Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 1 - 9
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1916
https://journal.fkpt.org/index.php/MIS

Designing Game File Compression Applications With Implementing the Stout Code Algorithm

Iqbal Rahmadana
Faculty of Computer Science & Information Technology, Informatics Engineering Study Program,
Budi Darma University, Medan, Indonesia
Email: iqbal.rahmadan74@gmail.com,
Email Correspondent: iqbal.rahmadan74@gmail.com

Abstract- Games today are increasingly modern and have stunning graphics so that the size of the game files contained in the game becomes very large, because these games are increasingly modern and have very stunning graphics so that the graphics are very realistic. Games that are now available on computers or smart phones have stunning graphics and each one must have a game file that is on the storage media. Game files contained in the storage media can cause the capacity of the storage media to reach its limit, and game files on the computer version usually require that the game be downloaded or files that have not been installed, in contrast to the mobile version, in the mobile version the requirements for playing the game must download the application that has been provided to download the game. The problem in this research is that there is a game file whose size is very large so that it will be a problem for storage media and this can cause the performance of devices such as computers or smart phones to become slower.
Therefore for the solution in the previous paragraph, namely by using the compression technique. Compression is reducing the size, or in computer science compression is reducing the bit size of the data contained on the storage media so that the capacity space on the storage media is decreasing. The type of compression technique used in this study is lossless. Lossless compression is a compression technique, in which the use of this lossless compression technique, when compressed data is not the same as the data before compression and the data after compression can be decompressed into data like the original before compression. In this study, we will design a special compression application for game files, where game files have a large size so that it can cause the capacity of the storage media to decrease and this can cause performance on devices such as computers or smart phones to slow down. . And in this research, to compress the game file, it will apply an algorithm, namely the stout code algorithm. The stout code algorithm is one of the algorithms related to data compression, and in this study we can see the results of game file compression by applying the stout code algorithm.
After doing the compression and decompression process using the stout code algorithm, the application for game file compression that has been designed before can be run, namely by running the application, then press the file search button to find the game file to be compressed, then press the save file button to save the game file will be saved, then press the compression button to compress the game file and the game file is successfully compressed, to decompress do the same thing, find and save the file then press the decompress button and the game file is successfully decompressed.

Keywords: Data Compression, Algorithm, Game Files, Stout Code, Storage Media

1. INTRODUCTION

In this increasingly modern era, the gaming industry is currently developing very rapidly and technological advances are becoming increasingly sophisticated in the world of the gaming industry, so that games that have been released to people all over the world will be very entertained by the games that have been created in the world. gaming industry, the more popular the game is among the people, the game will become a competition among local people and the world. Games that are currently popular are not only on computers (PCs), but also on smart phones which are becoming increasingly sophisticated and modern. For those who like playing games, of course you need a lot of storage to store these games, whether you store light or heavy games. Current games are becoming more and more sophisticated in terms of graphics, so that games are now almost close to real life, this is because the graphics in games are becoming more and more stunning and the size of the game files is getting bigger. But it will be useless even if someone whose hobby is just playing games on a computer and saving lots of game files, so that the capacity of the storage media has reached its limit, that will be a problem even if they have a computer that has a large capacity. Therefore, the solution to this problem uses compression techniques which aim to reduce the size of the game file, in order to lighten the load on the capacity of the storage media.
Some of the game files are very large in size. So you have to need a large storage media too, if the game files that have accumulated on the storage media have reached their limit, it will be a burden on the storage media. Because there are game files that have very large file sizes, this will hinder the process of sending data to the storage media, causing the process of sending data to the storage media to fail on the data storage media whose capacity has reached its maximum.
To overcome the problem above, use compression techniques to reduce the bit size of the game file by applying the stout code algorithm, which is to lighten the load on the capacity of the storage media which has reached its limit.
Previously in 2019, Mr Surya Darma Nasution had researched compression, only with a different case, namely compression of text files by applying the same algorithm in this research, namely the stout code algorithm, which in his research was aimed at reducing the size ratio of the files. text[1].
In this research you will be expected to know how to compress game files by applying the stout code algorithm. So in this research we can study the stout code algorithm, to compress game files in order to reduce the bit size of game files so that it can lighten the capacity load on storage media which has reached its limit.

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 1 - 9
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1916
https://journal.fkpt.org/index.php/MIS

2. RESEARCH METHODOLOGY

2.1 Data Compression

Data compression is the process of encoding information using other bits that have a lower value than the unencoded data representation [1]. Compression has the meaning of reducing a size, in computer science compression is reducing the size of data in a program, be it a program from photo files, videos, applications, and so on, which aims to expand the capacity of data storage media, be it data storage media. hard disk, flash disk, SSD, and so on, if the data storage media whose capacity has reached the limit will cause the performance of the computer to become slower and when you want to transfer data, the process that will be required will be slow, therefore using this method This compression is really needed if the space capacity of the data storage has reached the limit and if you want to transfer data smoothly, so that the performance of the computer will be lighter and more efficient [4].

2.2 Data Compression Techniques

Data compression has two techniques, including the following:

1. Lossless Compression
   This technique compresses data, where the data from the compression results can be decompressed again and the results are exactly the same as when the data was not compressed. This technique is used if necessary when data after compression must be extracted or decompressed again exactly the same, for example: RAR, ZIP, 7-ZIP, and so on. And it is usually used when data accuracy is very important, for example: text/binary data, program data, images (GIF, PNG), and so on. Sometimes there is data that after being compressed using lossless techniques the data size becomes larger or the same [3].
2. Lossy Compression
   In this compression technique, the data from the decompression results is not the same as the data when the data was compressed, but the data is sufficient for use. Examples of data used using this compression technique are: BMP, JPEG, MPEG, WMA, MP3 and so on. etc. The advantage of this lossy compression technique is that the file size is smaller compared to lossless, but it still meets the requirements for use. Usually, this lossy compression technique will remove parts of the data that are actually not very useful, cannot be felt, cannot be seen by humans, so that humans can still assume that the data can still be used even though the condition has been compressed [3]

2.3 Compression Ratio

Compression Ratio is a calculation of system performance from the start of the compression process until it reaches the end of compression completion, so that it becomes a data count representing the final result. The formula for calculating the data compression ratio is written as follows:

$$Cr = \left(\frac{Ukuran\ Data\ Setelah\ Di\ Kompresi}{Ukuran\ Data\ Sebelum\ Di\ Kompresi}\right) \times 100\% \qquad (1)$$

(1)

2.4 Decompression

Decompression is the opposite when carrying out a data compression process, where the data is returned to new data after being produced through the compression process and will become new initial data. The data produced through the decompression process is exactly the same as the original data, namely when the data was before the compression process was carried out at all, therefore the data previously was called lossless compression. And vice versa, if the data produced through the decompression process is not exactly the same as the original data before decompression [3].

2.5 Game Files

A game file is a file that contains program data from the results of making a game, where the data can be accommodated in formats with the extension .exe, .bin, .dll, .cgf, and so on. Game files are created using programming software that is specifically designed to create games using basic methods such as understanding an algorithm and understanding coding from a programming language. The stage process for creating a game file is that you have to understand the use of software, algorithms, programming languages, debugging, and so on, and from stages like that, the game file is roughly the beginning of being created.

2.6 Stout Code Algorithm

In stout code, the variable-length for integer code is similar to the elias omega and even-rodeh codes. The codewords generated by the stout code algorithm depend on the choice of parameter l which is greater than or equal to 2 [1]. The stout code algorithm was introduced by Quentin Stout with two families, namely recursive Rl and Sl [3]. In the Rl family, more and more group lengths are read until the group is found and followed by 0. Use the notation L = 1 + ? Log2 n? and is symbolized by the binary representation B (n, l) of the l – bits (beta code) of the integer n. So, B (12, 5) = 01100. For l > 2, the prefix can be interpreted as:

$$R_l(n) = \begin{cases} B(n,l), & \text{for } 0 \le n \le 2^l - 1, \\ R_l(L)B(n,L), & \text{for } n \ge 2^l. \end{cases} \tag{2}$$

Those familiar with the Even-Rodeh Code may already know that this code is identical to R3. Furthermore, the Elias-Omega Code is between R2 and R3 with two differences, namely:

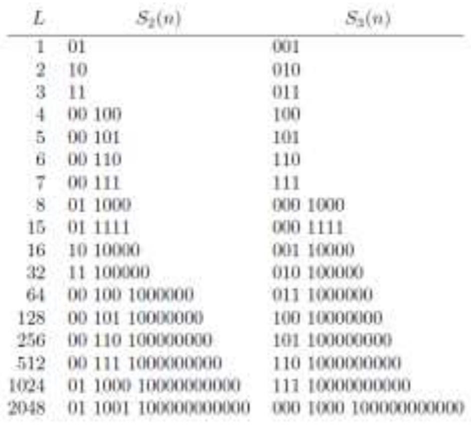
1. The omega code encodes the quantity Li = 1.
2. Separator 0 is placed to the right of n.

$$\begin{aligned} R_2(985) &= 11\ 100\ 1010\ 1111011001 & R_2(31{,}925) &= 11\ 100\ 1111\ 111110010110101 \\ R_3(985) &= 100\ 1010\ 1111011001 & R_3(31{,}925) &= 100\ 1111\ 111110010110101 \\ R_4(985) &= 1010\ 1111011001 & R_4(31{,}925) &= 1111\ 111110010110101 \\ R_5(985) &= 01010\ 1111011001 & R_5(31{,}925) &= 01111\ 111110010110101 \\ R_6(985) &= 001010\ 1111011001 & R_6(31{,}925) &= 001111\ 111110010110101 \end{aligned}$$

The second family of stout codes has a similar method, but with a different prefix denoted by S_l(n). For small values of 1, this group offers some improvements compared to the Rl code. In particular this eliminates a bit of redundancy in the Rl code because the group length cannot be 0 (which is why the group length in the omega code encodes Li – 1 and not Li). The prefix Sl is similar to the prefix Rl with the difference that the group length for Li encodes Li – 1 – l. The prefix Sl (n) is defined recursively by:

$$S_l(n) = \begin{cases} B(n,l), & \text{for } 0 \le n \le 2^l - 1, \\ R_l(L-1-l)B(n,L), & \text{for } n \ge 2^l. \end{cases} \tag{3}$$

Table 2.1 lists the prefixes S2(n) and S3(n) and explains the regularity. See the leftmost column includes the L value, i.e., the length of the encoded integer, and not the integer itself. The group length maintains its value until the groups following it all become 1, at which point the group's points are increased by 1 and the next group's points are reset to 10...0. All group lengths, except perhaps the leftmost one, start with 1. Behavior this is the result of the choice Li – 1 – l.



L	$S_2(n)$	$S_3(n)$
1	01	001
2	10	010
3	11	011
4	00 100	100
5	00 101	101
6	00 110	110
7	00 111	111
8	01 1000	000 1000
15	01 1111	000 1111
16	10 10000	001 10000
32	11 100000	010 100000
64	00 100 1000000	011 1000000
128	00 101 10000000	100 10000000
256	00 110 100000000	101 100000000
512	00 111 1000000000	110 1000000000
1024	01 1000 10000000000	111 10000000000
2048	01 1001 100000000000	000 1000 100000000000

Picture 1. Codes S2(n) and S3(n)

The prefix S2(64), for example, starts with the 7-bit group 100000 = 64 and depends on it S2(7-1-2) = S2(4) = 00 | 100. It is emphasized once again that table figure 2.4 only includes prefixes, not complete codewords. Once this is understood, it is not difficult to do it, you can see that the second stout code is a code prefix. Once a codeword is given, it will not be a prefix to another codeword. So, for example, the prefix of all codewords for 64-bit integers starts with the prefix 00 100 of the 4-bit integer, but every codeword for 4-bit integers has 0 and follows 00 100, whereas for 64-bit codewords it follows has 1 and following 00 100.

3. RESULT AND DISCUSSION

3.1 Discussion

In this research, we will analyze the performance of the stout code algorithm in game file compression in order to know the exact results in accordance with the analysis of game files that are compressed by applying the stout code algorithm and also find out the results of decompression of game files using the applied stout code algorithm. The stout code algorithm is an algorithm for compressing data, one of its functions is to reduce the bit size of the data in order to reduce the capacity of the storage media and when sending data it becomes faster and more precise. In this research, the stout code algorithm is used to compress game files and to change the size of the original file and the bits will be reduced.

The procedure for compressing game files carried out in this research is by searching/selecting the game file to be compressed, then carrying out the compression process on the game file which applies the stout code algorithm, then after the compression process is complete the size of the game file will be smaller. Then the decompression process will apply the same stout code algorithm as the compression process, then the difference is in this decompression process, the game file that was previously compressed has a smaller file size, so this decompression process does the opposite, the decompression process will restore the size. the file is as normal as before, what is meant is the size of the game file as it was before carrying out the compression process. Below in Figure 1 we will show the procedure for the compression and decompression process of game files.

3.2 Preparation Before Implementing the Stout Code Algorithm

In the analysis carried out in this research, game files will be compressed by applying the stout code algorithm, and this research will also discuss decompression of game files by applying the stout code algorithm.
Before carrying out the game file compression process using the stout code algorithm, the first thing that must be done is to provide the game file first so that you can find out where the game file will be compressed and you must know what the name of the game file is, its size, and extension. in the game file. The following is an example of a game file that will be compressed using the stout code algorithm in figure 1 below..

Figure 1. Example of a game file that will be compressed

In Figure 1 above, an example of a game file that will be compressed is provided, before carrying out the compression process using the stout code algorithm. In the table below is the status of the game file based on the image above, and the game file that will be compressed using the stout code algorithm.

Table 2. Status of Game Files to be Compressed

Name File	Ekstensi File	Size File
hz_setup.exe	.exe	20.5 MB (21.564.500 Byte)

After the game file has been provided and the status of the game file is known, the game file is then entered into the HxD software to obtain and determine a sample of the hexadecimal numbers, so that the hexadecimal numbers can be applied to the stout code algorithm. And the reason why it is entered is a hexadecimal number, it is because it is a data compression technique when carrying out calculations manually using an algorithm related to data compression techniques, while when carrying out data compression techniques it does not require a sample of hexadecimal numbers, for example compression of image files. and text files, because image files take samples in the form of image numbers in the image and in the case of text files only the text can be used as a sample to be applied in a data compression algorithm. Apart from being able to find out the hexadecimal numbers in a file, this HxD software can also find out the ASCII characters in the file. Apart from the HxD software, there is one software that is similar to the HxD software, this software is Binary Viewer which has the same function as the HxD software, namely being able to find out hexadecimal numbers and ASCII characters in a file.
As shown in the image below, the game file will be entered into the HxD software and will take or determine a sample of 16 digits of hexadecimal numbers in the game file, and will be applied to the stout code algorithm.

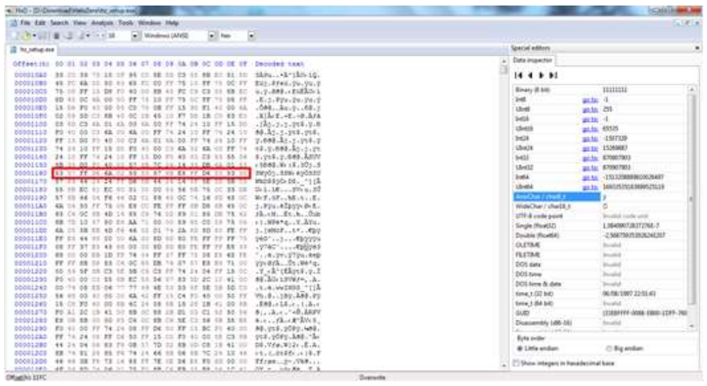


Figure 2. Sample of Hexadecimal Numbers in Game Files in HxD Software

After determining a sample of hexadecimal numbers in the HxD software in the game file in the image above, the following table below shows a sample of hexadecimal numbers totaling 16 digits.

Table 3. Samples of Hexadecimal Numbers in Game Files

53	57	FF	D6	6A	02	53	53	57	8B	E8	FF	D6	53	53	55

3.2.1 Case Example of Game File Compression Process Using the Stout Code Algorithm

After obtaining or having determined a sample of hexadecimal numbers totaling 16 digits, then determine the frequency of the hexadecimal number which is intended to determine the hexadecimal characters that appear most frequently in the sample game file of hexadecimal numbers, then continue with the next step. Following are the steps to compress game files using the stout code algorithm.

1. Make a list of the frequency of occurrence of each character that appears most often in a sample of hexadecimal numbers and sort them from largest frequency to smallest frequency.

Table 4. List of Character Appearances with the Largest to the Smallest Frequency

No	Heksadesimal	Biner	Bit	Frekuensi	Bit * Frekuensi
1	53	01010011	8	5	40
2	57	01010111	8	2	16
3	FF	11111111	8	2	16
4	D6	11010110	8	2	16
5	6A	01101010	8	1	8
6	02	00000010	8	1	8
7	8B	10001011	8	1	8
8	E8	11101000	8	1	8
9	55	01010101	8	1	8
		Jumlah Bit * Frekuensi			128 Bit

2. The next step is to look for the codeword for each existing character, which is based on the data in the table above. And to find the codeword, the first thing to do is determine the value of l, the value of l can use l = 2 and l = 3 according to the data in table 2, and in this study we will use l = 2. Then after the value of l has been determined, then enter the following formula:
   A. For the value 0 = n = 2¬¬¬¬l – 1
      because the value of l = 2, then 0 = n = 22 - 1 □ 0 = n = 3
      a n = 1, then the resulting codeword is the binary value of n which is 1 and taken as much as the value of l is 2 so that the resulting codeword is: 01

b n = 2, then the resulting codeword is the binary value of n which is 10 and taken as much as the value of l is 2 so the resulting codeword is: 10

c n = 3, then the resulting codeword is the binary value of n which is 11 and taken as much as the value of l is 2 so the resulting codeword is: 11

B. For the value n = 2l
because the value of l = 2, then n = 22 □ n = 4

a n = 4, with a binary value of 100, then the value L = 3 is taken from the bit length of the binary value n and the next step is to find the R value.
R2 (3-1-2) = 0, the binary value is 0 and the value of l is taken as much as 2 so that the value of R = 00
R¬l (L-1-l) B (n, L) = 00 100

b n = 5, with a binary value of 101, then the value of L = 3
R2 (3-1-2) = 0, the binary value is 0 and the value of l is taken as much as 2 so that the value of R = 00
Rl (L-1-l) B (n, L) = 00 101

c n = 6, with a binary value of 110, then the value of L = 3
R2 (3-1-2) = 0, the binary value is 0 and the value of l is taken as much as 2 so that the value of R = 00
Rl (L-1-l) B (n, L) = 00 110

d n = 7, with a binary value of 111, then the value of L = 3
R2 (3-1-2) = 0, the binary value is 0 and the value of l is taken as much as 2 so that the value of R = 00
Rl (L-1-l) B (n, L) = 00 111

e n = 8, with a binary value of 1000, then the value of L = 4
R2 (4-1-2) = 1, the binary value is 1 and the value of l is taken as much as 2 so that the value of R = 01
Rl (L-1-l) B (n, L) = 01 1000

f n = 9, with a binary value of 1001, then the value of L = 4
R2 (4-1-2) = 1, the binary value is 1 and the value of l is taken as much as 2 so that the value of R = 01
Rl (L-1-l) B (n, L) = 01 1001

3. After forming the codeword, the next step is to replace all the characters with the codeword that has been generated, and the process can be seen in table 3.4 below*:*

Table 5. Changing All Characters to Codewords

n	Heksadesimal	Frekuensi	Codeword	Bit	Frekuensi * Bit
1	53	5	01	2	10
2	57	2	10	2	4
3	FF	2	11	2	4
4	D6	2	00 100	5	10
5	6A	1	00 101	5	5
6	02	1	00 110	5	5
7	8B	1	00 111	5	5
8	E8	1	01 1000	6	6
9	55	1	01 1001	6	6
		Jumlah Frekuensi * Bit			55 Bit

Based on the table above, after changing all the hexadecimal characters to codewords, all the characters in the hexadecimal sample will produce a bit string or in other words will produce a new bit in the hexadecimal character, as follows in the table below the ordering of the sample of all hexadecimal numbers. will be changed to the codeword value generated in the previous steps, thus producing a bit string for all hexadecimal samples.

Table 6. Bit strings generated for all samples of hexadecimal numbers

53	57	FF	D6	6A	02	53	53
(01)	(10)	(11)	(00 100)	(00 101)	(00 110)	(01)	(01)

57 (10)	8B (00 111)	E8 (01 1000)	FF (11)	D6 (00 100)	53 (01)	53 (01)	55 (01 1001)

Dengan total bit yaitu 55 *bit*

From the above process, the resulting bit string is "0110110010000101001 10010110001110110001100100010101 1001". The next stage is to add padding and flagging. In a computer, 1 character is represented by an ASCII (American Standard Code For Information Interchange) number of 8 bits in binary numbers. If it turns out that the amount of data in these bits is not a multiple of 8. Then a new variable is formed as an addition to the data in these bits so that the bits in the data are divisible by 8, this variable is called padding and flagging. The following is the addition of padding and flagging to the bit string based on the results from the table above.

A. Adding Padding
In a sample hexadecimal number, the resulting bit string is:
"0110110010000101001100101100011101100011001000101011001" With a total of 55 bits. And 55 is not divisible by 8 and leaves 7, or in other words:
55 Mod 8 = 7
Denote the remainder of the quotient by “n”, then enter the following formula to add padding:
7 – n + “1”
7 – 7 + “1” = 1 □ There is no addition of the number 0 because 7-7 has no remainder

B. Addition of Flagging
To add flagging, you can use the following formula:
9 – n
9 – 7 = 2 = 00000010 □ Binary number in number 2
C. Addition Results Using Padding and Flagging
So the addition of padding and flagging will be as follows:
“0110110010000101001100101100011101100011001000101011001100000101” □ With a total of 64 bits.
The next step in the table below divides the bit string into bits, then converts it into a character.

Table 7. Bit string divided into 8 bits and converting them into characters

Biner	Desimal	Karakter
01101100	108	l
10000101	133	à
00110010	50	2
11000111	199	Ç
01100011	99	c
00100010	34	“
10110011	179	³
00000101	5	♣

Hasil dari proses kompresi menghasilkan karakter berikut ini:
là2Çc“³♣

With a total bit length of 64 bits, the performance results of the stout code algorithm can be calculated by:

A. Compression Ratio □ Compression Ratio (CR)
Compression ratio is the percentage between data that has been compressed and data that has not been compressed.

$$Cr = \left(\frac{Ukuran\ Data\ Setelah\ Di\ Kompresi}{Ukuran\ Data\ Sebelum\ Di\ Kompresi}\right) \times 100\% \quad (3)$$

$$Cr = \frac{64}{128} \times 100\%$$

$$Cr = 50\% \; (3)$$

B. Space Saving □ Space Saving (SS)
In saving space between uncompressed data and uncompressed data.

$$SS = 100\% - Cr \quad (4)$$

$$SS = 100\% - 50\%$$
$$SS = 50\% \quad (4)$$

From the compression process on the game file above using the stout code algorithm, the game file data obtained before compression is 128 bits and the compression results obtained on the game file using the stout code algorithm are 64 bits, which means if 1 character is equal to 8 binary bits or equal to 1 byte in computer units, therefore the size of the compressed game file is 8 bytes, the size of the previously compressed game file in the case example in this study is 20.5 MB (21,564 ,500 bytes) and after compression using the stout code algorithm the size of the game file is 20.5 MB (21,564,492 bytes). The following is the table below on the status of game files that have been compressed using the stout code algorithm, as follows:

Table 8. Size Results of Game Files After Compression Using the Stout Code Algorithm

Nama File	Ekstensi File	Size (Ukuran) File
hz_setup.exe	.exe	20.5 MB (21.564.**492** Byte)

3.2.2 Decompression Process

After converting the characters into bit strings, then remove the padding and flagging. To eliminate padding and flagging by taking the last 8 bits and changing them to a decimal number, then denote it with "n", below in the table below are the results of the previous compression process based on the data table and will be decompressed.

Table 9. Game file data that has been previously compressed and will be decompressed

Karakter	Desimal	Biner
l	108	01101100
à	133	10000101
2	50	00110010
Ç	199	11000111
c	99	01100011
“	34	00100010
³	179	10110011
♣	5	00000101
Hasil dari proses kompresi menghasilkan karakter berikut ini: là2Çc“³♣		

A. Takes the Last 8 Bits in the Bit String
n = 00000010 = 2 □ Changes to a decimal number
Then use the formula "7+n" as follows:
7 + n
7 + 2 = 9 □ Remove from the bit string the last 9 bits
□ The last 9 bits of the bit string have not yet been removed:
“0110110010000101001100101100011101100011001000101011001100000101” □ Bits marked in red are removed
□ The last 9 bits of the bit string when removed:
“0110110010000101001100101100011101100011001000101011001”, the last 9 bits have been removed.

Based on the previous decompression results where the initial value of the bit string during the compression process was 64 bits and became 55 bits again, in accordance with the results obtained at the previous stage during the performance steps of using the stout code algorithm on hexadsimal samples , thus producing the codeword value, due to the reduction of the last 9 bits of the bit string in the hexadecimal sample during the decompression process, the value obtained in the hexadecimal sample is in accordance with table 3.5, previously the bit string was 55 bits. Below is table 3.8 the results of the decompression process which appear to be in accordance with those in table 3.5 previously.

Management of Information System Journal
Vol 2, No 3, Juli 2024, Hal. 1 - 9
ISSN: 2964-9455 (media online)
DOI: 10.47065/mis.v2i3.1916
https://journal.fkpt.org/index.php/MIS

Table 10. The results of the decompression process become a bit string result at the beginning of 55 bits

53 (01)	57 (10)	FF (11)	D6 (00 100)	6A (00 101)	02 (00 110)	53 (01)	53 (01)
57 (10)	8B (00 111)	E8 (01 1000)	FF (11)	D6 (00 100)	53 (01)	53 (01)	55 (01 1001)
Dengan total bit yaitu 55 *bit*							

Table 11. Game file data that has been decompressed becomes the data it was before compression using the Stout Code algorithm

No	Heksadesimal	Biner	Bit	Frekuensi	Bit * Frekuensi
1	53	01010011	8	5	40
2	57	01010111	8	2	16
3	FF	11111111	8	2	16
4	D6	11010110	8	2	16
5	6A	01101010	8	1	8
6	02	00000010	8	1	8
7	8B	10001011	8	1	8
8	E8	11101000	8	1	8
9	55	01010101	8	1	8
		Jumlah Bit * Frekuensi			128 Bit

Table 12. Size Status of Game Files After Decompression

Nama File	Ekstensi File	Size (Ukuran) File
hz_setup.exe	.exe	20.5 MB (21.564.**500** Byte)

4. CONCLUSION

Based on the application of the Stout Code algorithm, with this algorithm it can be seen that a game file whose initial size was 20.5 MB (21,564,500), becomes 20.5 MB (21,564,492)..

REFERENCES

[1] S. D. Nasution, "Data Compression Using Stout Codes," vol. 3, no. 1, pp. 28–33, 2019.
[2] D. Salomon, *Data Compression The Complete Reference FourthEdition*, vol. 53, no. 9. 2007.
[3] R. Goyena and A. . Fallis, *Handbook of Data Compression 5th*, vol. 53, no. 9. 2019.
[4] Nasution and B. Purba, "PENERAPAN ALGORITMA EVEN-RODEH PADA APLIKASI KOMPRESI FILE," no. November, 2019.
[5] Tominanto and Subinarto, "TEKNOLOGI INFORMASI KESEHATAN III - ALGORITMA DAN PEMROGRAMAN," no. Agustus, 2018. S. D.
[6] Henderi, "Object Oriented Modelling With Unified Modeling Language (Uml)," *5 Novemb. 2009*, no. June, p. 77, 2009.
[7] S. Santoso and R. Nurmalina, "Perencanaan dan Pengembangan Aplikasi Absensi Mahasiswa Menggunakan Smart Card Guna Pengembangan Kampus Cerdas (Studi Kasus Politeknik Negeri Tanah Laut)," *J. Integr.*, vol. 9, no. 1, pp. 84–91, 2017.
[8] T. Herman, A. Jones, M. Macdonald, and R. Rajan, *Visual Basic 2008 Recipes*. 2008.