Journal of Physics: Conference Series

PAPER • **OPEN ACCESS**

Comparative Analysis Run-Length Encoding Algorithm and Fibonacci Code Algorithm on Image Compression

To cite this article: S M Hardi *et al* 2019 *J. Phys.: Conf. Ser.* **1235** 012107

IOP Conf. Series: Journal of Physics: Conf. Series **1235** (2019) 012107 doi:10.1088/1742-6596/1235/1/012107

Comparative Analysis Run-Length Encoding Algorithm and Fibonacci Code Algorithm on Image Compression

S M Hardi[1*], B Angga [2], M S Lydia [3], I Jaya [4], J T Tarigan [5]
[1,2,3,4,5] Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia.

*Email: vani.hardi@usu.ac.id , bayuangga29@gmail.com

Abstract. Compression purpose to reduce the redundancy data as small as possible and speed up the data transmission process. To solve the size problem in saving data and transmission process, we use Run Length Encoding and Fibonacci Code algorithm to do compression process. Run Length Encoding and Fibonacci Code algorithm is a type of lossless data compression used in this research, which performance will be measured by comparison parameters *of the Compression Ratio* (CR), *Redundancy* (RD), *Space Saving* (SS) and Compression Time. The compression process is only done on image files with Bitmap format (*.bmp) and encode using Run Length Encoding or Fibonacci Code, then perform the compression process. The final result of the compression is file with extension *.rle or *.fib which contains compressed information that can be decompressed back. The output of the decompression result is an original image file that is stored with *.bmp extension. Fibonacci algorithm will give a better compressed size on image color, while in a grayscale image Run Length Encoding will give a better compressed size. Based on the results of research at two different types of images, each algorithm has its own advantages. Fibonacci Code algorithm is better for color image compression while Run-Length algorithm Encoding is better for grayscale image compression.

1. Introduction

Data compression is the process of converting an input data stream (the source stream or the original raw data) into another data stream (the output, the bit stream, or the compressed stream) that has a smaller size. Data compression is popular for two reasons, people like to accumulate data and hate to throw anything away. People hate to wait a long time for data transfers. When sitting at the computer, waiting for a Web page to come in or for a file to download, naturally feel that anything longer than a few seconds is a long time to wait [1].

There are two major families of compression techniques when considering the possibility of reconstructing exactly the original source. They are called lossless and lossy compression [4]. Lossless compression techniques, as their name implies, involve no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossy compression techniques involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly [2].

Run-Length Encoding algorithm is a type of lossless data compression, Run-Length Encoding is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run [3]. The Fibonacci coding is a data compression technique that based on Fibonacci series. It produces static variable length code for representing the data, the Fibonacci numbers are the numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones: 1;1;2;3;5;8;13;21;34;55;89;144 [2]. Benefit of image compression is to reduce the amount of data required for representing sampled digital images and therefore reduce the cost for storage and transmission [3].

2. Method

In this implementation there are 2 main menus namely: Compression to compress original image and Decompression to decompress compressed image back to the original image.

2.1. *Fibonacci Code Algorithm*

Steps for compression:

1. Get all the frequency of all color value on each pixel in the selected image.
2. Sort the pixel value by the frequencies in descending order.
3. Compute the Fibonacci code of each ranking.
4. Output the ranking as the header of the compressed file.
5. Reread the input file, using the code table to generate output to the compressed file.

Steps for decompression:

1. Read compressed file data.
2. Read compressed file value character by character until the number 11 found.
3. Then the entire binary code decoded to corresponding unique character.

250	250	7	7	7	7	250	250	250	250
254	254	1	1	1	7	7	7	7	7
7	7	7	7	3	3	3	3	3	3
254	254	5	5	5	5	250	250	250	250
5	5	5	5	246	246	246	246	246	246
7	7	7	246	246	246	246	250	250	250
254	254	3	3	3	3	3	5	5	5
7	7	7	7	7	7	254	254	254	254
0	0	0	0	0	0	0	0	0	0
1	1	1	254	254	250	250	5	5	5

Figure 1. Sample of 8bit grayscale image value

From Figure 1, the steps for compression of the pixel are as follows:

2.1.1. Compression Process

Table 1. Fibonacci Code [2]

Position	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Rank's Fibonacci Representation**	**Rank_ Fibonacci_ Code =+{suffix 1}**
Fibonacci _Value F(n)	**1**	**2**	**3**	**5**	**8**	**13**	**21**	**34**	**55**	**89**	**144**		
Rank													
1	1											1	11
2	0	1										01	011
3	0	0	1									001	0011
4	1	0	1									101	1011
5	0	0	0	1								0001	00011
6	1	0	0	1								1001	10011
7	0	1	0	1								0101	01011
8	0	0	0	0	1							00001	000011
9	1	0	0	0	1							10001	100011
10	0	1	0	0	1							01001	010011
...													
147	1	1	0	0	0	0	0	0	0	0	1	11000000001	110000000011

a. Create a table based on pixel frequency (descending order) as showed in table 2.

Table 2. Pixel Frequency Distribution

Value	Binary	bit	Frequency	bit x Frequency
7	00000111	8	22	176
250	11111010	8	15	120
5	00000101	8	14	112
254	11111110	8	12	96
3	00000011	8	11	88
0	00000000	8	10	80
246	11110110	8	10	80
1	00000001	8	6	48
	Code Word Total Bits			800

b. Compute the Fibonacci code of each ranking and output the ranking as the header of the compressed file.

Table 3. Fibonacci code representation of frequency of pixel sample

Value	Frequency	Fibonacci Code	Bit	Bit x Frequency
7	22	11	2	44
250	15	011	3	45
5	14	0011	4	56
254	12	1011	4	48
3	11	00011	5	55
0	10	10011	5	50
246	10	01011	5	50
1	6	000011	6	36
	Code Word Total Bits			384

c. Compressed data

Reread the input file, using the code table to generate output to the compressed file, finally the result of compression is
“01101111111111101101110111011000011000011000011111111111111111111100011000110001100011000110
0011000111011101100110011001100110110110110110011001100110011011011011011001100110
0110101101011010110101101011010111111110101101011010110101101101101110111011000110 0011
00011000110001100110011001111111111111111011101110111011100111001110011100111001110 0111
001110011100111001100001100001100001110111011011011001100110011”
Uncompressed size = 10 x 10 x 8bit = 800 bit
Compressed size = 384 bit
The compression ratio and space savings calculated for input data set:
Compression Ratio= 384:800 = 0.48:1
Space Savings =1-(384/800) =0.52 = 52%

2.1.2. Decompression Process

a. The decoding process read the input, one by one until the number 11 found.

Step 1:
“01101111111111101101110111011000011000011000011111111111111111111100011000110001100011000110
0011000111011101100110011001100110110110110110011001100110011011011011011001100110
0110101101011010110101101011010111111110101101011010110101101101101110111011000110 0011
00011000110001100110011001111111111111111011101110111011100111001110011100111001110 0111
001110011100111001100001100001100001110111011011011001100110011”

IOP Conf. Series: Journal of Physics: Conf. Series **1235** (2019) 012107 doi:10.1088/1742-6596/1235/1/012107

b. Read table of fibonacci code of each ranking and change the value.

Step 2:
"2500111111111110110111011101100001100001100001111111111111111111100011000110001100011000110001110111011001100110011001101101101101100110011001100110110110110110011001100110011010110101101011010110101101011111111010110101101011010110110110111011101100011000110001100011000110011001100111111111111111011101110111011100111001110011100111001110011100111001110011100111001100001100001100001110111011011011001100110011"

Step 20:
"2502507777250250250250101110110000110000110000111111111111111111110001100011000110001100011000111011101100110011001100110110110110110011001100110011011011011011001100110011001101011010110101101011010110101111111101011010110101101011011011011101110110001100011000110001100011000110011001100111111111111111011101110111011100111001110011100111001110011100111001110011100111001100001100001100001110111011011011001100110011"

Step 200:
"25025077772502502502502542541117777777777333333254254555525025025025055552462462462462462467772462462462462502502502542543333355577777725425425425400000000001112542542502505 55"

2.2. *Run-Length Encoding Algorithm*

Steps for compression:

1. Check of current value with the neighbors value, when current value same with the neighbors , then combine the values into one , and add counter to the values.
2. Go to the next value, if current value is not same with the neighbors value then save current value and repeat the first step back.
3. After process 1 and 2 finished, then save compression result.

Steps for decompression:

1. Check the current value with the neighbors value, neighbors value is the amount of current value.
2. Wrote the current value as much as the neighbor's value who has been checked.
3. Go to the next value and repeat the first step and second step until all back to the original values.

From Figure 1, the steps for compression of the pixel are as follows:

2.2.1. Compression Process

a. Check of current value with the neighbors value, when current value same with the neighbors , then combine the values into one , and add counter to the values.

Step 1:
"(250,2)777725025025025025425411177777777773333332542545555250250250250555524624624624624624677724624624624625025025025425433333555777777254254254254000000000011125425425025055 5"

b. Go to the next value, if current value is not same with the neighbors value then save current value and repeat the first step back.

Step 2:
"(250,2)(7,4)250250250250254254111777777777733333325425455552502502502505555246246246246246246777246246246246250250250254254333335557777772542542542540000000000111254254250250555"

Step 3:
"(250,2)(7,4)(250,4)254254111777777777733333325425455552502502502505555246246246246246246777246246246246250250250254254333335557777772542542542540000000000111254254250250555"

Step 26:
"(250,2)(7,4)(250,4)(254,2)(1,3)(7,5)(7,4)(3,6)(254,2)(5,4)(250,4)(5,4)(246,6)(7,3)(246,4)(250,3)(254,2)(3,5)(5,3)(7,6)(254,4)(0,10)(1,3)(254,2)(250,2)(5,3)"

c. After process 1 and 2 finished, then save compression result.
Compression result:
250 2 7 4 250 4 254 2 1 3 7 5 7 4 3 6 254 2 5 4 250 4 5 4 246 6 7 3 246 4 250 3 254 2 3 5 5 3 7 6 254 4 0 10 1 3 254 2 250 2 5 3
Total = 52 pixel
Uncompressed size = 10 x 10 x 8bit = 800 bit
Compressed size = 52 x 8bit = 416 bit
The compression ratio and space savings calculated for input data set:
Compression Ratio= 416:800 = 0.52:1
Space Savings =1-(416/800) =0.48 = 48%

2.2.2. Decompression Process
a. Check the current value with the neighbors value, neighbors value is the amount of current value.
Step 1:
"(250,2)(7,4)(250,4)(254,2)(1,3)(7,5)(7,4)(3,6)(254,2)(5,4)(250,4)(5,4)(246,6)(7,3)(246,4)(250,3)(254,2)(3,5)(5,3)(7,6)(254,4)(0,10)(1,3)(254,2)(250,2)(5,3)"

b. Wrote the current value as much as the neighbors value who has been checked.
Step 2:
"250250(7,4)(250,4)(254,2)(1,3)(7,5)(7,4)(3,6)(254,2)(5,4)(250,4)(5,4)(246,6)(7,3)(246,4)(250,3)(254,2)(3,5)(5,3)(7,6)(254,4)(0,10)(1,3)(254,2)(250,2)(5,3)"

c. Go to the next value and repeat the first step and second step until all back to the original values.
Step 3:
"250250(7,4)(250,4)(254,2)(1,3)(7,5)(7,4)(3,6)(254,2)(5,4)(250,4)(5,4)(246,6)(7,3)(246,4)(250,3)(254,2)(3,5)(5,3)(7,6)(254,4)(0,10)(1,3)(254,2)(250,2)(5,3)"
Step 4:
"2502507777(250,4)(254,2)(1,3)(7,5)(7,4)(3,6)(254,2)(5,4)(250,4)(5,4)(246,6)(7,3)(246,4)(250,3)(254,2)(3,5)(5,3)(7,6)(254,4)(0,10)(1,3)(254,2)(250,2)(5,3)"
Step 52:
"2502507777250250250250254254111777777777733333325425455552502502502505555246246246246246246777246246246246250250250254254333335557777772542542542540000000000111254254250250555"

3. Result and Discussion

The experiments are conducted on the Windows 8.1 Notebook which has AMD A10 processor with 64-bit architecture and 8192MB RAM. The development environment being used for coding C# scripts is Visual Studio 2012. The results of the experiments of each set are presented in Tables 4, 5, 6 and 7 as follows.

Table 4. The Compression result using Fibonacci Code for 4 sample color images (RGB)

Resolution (pixel)	Original Size (kb)	Compressed Size (kb)	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
100 x 100	300.56	242.79	80.78	19.22	15.6492
200 x 200	800.56	584.23	72.98	27.03	15.6259
300 x 300	2700.56	2312.29	85.62	14.38	78.1249
400 x 400	4800.56	4056.66	84.50	15.50	156.268

It can be seen in table 3 that the average compression ratio is 80.97%, the average space saving is 19.0325% and the average compression time is 66.417ms.

Table 5. The Compression result using Run-Length Encoding for 4 sample color images (RGB)

Resolution (pixel)	Original Size (kb)	Compressed Size (kb)	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
100 x 100	300.56	395.98	131.75	-31.75	0

200 x 200	800.56	1598.56	199.68	-99.68	15.6583
300 x 300	2700.56	3546.62	131.33	-31.33	15.6246
400 x 400	4800.56	6293.50	131.10	-31.10	15.6333

It can be seen in table 4 that the average compression ratio is 148.465%, the average space saving is -48.465% and the average compression time is 11.72905ms.

Table 6. The Compression result using Fibonacci Code for 4 sample grayscale images

Resolution (pixel)	Original Size (kb)	Compressed Size (kb)	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
100 x 100	300.56	230.86	76.81	23.19	15.5955
200 x 200	800.56	794.36	66.17	33.83	15.6948
300 x 300	2700.56	1584.11	58.66	41.34	62.4649
400 x 400	4800.56	2498.24	52.04	47.96	93.8025

It can be seen in table 5 that the average compression ratio is 63.42%, the average space saving is 36.58% and the average compression time is 46.88943ms.

Table 7. The Compression result using Run-Length Encoding for 4 sample grayscale images

Resolution (pixel)	Original Size (kb)	Compressed Size (kb)	Compression Ratio (%)	Space Saving (%)	Compression Time (ms)
100 x 100	300.56	118.64	39.47	60.53	0
200 x 200	800.56	386.70	32.21	67.79	0
300 x 300	2700.56	761.36	28.19	71.81	15.616
400 x 400	4800.56	1090.16	22.71	77.29	15.6246

It can be seen in table 6 that the average compression ratio is 30.645%, the average space saving is 69.355% and the average compression time is 7.81015ms.

4. Conclusion

The conclusion in this research are as follows:

- Fibonacci Code algorithm will give a better compression ratio and space saving/redundancy than Run Length Encoding algorithm on image color
- Run Length Encoding algorithm will give a better compression ratio, space saving/redundancy and compression time than Fibonacci Code algorithm on grayscale image

Based on the results of research at two different types of images, each algorithm has its own advantages. Fibonacci Code is better for color image compression while Run-Length Encoding is better for grayscale image compression.

Acknowledgments
The authors gratefully thank to all who have supported in the completion of this research. And to the Universitas Sumatera Utara that has given me the opportunity to publish this paper.

References

[1] Salomon, D. 2004. *Data Compression The Complete Reference*. Third Edition. New York: Department of Computer Science California State University, Northridge, Springer-Verlag 1-2.
[2] Bhattacharyya, S. 2017. Complexity Analysis of a Lossless Data Compression Algorithm using *Fibonacci* Sequence. *International Journal of Information Technology (IJIT).* Volume 3(3).
[3] Kaur, K., Saxena, J., dan Singh, S. 2017. Image Compression Using Run Length Encoding (RLE).*International Journal on Recent and Innovation Trends in Computing and Communication.* Volume 5(5).
[4] Mengyi Pu, Ida. 2006. *Fundamental Data Compression*. First Edition. London: Elsevier.