Data compression tool using Huffman Algorithm

Vern Cervantes   
*College of Computer and Information Science*   
*Polytechnic University of the Philippines*Manila, Philippines  
vacervantes@iskolarngbayan.pup.edu.ph

Jerico Glico   
*College of Computer and Information Science*   
*Polytechnic University of the Philippines*Manila, Philippines  
jtglico@iskolarngbayan.pup.edu.ph Freanne Lei Concepcion   
*College of Computer and Information Science*   
*Polytechnic University of the Philippines*Manila, Philippines  
flfconcepcion@iskolarngbayan.pup.edu.ph

Jirro Dave Reoloso   
*College of Computer and Information Science*   
*Polytechnic University of the Philippines*Manila, Philippines  
jdcreoloso@iskolarngbayan.pup.edu.ph Aaron Bien De Guzman   
*College of Computer and Information Science*   
*Polytechnic University of the Philippines*Manila, Philippines  
aaronbiensdeguzman@iskolarngbayan.pup.edu.ph

# **Introduction And Background**

Due to the exponential growth of digital information, it is becoming crucial for data to be efficiently stored and transmitted. File storage and even servers have its limited capacity to store data. To address the aforementioned issue, from a software perspective, compression is a solution, specifically lossless compression. Data compression is the process of reducing the amount of data needed of storage or transmission of a given piece of information typically by use of encoding techniques, in order to reduce the consumption of critical resources, such as storage and bandwidth, or even for security reasons. Compression serves both to save storage space, and to save transmission time.

One of the common data that is being used and utilized is text. Being a heavily used type of data, text compression has important application in the areas of data transmission and data storage because it requires efficient ways to store and transmit different types off data such as text, image, audio, and video, and hence reduce execution time and memory size [1]. Text compression are categorized as lossy or lossless compression. Lossy compression compresses data but with loss of some data or quality is reduced while lossless compression compresses but maintains the quality of data.

This project focuses on utilizing lossless compression of text using Hoffman algorithm. The aim of this compression program is to produce a new file, as short as possible, containing the compressed version of the same text, without any loss of information. Using lossless compression, it allows the original data to be entirely reconstructed from the compressed data. On the contrary, by using the Huffman algorithm, text compression process is done by using the principle of the encoding; each character is encoded with a series of several bits to produce a more optimal result. Aside from creating a data compression tool, this study aims to investigate the effectiveness of the Huffman algorithm in the compression of text.

# Related Works

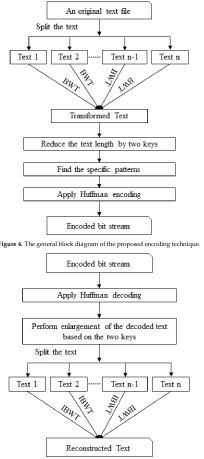
****

Figure 1. Diagram used on this project proposal based on the first related work (Burrows Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding)

**Burrows-Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding**

Rahman and Hamada (2020) proposed a Burrows–Wheeler transform and pattern matching-based lossless text compression algorithm that uses Huffman coding in order to achieve an excellent compression ratio. The researchers introduce an algorithm with two keys that are used in order to reduce more frequently repeated characters after the Burrows–Wheeler transform. The researchers then find patterns of a certain length from the reduced text and apply Huffman encoding. They compare their proposed technique with state-of-the-art text compression algorithms. Finally, the researchers conclude that the proposed technique demonstrates a gain in compression ratio when compared to other compression techniques. Another finding is that a small problem exist with their proposed method because it does not work very well for symmetric communications like Brotl. [2]

**An Enhanced Text Compression System Based on ASCII Values and Huffman Coding**

Rani and Singh (2016) have developed various systems to compress the text data to reduce data transfer cost and to increase the performance of the communication channel. Basically, Data compression is a process by which a file (Text, Audio, and Video) may be transformed to another (compressed) file, such that the original file may be fully recovered from the original file without any loss of actual information. This process may be useful if one wants to save the storage space. Also compressed files are much more easily exchanged over the internet since they can be uploaded and downloaded much faster. In their study, a data compression algorithm is represented using dynamic text compression algorithm which uses Huffman Coding to compress and decompress the text data. This algorithm improves the performance of Huffman Algorithm and gives better results. [3]

**The performance of text file compression using Shannon-Fano and Huffman on small mobile devices**

Mantoro et.al (2017) proposed an improved Huffman algorithm which were evaluated and also the possible compression method on smartcard has been discussed. The goal of their study is to produce a compression method that produces text files with the smallest ratios and the fastest compression and de-compression times especially on mobile devices. Their study also improves file compression performance in order to obtain better file size, compression ratio, and compression time especially on text processing. [4]

**Efficient Data Compression Scheme using Dynamic Huffman Code Applied on Arabic Language**

Ghwanmeh et al (2006) in their study, employed the dynamic Huffman coding on data compression with variable length bit coding, on the Arabic language. Experimental tests have been performed on both Arabic and English text. A comparison is made to measure the efficiency of compressing data results on both Arabic and English text. Also a comparison is made between the compression rate and the size of the file to be compressed. Based on the study, It has been found that as the file size increases, the compression ratio decreases for both Arabic and English text. The experimental results from the study show that the average message length and the efficiency of compression on Arabic text is better than the compression on English text. Also, results show that the main factor which significantly affects compression ratio and average message length is the frequency of the symbols on the text. [5]

**Entropy of Malayalam language and text compression using Huffman coding**

Kuruvila and Gopinath (2014) performed an informational analysis paper on the Entropy of Malayalam language. Entropy of Malayalam language is calculated and is obtained as 4.8 bits per character. The Malayalam text compressor discussed in their study, follows a Huffman coding technique which takes both Malayalam and English alphabets along with arithmetic numbers and most probable character is represented by less number of bits. Based on the result of their study, it is found that the Huffman compression algorithm achieves a compression ratio of 66 percentage for a standard Malayalam database taken. A comparison is made on compression ratio for different databases taken. [6]

**Multiple Subgroup Data Compression Technique Based on Huffman Coding**

Shukla et al (2009) proposed an intelligent compression method in their study based on switching code information compression technique with the utility of adaptive Huffman coding. The programming development of the structure encodes usually occurring characters with shorter bit code and infrequently occurring appearing characters with longer bit codes, same code can be use for three symbols for numbers, alphabet, and special character. The decoding process regenerate the encoded data by expanding the encoded data back to the original data and works very much like the encoder process. The proposed method is more effective than adaptive Huffman coding because it reduce the codeword length of the characters and can be use same codeword for three different groups character. Their size of the symbol-codeword table is also reduces which transmitted with the compressed data. This process enhancing compression efficiency up to 12% more than adaptive Huffman coding which provide compression ratio up to 52.51% (+12%). Based on their study, the propose system is very impressive for compressing text file, which hold numbers and special symbols approximately equi-probable to alphabets like Mathpsilas Books/database files in a real time environment. [7]

**An Adaptive Multiple Order Context Huffman Compression Algorithm Based on Markov Model**

Yonghua et al (2016) proposed in their study, an adaptive multiple order context Huffman compression algorithm based on Markov chain. Firstly, the data to be compressed is traversed, and the character space of the data and the times that one character transfers to its neighboring character are figured out. According to the statistical results, the researchers calculate the one-step transition probability matrix and the multi-step transition probability matrix. When the conditional probability between two adjacent characters is greater than the set threshold value, the adjacent characters are merged and compressed as an independent encoding unit. Improve the compression efficiency by increasing the length of the compression characters. Based on the experimental results of the study, it shows that the algorithm achieves good compression efficiency. [8]

**Performance Improvement Of Bengali Text Compression Using Transliteration And Huffman Principle**

In the study of Hussain et al (2016), a new compression technique was proposed based on transliteration of Bengali text to English. Compared to Bengali, English is a less symbolic language. Thus transliteration of Bengali text to English reduces the number of characters to be coded. Huffman coding is well known for producing optimal compression. When Huffman principal is applied on transliterated text significant performance improvement is achieved in terms of decoding speed and space requirement compared to Unicode compression. [9]

**A Data Compression Scheme for Chinese Text Files Using Huffman Coding and a Two-Level Dictionary**

Ong and Shell-Ying (1995) in their study presented a data compression scheme for Chinese text files. Due to the skewness of the distribution of Chinese ideograms, the Huffman coding method is adopted. By storing the frequencies of the encoding symbols rather than their Huffman codes in a dictionary, applying differential coding where it saves space, and structuring the dictionary in the Huffman coding scheme into a two-level dictionary structure, the algorithm produces significant improvement on the compression results. The proposed method is evaluated by comparing its performance with three well-known compression algorithms. This algorithm should also be applicable to other ideogram-based or oriental-language texts. Based on the study, it has the potential to reduce the dictionary size in a bigram- or trigram-based semi-adaptive compression scheme for English texts. [10]

**Extending Huffman coding for multilingual text compression**

In the study of Chi-Hung et al (1995) two new algorithms that are based on the 16-bit or 32-bit sampling character set and on the unique features of languages with a large number of distinct characters to improve the data compression ratios for multilingual text documents were proposed. The researchers choose Chinese language using 16 bit character sampling as the representative language in their study. The first approach, called the static Chinese Huffman coding, introduces the concept of a single Chinese character in the Huffman tree. The experimental results of their study showed that the improvement in compression ratio were obtained. The second approach, called the dictionary-based Chinese Huffman coding, includes the concept of Chinese words in the Huffman coding. [11]

**Design and Implementation of Huffman Decoder for Text data Compression**

Swapnah and Ramesh (2015) presented in their paper a Huffman decoder based on new binary tree method for improving usage of memory and Bandwidth for Text data Compression. The work mainly deals with the implementation of Huffman decoder on a Xilinx 14.7 version, using Verilog Hardware Description Language. [12]

**Compression Using Huffman Coding**

Data compression is also called as source coding. It is the process of encoding information using fewer bits than an un-coded representation is also making a use of specific encoding schemes. Compression is a technology for reducing the quantity of data used to represent any content without excessively reducing the quality of the picture. It also reduces the number of bits required to store and/or transmit digital media. Compression is a technique that makes storing easier for large amount of data. There are various techniques available for compression. Huffman algorithm is one and can be compare with other common compression techniques like Arithmetic, LZW and Run Length Encoding. [13]

**The Application of Text Compression to Short Message Service Using Huffman Table**

In the study of Ahmad et al (2011), a Huffman table as an application text compression on SMS was presented in order to compress and decompress when sending and receiving message and as to overcome the limitation on short message services. Their application is made by J2ME and will run on mobile phones based on MIDP 2.0. The results of their study show that Huffman table can be used in the compression text on SMS services. Generally Huffman table has an average compression ratio of 28.73 %. Overall SMS compression testing system by using the Huffman table will produce good compression if the compressed data to be composed of the characters with a shorter length code (lower case letter) contained in the Huffman table, and vice versa if the SMS data consists of characters which have a longer code (data consisting of all upper case) contained in the table Huffman compression results become less effective. [14]

**Study on Various Data Compression Types and Techniques**

Huffman coding is an entropy encoding strategy used for lossless data compression, according to Muthuchamy (2018). The technique relates to the usage of variable length code tables for encoding a source symbol. As a result, for symbol-by-symbol coding with a known input probability distribution, Huffman coding is the best option. The study will aid the researchers in comprehending how Huffman coding will influence the researcher’s objectives. [15]

**Text Compression using Huffman Coding**

According to Sayood (2018), he created the probabilities of the occurrence of the English Alphabet based on a specific chapter on the study using Huffman coding, resulting in a file size reduction of around 70,000 bytes to 43,000 bytes. The study will be used by the researchers to acquire a better compression, as suggested by Sayood, in order to remove the structure present in the form of correlation between the symbols in the file. [16]

**Advantages of Huffman Coding**

As stated by Mengyi Pu (2006), the Huffman algorithm has the advantages of requiring no preprocessing and having a low overhead of using the uncompressed version of the symbol only at their first occurrence, and it can be allied to other types of files beside text files, such as objects or bytes in an executable file. [17]

**Disadvantages of Huffman Coding**

Mengyi Pu (2006) also mentioned that the Huffman algorithm has two drawbacks: first, it is not optimal unless all of the probabilities are negative power of 2, which means that in most cases, there is a gap between the average number of bits and entropy; second, despite the availability of some clever methods for counting the frequency of each symbol fairly quickly, it can be very slow when counting the frequency of each symbol. To avoid errors, the researchers will keep this in mind when working on the project. [18]

# **Objectives of the project**

## General Objectives

The overall objective of this project is to create an application that compresses text using Hoffman algorithm.

## Specific Objectives

To achieve the goal, we must:

* Ideate on how to compress text.
* Create a pseudocode on text compression using Huffman algorithm and simulate to create a proof of concept.
* Create a prototype application on compressing text based on the pseudocode and run several tests and corner cases to ensure the functionality of the application.
* Debug if problem arises, update, and compress the code if necessary.

# **Scope and limitations of the project**

The study aims to generate a text file compression tool that utilizes Huffman Algorithm. The study also investigates the effectiveness of the Huffman algorithm in the area of text compression and demonstrates this process using programming. This paper shall consider only the lossless compression scheme which preserves all original data, and has not yet taken under consideration the use of different types of compression schemes such as digitization, color representation, digital data representation etc. This study will only focus on the compression of text files, and does not yet incorporate compression of images, videos, or other types of digital media. There are various compression algorithms available, but this study will focus on the implementation of the Huffman coding technique to create the data compression tool.

# System of design and methodology

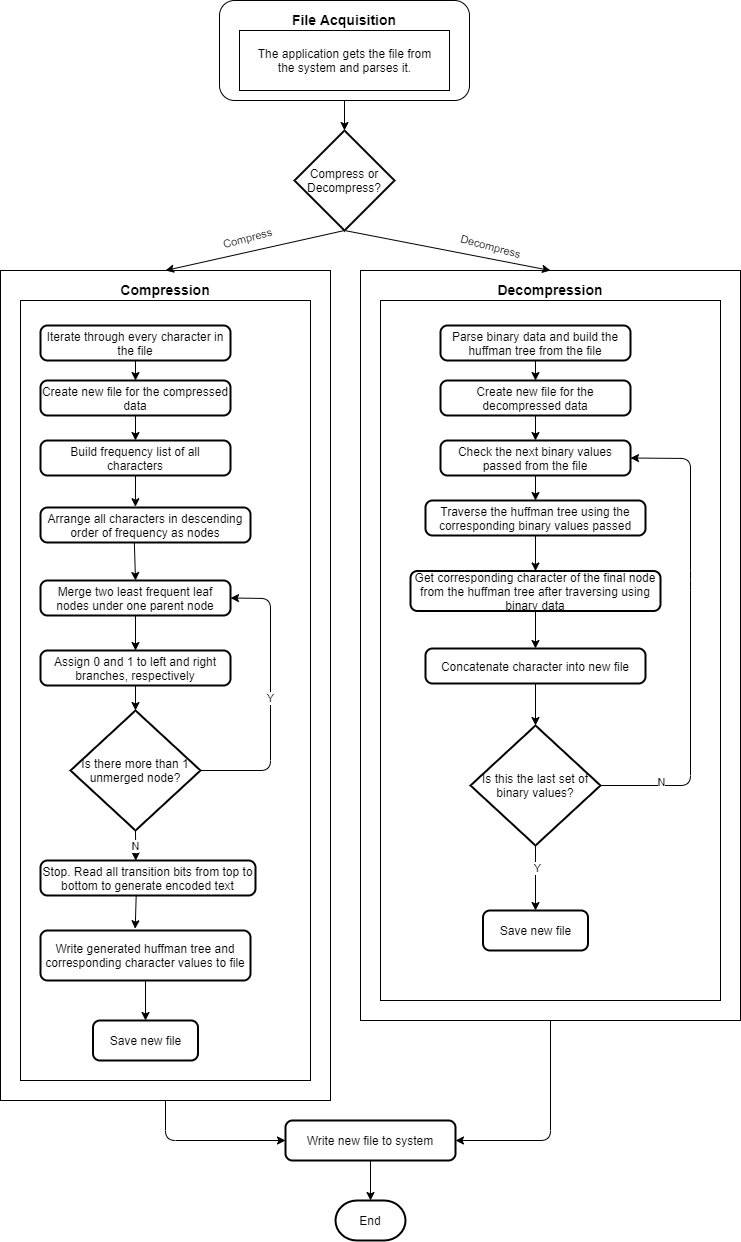


Figure 2: System Architecture

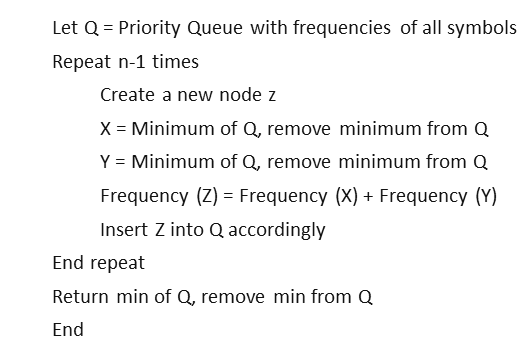


Figure 3: Pseudocode of the Program

The system begins with File Acquisition in which the user must provide the directory to a file that will be the file that the application will get data from to compress using Huffman Algorithm.

Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol or character.

As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols, which ultimately results in a small file size. To make sure Huffman's method is efficiently implemented, an optimized code is implemented in time linear to the number of input weights if these weights are sorted.

For every input:

Our goal:

where

Equation 1. Goal for Compression

However, it is important to note that Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding or asymmetric numeral systems if better compression ratio is required.

# **Results and discussion**

This section describes the results generated by the proposed system. We conducted the testing process on sample text files of different sizes and compressed it with Huffman algorithm. Compression ratio coefficient functioned our main performance parameter used to evaluate the efficiency of the method based on our sample texts. Results of the comparison between these files size with the original size (bytes) are shown in Table 1 and 2. We also made a comparison with some other algorithms namely, LZW, GZIP, and RUNLENGTH, in order to compare Huffman algorithm with other lossless compression techniques, and to show the usefulness of our utilized algorithm. Another part of our discussion tackles the multi-level compression of our proposed system. With this we would compress and decompress for about three consecutive times on random sample text files of different sizes.

Performance Parameters: Performance evaluation of the proposed algorithm is done using one parameter which is Compression Ratio.

Compression ratio: Compression ratio is defined as the ratio of size of the compressed file to the size of the source file. Compression ratio= (Compressed size \* 100)/Original file size.

The input file which is to be compressed using Huffman algorithm is selected from the specific location and displayed in the list box as shown in the following figure 4.

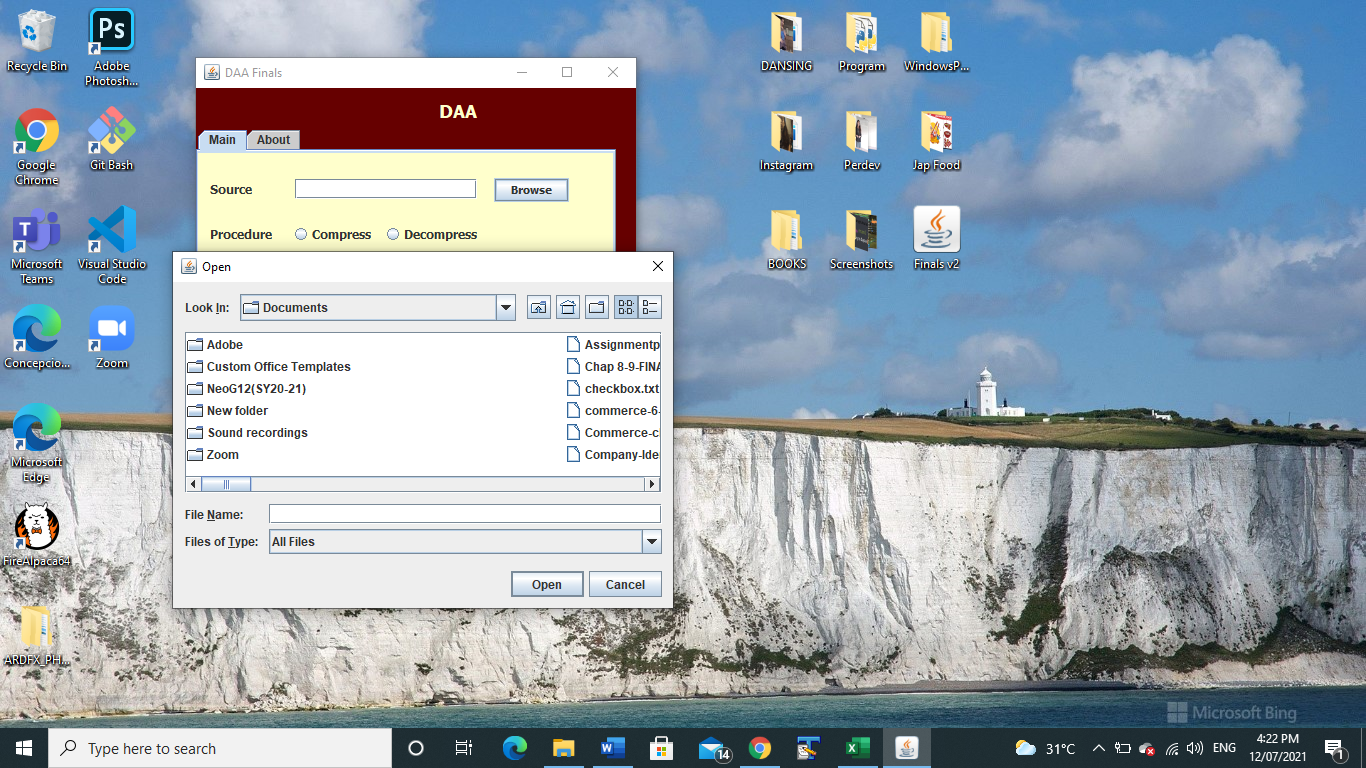


Figure.4 – System’s running window for file browsing

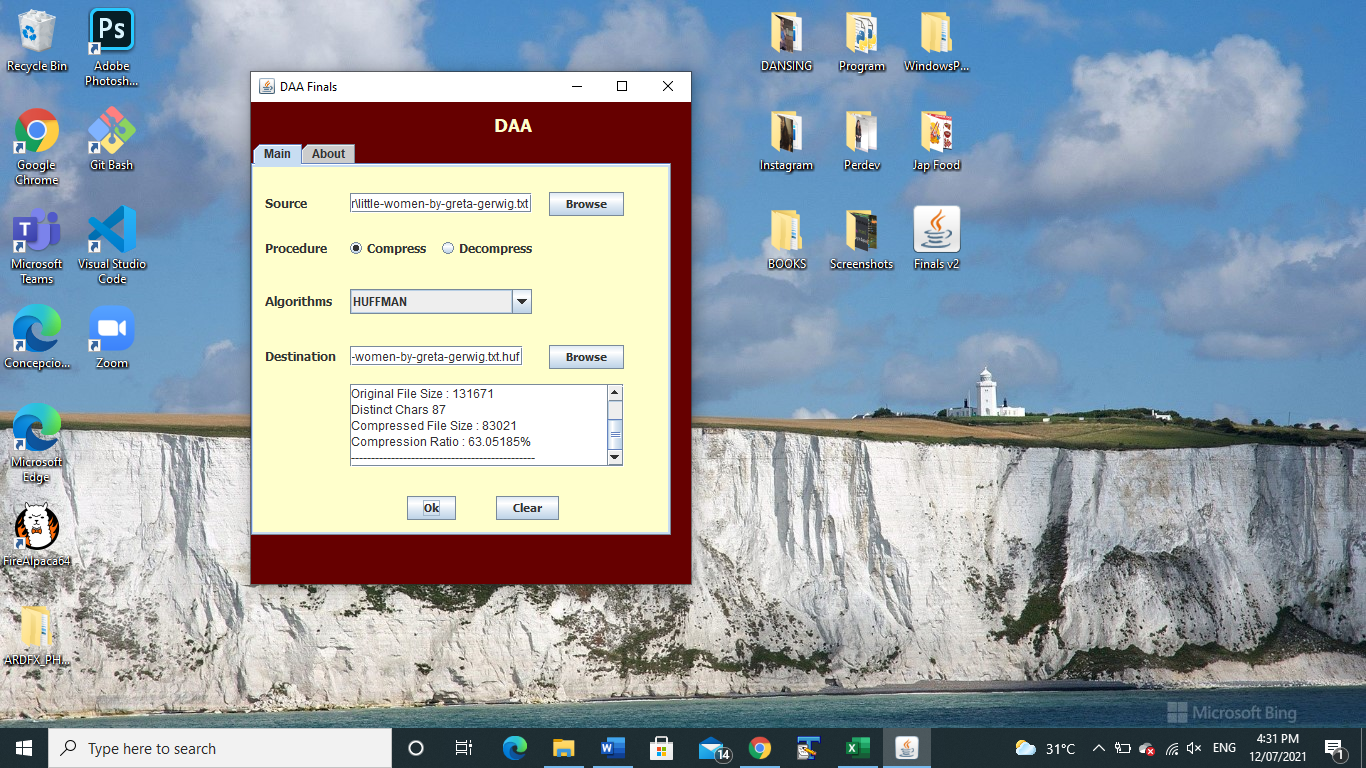


Figure.5 – Compression results using Huffman Algorithm

1. Efficiency test of the Huffman Algorithm

To test the efficiency of the Huffman algorithm in the field of text compression, we have conducted multiple attempts of compression on different sample texts with different sizes. We have noted the original file size and the compressed file size to show comparison. The compression ratio is also showed on the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| Input text size (in bytes) | Output of the system (in bytes) | Compression Ratio of the system (in %) | Distinct Chars |
| 132 | 126 | 95.454544% | 23 |
| 273 | 257 | 94.13919% | 40 |
| 570 | 377 | 66.14035% | 30 |
| 1,264 | 821 | 64.95253% | 41 |
| 131,671 | 83,021 | 63.05185% | 87 |
| 1,256,167 | 720,202 | 57.3333% | 91 |

Table 1 – the table above shows the various trials conducted to determine the compression ratio achieved by the system for random data set.

We can see that the system is fully functional even on smaller text files. Based on the output of the system, we can see the apparent difference on file sizes. The sample text of the story Moby Dick with a file size of 1,256,167 bytes is reduced to 720,202 bytes, which ensued a difference of 535,965 bytes. The system performed exceptionally at 57.33% compression ratio. The system’s functionality can also be observed at smaller file sizes, yet the difference is not as evident.

1. Comparison of Huffman Algorithm to other compression techniques

In this study, we also attempted to compare the performance of the Huffman algorithm to other text compression algorithms. The original file sizes and the compressed file sizes are shown on the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Original | Huffman | LZW | GZIP | Runlength |
| 132 | 126 | 154 | 116 | 134 |
| 273 | 257 | 306 | 208 | 274 |
| 570 | 377 | 523 | 334 | 571 |
| 1,264 | 821 | 925 | 424 | 1,266 |
| 131,671 | 83,021 | 69,057 | 52,842 | 131,660 |
| 1,256,167 | 720,202 | 684,342 | 510,230 | 125,5886 |
| TOTAL | | | | |
| 1390077 | 804,804 | 755,307 | 564,154 | 1,389,791 |

Table 2 – Shows the original file size and compressed size using different algorithms.

We have also computed the total output of the system and used it as a means of comparison. These collective outputs of the system, when compared to the input text size, clearly showed how much data was compressed. From the original size of 1,390,077 bytes, GZIP performed exceptionally and reduced the size to 564,154 bytes. Huffman algorithm, however, was only able to reduce the file size to 804,804 bytes, which has a difference of 240,650 bytes when compared to the GZIP algorithm.

From the histogram below, we can see that the Huffman algorithm performed of inferior quality compared to GZIP algorithm but is almost on par with the LZW algorithm.

Figure.6 – The figure above shows the comparison of the compressed file sizes from the original file sizes, using different algorithms

1. Multi-level compression & decompression on random sample text files of different sizes for about three consecutive times.

|  |  |
| --- | --- |
| File | |
| Input text size (in bytes) | Description |
| 1,861 | Happy – Pharrell Williams Lyrics |
| 1,998 | Dear John – Taylor Swift Lyrics |
| 2,309 | Some Nights – Fun Lyrics |
| 131,314 | Sherlock Holmes Script |
| 134,933 | Coco Script |
| 151,832 | The Greatest Showman Script |
| 1,256,167 | The Moby Dick Story |

|  |  |
| --- | --- |
| 1,663,104 | Space Trilogy Story |
| 3,324,842 | War And Peace Story |

|  |  |  |
| --- | --- | --- |
| 1st Compression | | |
| Input text size  (in bytes) | Output Size  (in bytes) | Compression Ratio |
| 1,861 | 1,181 | 63.460506% |
| 1,998 | 1,286 | 64.364365% |
| 2,309 | 1,448 | 62.711132% |
| 131,314 | 83,658 | 63.708363% |
| 134,933 | 89,018 | 65.972% |
| 151,832 | 99,563 | 65.57445% |
| 1,256,167 | 720,202 | 57.3333% |
| 1,663,104 | 951,166 | 57.192215% |
| 3,324,842 | 1,916,892 | 57.653625% |
| 2nd Compression | | |
| Input text size  (in bytes) | Output Size  (in bytes) | Compression Ratio |
| 1,181 | 1,779 | 150.63506% |
| 1,286 | 1,902 | 147.90047% |
| 1,448 | 2,075 | 143.3011% |
| 83,658 | 83,732 | 100.088455% |
| 89,018 | 89,042 | 100.02696% |
| 99,563 | 99,364 | 99.800125% |
| 720,202 | 709,975 | 98.57999% |
| 951,166 | 940,666 | 98.896095% |
| 1,916,892 | 1,890,166 | 98.60577% |
| 3rd Compression | | |
| Input text size  (in bytes) | Output Size  (in bytes) | Compression Ratio |
| 1,779 | 2,437 | 136.98708% |
| 1,902 | 2,564 | 134.80547% |
| 2,075 | 2,743 | 132.19276% |
| 83,732 | 84,401 | 100.79898% |
| 89,042 | 89,722 | 100.76369% |
| 99,364 | 100,041 | 100.681335% |
| 709,975 | 710,614 | 100.090004% |
| 940,666 | 941,346 | 100.07229% |
| 1,890,166 | 1,890,761 | 100.03148% |

Table 3 – This table shows the multilevel-compression results conducted on different text files with various sizes for three compression.

Figure.7 – The figure above shows the multi-level compression of random sample text files using three consecutive compression

Table 3 showcase the different stages of compression that are compared towards one another. From the table it can be seen that on the first stage of compression, our output file sizes show significant difference and apparent reduction. However, since our system was made to rapidly and significantly reduce the file size on the first phase of compression, we can see that the output file sizes gradually increased at stages 2 and 3. Hence, our system has already exceeded its own capacity on the first stage of compression. This is due to the core foundation of the Huffman algorithm to create tress, the file size continuously increases as we proceed to deeper levels of compression, since the algorithm links and creates new set of trees every time the user tries to compress the file. So the more in depth we are, or the more compressed our files are, the larger it becomes, since we have already reached the lowest possible file size, we are just connecting more trees to it, hence, creating larger file sizes.

We have also incorporated a decompression part in our system. The text decompression takes the compressed file and expands it into its original form. The decompression part provides the original file as output. That is converting an input data stream into another data stream that has an original size of input file. The decompression process is activated through decompression button using Huffman coding. The decompressed text file is stored in a separate file. However, the downside of decompression is requiring the user to constantly rename or delete the file since the program overwrites the file. Therefore, to know if the decompression part functions properly, the program requires the user to rename or delete the original text file.

# **Conclusion**

The main of goal of this study is to build a text compression tool and measure the efficiency of Huffman algorithm, with the use of compression ratio to compare and analyze the results. Along with this, we compared Huffman coding to other compression techniques. We can conclude that the Huffman coding technique shows that the compression performance is efficient but not entirely better than other compression techniques. The compression rate is average, yet still functional even in smaller file sizes. By also performing a multi-level compression and decompression, we have also observed the efficiency and capacity of our system. Confidently, we can infer that Huffman will open a scope in the field of text compression.

As a research direction, we can suggest based from our study and investigation that Huffman provides satisfactory results. However, it is relatively slower than LZW and GZIP algorithms. Huffman will perform well and give better results from every point of view if we can increase its encoding and decoding speed.

# **Future works/ recommendation**

The data compression system tool based on Huffman Algorithm built on this study works only with text data written in single language. In other words, the system works only on ASCII dataset which can be extended to work with Unicode data for future work. We also recommend adopting various techniques for compression, on different types of digital media.

# **Calendar of Activities**

|  |  |  |
| --- | --- | --- |
| Month | Activity | Members |
| May 28, 2021 | Proposal Submission | All Members |
| June |  |  |
| 1-5 | ● Revising of introduction and background  ● Start doing thegoals, objectives and scope | ● Concepcion, Cervanted, Reoloso  ● De Guzman, Glico |
| 6-12 | ●Revising of goal, objectives and scope  ●Start doing the methodology | ●De Guzman, Glico, Reoloso  ●Concepcion, Cervantes |
| 13-19 | ●Revising of methodology  ●Start doing the code | ●Concepcion, Cervantes  ●Reoloso |
| 20-26 | ●Improving the code | ● Reoloso |
| 27- July 3 | ● Revising of the paper when needed if the code has changed | ●Concepcion, Cervantes, De Guzman, Glico |
| July |  |  |
| 4-10 | ● Finalization of the coding part of the project | ● All Members |
| 11-15 | ● Finalization of the written part of the project | ●All Members |
| July 16, 2021 | ● Final Project Submission | ● All Members |

##### References

1. Elabdalla, A. R. and Irshid, M. I., “An efficient bitwise Huffman coding technique based on source mapping”. Computer and Electrical Engineering 27 (2001) 265 – 272
2. Rahman, M.A., & Hamada, M. (2020). Burrows-Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding. Symmetry, 12, 1654.
3. Rani, M., & Singh, V. (2016). An Enhanced Text Compression System Based on ASCII Values and Huffman Coding.
4. Mantoro, T., Ayu, M.A., & Anggraini, Y. (2017). The performance of text file compression using Shannon-Fano and Huffman on small mobile devices. 2017 International Conference on Computing, Engineering, and Design (ICCED), 1-5.
5. Ghwanmeh, S., Al-Shalabi, R., & Kanaan, G. (2006). Efficient Data Compression Scheme using Dynamic Huffman Code Applied on Arabic Language. Journal of Computer Science, 2, 885-888.
6. Kuruvila, M., & Gopinath, D. (2014). Entropy of Malayalam language and text compression using Huffman coding. 2014 First International Conference on Computational Systems and Communications (ICCSC), 150-155.
7. P. K. Shukla, P. Rusiya, D. Agrawal, L. Chhablani and B. S. Raghuwanshi, "Multiple Subgroup Data Compression Technique Based on Huffman Coding," 2009 First International Conference on Computational Intelligence, Communication Systems and Networks, 2009, pp. 397-402, doi: 10.1109/CICSYN.2009.86.
8. Huo, Y., Wang, Z., Wang, J., Qu, K., & Yang, Y. (2016). An Adaptive Multiple Order Context Huffman Compression Algorithm Based on Markov Model. CollaborateCom.
9. Hossain, M.M., Habib, A., & Rahman, M. (2016). Performance Improvement Of Bengali Text Compression Using Transliteration And Huffman Principle.
10. Ong, G., & Huang, S. (1995). A Data Compression Scheme for Chinese Text Files Using Huffman Coding and a Two-Level Dictionary. Inf. Sci., 84, 85-99.
11. Chi, C., Kan, C., Cheng, K., & Wong, L. (1995). Extending Huffman coding for multilingual text compression. Proceedings DCC '95 Data Compression Conference, 437-.
12. Swapna, R., & Ramesh, P. (2015). Design and Implementation of Huffman Decoder for Text data Compression.
13. Sharma, M. (2010). Compression Using Huffman Coding.
14. Ahmad, Erwin , & Saparudin (2011). The Application of Text Compression to Short Message Service Using Huffman Table.
15. Muthuchamy, K. (2018). A Study on Various Data Compression Types and Techniques.
16. Sayood, K. (2018). Introduction to Data Compression (Fifth Edition).
17. Mengyi Pu, I. (2006). Fundamental Data Compression.