Comparison study of Fibonacci code algorithm and Even-Rodeh algorithm for data compression

D Rachmawati*, M A Budiman and M A Subada

Computer Sciences Department, Computer Sciences and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia

*Corresponding author: dian.rachmawati@usu.ac.id

Abstract. The need for greater storage capacity is the cause of various compression techniques. By compressing, large data will be reduced in size to save storage capacity. This research will use Even-Rodeh code algorithm and Fibonacci code algorithm to compress the data. The performance of algorithms will be measured by Bitrate, Compression Ratio, Space Saving, and Compression Time (millisecond) and Time Decompression (millisecond). Compression is done by reading String in the text file, then Even-Rodeh Code algorithm and Fibonacci code algorithm create String code and perform compression process. The result of compression is the file with extension *. C and *. Fib which contains character information and String Bit. The results of compression can be decompressed to the original text file. The system test used a sample of Strings consisting of one type of character (Homogen String) and String comprising more than one kind of character (Heterogenous String) is stored in the text file by extension * .doc, * .docx, * .pdf. From the compression process on the homogenous String, the Fibonacci code algorithm is more efficient about the compression ratio of 0.25 and decompression time is 0.011 milliseconds compared with the Even-Rodeh code algorithm. While in the compression process on heterogeneous String Fibonacci Code algorithm is better than the Even-Rodeh code algorithm with an average Compression Ratio of 0.65 and decompression time faster with an average of 0.293 milliseconds.

1. Introduction

The need for greater storage capacity is the cause of the emergence of various compression techniques. By doing compression, large data will decrease in size. In addition to storage problems in sending data size data greatly affects delivery [1]. File sending can be done quickly if the file is small in size, but not all data which people send have a small size of capacity so that we need a large capacity memory to store all the data needed and an unstable internet connection speed requires a relatively long time in the process of sending data. Therefore, in large-scale data transmission, the compression process is also needed. Data compression has been widely applied to various types of data, text, images, and video [2].

Compression is useful for minimizing files that are large. Fibonacci code compression method is a type of compression method based on coding with the Universal code system where the positive value of the integer of the data to form a binary code word by utilizing the Fibonacci code number [3].

Another algorithm that can be used for known compression is the Even-Rodeh code algorithm. Even-Rodeh code algorithm is a data compression algorithm that encodes characters by using a series of bits representing each character based on the frequency of each character. The Even-Rodeh code algorithm is a compression algorithm developed by Shimon Even and Michael Rodeh code in 1978 [4].
In this study, a comparison will be made to the two algorithms namely Fibonacci code and Even-Rodeh code to compress data.

2. Methods

2.1. Compression

The size of the data used by humans will require a very large place. Today many programmers are trying to reduce data (data compression). The purpose of reducing the data is to save storage space (memory) so that the storage area does not quickly exceed the capacity and the process of data transfer will be faster. Traditional compression algorithms typically operate on individual bytes [5].

Compression is a stage or process to change the flow of new data flow data that has a smaller data size (data that has been compressed) [6]. The data compression algorithm has two outlines, Lossless compression and Lossy compression [7].

1. Lossless compression is a data compression method that allows the original data to be completely reconstituted, without losing any information. Lossless compression has a lower degree of compression, but the data accuracy maintained between before and after the compression process.

2. Lossy compression is a data method that does not produce original data before compression [8]. Although the difference is quite close, this type of compression is not good for data compression like text data but is often used in audio data, files, and images. This method provides a higher degree of compression.

2.2. Fibonacci codes algorithm

Leonardo Pisano Fibonacci was an Italian mathematician, considered a great medieval Matematiater; he was instrumental in reviving ancient mathematics. In his book Liber Abaci, introducing Europe with Hindu-Arabic notation for numbers.

Table 1. The table of Fibonacci codes

| Fibonacci Code Table |
|----------------------|
| 1  11     7  1011  |
| 2  011    8  000011 |
| 3  0011   9  100011 |
| 4  1011   10  010011 |
| 5  00011 11  001011 |
| 6  10011  12  101011 |

Currently, the Fibonacci series is one of the famous mathematical objects. Fibonacci numbers are defined as numbers whose terms are the sum of the two previous terms [9]. The Fibonacci number can be shown as a number sequence number: 0,1,1,2,3,5,8,13,21,34,55,89,144, ... Table 1 is the Fibonacci code table [10].

2.3. Even-Rodeh code algorithm

One known compression algorithm is the Even-Rodeh code algorithm. Even-Rodeh code Algorithm is a data compression algorithm that encodes characters with a series of bits representing characters created based on the frequency of each character. The Even-Rodeh code algorithm is a compression algorithm developed by Shimon Even and Michael Rodeh in 1978 [7].

The stage of building the Even-Rodeh code with n as an index of characters is as follows [11]:

1. Calculate bit length.
2. If the length of the bit is 0 <= n <= 3 then the value of n is changed to binary, adding 0 in front of the binary value so that the bit becomes 3 digits.
3. If the bit length is \(4 \leq n \leq 7\) then the value of \(n\) is converted to binary, adding 0 behind the binary value so that the bit becomes 4 digits.

4. If the length of the bit \(n \geq 8\) then the value of \(n\) is changed to binary, add 0 behind the binary value then the number is added in front of the binary value as many as the number of digits in the binary value.

5. Some of the Even-Rodeh code codes and the number of Even-Rodeh code codes based on the frequency of characters can be seen in Table 2 and Table 3.

| \(N\) | Even-Rodeh Code |
|------|------------------|
| 0    | 000              |
| 1    | 001              |
| 2    | 010              |
| 3    | 011              |
| 4    | 100 0            |
| 7    | 111 0            |
| 8    | 100 1000 0       |
| 15   | 100 1111 0       |
| 16   | 100 10000 0      |
| 32   | 110 10000 0      |
| 100  | 111 1100100 0    |

### Table 2. Even-Rodeh code algorithm

| \(N\) | Even-Rodeh Code |
|------|------------------|
| 0    | 3               |
| 4-7  | 4               |
| 8-15 | 8               |
| 16-31| 9               |
| 32-63| 10              |
| 61-127| 11             |

### Table 3. Number of codes the Even Rodeh algorithm based on character variations

3. **Results and Discussion**

There is a text file containing strings "MHD ALI SUBADA ILMU KOMPUTER." The size of original string is shown in Table 4.

Based on the ASCII code, one character is worth eight bits of a binary number. So that 28 characters on the string have a binary value of 224 bits. Before performing the compression process, characters are first sorted from the largest to the smallest frequency.

Compression analysis process of text files using the Fibonacci Code algorithm. 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 "MHD ALI SUBADA COMPUTER SCIENCE." It is shown in Table 5. string size that has been compressed using the Fibonacci Code algorithm.
Table 4. Size of string before compressed

| Character | Frequency | ASCII Code | ASCII Code (Binary) | Bits x Frequency |
|-----------|-----------|------------|---------------------|------------------|
| SP        | 4         | 32         | 00100000            | 8 x 32           |
| M         | 3         | 77         | 01001101            | 8 x 24           |
| A         | 3         | 65         | 01000001            | 8 x 24           |
| U         | 3         | 85         | 01010101            | 8 x 24           |
| D         | 2         | 68         | 01000100            | 8 x 16           |
| L         | 2         | 76         | 01001100            | 8 x 16           |
| I         | 2         | 73         | 01001001            | 8 x 16           |
| H         | 2         | 72         | 01001000            | 8 x 8            |
| SP        | 1         | 83         | 01010011            | 8 x 8            |
| B         | 1         | 66         | 01000010            | 8 x 8            |
| K         | 1         | 75         | 01001011            | 8 x 8            |
| O         | 1         | 79         | 01001111            | 8 x 8            |
| P         | 1         | 80         | 01010000            | 8 x 8            |
| T         | 1         | 84         | 01010100            | 8 x 8            |
| E         | 1         | 69         | 01000101            | 8 x 8            |
| R         | 1         | 82         | 01010010            | 8 x 8            |

Total of Bits: 224

And the next is to rearrange the codes in Table 6 according to the position of the characters in the string and sorted from the largest to the smallest frequency. So, the bit string that has been compressed is

“01100001110001110001110001110011001101011110011001101000110011011011011011011011000011010100000011110010111010110110000101110111000011010001101000110010011”.

First, add the padding bits and flags before the results of the compression are written into the file. Adding to the bit padding and flag bit is done by referring to the remaining number of bits divided by 8. The sum of the results of compression is 129 bits; because 129 is not divisible by 8, it can be obtained padding0000000.

“0110000111000000111100011100110110110110011001101101011100110100110100110110110110110110000110101000000010111000011010001101000110000000".

From the addition of 7 bits of padding, flag bit 000000111 is obtained, and a new string is obtained as follows:

“0110000111000111000111001101011110001110110101011001100110110110110100011010100000001111001011101011011000010111011100001101000110100011000000000001111”

Then the total bits obtained are 144 bits. From the results of compression with the Fibonacci Code algorithm above, the compression performance can be calculated, namely:

\[
\text{Bitrate} = \frac{\text{compressed bit size}}{\text{number of the unique symbol in the text}}
\]

\[
\text{Bitrate} = \frac{144}{16} = 9
\]
Table 5. String size that has been compressed by using the Fibonacci code algorithm

| Character | Frequency | Fibonacci code | Bits | Bit x Frequency |
|-----------|-----------|----------------|------|-----------------|
| Sp        | 4         | 11             | 3    | 8               |
| M         | 3         | 011            | 3    | 9               |
| A         | 3         | 0011           | 4    | 12              |
| U         | 3         | 1011           | 4    | 12              |
| D         | 2         | 00011          | 5    | 10              |
| L         | 2         | 10011          | 5    | 10              |
| I         | 2         | 01011          | 5    | 10              |
| H         | 1         | 000011         | 6    | 6               |
| S         | 1         | 100011         | 6    | 6               |
| B         | 1         | 010011         | 6    | 6               |
| K         | 1         | 001011         | 6    | 6               |
| O         | 1         | 101011         | 6    | 6               |
| P         | 1         | 0000011        | 7    | 7               |
| T         | 1         | 1000011        | 7    | 7               |
| E         | 1         | 0100011        | 7    | 7               |
| R         | 1         | 0010011        | 7    | 7               |

**TOTAL of BITS** 129

b. Compression Ratio (CR)

\[
CR = \frac{\text{the size data after compressed}}{\text{the size data before compressed}}
\]

\[
CR = \frac{144}{224}
\]

CR = 0.64

c. Space Saving

\[
Space\ Savings = \left(1 - \frac{\text{the size of the data that has been compressed}}{\text{the size of the data before it compressed}}\right) \times 100\%
\]

\[
Space\ Savings = \left(1 - \frac{144}{224}\right) \times 100\%
\]

Space Saving = 36%

The compression process for the Even-Rodeh Code algorithm can be seen in Table 6.

The next step, rearranging the characters, strings, and codes in the table. The string that has been read is “MHD ALI SUBADA ILMU KOMPUTER”.

So, the result of the string bit being compressed is “0011110100000001010101100000111001001001000010001100100101100011001001110010011100100111100000000000110”.

First, add the padding and flag bits before the results of the compression are written into the file. Adding to the bit padding and bit flag is done by referring to the remaining number of bits divided by 8. The sum of the compression results is 130 bits; because 130 is not divisible by 8, it can be obtained padding 000000.

“0011110100000001010101100000111001001001000010001100100101100011001001110010011110000000000000110."

From the addition of 6 bits of padding, flag bit 000000110 is obtained. And a new String bit is obtained, namely;

“00111101000000010101011000001001000011100100100100001000110010011001011000110010011111100000000000110.”

Then the total bits obtained are 144 bits.
Table 6. String size after compressed with Even-Rodeh code

| n | Character | Frequency | Even - Rodeh Code | Bits | Bit x Frequency |
|---|-----------|-----------|-------------------|------|----------------|
| 0 | Sp        | 4         | 000               | 3    | 12             |
| 1 | M         | 3         | 001               | 3    | 9              |
| 2 | A         | 3         | 010               | 3    | 9              |
| 3 | U         | 3         | 011               | 3    | 9              |
| 4 | D         | 2         | 1000              | 4    | 8              |
| 5 | L         | 2         | 1010              | 4    | 8              |
| 6 | I         | 2         | 1100              | 4    | 8              |
| 7 | H         | 1         | 1110              | 4    | 4              |
| 8 | S         | 1         | 100 100 0         | 7    | 7              |
| 9 | B         | 1         | 100 1001 0        | 8    | 8              |
| 10 | K       | 1         | 100 1010 0       | 8    | 8              |
| 11 | O       | 1         | 100 1011 0       | 8    | 8              |
| 12 | P       | 1         | 100 1100 0       | 8    | 8              |
| 13 | T       | 1         | 100 1101 0       | 8    | 8              |
| 14 | E       | 1         | 100 1110 0       | 8    | 8              |
| 15 | R       | 1         | 100 1111 0       | 8    | 8              |

Total of Bits 130

From the compression results with the Even-Rodeh Code algorithm, the compression performance can be calculated, namely:

a. **Bitrate**

\[
\text{Bitrate} = \frac{\text{compressed bit size}}{\text{number of unique symbols in the text}}
\]

\[
\text{Bitrate} = \frac{144}{16} = 9
\]

b. **Compression Ratio (C_R)**

\[
C_R = \frac{\text{the size of the data after being compressed}}{\text{the size of the data before it compressed}} \times 100\%
\]

\[
C_R = \frac{144}{224} = 0.636
\]

c. **Space Saving**

\[
\text{Space Saving} = \left(1 - \frac{\text{the size of the data that has been compressed}}{\text{the size of the data before it is compressed}}\right) \times 100\%
\]

\[
\text{SS} = \left(1 - \frac{144}{192}\right) \times 100\% = 36\%
\]

3.1. **Homogeneous String Testing**

The result table of String Homogeneous test with Fibonacci code and Even-Rodeh code can be seen in Table 7, Table 8, Table 9, and Table 10.
### Table 7. String homogen test results with Fibonacci code

| Amount of Char | Type of Char | Bit Before Compression | Bit Compressed | Bitrate | C<sub>R</sub> | SS | Compression Time | Decompression Time |
|----------------|--------------|-------------------------|----------------|---------|------------|----|-----------------|------------------|
| 200            | 1            | 408                     | 136            | 408     | 0.25       | 0.62| 0.002ms         | 0.006ms          |
| 300            | 1            | 608                     | 204            | 608     | 0.25       | 0.62| 0.003ms         | 0.007ms          |
| 400            | 1            | 808                     | 286            | 808     | 0.25       | 0.62| 0.005ms         | 0.009ms          |
| 500            | 1            | 1008                    | 336            | 1008    | 0.25       | 0.62| 0.006ms         | 0.011ms          |
| 600            | 1            | 1208                    | 404            | 1208    | 0.25       | 0.62| 0.008ms         | 0.016ms          |
| 700            | 1            | 1408                    | 468            | 1408    | 0.25       | 0.62| 0.011ms         | 0.029ms          |
| Average        |              |                         |                |         |            |     | 0.005ms         | 0.011ms          |

### Table 8. Homogeneous string testing results with Even-Rodeh code

| Amount of Char | Type of Char | Bit Before Compression | Bit Compressed | Bitrate | C<sub>R</sub> | SS | Compression Time | Decompression Time |
|----------------|--------------|-------------------------|----------------|---------|------------|----|-----------------|------------------|
| 200            | 1            | 1600                    | 104            | 608     | 0.38       | 0.62| 0.002ms         | 0.004ms          |
| 300            | 1            | 2400                    | 152            | 912     | 0.38       | 0.62| 0.003ms         | 0.006ms          |
| 400            | 1            | 3200                    | 204            | 1208    | 0.38       | 0.62| 0.004ms         | 0.012ms          |
| 500            | 1            | 4000                    | 252            | 1512    | 0.38       | 0.62| 0.004ms         | 0.013ms          |
| 600            | 1            | 4800                    | 304            | 1808    | 0.38       | 0.62| 0.006ms         | 0.010ms          |
| 700            | 1            | 5600                    | 352            | 2112    | 0.38       | 0.62| 0.007ms         | 0.074ms          |
| Average        |              |                         |                |         |            |     | 0.004ms         | 0.019ms          |

### Table 9. Heterogeneous string test results with Fibonacci code

| Amount of Char | Type of Char | Bit Before Compression | Bit Compressed | Bitrate | C<sub>R</sub> | SS | Compression Time | Decompression Time |
|----------------|--------------|-------------------------|----------------|---------|------------|----|-----------------|------------------|
| 200            | 36           | 1600                    | 452            | 48      | 0.65       | 0.27| 0.013ms         | 0.036ms          |
| 300            | 36           | 2400                    | 756            | 82      | 0.65       | 0.25| 0.023ms         | 0.027ms          |
| 400            | 36           | 3200                    | 996            | 106     | 0.65       | 0.27| 0.026ms         | 0.041ms          |
| 500            | 36           | 4000                    | 1228           | 133     | 0.65       | 0.27| 0.031ms         | 0.05ms           |
| 600            | 36           | 4800                    | 1472           | 160     | 0.65       | 0.26| 0.037ms         | 0.057ms          |
| 700            | 36           | 5600                    | 1732           | 188     | 0.65       | 0.26| 0.04ms          | 0.064ms          |
| Average        |              |                         |                |         |            |     | 0.028ms         | 0.045ms          |

### 4. Conclusion

From the second test, the algorithm can reduce the file size and restore the original file. For the same (homogeneous) character based on Compression Ratio (C<sub>R</sub>). Fibonacci code algorithm is better than Even-Rodeh code algorithm with average Compression Ratio of 0.25%. While different characters (heterogeneous) Fibonacci code algorithm is better than Even-Rodeh code algorithm with Compression Ratio 0.65%. Regarding time for the same (homogeneous) data, the Even-Rodeh code algorithm is better for Returns the former compressed file to the original text file, with an average of 0.004 milli seconds. For different (heterogeneous) characters the Fibonacci code algorithm is better for recovering text files from the original file compression, with an average of 0.279 milli seconds.
Table 10. Test results of heterogeneous strings with Even-Rodeh code

| Amount of Kar | Type of Kar | Bit Before Compressed | Bit Compressed | Bitrate | $C_R$ | SS | Compression Time | Decompression Time |
|--------------|-------------|------------------------|----------------|---------|------|----|-----------------|-------------------|
| 200          | 36          | 1600                   | 460            | 53      | 0,73 | 0,27 | 0,029ms         | 0,02ms            |
| 300          | 36          | 2400                   | 792            | 92      | 0,75 | 0,25 | 0,027ms         | 0,03ms            |
| 400          | 36          | 3200                   | 1008           | 118     | 0,73 | 0,27 | 0,029ms         | 0,044ms           |
| 500          | 36          | 4000                   | 1284           | 149     | 0,73 | 0,27 | 0,033ms         | 0,057ms           |
| 600          | 36          | 4800                   | 1540           | 179     | 0,74 | 0,26 | 0,04ms          | 0,067ms           |
| 700          | 36          | 5600                   | 1804           | 210     | 0,74 | 0,26 | 0,042ms         | 0,071ms           |
| Average      |             |                        |                | 133     | 0,73 | 0,2  | 0,033ms         | 0,048ms           |

5. Acknowledgments
We gratefully acknowledge that this research is supported by Fund Dissemination IPTEKS Research Results for Lecturers / Researchers Universitas Sumatera Utara.

References
[1] Mantoro T, Ayu M A and Anggraini Y 2017 Int. Conf. Comput. Eng. Des. 1
[2] Walder J, Krátký M, Bača R, Platoš J and Snášel V 2012 Inf. Sci. 183 66
[3] Yang H and Fu R 2011 Int. Conf. Multimed. Technol. Hangzhou 1969
[4] David S 2007 Variabel-Lengh Codes for Data Compression (USA: Computer Science Department California State University)
[5] Gleave A and Steinrueven C 2017 Proc. Data Compress. Conf. Snowbird, UT, USA
[6] David S and Giovanni M 2010 Handbook of Data Compression. Fifth Edition (London: Computer Science Department California State University)
[7] Budiman M A and Rachmawati D 2017 IOP Conf. Ser.: Mater. Sci. Eng. 180 012062
[8] Rachmawati D, Budiman A B and Saffiera CA 2018 IOP Conf. Ser.: Mater. Sci. Eng. 300 012040
[9] Klein S T and Ben-Nissan M K 2009 Comput. J. 53 701
[10] Klein S T and Shapira D 2018 Proc. Data Compress. Conf. Snowbird, UT, USA
[11] Rachmawati D, Budiman M A and Batubara M M 2018 Int. Conf. Comput. Control Inform. Appl. 54