Journal of Theoretical and Applied Information Technology

30th September 2023. Vol.101. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARISON OF STOUT CODE AND FIBONACCI CODE ALGORITHM FOR FILE COMPRESSION BASED ON ANDROID

HANDRIZAL[1], T. HENNY FEBRIANA HARUMY[2], FADHLI IBRAHIM SIREGAR[3]

[1,2,3]Department of Computer Science, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia

E-Mail: handrizal@usu.ac.id

ABSTRACT

In the current era, human activities are closely intertwined with the utilization of computers and the internet. Engaging in activities with computers necessitates data storage, whether through cloud-based systems or physical storage mediums. The size of data becomes a critical factor in optimizing resource efficiency for these activities. As a result, ongoing research focuses on data size reduction techniques to enhance overall efficiency. Data compression is the process of converting data into smaller sizes. There are various data compression algorithms. Stout Code algorithm and Fibonacci Code algorithm will be used in this study. We built an Android application and performed a text file compression test using these two algorithms to compare their performance. The comparison parameters that will be used are compression ratio, compression time, and decompression time. The test results indicate that the Stout Code algorithm outperforms the Fibonacci Code algorithm in terms of compression ratio for both homogeneous and heterogeneous strings. The average compression ratios for the Stout Code are 1.949 and 1.159, while for the Fibonacci Code, they are 1.943 and 1.064, respectively. However, concerning compression time and decompression time, the Fibonacci Code algorithm proves to be more efficient. Its average compression times for homogeneous strings and heterogeneous strings are 2437 ms and 2855.429 ms, whereas the Stout Code algorithm takes an average of 2564.857 ms and 3021.571 ms. Similarly, for decompression time, the Fibonacci Code algorithm outperforms the Stout Code algorithm with average times of 349.571 ms for homogeneous strings and 853.857 ms for heterogeneous strings, while the Stout Code algorithm shows average times of 456 ms and 1016.143 ms, respectively. The results lead to the conclusion that the Stout code algorithm outperforms in reducing file sizes, whereas the Fibonacci code algorithm excels in terms of speed.

Keywords: *Stout Code, Fibonacci, Android, Algorithm, Compression*

1. INTRODUCTION

Human activities in this era generally have a relationship with the use of computers and the internet. When using a computer to perform an activity, data storage is required. Whether in the form of cloud storage or physical storage, the issue of data size is important to the efficiency of resources in carrying out activities. Therefore, research in trying to reduce the size of data is continuously conducted.

Data compression is the process of converting input data into output data that is smaller in size [1]. The technique of compressing data plays an important role in determining how much size is reduced [2]. An effective compression technique will make the data size smaller than before. Various algorithms have been developed for this problem. Two of them use the Stout code and Fibonacci code.

Stout code is a recursive algorithm discovered by Quentin Stout in 1980. This algorithm consists of two types called families. The first family is called Rℓ and the second is called Sℓ. The defining parameter of this algorithm is an integer number greater than or equal to two. This parameter is called ℓ. The Sℓ family has several advantages over the Rℓ family for small ℓ values. In research [3] it was found that this algorithm is suitable for compressing text.

The Fibonacci code algorithm is an algorithm whose compression method uses the Fibonacci sequence which is converted into binary to form the code [4]. Research [5] found that the Fibonacci code algorithm is better than Even-Rodeh Code in compressing text on both homogeneous and heterogeneous characters.

Based on previous research, it has been found that both algorithms are suitable for text compression. In this research, we want to examine the performance comparison of the two algorithms using Android-based applications. The comparison parameters used are compression ratio, compression time, and decompression time. The results of this research are expected to provide an understanding of the advantages and disadvantages of both algorithms to make it easier to choose the appropriate text compression algorithm.

2. LITERATURE REVIEW

2.1 Compression

Data compression is the science or art of representing information in a compact form. Data compression is done by identifying and using the structure that exists in the data. Data compression is needed because of the increasing amount of information humans produce or store digitally [6].

Techniques in data compression can be broadly divided into two parts, namely lossy data compression and lossless data compression.

Lossy compression techniques are compression techniques that involve partial loss of information. Data that has been compressed using lossy techniques generally cannot be recovered or reconstructed precisely. However, data compressed using this technique can generally obtain a much higher compression ratio than using lossless compression techniques [7].

Lossless compression technique as the name suggests, does not involve any loss of information. If data has been compressed using this compression technique, the original data can be recovered back to its original state without any data loss. Lossless compression is generally used for applications that cannot tolerate differences between the original data and the reconstructed data [8].

2.2 Stout Code Algorithm

The Stout code algorithm is an algorithm discovered by Quentin Stout in 1980. The codeword generated by the Stout code algorithm depends on a parameter ℓ that is chosen with the condition that it is greater than or equal to two [3].

In the R_ℓ family, the prefix is defined as:

$R_\ell(n) = B(n, \ell)$, for $0 \le n \le 2^\ell - 1$

$R_\ell(n) = R_\ell(L)B(n, \ell)$, for $n \ge 2^\ell$

B(n, ℓ) is a binary value n taken as ℓ bits. For example, B(1, 3) will produce a binary value of 1 taking as many as 3 bits, namely 001. L is the number of digits in the binary value n. For example, if n is 4, then the binary value of n is 100. There are three digits in the binary value of n, so the value of L is 3.

The codeword for the S_ℓ family is formed by the same method using a different prefix. This prefix is denoted by $S_\ell(n)$. The S_ℓ family has an advantage over the R_ℓ family for small ℓ values (Nasution, 2019).

In the S_ℓ family, the prefix is defined as:

$S_\ell(n) = B(n, \ell)$, for $0 \le n \le 2^\ell - 1$

$S_\ell(n) = R_\ell(L - 1 - \ell)B(n, \ell)$, for $n \ge 2^\ell$

2.3 Fibonacci code algorithm

The Fibonacci code algorithm is an algorithm whose compression method uses the Fibonacci sequence converted into binary to form the code. The formation of Fibonacci codes is based on the fact that a positive integer n can be expressed uniquely as the sum of different Fibonacci numbers [9].

Table 1. Fibonacci codes table

n	*Fibonacci code*	n	*Fibonacci code*
1	11	7	01011
2	011	8	000011
3	0011	9	100011
4	1011	10	010011
5	00011	11	001011
6	10011	12	101011

2.4 Problem analysis

In this research, the problem raised is the comparison of the Stout code algorithm and the Fibonacci code algorithm in compressing text files in Android-based applications. To make it easier to recognize the factors that cause this problem and the relationship between these factors, the authors describe the following Ishikawa diagram.

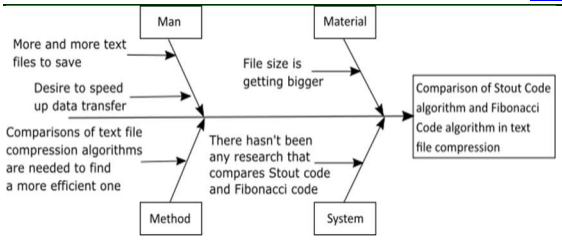

Figure 1. Ishikawa Diagram

The figure shows that the main problem of this study is to compare the Stout code and Fibonacci code algorithms in compressed text files. Then four factors cause the main problem, namely human, material, method, and system. In these four factors, there are details of problems related to each factor described by arrows pointing to the arrow belonging to the factor.

3. RESULTS AND DISCUSSIONS

There is a text file containing the string “FADHLI SIREGAR." The size of the original string is shown in Table 2.

Table 2. Size of string before compressed

n	Character	Frequency	ASCII Binary	Bit	Frequency x Bit
1	A	2	01000001	8	16
2	I	2	01001001	8	16
3	R	2	01010010	8	16
4	F	1	01000110	8	8
5	D	1	01000100	8	8
6	H	1	01001000	8	8
7	L	1	01001100	8	8
8	space	1	00100000	8	8
9	S	1	01010011	8	8
10	E	1	01000101	8	8
11	G	1	01000111	8	8
		Total Bits			112

Based on the ASCII code, one character is worth eight bits of a binary number. So 14 characters on the string have a binary value of 112 bits. Before performing the compression process, characters are first sorted from the largest to the smallest frequency.

The compression analysis process of text files using the Stout Code algorithm. Below is an example of the compression process of a text file using the Stout Code algorithm. There is a text file containing the string "FADHLI SIREGAR." It is shown in Table 3. string size that has been compressed using the Stout Code algorithm.

Table 3. String size that has been compressed by using the Stout code algorithm

n	Character	Frequency	*Stout code*	Bit	Frequency x Bit
1	A	2	01	2	4
2	I	2	10	2	4
3	R	2	11	2	4
4	F	1	00100	5	5
5	D	1	00101	5	5
6	H	1	00110	5	5
7	L	1	00111	5	5
8	space	1	011000	6	6
9	S	1	011001	6	6
10	E	1	011010	6	6
11	G	1	011011	6	6
		Total Bits			56

Next, exchange the ASCII binary code of each character in the string "FADHLI SIREGAR" according to the Stout code that has been determined from Table 3. It is shown in Table 4. character conversion to Stout Code.

Table 4. Character Conversion to Stout code

F	A	D	H	L	I	space
00100	01	00101	00110	00111	10	011000
S	I	R	E	G	A	R
011001	10	11	011010	011011	01	11

Then the bit string is obtained as follows:

“00100010010100110001111001100001100110110110100110110111”.

The next step is to add padding bits and flag bits.

In this case, the bits in the bit string are 56. Since the number of bits in the bit string is divisible by 8, then the number of bits in the padding is 0 or there is no need for bit padding at all.

The flag bit is 8 bits of binary value from the number of bits in the padding bit which is 00000000. Then the result after compression is:

“00100010010100110001111001100001100110110110100110110111100000000”.

The compression ratio is obtained as follows:

$$Compression\ Ratio = \frac{Data\ size\ before\ compressed}{Data\ size\ after\ compressed} = \frac{112\ bits}{64\ bits} = \frac{7}{4} = 1.75$$

Below is an example of the compression process of a text file using the Fibonacci code algorithm. There is a text file containing the string "FADHLI SIREGAR." It is shown in Table 5 that string size has been compressed using the Fibonacci code algorithm.

Table 5. String size that has been compressed by using the Fibonacci code algorithm

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

n	Character	Frequency	*Fibonacci code*	Bit	Frequency x Bit
1	A	2	11	2	4
2	I	2	011	3	6
3	R	2	0011	4	8
4	F	1	1011	4	4
5	D	1	00011	5	5
6	H	1	10011	5	5
7	L	1	01011	5	5
8	space	1	000011	6	6
9	S	1	100011	6	6
10	E	1	010011	6	6
11	G	1	001011	6	6
		Total Bits			61

Next, exchange the ASCII binary code of each character in the string "FADHLI SIREGAR" according to the Fibonacci code that has been determined from Table 5. It is shown in Table 6. character conversion to Fibonacci Code.

Table 6. Character Conversion to Fibonacci code

F	A	D	H	L	I	space
1011	11	00011	10011	01011	011	000011
S	I	R	E	G	A	R
100011	011	0011	010011	001011	11	0011

Then the bit string is obtained as follows:

"10111100011100110101101100001110001101100110100110010111100110000000011".

The next step is to add padding bits and flag bits.

In this case, the bits in the bit string are 61. Since the number of bits in the bit string when divided by 8 leaves the remainder 5, an additional 3 bits are needed. So, the padding bit is 000.

The flag bit is 8 bits of binary value from the number of bits in the padding bit which is 00000011. Then the result after compression is:

"10111100011100110101101100001110001101100110100110010111100110000000011".

The compression ratio is obtained as follows:

$$Compression\ Ratio = \frac{Data\ size\ before\ compressed}{Data\ size\ after\ compressed} = \frac{112\ bits}{72\ bits} = \frac{14}{9} = 1.56$$

3.1. Implementation and Testing

Two Fragments are used in the system, namely the Compression Fragment and the Decompression Fragment.

Journal of Theoretical and Applied Information Technology
30th September 2023. Vol.101. No 18

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

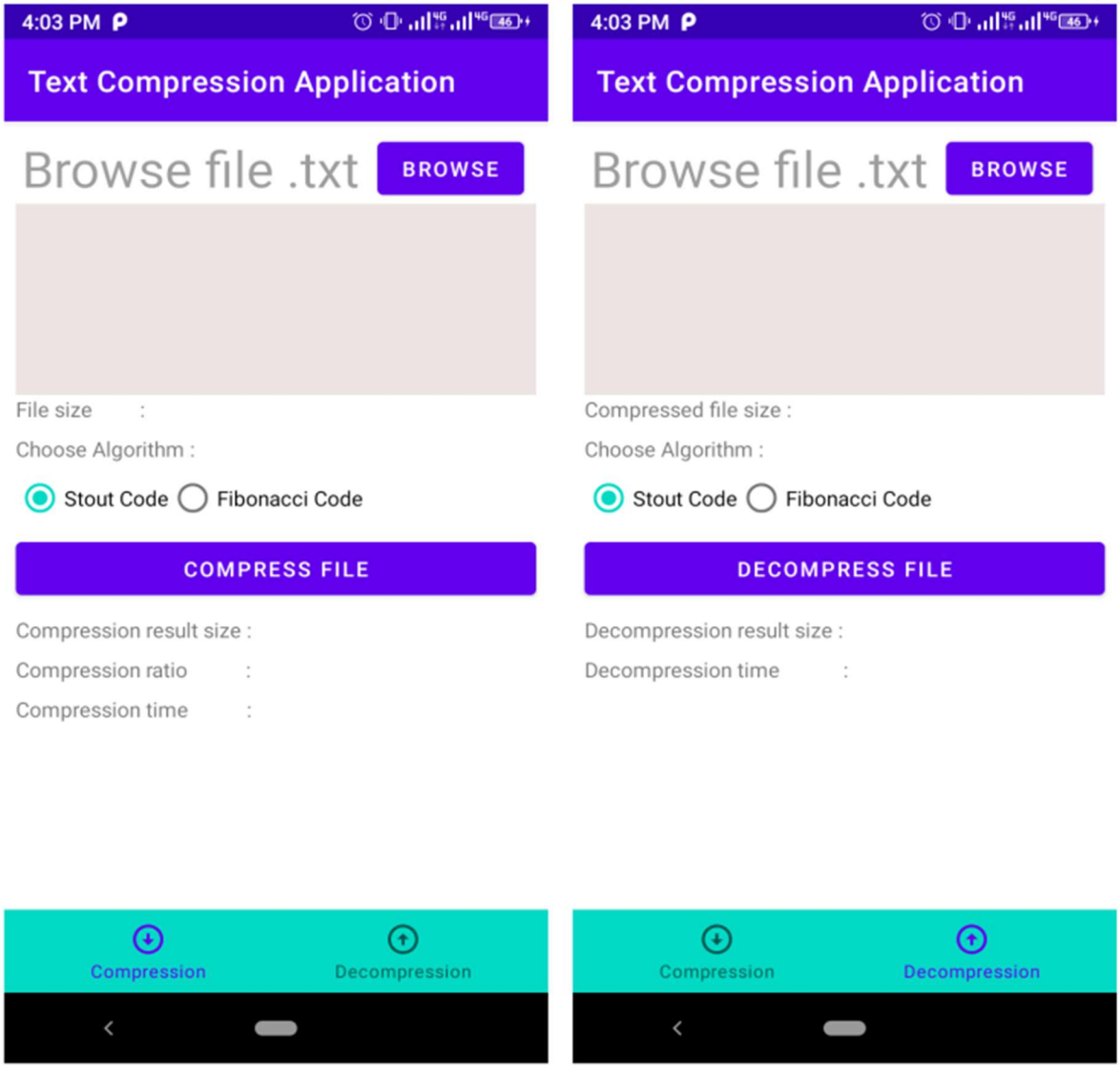

Figure 2. Compression (left) and decompression (right) fragment

3.2. Compression testing

In the testing phase of the compression process, the first thing the user does is enter a text file. Next, users can choose which algorithm they want to perform the compression process. After that, the user presses the compress button to run the system. Then the file will be saved in the root folder of the user's internal Android storage. Furthermore, the system will produce a calculation of the size of the compression result, compression ratio, and compression time.

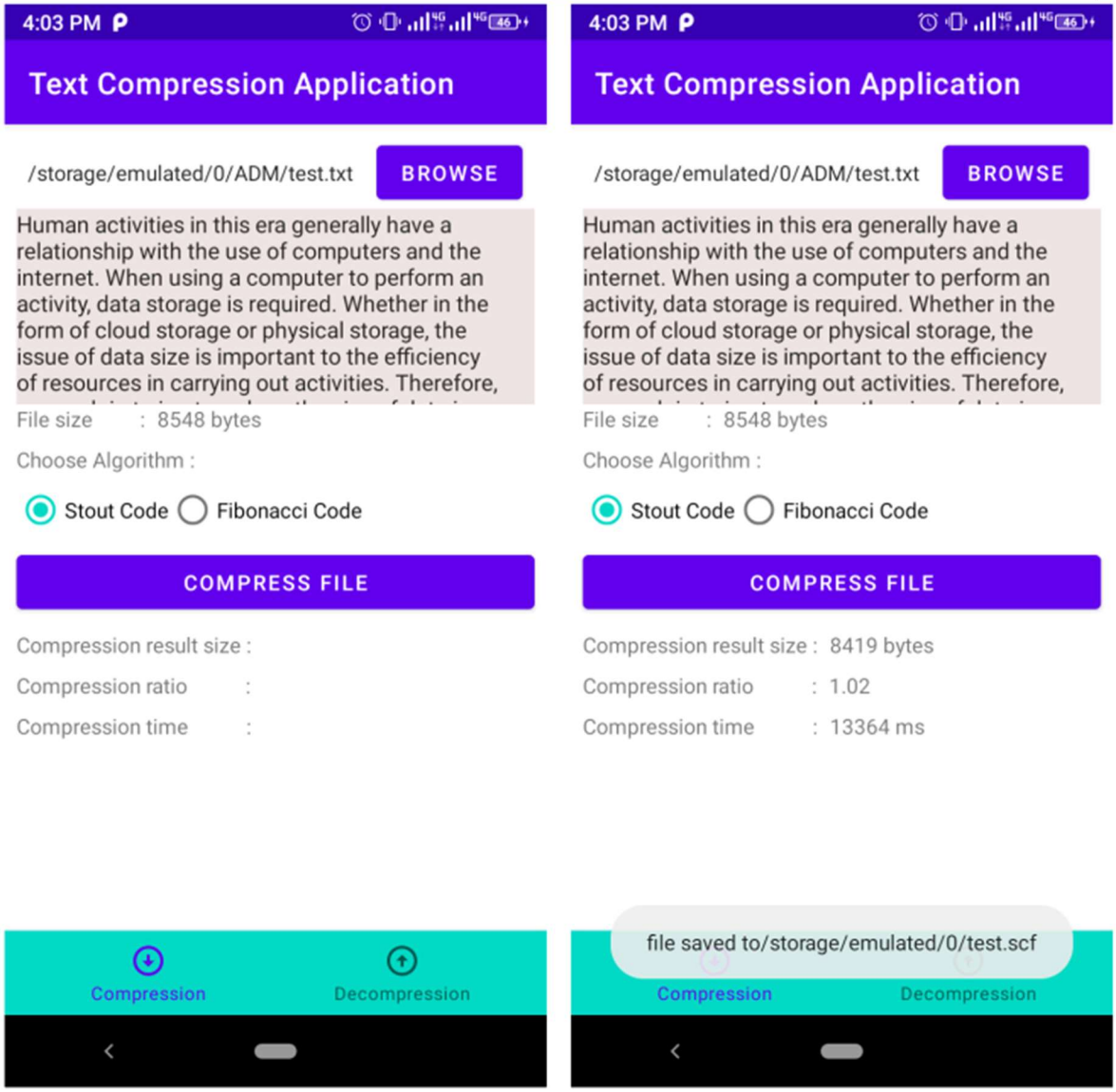

Figure 3. Compression testing

3.3. Decompression testing

In the testing stage of the decompression process, the first thing the user does is enter a text file that has been compressed. Next, the user can choose which algorithm is used to perform the decompression process according to the compressed file extension. After that, the user presses the decompress button to run the decompression process. Then the file will be saved in the root folder of the user's internal Android storage. The system will produce a large calculation of the decompression results and decompression time.

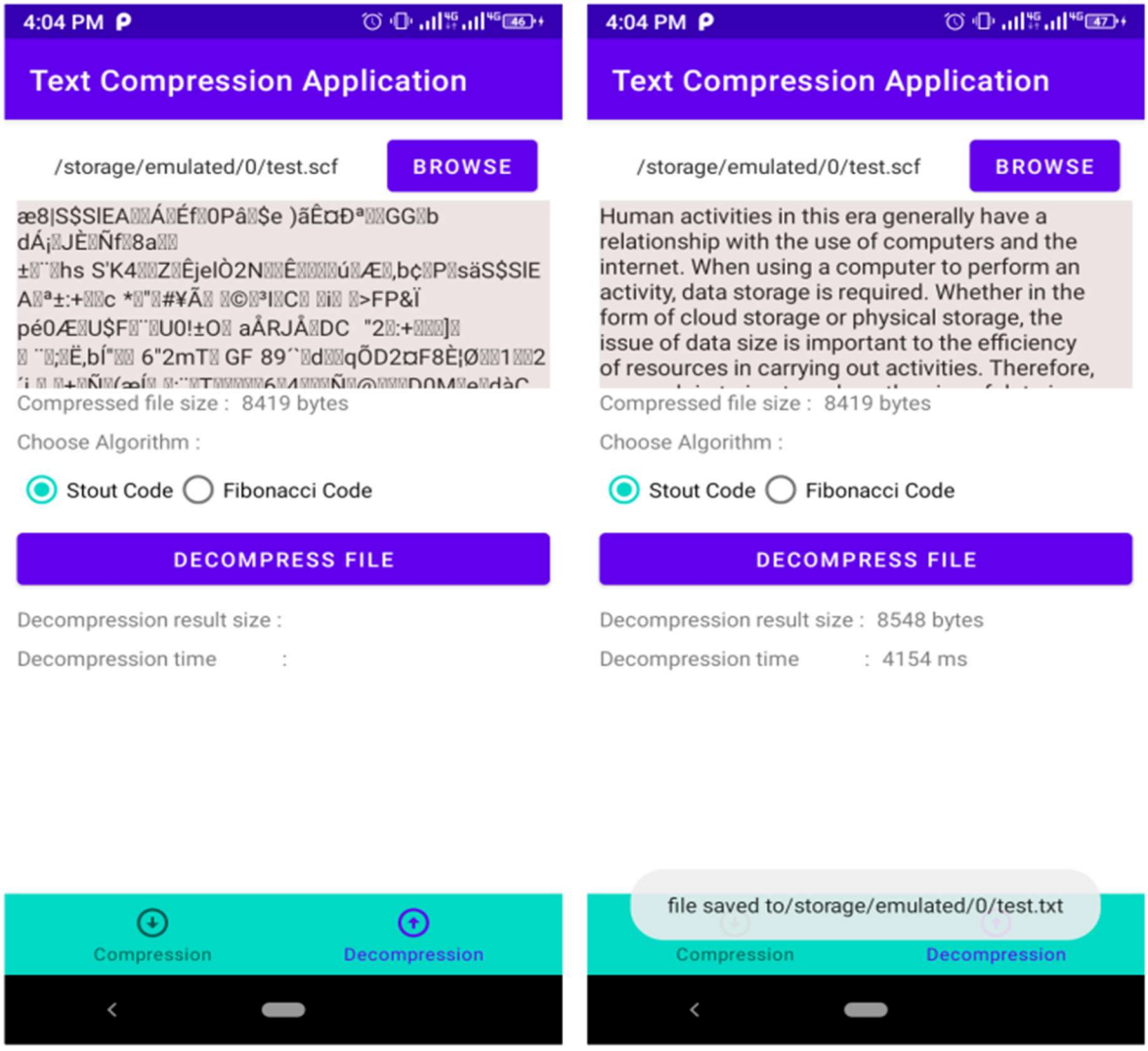

Figure 4. Decompression testing

3.4. Homogenous and Heterogenous String Testing

The result table of String Homogeneous and heterogenous test with Stout code and Fibonacci code can be seen in Table 7, Table 8, Table 9, and Table 10.

Table 7. Homogenous string test results with Stout code

Stout code					
Total Characters	Size Before Compression (bits)	Size After Compression (bits)	Compression Ratio	Compression Time	Decompression Time
100	100	55	1,82	50	14
200	200	106	1,89	33	13
500	500	255	1,96	55	36
1000	1000	506	1,98	155	60
2000	2000	1006	1,99	436	134
5000	5000	2506	2	3222	565
10000	10000	5006	2	14003	2370

Averages	1,949	2564,857	456

Table 8. Homogenous string test results with Fibonacci code

Fibonacci code					
Total Characters	Size Before Compression (bits)	Size After Compression (bits)	Compression Ratio	Compression Time	Decompression Time
100	100	56	1,79	22	17
200	200	106	1,89	20	16
500	500	256	1,95	36	23
1000	1000	506	1,98	134	51
2000	2000	1006	1,99	425	124
5000	5000	2506	2	2700	510
10000	10000	5006	2	13722	1706
Averages			1,943	2437	349,571

Table 9. Heterogenous string test results with Stout code

Stout code					
Total Characters	Size Before Compression (bits)	Size After Compression (bits)	Compression Ratio	Compression Time	Decompression Time
100	100	98	1,02	7	10
200	200	180	1,11	20	20
500	500	428	1,17	47	37
1000	1000	840	1,19	202	131
2000	2000	1665	1,2	606	262
5000	5000	4140	1,21	3563	1342
10000	10000	8265	1,21	16706	5311
Averages			1,159	3021,571	1016,143

Table 10. Heterogenous string test results with Fibonacci code

Fibonacci code					
Total Characters	Size Before Compression (bits)	Size After Compression (bits)	Compression Ratio	Compression Time	Decompression Time
100	100	106	0,94	4	7
200	200	195	1,03	14	16
500	500	466	1,07	60	31
1000	1000	915	1,09	167	106
2000	2000	1815	1,1	544	256
5000	5000	4515	1,11	3663	1114
10000	10000	9015	1,11	15536	4447
Averages			1,064	2855,429	853,857

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

3.5. Difference from prior research

In previous studies, it has been found that the Stout code and Fibonacci code algorithms are suitable for compressing text. The Fibonacci code algorithm was also found to be better than the Even-Rodeh code algorithm for compressing text on both homogeneous and heterogeneous characters. In this study, the authors created an android application to compare the Stout code and Fibonacci code algorithms through parameters such as compression ratio and compression time to determine the effectiveness and efficiency of the two algorithms. Some of the benefits of this research are:

- Understanding how the Stout code algorithm and the Fibonacci code algorithm perform compression and decompression processes.
- Obtaining the results of a performance comparison of the Stout code and Fibonacci code algorithms on Android-based applications.

Getting an application that is capable of compressing and decompressing text files using the Stout code and Fibonacci code algorithms.

4. CONCLUSION

The test results show that based on the compression ratio, the Stout Code algorithm is better with an average of 1,949 for homogeneous strings and 1,159 for heterogeneous strings while the Fibonacci Code algorithm has an average of 1,943 for homogeneous strings and 1,064 for heterogeneous strings.

Based on the compression time, the Fibonacci Code algorithm is better with an average of 2437 ms for homogeneous strings and 2855,429 ms for heterogeneous strings while the Stout Code algorithm has an average of 2564,857 ms for homogeneous strings and 3021,571 ms for heterogeneous strings.

Based on the decompression time, the Fibonacci Code algorithm is also better with an average of 349.571 ms for homogeneous strings and 853.857 ms for heterogeneous strings while the Stout Code algorithm has an average of 456 ms for homogeneous strings and 1016,143 ms for heterogeneous strings.

The results of the compression test on both homogeneous and heterogeneous text files lead to the conclusion that the Stout code algorithm outperforms in reducing file sizes, whereas the Fibonacci code algorithm excels in terms of speed.

REFERENCES

[1]. Wang, L., Zhang, X., Yang, K., Yu, L., Li, C., Hong, L., ... & Zhu, J. (2022). Memory replay with data compression for continual learning. arXiv preprint arXiv:2202.06592.

[2]. Jayasankar, U., Thirumal, V., & Ponnurangam, D. (2021). A survey on data compression techniques: From the perspective of data quality, coding schemes, data type, and applications. Journal of King Saud University-Computer and Information Sciences, 33(2), 119-140.

[3]. Nasution, S. D. (2019). Data Compression Using Stout Codes. The IJICS (International Journal of Informatics and Computer Science), 3(1), 28-33.

[4]. Bhattacharyya, S. (2017). Complexity analysis of a lossless data compression algorithm using Fibonacci sequence. International Journal of Information Technology (IJIT), 3(3).

[5]. Rachmawati, D., Budiman, M. A., & Subada, M. A. (2019, October). Comparison study of Fibonacci code algorithm and Even-Rodeh algorithm for data compression. In Journal of Physics: Conference Series (Vol. 1321, No. 3, p. 032015). IOP Publishing.

[6]. Otair, M., Abualigah, L., & Qawaqzeh, M. K. (2022). Improved near-lossless technique using the Huffman coding for enhancing the quality of image compression. Multimedia Tools and Applications, 81(20), 28509-28529.

[7]. Barman, R., Badade, S., Deshpande, S., Agarwal, S., & Kulkarni, N. (2022). Lossless data compression method using deep learning. In Machine Intelligence and Smart Systems: Proceedings of MISS 2021 (pp. 145-151). Singapore: Springer Nature Singapore.

[8]. Hughes, J. (2023). Comparison of lossy and lossless compression algorithms for time series data in the Internet of Vehicles.

[9]. Hardi, S. M., Angga, B., Lydia, M. S., Jaya, I., & Tarigan, J. T. (2019, June). Comparative analysis run-length encoding algorithm and fibonacci code algorithm on image compression. In Journal of Physics: Conference Series (Vol. 1235, No. 1, p. 012107). IOP Publishing.