Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

Combination of Sequitur and Leveinstein Algorithms for Text File Compression

Muhammad Hasan
Faculty of Computer Science & Information Technology, Informatics Engineering Study Program, Budi Darma University, Medan, Indonesia
Email: mhasanhsb@gmail.com
Email Correspondent: mhasanhsb@gmail.com

Abstract- - Data is one of the main things in computer science engineering (ICT) problems, data can mean a situation, image, sound, letter, number, mathematics, language or other symbols. Facts that often occur about data are the need for storage capacity and the need for Data transfer time, which is a case that must be observed in this requirement, is caused by the increasing amount of data that must be stored in RAM memory. Based on the results of the tests carried out: Files compressed by the sequitur algorithm make it easier to use the internet so that the time required will be shorter and the possibility of download and upload jobs failing will be smaller. Then transferring files over the network will be faster, the delivery time depends on whether the provider is fast or not and the size of the file to be sent and helps in reducing the size of the file so that it can reduce the storage capacity of a memory / RAM.
A text file is a collection of various related information in text form, text files originating from documents, letters, numbers and symbols will be compressed with a sequitur algorithm. The need for large storage capacity seems increasingly important. This need is caused by the increasing amount of data stored over time, especially for the business and banking world. These companies generally need very large capacity to store all important data and files.
The Leveinstein algorithm is a development of the sequitur algorithm, where a file approximates the difference between a string and the source string. The difference value between strings is expressed as the edit distance value. In contrast to the Leveintein algorithm which works on a character-by-character basis, the sequitur algorithm operates by enforcing uniqueness diagram constraints and usability rules.

Keywords: Sequitur Algorithm, Levenstein Algorithm, Text File.

1. INTRODUCTION

With the rapid development of technology, computerization has become a major part of the world of work. Data that must be stored every day will make computerized storage capacity increasingly low. The larger the size of the data stored, the smaller the computerized storage capacity. This problem is often faced by computer users who lack sufficient data storage media
To overcome the problem of data storage and access speed, one way is by compressing the stored information data. The information in question is information in the form of text (a file containing plain text). As we know, the size of text files is smaller compared to the size of images/images, and we also know that some agencies usually always store text data for a long period of time because it will be needed at any time. However, to streamline data storage space and obtain faster access times, it is deemed necessary to compress this text file.
Data compression is a technique for reducing the file size of the original file. All file types can be subjected to a compression process. One of them is a text file. The sequitur algorithm is a compression algorithm that can be used to compress the file. Levenstein's algorithm is an algorithm used to find the fewest number of operations to transform a string into another string. This algorithm performs compression using a formula, where the text is replaced with an index obtained from a "dictionary".
In 2013, Adi Adiono researched text file compression techniques using the sequitur and Levenstein methods, which showed that the results of text file compression did not change the quality of the text file, it only reduced the bits in the file. Likewise, in the decompression process, the result will be the same as the original file, even if it has been repeated once or twice. So the sequitur and levenstein methods are an alternative for compressing text file data [2].
With this, the author uses the Sequitur and Levenstein algorithm to find out how the compression performance is, if it is done by compressing text files to provide benefits in storage and requires less memory space compared to uncompressed text files.

2. RESEARCH METHODOLOGY

2.1. Compression

The compression process is the process of reducing the size of data to produce a digital representation that is compact but can still represent the quantity of information contained in the data. The term compression is often also called source coding, data compression, bandwidth compression, and signal compression [2].
Data and information are two different things. Data contains information. However, not all parts of the data are related to that information or in a piece of data there are parts of data that are repeated to represent the same information. The success of data compression depends on the size of the data itself and the type of data that allows it to be compressed, usually several components in the data that are more general in nature than others are often used in data compression algorithms that utilize this property.

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

This is called redundancy. The greater the redundancy in the data, the higher the success rate of data compression. In the data compression process, there is a general concept of probability which shows a measure of how much information is contained in a data series or what is called entropy which can be represented mathematically (Ferrianto Gozali & Mervyn, 2004).

2.1.1. Benefits of Compression

Some of the benefits of data compression are:

1. Reduce data size
2. Simplifies the process of storing data on media
3. Speed up the sending process on various types of connections.

Based on the type of code map used to convert the initial message (input file contents) into a set of codewords, compression methods are divided into 2 (two) groups, namely:

1. Static Method
   Uses a code map that is always the same. This method requires two phases (two-pass): The first phase is to calculate the probability of appearance of each character/symbol and determine the code map, and the second phase is to convert the message into a code to be transmitted.
2. Dynamic (adaptive) method
   Uses code maps that may change over time. This method is called adaptive because the code map is able to adapt to changes in the characteristics of the file contents during the compression process. This method is one-pass, because it only requires one reading of the file contents [3].

2.2. Types of Compression

Based on the information content of the data compression results, data compression can be grouped into 2 (two) types, namely lossless compression and lossy compression.

2.2.1. Lossy Compression

Lossy Compression is compression where the decompression results of the compressed text file are not the same as the original text file because there is information lost, but it can still be detected by eye perception. The eye cannot distinguish small changes in the image. This method produces a higher compression ratio than the lossless method. Most lossy data compression settings have different compression levels. This is done so that the compression is more effective and the information contained in the message is not changed and lost. Lossy data compression is effective when applied to digitized analog data storage such as video images and sound.

Lossy Compression data compression which produces compressed data files that cannot be returned to data before being completely compressed. When the compressed data is decoded again, the decoded data cannot be returned to be the same as the original data but there are parts of the data that are lost. [8]

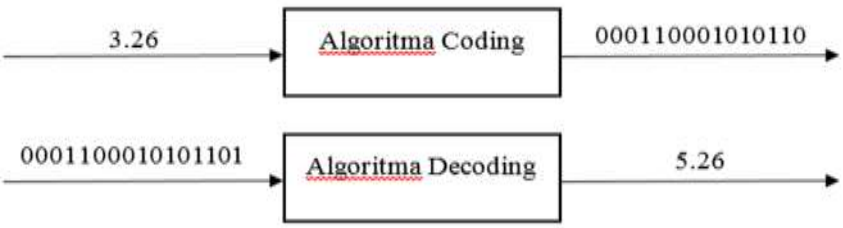

Figure 1.. Lossy Compression illustration

2.2.2. Lossless compression

Lossless compression is image compression where the decompression results of the compressed text file are not the same as the original text file because there is information lost. Unfortunately, the compression ratio of text files with this method is very low. Many applications require seamless compression, such as radiography applications, compression of text files resulting from medical diagnoses or detailed images, where even the slightest loss of images will cause unexpected results.

In lossless compression, because it must maintain the perfection of information, there is only a coding and decoding process, there is no quantitation process. This type of compression is suitable for application to database files, spread sheets, word processing files, biomedical images, and so on. [2].

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

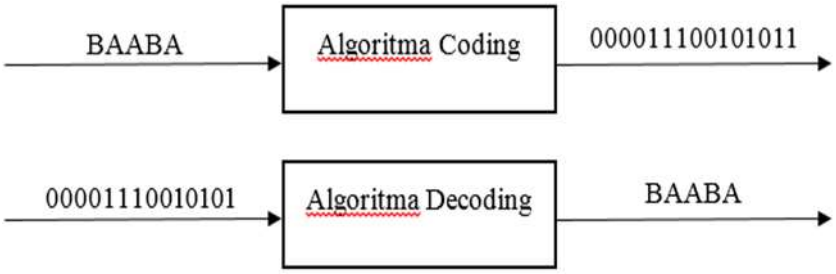

Figure 2. Lossless Compression illustration

2.3. Levenstein's algorithm

This Levenstein algorithm is known for non-negative integers and was created by Vladimir Leveinstein in 1968. Both encoding and decoding are multi-step processes (Salomon, 2007).
The Levenstein code for zero is a single 0. For the positive number code n, here are the encoding steps.

1. Set the first number of C to 1. Place the code-so-far in the empty string.
2. Take the binary value of n without the leading 1 and add it to the code so far
3. This.
4. Denote M as the number of bits added in stage 2.
5. If M ≠ 0, add 1 to C and do step 2 again, but with the value M instead of n.
6. If M = 0, add 1 followed by 0 in C to the code-so-far and stop.
7. For the example case, we assume n = 18. The binary value of 18 is 10010, we take 0010 and add it after 1. Add 1 to the code, then count the number of characters in 0010, we get 4. Add back 1, Take the value 00, add behind that addition. Repeat these steps until we get the value M = 0, then add 10. So 18 in the Levenstein code is 11110|0|00|0010.

Table 1. Levenstein Code Table (Salomon, 2007)

N	Kode Levenstein	N	Kode Leveinstein
0	0	9	11100 1 001
1	10	10	11100 1 010
2	110 0	11	11100 1 011
3	110 1	12	11100 1 100
4	1110 0 00	13	11100 1 101
5	1110 0 01	14	11100 1 110
6	1110 0 10	15	11100 1 111
7	1110 0 11	16	111100 1 00 0000
8	11100 1 000	17	111100 1 00 0001

Decoding is done as follows:

1. Set C with the number of consecutive numbers 1 before the first number 0.
2. If C = 0, the decoded value is zero, stop.
3. Set N = 1, and repeat step 4 (C-1) times.
4. Read N bits, add 1, and assign the resulting bitstring to N (thus removing the previous value from N). The string assigned to N in the last iteration is its decoded value.

For example, let's take the Levenstein code 11110|0|00|0010. The decoder first counts the number of 1s before 0s and finds that there are 4 iterations. Then read 0 and add the leading 1 to 10. Decode 10 and find that the number of bits in the binary representation of the codeword length is equal to 2. Then read 00 and add the leading 1 to 100. This means that the length of the binary representation is equal to 4. Finally, read 0111 , add prefix 1 and decode 10111, which is 23

3. RESULT AND DISCUSSION

3.1. Implementation of the Sequitur Algorithm

Before carrying out the compression process using the sequitur algorithm, the author determines the string that will be used. Suppose a text file contains the string “CHILDREN_MAIN_GAME”. It must be known in advance that the string "ANAK-ANAK_MAIN_PERMAINAN" consists of 24 characters, where 1 character is equal to (=) 1 bit and 1 bit is equal to (=) 8 bits, then the total string before compression is 24 bits or equal to (=) 192 bits.

To find out the size of the string "ANAK-ANAK_MAIN_ GAMES", you can see the table as follows:

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

Table 2 String Size Before Compression

Karakter	Frekuensi	ASCII Desimal	ASCII Binary	Bit	Bit x Frekuensi
A	7	65	01000001	8	56
N	5	78	01001110	8	40
Sp (spasi)	3	32	00100000	8	24
K	2	75	01001110	8	16
M	2	77	01001011	8	16
I	2	73	01001001	8	16
P	1	80	01000101	8	8
E	1	69	01000101	8	8
R	1	82	01000111	8	8
		Jumlah bit x frekuensi			192

A text file that has known strings and the resulting number of each frequency of occurrence, then the next process is the compression process using the sequitur algorithm.
In the sequitur algorithm, the existing string is then processed to get a duplicate string or in other words 2 characters or 2 letters that appear more than once. Then the character will be checked, if a pair appears more than once then the pair is changed to a non-terminal symbol.
Initial string : CHILDREN_MAIN_GAME
String to lowercase : kids_play_games

3.2 Sequitur Algorithm Process

As explained above, a string that has been converted to lowercase will then be checked to see if any pairs appear more than once. From the string "anak_anak_main_permainan", the character pair "an" appears three times. So according to the principle of performance of this sequitur algorithm, the character pair "an" will be changed to a non-terminal symbol, namely to "A". So a new string is formed as "Aak_Aak_main_permainA".
Then, from the new string, it will be checked again to see if there are pairs of characters that appear more than once. After checking again, it turned out that the character pair "Aa" appeared more than once, namely twice. Therefore, the character pair "Aa" will be changed to a non-terminal symbol, namely the symbol "B". So that it forms the string "Bk_Bk_main_permainA". In the new string, it will be checked to see if any pair of characters appears more than once. After checking again, it turned out that the character pair "Bk" appeared more than once, namely twice. Therefore, the character pair "Bk" will be changed to a non-terminal symbol, namely the symbol "C". So that it forms the string "C-C_main permainA".
Then, from the new string, it will be checked again to see if there are pairs of characters that appear more than once. After checking again, it turned out that the character pair "ma" appeared more than once, namely twice. Therefore, the character pair "ma" will be changed to a non-terminal symbol, namely the symbol "D". So that it forms the string "C-C_Din_perDinA". In the new string, it will be checked to see if any pair of characters appears more than once. After checking again, it turned out that the character pair "Di" appeared more than once, namely twice. Therefore, the character pair "Di" will be changed to a non-terminal symbol, namely the symbol "E". So that it forms the string "C-C_En perEnA".
In the new string, it will be checked to see if any pair of characters appears more than once. After checking again, it turned out that the character pair "En" appeared more than once, namely twice. Therefore, the character pair "En" will be changed to a non-terminal symbol, namely the symbol "F". So that it forms the string "C-C_F_perFA".
The overall process of this sequitur algorithm can be contained in table form which is presented as follows:

Table 3. Sequitur Algorithm Process

Process	Strings	Duplicate Diagram on String	Symbol Formation (Non-Terminal)	Rule	Resulting String	Delete Rule
1	anak- anak main permainan	An	A	A = an	Aak Aak main permainA	-
2	Aak Aak main permainA	An Aa	A B	A = an B = Aa	Bk Bk main permainA	-

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

3	Bk Bk main permainA	An Aa Bk	A B C	A = an B = Aa C =Bk	C C main permainA	-

Table 4. Sequitur Algorithm ProcessContinued

Process	Strings	Duplicate Diagram on String	Symbol Formation (Non-Terminal)	Rule	Resulting String	Delete Rule
1	C C main	An	A	A = an	C C Din	
	permainA	Aa	B	B = Aa	perDinA	-
		Bk	C	C = Bk		
		Ma	D	D = ma		
2	C C Din	An	A	A = an	C C En perEnA	
	perDinA	Aa	B	B = Aa		-
		Bk	C	C = Bk		
		ma	D	D = ma		
		Di	E	E = Di		
3	C C En	An	A	A = an	C C F perFA	
	perEnA	Aa	B	B = Aa		-
		Bk	C	C = Bk		
		Ma	D	D = ma		
		Di	E	E= Di		
		En	F	F =En		
4	C C F	An	A	A = an		
	perFA	Aa	B	B = Aa	-	-
		Bk	C	C = Bk		
		Ma	D	D = ma		
		Di	E	E= Di		
		En	F	F =En		

3.3 Dictionary

In this case, the dictionary functions as a replacement for a temporary database, which means that in the process this dictionary will store the results of processes that have been carried out in the previous process to produce return data in the decompression process later. The dictionary of the processes that have been carried out can be seen in the following table:

Table 5. Dictionary

Proses	Aturan Rule
1.	A = an
2.	A = an
	B = Aa
3.	A = an
	B = Aa
	C = Bk
4.	A = an
	B = Aa

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

	C = Bk
	D =ma
5.	A = an
	B = Aa
	C = Bk
	D = ma
	E = Di
6	A = an
	B = Aa
	C = Bk
	D = ma
	E = Di
	F = En

3.4. Sequitur Algorithm Decompression

This decompression process aims to make the string that has been compressed return to its original state before being compressed. Therefore, every compression process is always accompanied by decompression. From this statement, the author will explain how to decompress the sequitur algorithm with previously processed strings.
It is known that the string that has been compressed is the string "C-C_F_perFA". In the sequitur algorithm, this algorithm process has rules where in these rules there are symbols A, B, C, D. E, and F. Each symbol has a pair of repeated characters.
For more clarity on the decompression algorithm sequitur can be seen in the following table:

Table 6 Sequitur Decompression Process

Process	*Compression Strings*	Rules	Rule Description	*Strings After Decompression*
1	C-C_F_perFA	A= an	Simbol “A” diganti dengan pasangan karakter “an”	Anak-anak _F_perFA
2	Anak-anak _F_perFA	B = Aa	Simbol “B” diganti dengan *rule* “Aa”	Aak-aak _F_perFA
3	Aak-aak _F_perFA	C = Bk	Simbol “C” diganti dengan *rule* “Bk”	Bk-Bk _F_perFA
4	C-C _main_permainA	D=ma	Simbol “D” diganti dengan *rule* “ma”	C-C_Din_perDinA
5	C-C _Din_perDinA	E = Di	Simbol “E” diganti dengan *rule* “Di”	C-C _En_perEnA
6	C-C _En_perEnA	F= En	Simbol “F” memiliki *rule* “En”.	

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

anak- anak main

permainan

3.5. Levenstein Algorithm Compression

After carrying out the compression process using the sequitur algorithm, the next process is calculating the total bit string that has been processed using the Levenstein algorithm.

a. Bit calculation with the Levenstein algorithm can be seen as follows:

Table 7. Compression process with the Levenstein algorithm

Character	Frequency	Levenstein Code	Bit	Bit * Frequency
Sp (spasi)	2	0	1	2
C	2	10	2	4
F	2	1100	4	8
-	1	1101	4	4
A	1	1110000	7	7
P	1	1110001	7	7
e	1	1110010	7	7
R	1	1110011	7	7
		Jumlah Bit x Frekuensi		46 bit

After knowing the compression results, the compression results can be measured or calculated as follows:

$$Cr = 100\% - \frac{ukuran\ bit\ data\ setelah\ dikompresi}{ukuran\ bit\ data\ sebelum\ dikompresi} \times 100\%$$

$$Cr = 100\% - \frac{46\ bit}{192\ bit} \times 100\%$$

$$Cr = 100\% - 76\%$$

$$Cr = 24\%$$

From the results of the compression ratio calculation, it turns out that the compression process using the sequitur algorithm is 24% more efficient on the capacity available.

Combining the resulting Levenstein code bit strings, the following binary is obtained:

10110110 01100011 10001111 00101110 01111001 110000 (total number is 46 bits).

Check the length of the bit string.

If the remainder for the string length of 8 bits is 0, then add 8 "0" bits. Express it with the final bit. Meanwhile, 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 followed by the number "1" at the end of the bit string, denote it by L, denote it by bit. end. Because the number of bit strings that have been compressed is 46 bits and is not divisible by 7, the calculation is as follows:

N= (Number of compressed string bits)/7

$$N = \frac{Jumlah\ bit\ string\ hasil\ kompresi}{7}$$

$$= \frac{46}{7}$$

$$= 6$$

Based on the results of the operation above, it is concluded that the remainder for the 46 bit string is 6, so n = 6.

To find padding, look as follows:

= 7 – n + "1"

= 7 – 6 + "1" = 1

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

then the binary compression result above will be added with 1 bit "0" followed by bit "1", so that the bit padding is 11000001

Looking for Flagbits

To find flagbits, use the formula 9 - 6 where n is the number of padding bits so that it becomes 9 – 6 = 9 – 6 = 3, represented in binary form 00000011.

This flagbits value is converted into binary with 8 bit representation to become 00000011. This flagbits value will be added after the padding bits that were obtained previously.

The final result of the compression process (in binary form) is:

10110110 01100011 10001111 00101110 01111001 11000001 00000011

Grouping compressed binaries

The binary resulting from compression above will be grouped into 8 bits per group, so that characters can be produced from each group of bits, the results are shown in table 3.5.

Table 8. Grouping of compressed binaries

Biner	Desimal	Karakter
10110110	182	¶
01100011	99	C
10001111	143	
00101110	46	.
01111001	121	Á
1100000**1**	193	Á
00000011	3	
Jumlah Bit Hasil Kompresi		56 bit

h. Based on the compression results that have been obtained, the parameter values of compression ratio, compression ratio and redurancy can be calculated:

$$Rc = \frac{data\ bit\ size\ before\ compression}{data\ bit\ size\ after\ compression} = \frac{192\ bit}{56\ bit} = 3{,}42$$

$$Cr = \frac{udata\ bit\ size\ before\ compression}{data\ bit\ size\ after\ compression} \times 100\% = \frac{56\ bit}{192\ bit} \times 100\% = 29\%$$

$$Rd = 100\% - Cr = 100\% - 29\% = 71\%$$

Based on calculations using compression parameters for the contents of the material file above, the Rc result above before compression is 13.42 bits times the size of the data after compression. Meanwhile, the Cr result above means that after compression the data size is 29% of the data before compression and Rd is 72%, which is data that has been compressed or missing data.

3.6. Levenstein Decompression Process

After carrying out the compression process using the sequitur algorithm, the next process is calculating the total bit string that has been processed using the Levenstein algorithm. The binary material that has been compressed is:
10110110 01100011 10001111 00101110 01111001 11000001 00000011
The next step takes the last 8 bits of the string to be decompressed, represented in decimal form, namely 3, expressed as n, then uses the 7+3 formula to return the bits to their original state, namely 7+n = 7+3 = 10.
Then remove 10bits from the end of the bit string so it becomes
10110110 01100011 10001111 00101110 01111001 110000
Next, check the first bit, if it is in the Levenstein code table in the Levenstein code dictionary in table 3, then change the string accordingly.

Table 9. Bit Checking and Decompression Process

Index	Levenstein Value	Description	Character

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

1	1	There isn't any	
2	10	There is	C
3	1	There isn't any	
4	11	There isn't any	
5	110	There isn't any	
6	1101	There is	-
7	1	There isn't any	
8	10	There is	C
9	0	There isn't any	Spasi
10	11	There isn't any	
11	111	There isn't any	
12	11100	There is	F
13	11101	There isn't any	
14	111011	There isn't any	
15	1110110	There isn't any	

Table 10. Bit Checking and Advanced Decompression Process

Index	Levenstein Value	Description	Character
16	11101100	There isn't any	
17	0	There is	Spasi
18	11	There isn't any	
19	111	There isn't any	
20	1110	There isn't any	
21	1110001	There is	P
22	1110010	There is	E
23	1110011	There is	R
24	1	There isn't any	
25	11	There isn't any	
26	111	There isn't any	
27	1100	There is	F
28	11100	There isn't any	
29	111000	There isn't any	
30	1110000	There is	A

The checking process as in table 3.6 above is carried out until the initial string "C-C_F_perFA" is found

4. CONCLUSION

The conclusions that can be drawn after designing a text file compression application by applying the Sequitur algorithm are:

1. Applying the sequitur algorithm to carry out the encryption process for message characters to produce ciphertext. Carry out the process of grouping the characters into several columns, where each column will contain three message characters. Perform matrix multiplication between message characters and the key used. The values resulting from the matrix multiplication process will be modulated, so that the final result of the modulation process is a ciphertext.

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 23 - 32
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1914
https://journal.fkpt.org/index.php/MIS

2. Application of the Levenstein algorithm to compress text files containing previously encrypted text. The data is first analyzed by creating a table of the frequency of occurrence of each symbol. The frequency table has attributes in the form of ASCII symbols and frequencies. The data that has the smallest frequency of appearance is selected as the first node in this algorithm, from these two nodes a parent node is created which records the frequency of the first two nodes.
3. Reducing the size of the text file using the Sequitur and Levenstein algorithms, which can be done using compression techniques, where each character (plaintext) will be encrypted. The encryption process is the change of each message character into another form whose meaning will no longer be understood as a result of this change (ciphertext).
4. *Designing an application using Visual Basic 2008 for placing inscribed text files is a step that has a positive impact on securing text files as well as saving data storage capacity, where the positive impact is in the form of convenience and time effectiveness.*
5. *used to perform encoding processing and compression of abstracted text files.*

REFERENCES

[1] (2010). In E. Jubilee, *Rahasia Manajemen File* (p. 144). Indonesia: Elex Media Komputindo.
[2] Marimin. (2004). *Teknik Dan Aplikasi Pengambilan Keputusan Kriteria Majemuk.* Jakarta, Indonesia: Grasindo.
[3] (2016). In M. Muslihudin, Oktafianto, & A. Pramesta (Ed.), *Analisa Dan Perancangan Sistem Informasi Menggunakan Model Terstruktur Dan UML*. Yogyakarta, Indonesia: Andi.
[4] (2015). In H. Nugroho, *Matematika Diskrit Dan Implementasinya Dalam Dunia Teknologi Informasi.* Yogyakarta, Indonesia: deepublish.
[5] Nurdam, N. (2014). Sequence Diagram Sebagai Perkakas Perancangan Antarmuka Pemakai. *Ultimatics , VI* (2085-4552), 21-25.
[6] Putra, D. (2010). In D. Putra, & Westriningsih (Ed.), *Pengolahan Citra Digital.* Yogyakarta, Indonesia: Andi.
[7] (2010). In D. Salomon, & G. Motta, *Handbook Of Data Compression.* New York, Amerika: Springer London Dordrecht.
[8] Sulistyorini, P. (2009). Pemodelan Visual Dengan Menggunakan UML Dan Rational Noise. *Jurnal Tekhnik Informasi Dinamika , XIV* (0854-9524), 23-29.
[9] Suryasari, Callista, A., & Sari, J. (2012). Rancangan Aplikasi Customer Service Pada PT. Lancar Makmur Besama. *Jurnal Sistem Informasi (JSI) , IV* (2085-1588), 468-476.
[10] (2004). In F. Wahid, *Dasar-Dasar Algoritma Dan Pemrograman.* Yogyakarta, Indonesia: Andi.
[11] Wedianto, A., Sari, H. L., & H, Y. S. (2016). Analisa Perbandingan Metode Filter Gaussian, Mean Dan Median Terhadap Reduksi Noise. *Media Infotama , XII*, 21-30.
[12] Wibowo, H. R. (2014). *Buku Pintar VB.Net.* Jakarta, Indonesia: Elex Media Komputindo.