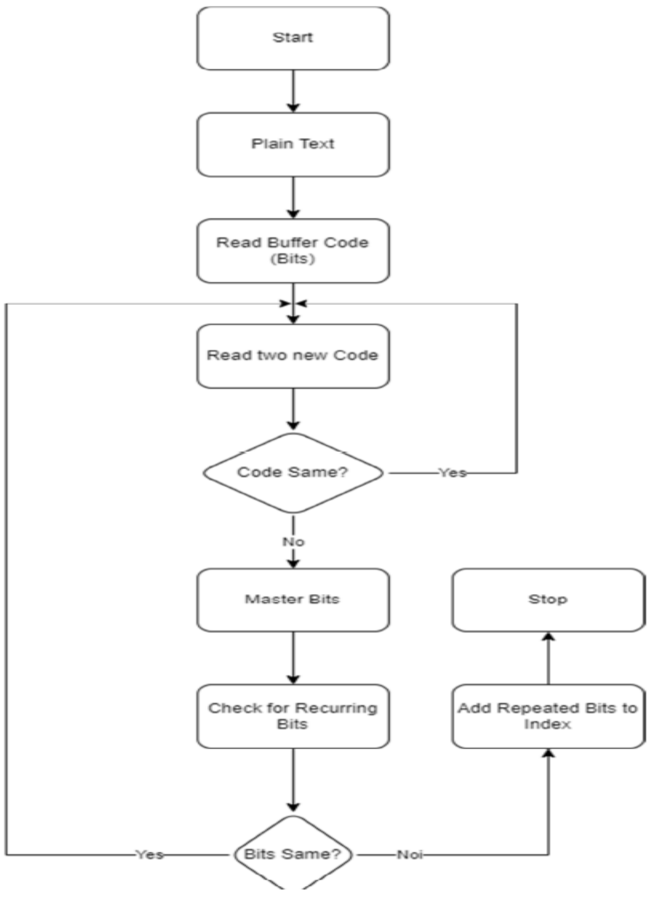


Fig 4-1: RLE Algorithm

Shannon Fano Algorithm

The ratio of data before compression to data after compression is the amount of compression. We know that encryption can make data compression better. Unfortunately, most standard Shannon Fano codes aren't very long, and this essay discussed two algorithms. To make the Shannon-Fano code shorter, In some applications, the improved Shannon-Fano coding method is used.

Huffman Technique

This Huffman compression technique has ways to do things. It is compression without loss. Lossless compression is a type of data compression that doesn't change the original data while making it smaller. Huffman's method is based on the fact that 8 bits are usually used to represent each ASCII character.

So, if there is a row of the letters "ABCD" in a file, there are 40 bits in the file, or 5 bytes. If every character is given a name, If we have a code like "A = 0," "B = 10," "C = 111," all we need is a 10 bit file (0010111110). Pay attention to the fact that codes must be unique; otherwise,they can’t be copied. built on top of other codes. We can use the Huffman method to compress and make codes that describe codes that must be unique. The number of bits is used to shorten the characters that show up most often.

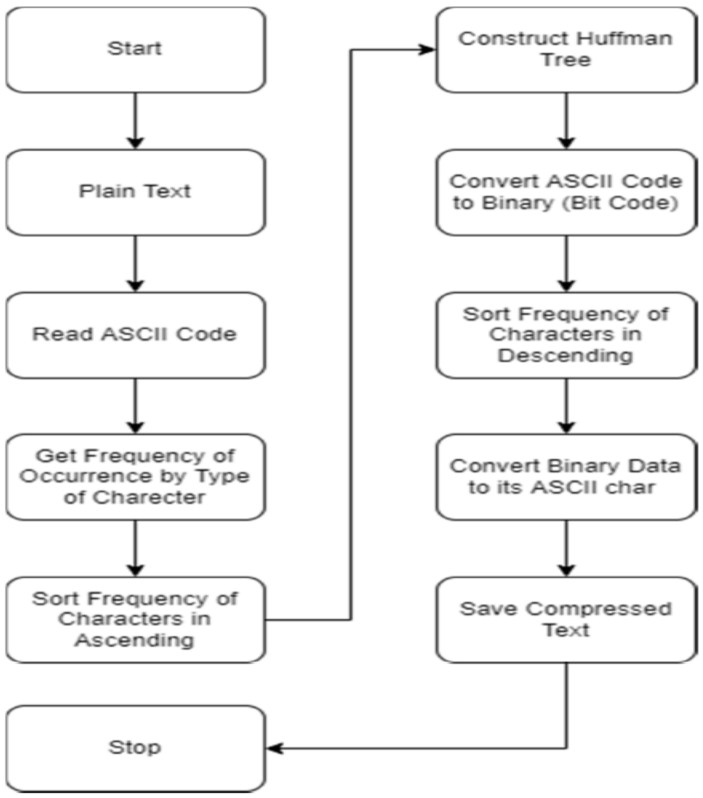

Start
Plain Text
Read ASCII Code
Get Frequency of Occurrence by Type of Charecter
Sort Frequency of Characters in Ascending
Construct Huffman Tree
Convert ASCII Code to Binary (Bit Code)
Sort Frequency of Characters in Descending
Convert Binary Data to its ASCII char
Save Compressed Text
Stop

Fig 5-1: Huffman Technique

Lempel Ziv Welch

LZW algorithm is a way to compress data without losing any of it. It does this by using a dictionary. LZW compression makes a dictionary as part of the compression process. ITZW There are a lot of ways to compress data. Before compression, the output must be smaller than the original file. The formula is a It has been updated to the Unicode standard and can be used very easily for any Bangla text compression. The idea behind this method is to start with a new character to make a new dictionary. In the LZW method, a variable word width dictionary is used to balance the compression and decompression of a file

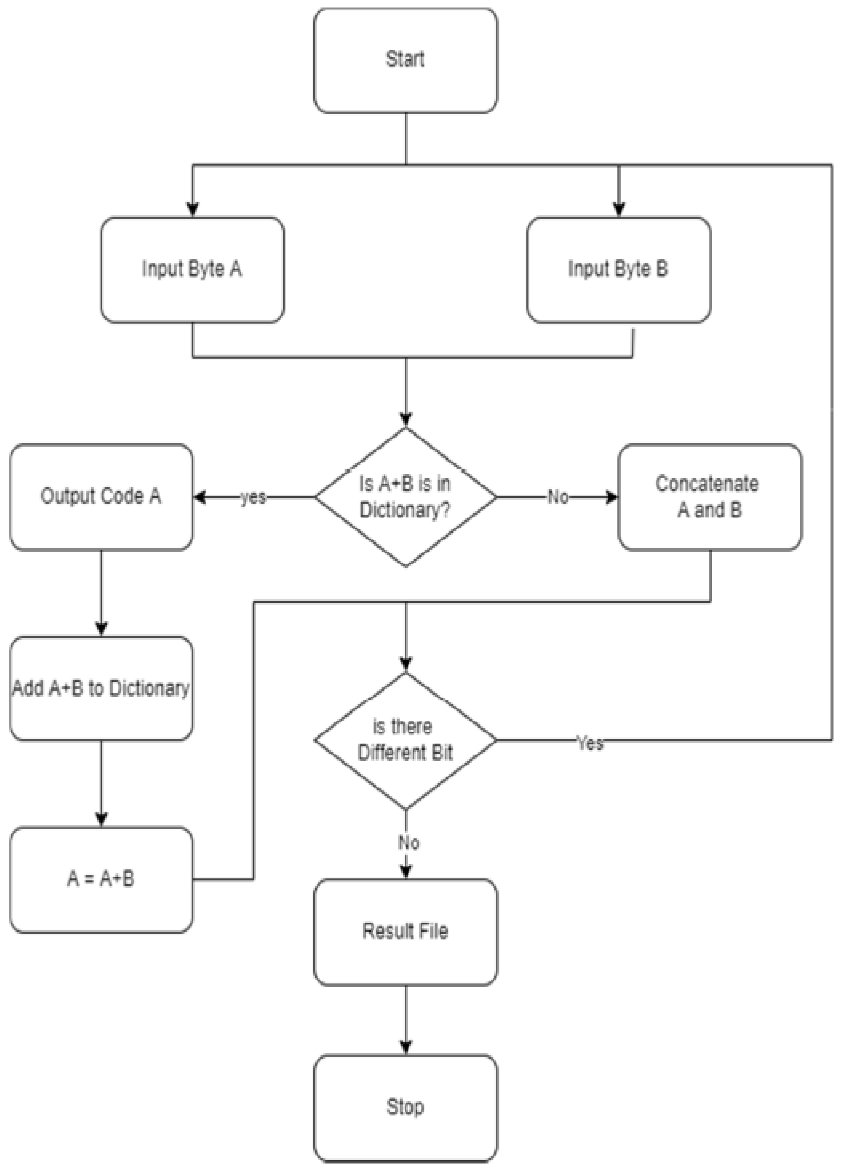


Fig 6-1: LZW Algorithm

Application

Byte-Efficient

“ByteEfficient” is a flexible and convenient Python companion, designed to reduce the size of data files using gzip compression, especially useful for scenarios involving large amounts of data that need to be stored or transmitted efficiently.

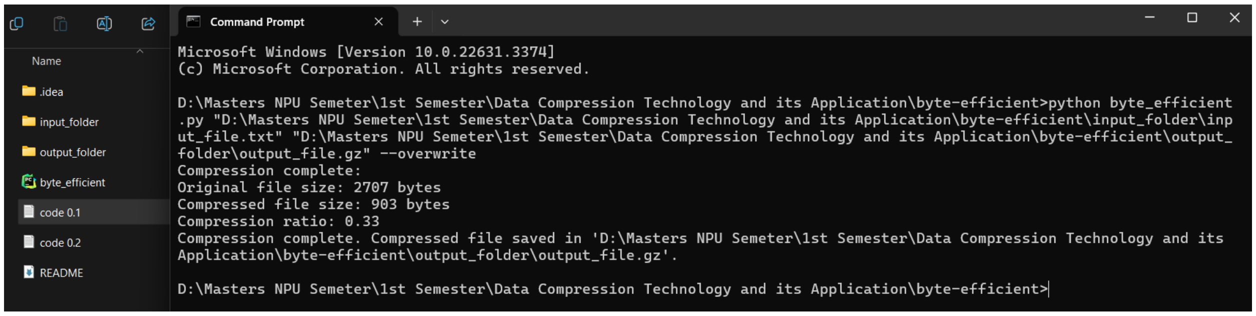
- Supports a wide range of file formats, such as text, CSV, JSON, and binary files, without compromising data integrity.
- Provides detailed information about the compression process, including original and compressed file sizes, and compression ratio.
- Accepts input and output file paths as command-line arguments for flexibility and customization.

Prerequisites & Installation

- Ensure you have Python 3.x installed on your system to run the script.
- Basic familiarity with running Python scripts from the command line.
- Run script file `python byte_efficient.py -input_folder -output_folder --overwrite` directly.

How to Run the program in Windows

- Python run command: python byte_efficient.py
- input folder : "D:\..\..\..\input_folder\input_file.txt"
- output folder : "D:\..\..\..\output_folder\ioutput_file.gz"



```
Microsoft Windows [Version 10.0.22631.3374]
(c) Microsoft Corporation. All rights reserved.

D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its Application\byte-efficient>python byte_efficient
.py "D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its Application\byte-efficient\input_folder\inp
ut_file.txt" "D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its Application\byte-efficient\output_
folder\output_file.gz" --overwrite
Compression complete:
Original file size: 2707 bytes
Compressed file size: 903 bytes
Compression ratio: 0.33
Compression complete. Compressed file saved in 'D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its
Application\byte-efficient\output_folder\output_file.gz'.

D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its Application\byte-efficient>
```

Fig 7-1: Running byte_efficient

Original file size was 2707 bytes then compressed to 903 bytes. The compression ratio was 0.33

Huffman Coding

Introduction

Huffman coding is one such algorithms that has become increasingly popular due to its efficiency in compressing large amounts of data into smaller sizes with minimal loss or distortion. As a data compression algorithm, Huffman coding is based on the frequency of occurrence of each character in the data. It is a lossless compression technique, which means that the original data can be exactly reconstructed from the compressed data. Huffman coding is widely used in various applications, such as image and video compression, text compression, and communication systems.

Encoding

Huffman coding creates a binary tree where each leaf node represents a character and the path from the root to the leaf represents the code for that character. The algorithm analyzes each symbol in a stream, assigns a code based on frequency count, and constructs a binary tree where frequently occurring characters have shorter codes. The data is encoded by traversing the tree, producing the code for each character. Huffman coding minimizes the average number of bytes needed to represent characters, resulting in smaller compressed files and significant storage and transmission savings.

Decoding

The decoding process in Huffman coding is a simple yet crucial step that enables the reconstruction of the original data from the compressed data. The decoding process requires the same Huffman tree that was used for encoding the data, so the tree must be transmitted or stored along with the encoded data to be able to decode it. The decoding process begins by navigating the Huffman tree using the code for each character to find the corresponding character in the data, starting from the root of the Huffman tree. Each code in the encoded data is then decoded by following the path of 0s and 1s that make up the code, and once a leaf node is reached, the character represented by that node is the decoded character for the corresponding code. This process is repeated until all the codes have been decoded, and the original data is reconstructed.

Explanation

This project presents an improved version of the traditional Huffman coding algorithm, by using dynamic trees that can be re-coded if the frequency of a symbol changes. This allows the algorithm to adapt to changing frequencies of the symbols in the input data, making it more efficient for data streams where the frequencies of the symbols change over time, such as in text as it is being typed. The proposed forward-looking algorithm is expected to be useful in various applications where data is streaming, such as text-based communications and multimedia streaming.

Instructions

The novel implementation is in the Huffman class, find in the file with the same name.

Two separate programs were written to access the aforementioned class.

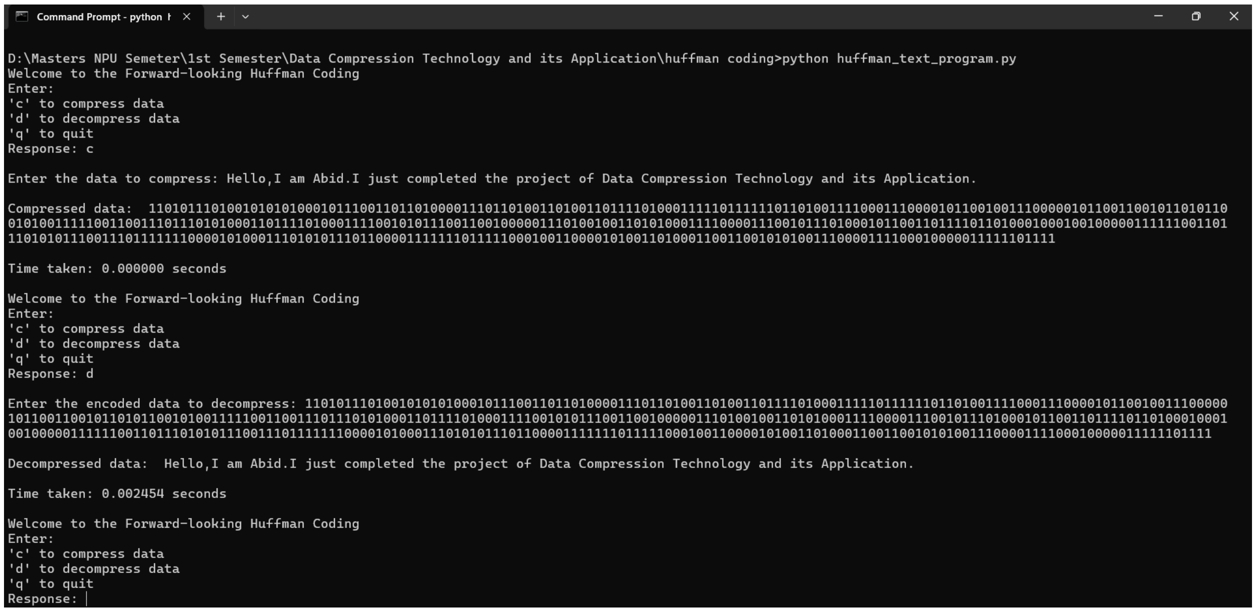
- The `huffman_text_program` allows users to compress and decompress strings or texts. This program performs the various operations and outputs the results in the terminal.

- The `huffman_files_program` allows users to compress and decompress files. Currently, only files containing texts are supported. The results are outputted in a text file, with a name determined by the user.

The user can use either of one of the program:

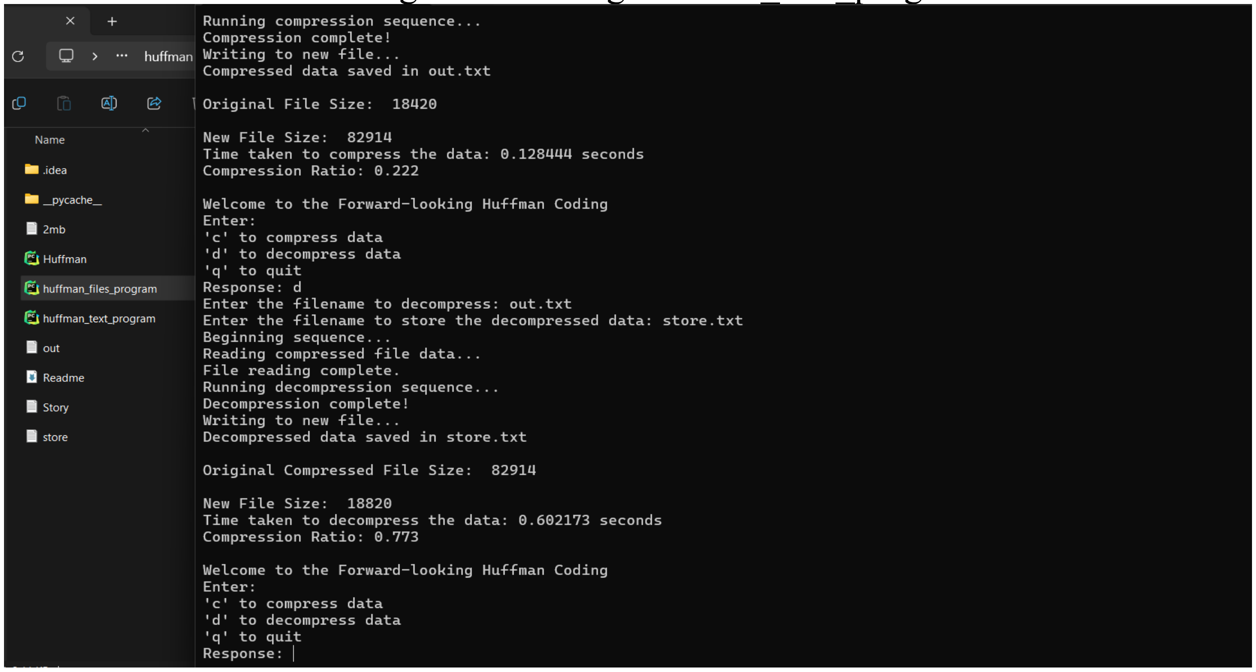
1. Running the 2 above program (can use 2mb.txt and story.txt files for testing if needed)

2. Then following the prompts that can be seen in screen



```
D:\Masters NPU Semeter\1st Semester\Data Compression Technology and its Application\huffman coding>python huffman_text_program.py
Welcome to the Forward-looking Huffman Coding
Enter:
'c' to compress data
'd' to decompress data
'q' to quit
Response: c

Enter the data to compress: Hello,I am Abid.I just completed the project of Data Compression Technology and its Application.

Compressed data:  1101011101001010101000101110011011010000111011010011010011011110100011111011111110110100111100011100001011001001110000010110011001011010110
0101001111100110011101110101000110111101000111100101011100110010000011101001001101010001111000011100101110100010110011011110110100010001001000001111111001101
1101010111001110111111100001010001110101011101100001111111011111000100110000101001101000110011001010100111000011110001000001111101111

Time taken: 0.000000 seconds

Welcome to the Forward-looking Huffman Coding
Enter:
'c' to compress data
'd' to decompress data
'q' to quit
Response: d

Enter the encoded data to decompress: 110101110100101010100010111001101101000011101101001101001101111010001111101111111011010011110001110000101100100111000000
1011001100101101011001010011111001100111011101010001101111010001111001010111001100100000111010010011010100011110000111001011101000101100110111101101000100010001
0010000011111100110111010101110011101111111000010100011101010111011000011111110111110001001100001010011010001100110010101001110000111100010000011111101111

Decompressed data:  Hello,I am Abid.I just completed the project of Data Compression Technology and its Application.

Time taken: 0.002454 seconds

Welcome to the Forward-looking Huffman Coding
Enter:
'c' to compress data
'd' to decompress data
'q' to quit
Response: 
```

Fig 8-1: Running huffman_text_program



```
Running compression sequence...
Compression complete!
Writing to new file...
Compressed data saved in out.txt

Original File Size:  18420

New File Size:  82914
Time taken to compress the data: 0.128444 seconds
Compression Ratio: 0.222

Welcome to the Forward-looking Huffman Coding
Enter:
'c' to compress data
'd' to decompress data
'q' to quit
Response: d
Enter the filename to decompress: out.txt
Enter the filename to store the decompressed data: store.txt
Beginning sequence...
Reading compressed file data...
File reading complete.
Running decompression sequence...
Decompression complete!
Writing to new file...
Decompressed data saved in store.txt

Original Compressed File Size:  82914

New File Size:  18820
Time taken to decompress the data: 0.602173 seconds
Compression Ratio: 0.773

Welcome to the Forward-looking Huffman Coding
Enter:
'c' to compress data
'd' to decompress data
'q' to quit
Response: 
```

Fig 8-2: Running huffman_files_program

Conclusion

The compression method can be used to make the file sizes as small as possible. In order to save space on a computer's hard drive, large amounts of data are shrunk down. It is possible to utilize data compression. using different compression algorithms on text, image, and video data. When it comes to compression, different techniques have pros and cons along with that we discussed the history of Data Compression Coding

Acknowledgement

I would like to express my sincere gratitude to the professor for his invaluable guidance and mentorship throughout the Data Compression Technology and its Application course. Deep knowledge, enthusiasm, and dedication significantly enriched my understanding of this complex subject.

Finally, I acknowledge using online resources for my report, whose contributions were instrumental in the successful completion of our Data Compression Technology and its Application report.

Thank you to everyone involved for their unwavering support and encouragement throughout this learning journey.

Reference

[1] "Data Compression Using Shannon Fano Algorithm Implemented By VHDL," by Mahesh Vaidya, Ekjot Singh Walia, and Aditya Gupta, IEEE International Conference on Advances in Engineering Technology Research, August 1–02,2014.

[2] "Bangla Text Compression Based on Modified Lempel-Ziv-Welch Algorithm," Linkon Barua, Pranab Kumar Dhar, Lamia Alam, and Isao Echizen, International Conference on Electrical, Computer, and Communication Engineering (ECCE), Bangladesh, February 16–8, 2017.

[3] "Using Improved Shannon-Fano-Elias Codes Data Encryption," Xiaoyu Ruan and Rajendra Katti, Proceedings of the ISIT Conference, North Dakota State University Fargo,North Dakota, July 9–14, 2006.

[4] "Data compression with Huffman and other algorithms," Harry Fernando, ITB, Bandung.

[5] International Journal of Advanced Research in Computer Science and Software Engineering, India, July 2015, Manjeet Kaur, "Lossless Text Data Compression Algorithm Using Modified Huffman Algorithm."