International Journal of Scientific Research and Management (IJSRM)
||Volume||12||Issue||09||Pages||1422-1427||2024||
Website: https://ijsrm.net ISSN (e): 2321-3418
DOI: 10.18535/ijsrm/v12i09.ec01

Text Compression Using the Shannon-Fano, Huffman, and Half–Byte Algorithms

Eko Priyono[1], Hindayati Mustafidah[2*]

[1,2]Informatics Engineering, Universitas Muhammadiyah Purwokerto, Indonesia

Abstract

Background and Objectives: File sizes increase as technology advances. Large files require more storage memory and longer transfer times. Data compression is changing an input or original data into another data stream as output or compressed data which is smaller in size. Existing compression techniques include the Huffman, Shannon-Fano, and Half-Byte algorithms. Like algorithms in computer science, these three algorithms offer advantages and disadvantages. Therefore, testing is needed to determine which algorithm is most effective for data compression, especially text data.

Methods: Applying the Huffman, Shannon-Fano, and Half-Byte algorithms to test their effectiveness in compressing text data. The text data as a sample in the research carried out is a text file containing abstracts from research articles published in scientific publications randomly selected from 100 journals. The abstract text used as data is in Indonesian.

Results: Based on test findings, the Huffman algorithm outperforms the Shannon-Fano and Half-Byte algorithms in terms of compression ratio. The Half-Byte algorithm has the lowest compression ratio compared to the Huffman and Shannon-Fano algorithm. The Half-Byte compression algorithm is based on the similarity of the first four bits of seven consecutive characters, whereas Huffman and Shannon-Fano algorithms employ the number of character appearances. The Huffman method can be considered for use in compressing Indonesian language text data according to its average compression ratio of 46.05%, while Shannon-Fano of 40.36%, and Half-Byte of 5.04%.

Keywords: compression ratio, text data, effectiveness in compressing

1. Introduction

The size of files increases as technology advances. This necessitates additional storage memory and significant transmission times. Not everyone has a significant storage capacity and a high-speed internet connection for file transfers. This issue can be addressed by the development of many file compression technologies, including data compression.

Data compression is a technique that converts an input data stream, namely original data, into another data stream, known as output or compressed data. Compressed output data has a smaller size (Salomon, 2007). Some extant compression techniques are the Huffman algorithm, Lempel Ziv Storer Szymanski (LZSS), Shannon-Fano, Half-Byte, Lempel Ziv Welch (LZW), and others.

Several research have created text compression techniques, including (Mizwar et al., 2017) which implemented the J-Bit Encoding Algorithm, and (Irliansyah et al., 2017), which implemented the Deflate method and the Goldbach Codes Algorithm. Two years later, (Darnita et al., 2019) invented the Sequitur method, which was also used to compress text data. Aside from that, (Saragih and Utomo, 2020) used the Prefix Code Algorithm to compress text data, (Rizky et al., 2020) used the Elias Delta Codes Algorithm, and (Simanjuntak, 2020) applied the Elias Delta Code Algorithm with Levenstein to compress text files.

The Huffman, Shannon-Fano, and Half-Byte algorithms investigated in this study offer advantages and downsides, as have previous studies (Siahaan, 2016) and (Puspabhuana, 2016). Other research that used the Huffman method include (Supiyandi and Frida, 2018), (Widatama and Saputro, 2019), (Mahmoudi and Zare, 2020), and (Pujianto et al., 2020). As a result, it is required to conduct tests to determine which technique is best for text data compression. The efficacy of a compression technique can be assessed by

comparing file sizes before and after compression. It can also be measured based on the algorithm's processing time (Sayood, 2017).

2. Method

The research was conducted using a qualitative and experimental technique. The experiment in question involves compressing text data with a compression method developed in Python computer language. The experiment data will next be examined to determine the most optimal algorithm, following the processes outlined in Figure 1.

Abstract texts in Indonesian were used to collect research data from journal articles selected at random from 100 sources. This text data is then converted in the *.txt format.

The three text data compression techniques, Huffman, Shannon-Fano, and Half-Byte, will be implemented next. The algorithm is implemented by creating a program in Python. Visual Studio coding, a coding editor, was utilized to assist with this process. After all of the algorithms have been implemented in a program, the files are compressed one at a time. Before compressing, it needs to be noted the size of each file.

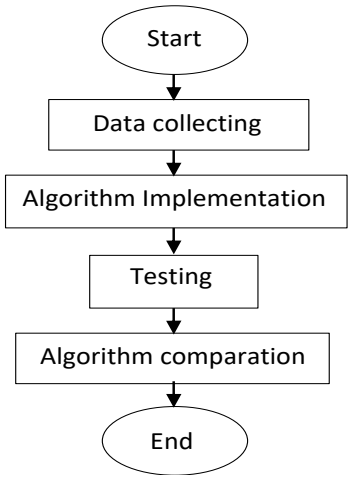

Figure 1: Research stages

The compression of each text file yields compressed characters in bit size. Each compressed file's compression ratio is then measured. The compression ratio measurements are then compared to see which algorithm is the most effective.

In this study, the three methods were compared based on their average compression ratio. The higher the compression ratio, the more efficient the algorithm is at compressing data. The algorithm with the highest average compression ratio is the most efficient or optimal.

The compression ratio calculation is shown in (1).

$$Ratio = 100\% - \left(\frac{compression\ yield\ capacity}{original\ file\ capacity}\ x\ 100\%\right) \quad (1)$$

The ratio is the result of compression and is used to assess the performance of a compression method (Supiyandi and Frida, 2018).

3. Results and Discussion

a. Research Data

The data in this study is presented in the form of abstract text from scientific articles. The sample size was 100 Indonesian language text data. Indonesian was chosen to standardize the data parameters. The abstract text was chosen because it is written in the form of sentences with a range of characters but a consistent systematic structure. The diversity of characters has an impact on the compression process. Abstract systematic homogeneity was employed as a benchmark for comparison. Aside from that, text in the form of sentences is more similar to how text is used in everyday life than repeated letters such as "AAAAABAAACCD".

The abstract content in the journal article retrieved is in the form of a PDF document, which must be translated into TXT format. The size of each text file is recorded before compression is performed so that it can be compared later to the compression results. Figure 2 shows an example of abstract text transferred into a text file.

b. Compression Process

The compression process is carried out by creating source code in Python to implement three compression algorithms: Huffman, Shannon-Fano, and Half Byte. The Huffman algorithm's initial step is to determine the probability or frequency with which a particular character appears. For example, the probabilities for the word "goods" are: a:2, b:1, r:1, n:1, and g:1. From this data, a binary tree known as a Huffman tree is produced. The following source code employs the Huffman technique to encode tree data in bit form. This bit format seeks to reduce file size.

The Shannon-Fano algorithm works similarly to Huffman. This approach creates a binary tree in order to obtain the binary code and then encodes it. The distinction between these two algorithms is the creation of a binary tree. Shannon-Fano creates binary trees from the bottom up, whereas Huffman does so from the top down.

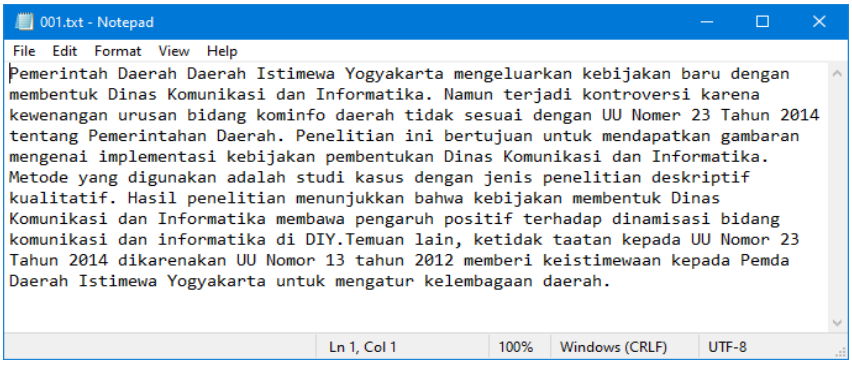

Figure 2: Example of a text file in txt format

The Shannon-Fano method is implemented in stages, beginning with counting and sorting character occurrences. Then, these sequences are divided into two to form a binary tree, which is repeated until all characters have their corresponding values. Once all of the characters have a binary representation, they are encoded using the new binary.

The Half-Byte algorithm uses the same four binary bits in consecutive characters. For example, in the word "yaaaaaay", the letter a has the same four sequential left bits, 0110, as seen in Table 1. The Half-Byte algorithm exploits this circumstance. When seven or more characters with the same first four bits are received in sequence, this algorithm compresses the data with a marker bit, then the first character of the same four-bit sequence, followed by the last pair of four bits in the next sequence, and finally with a closing bit.

Table 1: Binary words "yaaaaaaay" before and after compression

Binary Word "yaaaaaaay" Before Compression		Binary Word "yaaaaaaay" After Compression	
character	binary	character	binary
y	0111 1001	y	0111 1001
a	0110 0001	marker	1111 1110
a	0110 0001	a a	0001 0001
a	0110 0001	a a	0001 0001
a	0110 0001	a a	0001 0001
a	0110 0001	marker	1111 1110
a	0110 0001	y	0111 1001
y	0111 1001		

The initial step in implementing the Half-Byte technique is to transform the input text to binary. Following that, the start or left bit is split from the right or end bit. Next, we look for commonalities in each character's left bits. If there is, the left bit is removed, and a marker bit is assigned to the start and end bounds. The file is then encoded as a new compressed file.

c. Testing

Tests were conducted on 100 text sample files. The compression ratio and size of the findings are used in testing. The three approaches were tested by compiling a program in Python. Running the program produces

the following output: the original uncompressed text, symbols or letters in the text, symbol likelihood, size before and after compression, and bit representation after compression.

Figure 3 shows an example of a 6792-bit txt file used to test the Huffman method. The compression ratio attained was 45.45%, or 3785 bits. Similarly, we tested the Shannon-Fano (Figure 4) and the Half-Byte algorithm (Figure 5). With the identical text data example, the following two algorithms show compression file values of 4090 bits for the Shannon-Fano algorithm, namely with a compression ratio of 39.78%, and 6444 bits for the Half-Byte algorithm or with a compression ratio of 5.12%. The compression ratio produced by the evaluated algorithm is used to determine its effectiveness. The higher the compression ratio, the better the compression, or in other words, the more successful the algorithm.

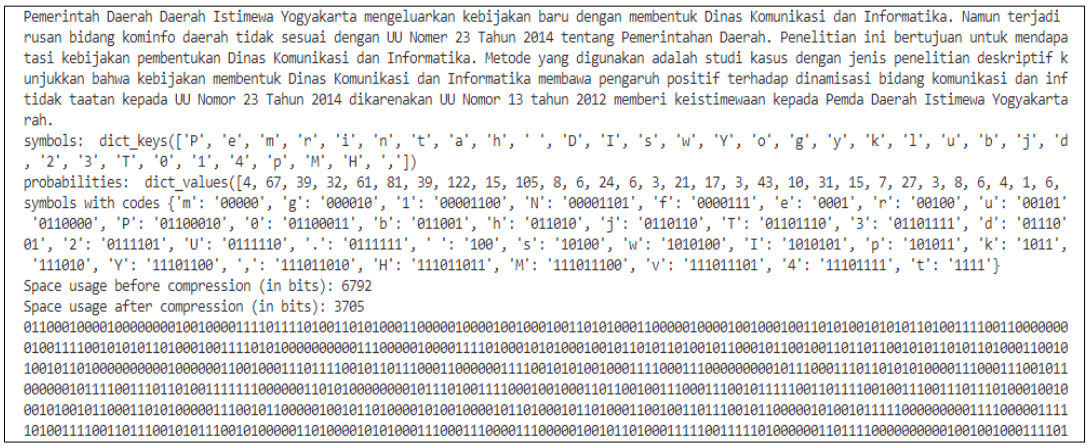

Figure 3: Huffman algorithm's data compression

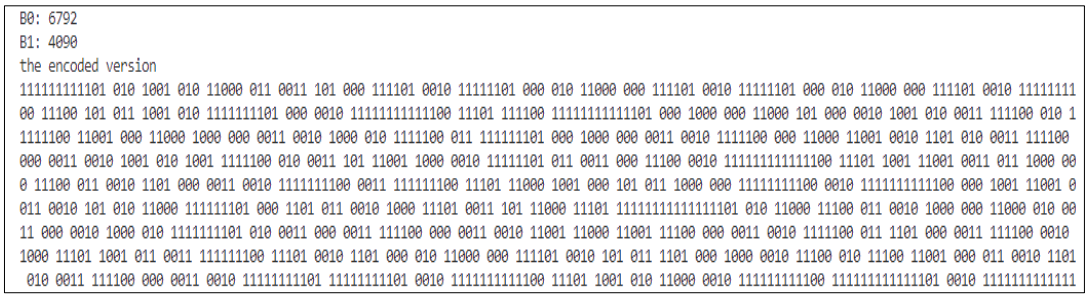

Figure 4: Shannon-Fano algorithm's data compression

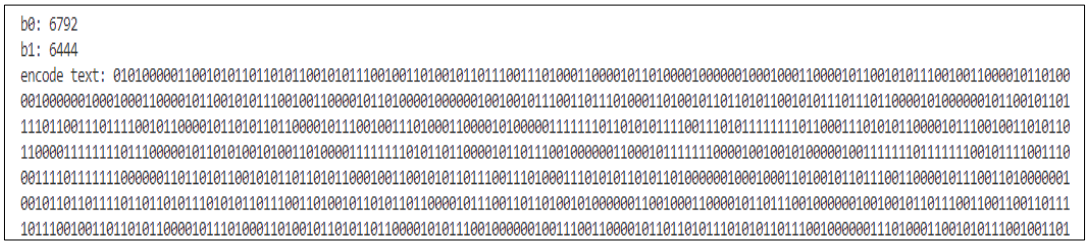

Figure 5: Half-Byte algorithm's data compression

Table 2 displays the compression results of the three methods, Huffman, Shannon-Fano, and Half-Byte, together with the average compression ratio shown in Figure 6.

Table 2: Test results for the Huffman, Shannon-Fano, and Half-Bye algorithms

File name	File size (Bit)				Compression ratio		
	Original	Huffman	Shannon-Fano	Half-Byte	Huffman	Shannon-Fano	Half-Byte
001.txt	6792	3705	4090	6444	45.45%	39.78%	5.12%
002.txt	6640	3465	3724	6338	47.82%	43.92%	4.55%
003.txt	6432	3493	3797	6056	45.69%	40.97%	5.85%
004.txt	7392	4015	4599	7035	45.68%	37.78%	4.83%

005.txt	7552	4108	4498	7214	45.60%	40.44%	4.48%
006.txt	10920	5822	6254	10521	46.68%	42.73%	3.65%
007.txt	8680	4527	4860	8330	47.85%	44.01%	4.03%
008.txt	18632	10342	11842	17679	44.49%	36.44%	5.11%
009.txt	10576	5736	6238	10050	45.76%	41.02%	4.97%
010.txt	10704	5706	6210	10170	46.69%	41.98%	4.99%
...	...	...	...	...	...	...	...
100.txt	7256	3778	4012	6892	47.93%	44.71%	5.02%

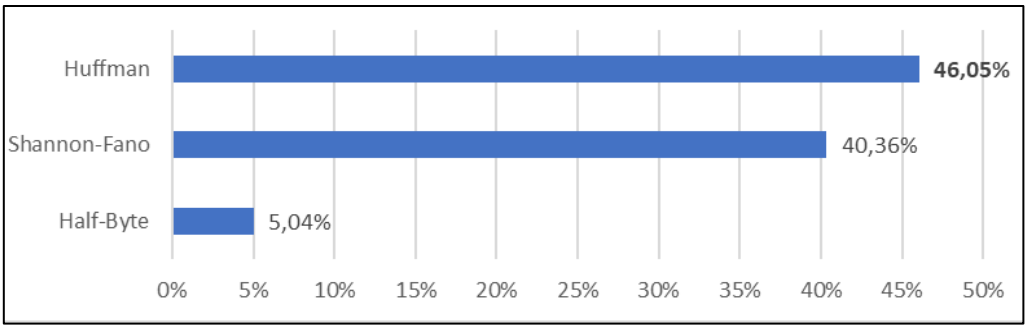

Figure 6: Graph of the average compression ratio of the Huffman, Shannon-Fano, and Half-Byte algorithms

Figure 6 shows that the Huffman method outperforms the other two algorithms. Because of differences in features, the Half-Byte algorithm has a low compression ratio compared to the Huffman and Shannon-Fano algorithm. Half-Byte compression is based on the similarity of the first four bits of seven consecutive characters, whereas Hufman and Shannon-Fano algorithms employ the number of character appearances. The sentences in the sample contain different characters or letters. Because of the wide range of characters, the likelihood of characters having the same and consecutive prefix bits is low. This has an effect on the characters that can be compressed; specifically, just a few characters can be compressed. As a result, there is only a tiny change in bit size between before and after compression.

The findings of comparing the text data compression algorithm with the more successful Huffman algorithm were also revealed in study results (Supiyandi and Frida, 2018), with a compression ratio of 71.43% compared to the Half-Byte technique's compression ratio of 22.33%. According to (Pujianto et al., 2020), the Huffman approach performed better in compression than the Run Length Encoding method. The Huffman method outperforms the Shannon-Fano technique in terms of compression gain for images of the same size and number of colors (Widatama and Saputro, 2019).

The Huffman algorithm is said to offer a higher compression ratio than numerous other compression algorithms for text and graphic data. However, this contradicts the research findings of [15]. The research focused on audio file compression (*.wav). According to the research findings, the Huffman approach has a compression ratio of 28.954%, whereas Run Length Encoding has a ratio of 46.77%.

4. Conclusion

The results of the investigation reveal that the Huffman algorithm outperforms the Shannon-Fano and Half-Byte methods. The Huffman method has an average compression ratio of 46.05%, Shannon-Fano at 40.36%, and Half-Byte at 5.04%. Thus, the Huffman method can be considered for use in compressing Indonesian language text data. Data compression techniques require further testing to determine their effectiveness on data other than text, such as video data, in order to acquire comprehensive information about the optimal compression algorithm.

Conflicts of Interest: “This research was carried out collaboratively between authors and there was no conflict of interest.”

References

3. Darnita, Y., Khairunnisyah, K. and Mubarak, H. (2019), “Text Data Compression Using the Sequitur Algorithm”, *SISTEMASI*, Vol. 8 No. 1, pp. 104–113.
4. Fatmawaty, F. and Mufty, M. (2020), “Comparative Analysis of Wav File Compression Using the Huffman Method and Run Length Encoding”, *Jurnal Teknologi Informasi Dan Terapan*, Vol. 7 No. 1, pp. 61–65.

5. Irliansyah, M.R., Nasution, S.D. and Ulfa, K. (2017), "Application of the Deflate Method and Goldbach Codes Algorithm in Text File Compression", *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, Vol. 1 No. 1.
6. Mahmoudi, R. and Zare, M. (2020), "Comparison of Compression Algorithms in text data for Data Mining", *Int. J. Adv. Eng. Manag. Sci.*, Vol. 6, pp. 231–235.
7. Mizwar, T., Ginting, G.L., Mesran, M., Fau, A., Aripin, S. and Siregar, D. (2017), "Implementation of the J-Bit Encoding Algorithm in Text File Compression", *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, Vol. 1 No. 1.
8. Pujianto, M., Prasetyo, B.H. and Prabowo, D. (2020), "Comparison of the Huffman Method and Run Length Encoding in Document Compression", *InfoTekJar J. Nas. Inform. Dan Teknol. Jar*, Vol. 5 No. 1, pp. 216–223.
9. Puspabhuana, A. (2016), *"Three Steps Comparison of Text Compression Techniques"*, President University.
10. Rizky, N.F., Nasution, S.D. and Fadlina, F. (2020), "Application of the Elias Delta Codes Algorithm in Text File Compression", *Building of Informatics, Technology and Science (BITS)*, Vol. 2 No. 2, pp. 109–114.
11. Salomon, D. (2007), *A Concise Introduction to Data Compression*, Springer Science & Business Media.
12. Saragih, S.R. and Utomo, D.P. (2020), "Application of the Prefix Code Algorithm in Text Data Compression", *KOMIK (Konferensi Nasional Teknologi Informasi Dan Komputer)*, Vol. 4 No. 1.
13. Sayood, K. (2017), *Introduction to Data Compression*, Morgan Kaufmann.
14. Siahaan, A.P.U. (2016), "Implementation of Huffman Text Compression Technique", *Jurnal Informatika Ahmad Dahlan*, Universitas Ahmad Dahlan, Vol. 10 No. 2, p. 101651.
15. Simanjuntak, L.V. (2020), "Comparison of the Elias Delta Code and Levenstein Algorithms for Text File Compression", *Journal of Computer System and Informatics (JoSYC)*, Vol. 1 No. 3, pp. 184–190.
16. Supiyandi, S. and Frida, O. (2018), "Comparative Analysis of Text Data Compression Using Huffman and Half-Byte Methods", *ALGORITMA: JURNAL ILMU KOMPUTER DAN INFORMATIKA*, Vol. 2 No. 1.
17. Widatama, K. and Saputro, W.T. (2019), "Comparison of the Performance of the Huffman Algorithm and the Shannon-Fano Algorithm in Compressing Image Files", *INTEK: Jurnal Informatika Dan Teknologi Informasi*, Vol. 2 No. 2, pp. 70–77.