The IJICS (International Journal of Informatics and Computer Science)
Vol 4 No 1, March 2020

ISSN 2548-8384 (online)
ISSN 2548-8449 (print)
Page 1 – 4
DOI 10.30865/ijics.v4i1.1348

Analysis of Even-Rodeh Code For Text Compression

Rian Syahputra

Department of Computer Science. STMIK Budi Darma, North Sumatera, Indonesia
Email: ryansyah93@gmail.com

Abstract–Text compression is used to reduce the repetition of letters so that the storage space used becomes smaller and transmission becomes faster. Techniques in text compression that can be used are using lossless compression. Lossless compression is a type of compression where when the decompression process occurs no data is lost. The Even-Rodeh Code algorithm is a lossless type of compression algorithm that replaces the initial bit code of the data with a code from the Even-Rodeh Code algorithm so that it can produce smaller data sizes.

Keywords: Text Compression, Even-Rodeh Code, Lossless

1. INTRODUCTION

Reducing the size of the data makes us get some benefits such as using less storage space and faster data transmission processes. Data reduction is usually done to save the transmission time and memory used. The process of reducing the size of this data is called compression. Compression is a process to reduce the size of data such as text, images, audio, and video [1]. Size reduction in the text is done by removing the same letter so that only the remaining one letter remains.

2. THEORY

2.1 Compression

Data compression is a way to reduce the size of a data so that it is stored more compactly and also to reduce the transfer time [2]. Compression aims to reduce the amount of data or bits needed to store and transmit data without excessively reducing the quality of the original data. Compressing something means you have data and you reduce its size [1], [3], [4].

2.2 Even-Rodeh Code

Even-Rodeh Code is a lossless compression type, it means no data will lost when it comes to decompression. To search the code from Even-Rodeh Code are explained below [2][5]:

1. If N < 4, then let c is the binary representation of N and lc is the length of c; the codeword of N is (3 – lc) times ‘0’ prepended to c. Thus, if N = 2, then c = ‘10’, lc = 2, so the codeword of N = 2 is (3 – 2) times ‘0’ prepended to ‘10’, which is ‘010’. If N >= 4 and N < 8, then the codeword is simply the binary representation of N prepended to ‘0’.
2. Therefore, if N = 5, then the codeword is ‘101’ prepended to ‘0’, which is ‘1010’.
3. If N >= 8, then let c is the binary representation of N, lc is the length of c, and bc is the representation of lc in binary; the codeword of N is bc prepended to c and prepended again to ‘0’. Hence, if N = 9, then c = ‘1001’, lc = 4, bc = ‘100’, so the codeword is ‘100’ prepended to ‘1001’ and prepended again to ‘0’, which is ‘10010010’.

Table 1. Even-Rodeh Code

n	Even-Rodeh Code
0	000
1	001
2	010
3	011
4	100 0
5	101 0
6	110 0

The step by step using Even-Rodeh Code are as follows:

ISSN 2548-8384 (online)
ISSN 2548-8449 (print)
Page 1 – 4
DOI 10.30865/ijics.v4i1.1348

1. Character sorting from largest frequency (which has the same value) to the smallest. If the frequencies are the same then sort it alphabetically.
2. Form a table for Even-Rodeh Code. The data bits are replaced with bits from Even-Rodeh Code table. Then count the bits in each value.
3. Make a bit string from the Even-Rodeh Code per character. But before make the bit string, first check the bit string length explained below:
    a. If the remainder for the length of the string bit to 8 is 0 then add 00000001. State the final bit.
    b. If the remainder for the length of the bit string to 8 is n (1, 2, 3, 4, 5, 6, 7) then add 0 as much as 7 - n + "1" at the end of the bit string. Express it with L. Then add the binary number from 9 - n. State with the final bit.
    c. Then change the binary per 8 bit to new character as compressed text.

Decompression step of Even-Rodeh Code are explained below:

1. Read compressed text and change it back to binary form.
2. Returns the binary to the initial bit string by reading the last 8 bits and as a decimal number. State the reading result with *n* then remove as much as 7 + *n* at the end of string bit.

3. RESULT AND DISCUSSION

For example, we have input string “budidarma”. First is to sort the frequency of the character from the largest to the smallest, see table 2.

Table 2. Sorting the input string

No.	Character	Frequency	Binary	Bit	Frequency x Bit
1	a	2	0110 0001	8	16
2	d	2	0110 0100	8	16
3	b	1	0110 0010	8	8
4	i	1	0110 1001	8	8
5	m	1	0110 1101	8	8
6	r	1	0111 0010	8	8
7	u	1	0111 0101	8	8
				Total	72

From the table 2 above the string “budidarma” has 72 bit size. Next is we change the binary of the character with the Even-Rodeh Code from table 1.

Table 3. Compression with Even-Rodeh Code

n	Character	Frequency	Even-Rodeh Code	Bit	Frequency x Bit
0	a	2	000	3	6
1	d	2	001	3	6
2	b	1	010	3	3
3	i	1	011	3	3
4	m	1	100 0	4	4
5	r	1	101 0	4	4
6	u	1	110 0	4	4
				Total	30

From table 3 we can see that compression with Even-Rodeh Code it can compressed the input string to 30 bit size with 41,67% (size before compression divide by size after compression) as compression ratio.

Arrange the new code as string bit for “budidarma”:

b	u	d	i	d	a	r	m	a
010	1100	001	011	001	000	1010	1000	000

From the string bit we have 30 bit as compressed size, then search the remainder of 30 for 8 is 6 and state as *n*. The remainder for the length of string bit is 6, so we add 0 as much as 7 - *n* + ”1” = 7 – 6 + “1” = 0 + “1” = 01 and state it with L and the padding is “01” bit at the end of the string bit.

ISSN 2548-8384 (online)
ISSN 2548-8449 (print)
Page 1 – 4
DOI 10.30865/ijics.v4i1.1348

01011000 01011001 00010101 00000001

After that we add the binary number from 9 - *n* = 9 – 6 = 3 = 00000011 and state it as final bit and add after padding.

01011000 01011001 00010101 00000001 00000011

Then we generate per bit to look for new character we get after compression of Even-Rodeh Code is complete. As you can see in table 4.

Table 4. Generating new code

String Bit	New Character
0101 1000	X
0101 1001	Y
0001 0101	NAK
0000 0001	SOH
0000 0011	ETX

The new character is the result of compressed initial text.

Decompression step of Even-Rodeh Code are as following:

1. Read the compressed text.
2. Change it back to binary form.

   01011000 01011001 00010101 00000001 00000011

3. Returns the binary to the original string bit by reading the last 8 bits and convert it to decimal number.

   00000011 = 3

   State the result with *n* then we remove as much as 7 + *n* = 7 + 3 = 10 from the last bit at the end of string bit.

   01011000 01011001 00010101 000000

Then we read the bit from left to right one by one and compare it with table Even-Rodeh Code. Read index-0 from the string bit is 0 then compare it with table 3. If not available, then join with the next index, we have index-0 + index-1 are = 01 then compare it with table 3. If not available, then join with the next index, we have index-0 + index-1 + index-2 are = 010. If available, change the bit with the found “b”. And then we read the next index is index-3 is 1 then compare it with table 3, and not available, so we continue the read index-3 + index-4 are 11 then compare it with table 3, and not available, continue read index-3 + index-4 + index-5 are 110 and not available, continue read index-3 + index-4 + index-5 + index-6 are 1100 and available we found “u” and so on until all string bit are replaced with the initial character.

010	1100	001	011	001	000	1010	1000	000
b	u	d	i	d	a	r	m	a

4. CONCLUSION

From section 3 we can summarized as follows:

1. Even-Rodeh Code is good for text compression that has large frequencies.
2. It has 41,67% of compression ratio meaning the size of left data after compression, and it means 58,33% data has been reduced.
3. As we can see the result of decompression has the same bit as the initial data, so it means Even-Rodeh Code is a lossless type compression.
4. For further research, we can compare it to other lossless type of compression.

ISSN 2548-8384 (online)
ISSN 2548-8449 (print)
Page 1 – 4
DOI 10.30865/ijics.v4i1.1348

REFERENCES

[1] M. Sharma, "Compression Using Huffman Coding," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, vol. 10, no. 5, p. 133, 2010.
[2] M. A. Budiman and D. Rachmawati, "On Using Goldbach G0 Codes and Even-Rodeh Codes for Text Compression," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 180, no. 1, pp. 1–7, 2017.
[3] S. D. Nasution and Mesran, "Goldbach Codes Algorithm for Text Compression," *IJournals Int. J. Softw. Hardw. Res. Eng.*, vol. 4, no. December, pp. 43–46, 2016.
[4] S. D. Nasution, G. L. Ginting, M. Syahrizal, and R. Rahim, "Data Security Using Vigenere Cipher and Goldbach Codes Algorithm," *Int. J. Eng. Res. Technol.*, vol. 6, no. 01, pp. 360–363, 2017.
[5] D. Salomon, D. Bryant, and G. Motta, *Handbook of Data Compression (Google eBook)*, Fifth. Springer, 2010.