DEVELOPMENT APPLICATION DATA COMPRESSION USING LEMPLE ZIV MARKOV CHAIN ALGORITHM (LZMA) AND LEMPLE ZIV STORER SYMANSKI ALGORITHM(LZSS) ON INTRANET

Subangkit Ramadhan[1]

[1] Teknik Informatika – Universitas Komputer Indonesia
Jl. Dipatiukur 112-114 Bandung
Email : subangkit.ramadhan@gmail.com[1]

ABSTRACT

In general, the size of the data residing at the moment has a very large size. So that when sending large data through the network would result in waste of the network resource usage when the data is accessed by other users and make the storage media is less efficient in storing such data.
File compression method used is the algorithm LZSS (Lemple Ziv Storer Symanski) and combined with LZMA method which is lossless. Expected with both these algorithms can reduce the time required in the data transmission process, will result in the value of the optimal data compression ratio so that the size of the data that has undergone the compression process will have a relatively smaller size compared to the size of the data before the compression.
From the results of the testing white box and black box testing can be concluded that the function of each application running properly. As well as the compression process can shorten the time in the file transfer process. The compression process can also save on the use of storage media.

Keywords : *Data compression, Algorithm, LZSS, LZMA, lossless, Intranet.*

1. PRELIMINARY

Data is important and has a very important role in the activities of computer use. In general, the size of the data residing at the moment has a very large size. So that when sending large data through the network would result in waste of the network resource usage when the data is accessed by other users and create more storage media is less efficient in storing such data.
There are studies that have been done previously with regard to an intranet network, including research conducted by Muhammad Nur Kamid with the title "Analysis of the effectiveness of Muria Kudus University Intranet Network" [1]. In the study explained that the existence of a condition delay when the traffic tends to be high when network usage increases, so it needs time and bandwidth usage during the transmission process will increase. Compression is the process of changing a set of data into a form of code in order to save storage space requirements and the time to perform data transmission. The data compression encode each character in the message or data to encode each character in the message or data with a shorter code [2].
There are studies that have been done previously associated with compression on LAN-based networks, including the research done by Ningning Hu with the title "Network Aware Data Transmission With Compression" [3]. It can be concluded, that the compression process can reduce the time required to perform the process of sending data. Only at the conclusion of the resulting application research separates the compression process and transfer so that the compression process is done individually first and then the result data transmitted from the compression process. Enabling the addition of time because the process happens separately.
Lemple Ziv Markov Chain algorithm (LZMA) is a data compression technique in which a renewal of LZ77. While Lemple Ziv algorithm Storer Symanski (LZSS) is a lossless data compression algorithm modified from LZ77, named after its creator James Storer and Thomas Szymanski (built on the work of Lempel and Jacob Ziv abraham). Referenced of the issue and the research that has been done before, the research that will be done is to conduct application development of data compression, using a combination of algorithms Lemple Ziv Markov Chain (LZMA) and algorithms Lemple Ziv Storer Symanski (LZSS) when data transmission occurs on the network LAN. Expected with both of these algorithms can reduce the time required in the data transmission process, will generate the value of the compression ratio optimal data so that the size of the data that has undergone the compression process will have a size that is relatively small compared to the size of the data before the compression and the use of storage media Data can be more efficient.

1.1 Data Compression

Compression means compress / shrink the size. Data compression is the process of encoding information using bits of information-bearing or other units

lower than the representation of the data that is not encoded with a particular encoding system [4]. Examples of simple compression we usually do in everyday life is the example we truncate the words we use but the word has been understood by any person or used in general. For example: the word "yang" compressed or shortened to the word "yg". Fundamental standards of data compression techniques have 2 compression techniques are lossy compression techniques and compression techniques loseless. For lossy compression technique there is some information that is omitted in the data during the compression process occurs. As for the technique of compression loseless opposite of lossy compression technique in which no information is lost during the compression process occurs.

1.1.1 Important Factors Data Compression

Dalam kompresi data, terdapat 4 (empat) faktor penting yang perlu diperhatikan, yaitu :

a. Time Process
b. Completeness
c. Ratio Compress
d. Optimaly

1.1.2 Type of Data Compression

In the process of data compression, the compression process generally has two types, namely:

1. Lossy Compression is the type of data compression that has the nature of the data source (native) and compressed data after the decompression process if it does not fit the results back to the data source. Generally this type of data compression used for the compression process on the data type of image [6].

   Gains on the type of compression is to make the value of the compression ratio (the ratio between the size of the original file with the file compression process) is quite high. This is because lossy way of working is to eliminate some data from a file, causing the data to a file that has been passed through the compression process is lost when the decompression process. However, the data that was eliminated usually is data that less attention or beyond the reach of humans, so the elimination of the data is not likely to affect the process of human interaction with the file. For example in pengkompresian text file, lossy compression eliminates data from a text file that has a very high frequency / low that many omit information in the data that are considered important. Some types of data can still tolerate this type of compression is the image, audio and video.

2. Loseless Compression is the type of data compression that has the nature of the data source (native) and compressed data after the decompression process does return will have the same file structure [6].

1.2 Lemple Ziv Storer Symanski Algorithm (LZSS)

Lemple Ziv Storer Symanski Algorithm (LZSS) is a data compression algorithm loseless modified type of LZ77 algorithm, named after its creator James Storer and Thomas Symanski (which builds on the work of Abraham Lempel and Jacob Ziv). Lemple Ziv Storer Symanski Algorithm (LZSS) one compression symbol sequence data (eg, byte-data) to identify the sequence of recurring symbols in the input, and replaces sequences of smaller symbols. Implementation Lemple Ziv Storer Symanski Algorithm (LZSS) can adjust the number of bits allocated by changing the length of the byte size (among other parameters) to get a pretty good compression performance [4].

1.2.1 How it Works LZSS Algorithm

How it works Lemple Ziv algorithm Storer Symaski (LZSS) consists of a literal bytes or partner (offset, match length). Literal bytes is a simple byte input that is copied directly to the output. pair (offset, match length) explaining the process of returning to an input offset distance of data that will be found a match length of the data, the algorithm uses two flags at the output, namely:

1. If *flag* "0", next byte literal.
2. If *flag* "1", a pair (offset,match length).

For Example :

1. *Input* data = "ABCDABCA"

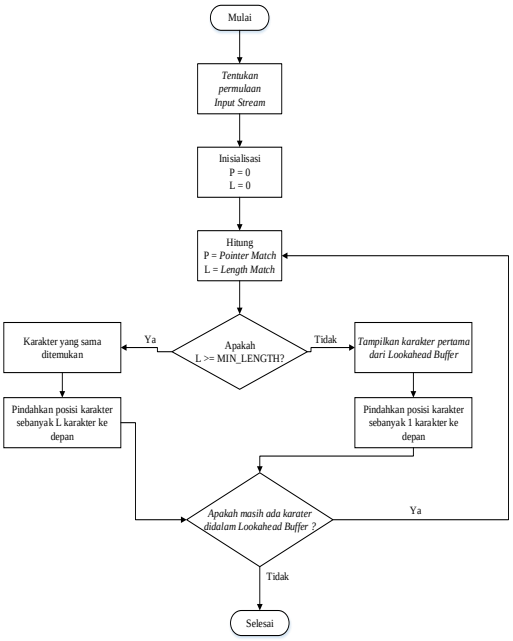
Start with byte "A", would have been there before? "NO". Enkodenya is a literal byte "0" followed by the data "A". Output data = 0 "A". Process continues until the next byte is equal to the previous byte. In other words, the output data into: 0 "A" 0 "and" 0 "C" 0 "D".

2. *Input* data = "ABCDABCA"

The next byte is "A", would have been there before? "YES" 4 previous bytes (offset = 4). How many bytes the same? 3 bytes of the same "ABC", it will be coded "1" (to identify that there is the same data) followed by (4.3). Output data = 0 "A" 0 "and" 0 "C" 0 "D" 1 (4.3).

3. *Input* data = "ABCDABCA"

The next byte is "A", would have been there before? "YES" 3 bytes before (offset = 3). How many bytes the same? 1 byte of the same "A", it will be coded "1" followed (3.1). Output data = 0 "A" 0 "and" 0 "C" 0 "D" 1 (4.3) 1 (3.1).



Picture 1 Flow Chart Of Algoritma LZSS

1.3 Lemple Ziv Markov Chain Algorithm (LZMA)

LZMA is the renewal of the LZ77 algorithm, using Markov chain and entropy coding [4]. This algorithm method using the input stream as part dictionary to be used in data compression. Encoder receiving input from right to left along with all the data going through the compression process. This method is also referred to as sliding windows. On this window is divided into two parts, the left is called the search buffer and the right side is called the look-ahead buffer, which contains the text to be compressed. As with LZ77, LZMA also produces an output that stores the distance, length, and the next symbol as a unified file compression results. Distance is the distance between the characters is sought in the look-ahead buffer that is the same as the character in the search buffer. Length is the length of the longest sequence of characters in the search buffer that matches the look-ahead buffer. Next symbol is a character in the look-ahead buffer after the longest sequence of characters selected. LZMA compression on the same principle with the deflate compression technique, but using the range Huffman coding in lieu of coding. In the real case, the result of the compression process using LZMA algorithm in the form of a file that contains a series of representations of the unitary character of the compressed file that consists of distance, length, and the next symbol.

1.3.1 How it Works LZMA Algorithm

How it works Lemple Ziv Markov Chain algorithm (LZMA) in the compression process is initiated by selecting the files to be compressed. The selected file will be recognized as an input stream that is a row of symbols / characters in length according to the size of large files. This algorithm will read the input stream by using the search buffer of size 2w bits and lookahead buffer. Where w is the rank value that is limited by the amount of memory in the hardware.

The compression process begins by reading the input stream in units of bytes and the last record n bytes are read, if it finds the same wording it will provide output values called tokens. Where the token has the following structure:

T = (x,y,Z)
x = *offset* (2 bit)
y = *length* (2 bit)
z = *next symbol* (1 *byte*)

During the encode process, the program also records the probability of each character are adjacent and form a Markov model in the arrangement directed graph.

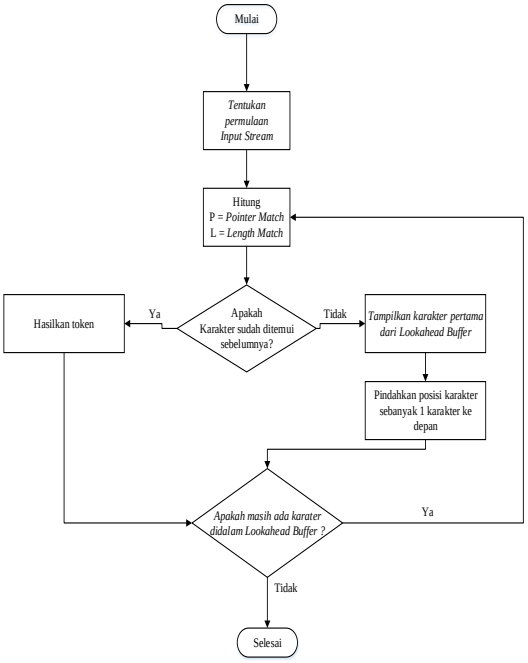
For example, the data will be compressed as follows: AABBCBBAABC.

Below is a table for the encoding process is presented in Table 1.

Table 1 hasil proses *encoding* algoritma LZMA

Step	Pos	Match	Token
1	1	--	(0,0,”A”)
2	3	A	(1,1,”B”)
3	5	B	(1,1,”C”)
4	6	BBB	(3,2,”A”)
5	11	ABA	(7,2,”C”)

The following is a description of table 1:
Encoding process: the process of encoding input Stream
Pos: position of each character in the input stream
Step: step encoding process;
Match: the same characters found
Token: rows of symbols comprising encoding process results from the offset value, length and the next character
From the above description, the flowchart for the process encode using LZMA algorithm can be presented as follows:

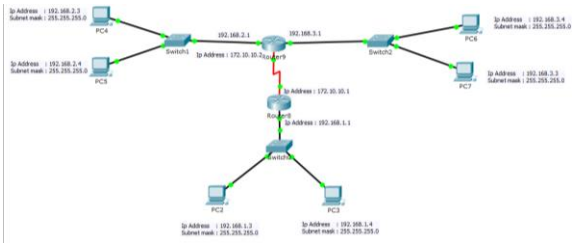


Picture 2 Flow Chart LZMA Algorithm

2. CONTENT OF RESEARCH

2.1 Network Scheme

To simulate this software, it takes intranet that is used in the process of file transfer and static routing has been implemented on the intranet network. Here is a network architecture that will be used in testing this software:



Picture 3 Network Scheme

Table 2 Table Routing Static

Router	Destination	Mask	Next Hop	Keterangan
Router8	192.168.1.0	255.255.255.0	-	*Direct Connection*
	192.168.2.0	255.255.255.0	172.10.10.2	*Indirect Connection*
	192.168.3.0	255.255.255.0	172.10.10.2	*Indirect Connection*
Router9	192.168.1.0	255.255.255.0	172.10.10.1	*Indirect Connection*
	192.168.2.0	255.255.255.0	-	*Direct Connection*
	192.168.3.0	255.255.255.0	-	*Direct Connection*

In the case of the Picture and Table above is an intranet network which has 3 different networks that have been associated with 2 routers and implement static routing systems can be seen in Table 2 above. Therefore, when a PC will make the process of data transfer, then the PC must menulusiri first predetermined path on the routing Table.

So based Picture 3 and Table 2 has been described, as an example of a scenario for data transmission to the buildup of the delivery process with ip 192.168.1.2 PC2 to PC4 with ip 192.168.2.3 using the software to be built can be seen as follows:

1. The PC2 will make the determination of the purpose and connect to the destination, in this example scenario goal of PC4 with ip 192.168.2.3.
2. After PC2 PC4 connected with the goal, then the next PC2 will make the process of data transfer, the software transfer process will begin with the process of compression algorithms using LZSS which will take place at PC2.
3. After the compression process on PC2 completed, the data will be forwarded to the gateway beforehand. Once the package is received by the gateway, then the package will be checked by the gateway for the specified destination address and the gateway will check the track that will be used in the delivery of the package.
4. After the address and track the delivery of the package is obtained, then the packet will be sent to the destination address by using the path that has been determined by the previous gateway.
5. Before the data packet is received by the destination, the compression process by using algorithms LZMA on the package will do.
6. Once the data is received by the destination, then this application will close the connection used in the transfer process.

So based on the example scenario above, the delivery process is expected if congestion occurs in the process of sending data packets as in Picture 3, is expected to reduce the workload of the router so that it can reduce the time delay that occurs due to accumulation of data packets at the router itself.

2.2 Encoding

The workings of the encoding process consists of the encoding process using LZSS first algorithm produces output results after the encoding process using the algorithm and process of encoding using Lemple Ziv Markov Chain algorithm (LZMA) against the results or output of the encoding process LZSS algorithm.

Video basically composed of a series of frames.

Each frame is an image. An image is represented by a field that each element represents the value of intensity. The points where the sampled image is referred to as picture elements or commonly known as pixels.
Step perform the encoding process on the source file is as follows:
1. First, the encoding process begins by reading a video file.
2. After that the file will be converted into an array of bytes that come from a collection of pixels of the source file.
3. After getting an array of bytes of the file, the second algorithm is looking for similarities between each bytes that are in the bytes array.
4. If found similarities then bytes will be encoded with the provisions of each algorithm.
5. Once all bytes array finishes traced, then the results of the search are saved into a new array of bytes, then a new array of bytes is reassembled and turned it into a file encoding format results lzsa

At this encoding process starts with the encoding process using algorithms LZSS first. Here are the stages in the process of encoding by using LZSS against input sample stream derived from the source file.
1. Input data = "7575759596" Start with byte "7", if it has been there before? "NO". It will be preceded by a literal byte "0" and followed by the byte "7". Then the output data = 0 7.
2. Input data = "7575759596" Followed by byte "5", if it has been there before? "NO". It will be preceded by a literal byte "0" and is followed by the byte "5". Then the output data = 07 05.
3. Input data = "7575759596" Followed by byte "7", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes that are designated with an existing byte is 2.Maka value offset = 2, and after the offset value determined will be checked against the next byte whether pre-existing or not. Because the next byte has similarities then penulusuran continued to be found which does not have the same byte before. So that the output data will be preceded by a literal byte "1" followed by the pair (offset, length match) ie in this example the pair is (2,3). Then the output data = 07 05 1 (2.3).
4. Input data = "7575759596" Followed by byte "9", if it has been there before? "NO". It will be preceded by a literal byte "0" and is followed by the byte "9". then Output data = 07 05 1 (2.3) 09.
5. Input data = "7575759596" Followed by byte "5", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes that are designated with an existing byte is 6.Maka offset value = 6, and after the offset value determined will be checked against the next byte whether pre-existing or not. Because the next byte has similarities then penulusuran continued to be found which does not have the same byte before. So that the output data will be preceded by a literal byte "1" followed by the pair (offset, length match) ie in this example the pair is (2,2). Then the output data = 07 05 1 (2.3) 09 1 (2.2).
6. Input data = "7575759596" Followed by byte "6", would have been there before? "NO". It will be preceded by a literal byte "0" and is followed by the byte "6". Then the output data = 07 05 1 (2.3) 09 1 (2.2) 06.
After the completion of data input with the encoding process using algorithms LZSS, the results of the process will be reassembled into a temporary file that will be used by subsequent encoding process. So the results of the encoding process using algorithms LZSS against input sample data. can be seen in Table 3.

Table 3 The Result of Encoding LZSS

Output
0 7
0 5
1 2 3
0 9
1 2 2
0 6

After getting the results of the encoding process using the LZSS algorithm, the result of the encoding process using LZSS will be accommodated into a temporary file which will then be used in the encoding process using LZMA algorithm. Here are the stages in the process of encoding by using LZMA on samples obtained from the input stream encoding results using LZSS that have been presented in Table 3.
1. Input data = "07051230912206" Start with byte "0", if it has been there before? "NO". Then the output data = <0.0, "0">.
2. Input data = "07051230912206" Followed by byte "7", if it has been there before? "NO". Then the output data = <0.0, "0"> <0.0, "7">.
3. Input data = "07051230912206" Followed by byte "0", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes that are designated with an existing byte is 2. Then the offset value = 2, and after the offset value determined will be checked against the next byte whether pre-

existing or not. Due to the subsequent bytes do not have in common then penulusuran stopped. So that the resulting output token is (2.1, "5"). Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5">.
4. Input data = "07051230912206" Followed by byte "1", if it has been there before? "NO". Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1">.
5. Input data = "07051230912206" Followed by byte "2", if it has been there before? "NO". Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1"> <0.0, "2">.
6. Input data = "07051230912206" Followed by byte "3", if it has been there before? "NO". Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1"> <0.0, "2"> < 0.0, "3">.
7. Input data = "07051230912206" Followed by byte "0", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes being designated with the existing bytes is 5. So the offset value = 5, and after the offset value determined will be checked against the next byte whether pre-existing or not. Due to the subsequent bytes do not have in common then penulusuran stopped. So that the resulting output token is (5.1, "9"). Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1"> <0.0, "2> <0 , 0, "3> <5.1," 9 ">.
8. Input data = "07051230912206" Followed by byte "1", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes being designated with the existing bytes is 5. So the offset value = 5, and after the offset value determined will be checked against the next byte whether pre-existing or not. Due next byte already have in common then penulusuran continued until the bytes which do not have the same length or until the end of the input stream. So that the resulting output token is <5.2, "2">. Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1"> <0.0, "2> <0 , 0, "3"> <5.1, "9"> <5.2, "2">.
9. Input data = "07051230912206" Followed by byte "0", if it has been there before? "YES". Due been found before then calculated how the distance between the bytes being designated by the pointer with bytes that have been there before. In this example, the distance between the bytes being designated with the existing bytes is 5. So the offset value = 5, and after the offset value determined will be checked against the next byte whether pre-existing or not. Due next byte already have in common then penulusuran continued until the bytes which do not have the same length or until the end of the input stream. So that the resulting output token is <5.1, "6">. Then the output data = <0.0, "0"> <0.0, "7"> <2.1, "5"> <0,0, "1"> <0.0, "2> <0 , 0, "3"> <8.1, "9"> <5.2, "2"> <5.1, "6">

Tokens generated from the encoding process can be seen in Table 4.

Table 4 The Result Of Encoding With LZMA

Output
<0,0,"0">
<0,0,"7">
<2,1,"5">
<0,0,"1">
<0,0,"2">
<0,0,"3">
<8,1,"9">
<5,2,"2">
<5,1,"6">

On Table 4 has described the results of an example of the encoding process that every output in the form of tokens in lieu of each byte contained in an array of bytes that is used as the source file. Once the encoding process is completed, the next is the establishment or re-writing of tokens generated into a new file format encoding results with new extensions which lzsa.

2.3 Software Testing

Software testing, conducted with the system environment based on the Picture 3 and the configuration of each Table routing in accordance with Table 2.

2.3.1 Black Box Testing

Based on test scenarios Black Box, the results of the test can be seen as follows:

1. Compression Testing

Black Box Testing for the compression process for the data right and wrong data can be seen in Table 5 and Table 6.

Table 5 Compress Process Testing Data Normal

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.avi	Perform the compression on process on the file	Show The message Compression Success	Accept

Table 6 Compress Process Testing Data One

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.mkv	Perform the compression on process on the file	Don't Show The message Compression Success	Reject

2. Testing process decompress

Black Box Testing for the decompression process for the data right and wrong data can be seen in Table 7 and Table 8.

Table 7 Testing Process Data Decompress Normal

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.lzsa	Perform decompression process on the file	Show the message Decompression Success	Accept

Table 6 One test of Process Decompress Data

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.rar	Perform decompression process on the file	Don't Show the message Decompression Success	Reject

3. Testing the transfer process

Black Box Testing for the transfer process to the data right and wrong data can be seen in Table 9 and Table 10.

Table 9 Normal Data Transfer Process Testing

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.avi	Perform the compression process and transfer files simultaneously with the desired destination	Show the message transfer success	Accept

Table 10 Data Transfer Process Testing False

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.rar	Perform the compression process and transfer files simultaneously with the desired destination	Don't Show the message transfer success	Reject

4. Choose the file enter the testing process

Black Box testing process to select files enter the data correctly and the data one can be seen in Table 11 and Table 12.

Table 11 Choose Process Testing Data File Enter Normal

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.avi	Detecting Input the address as the source file	Show the path of file input	Accept

Table 12 Testing Process Select File Enter Data One

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
File Input big_buck_bunny_360p214.rar	Detecting Input the address as the source file	Don't Show the path of file input	Reject

5. Testing process select an output location

Black Box testing to select the output location for the data right and wrong data can be seen in Table 13 and Table 14.

Table 13 Testing Process Select Location Data Output Normal

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
Alamat Lokasi Penyimpanan C:\Users \Documents	Detect the storage location of the file as output data	Show the output path	Accept

Table 14 Testing Process Select Location Output Data One

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
Alamat Lokasi Penyimpanan C:\Users \Documents	Detect the storage location of the file as output data	Don't Show the output path	Reject

6. Testing the connection process

Black Box testing for the connection to the data right and wrong data can be seen in Table 15 and Table 16.

Table 15 Testing Process Data connection Normal

Cases and Test Results (Data True)			
Input	Which are expected	Observation	Summary
Alamat IP Adrdress tujuan 192.168.1.5	Connecting with the desired destination	Show the message connected	Accept

Table 16 Testing Process Data Connection One

Cases and Test Results (Data One)			
Input	Which are expected	Observation	Summary
Alamat IP	Connecting	Show the	Reject

Adrdress tujuan 192.168.1.5	with the desired destination	message can't connect	

2.3.2 White Box Testing

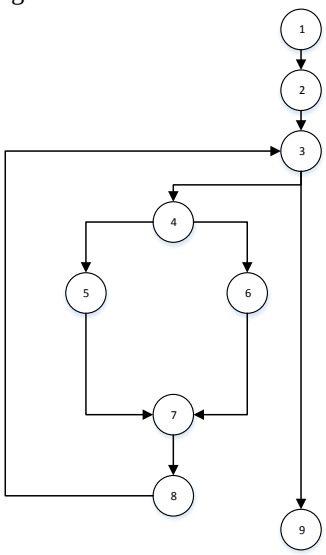
White Box Testing is used to ensure all orders and conditions on applications executed minimal, white box testing using two tools are used for the flow graph mengPicturekan groove of algorithms and graph matrix used for generating flow graph. The white box testing is as follows:

2.3.2.1 LZSS Algorithm Testing

The following is a pseudocode algorithm LZSS:

1: read the beginning of the input stream
2: initialization p = 0, l=0
3: Do look ahead buffer while there are still characters
Read *input stream*
Calculate p and l
4: if (l >= Min_Length)
5: save position of character on p
save p on record
character move forward as much as L
6: else
Save character on record
Move to next character
7: endif
8: endDo
9:end

Based on the above pseudocode then LZSS flow graph algorithm used can be seen in Picture 4.



Picture 4 Flow Graph Algoritma LZSS

Picture 4 of which have been described above, cyclomatic complexity can be calculated as follows:
V(G) = E – N +2
V(G) = 10 – 9 +2
V(G) = 3

where:
E = the number of edges in the flow graph
N = the number of nodes in the flow graph
So, cyclomatic complexity to Picture 4.7 is 3. Based on the cyclomatic complexity, then elected three paths consisting of
Path 1 = 1-2-3-9
Path 2 = 1-2-3-4-5-7-8-3-9
Path 3 = 1-2-3-4-6-7-8-3-9
To test the above base path used graph matrix. Graph matrix is a rectangular matrix which has the same size as the number of nodes in the flow graph. The graph matrix in the flow graph algorithm LZSS above are as follows:

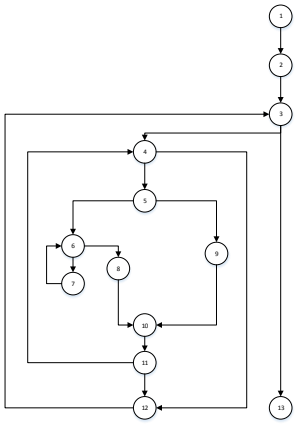
N	1	2	3	4	5	6	7	8	9	Sum-1
1		1								0
2			1							0
3				1					1	1
4					1	1				1
5							1			0
6							1			0
7								1		0
8			1							0
9										0
Jumlah +1										3

2.3.2.2LZMA Algorithm Testing

The following is a pseudocode algorithm LZMA:

1: read starters input stream
2: initialization p = 0, l=0
3: Do while *lookahead buffer* still has character
read *input stream*
4: Do while *search buffer* still has character
read *search buffer*
5: if (*byte di search buffer == byte onlookahead buffer*)
6: do while *byte* next *di lookahead buffer == byte di search buffer*
Calculate sum of the same byte
7: endDo
8: save the sum of byte on l
Save the same offset value of the same byte position on *search buffer* until the current byte position on p
Create the token
9: else
Create the token
10: endIf
11: endDo
12: endDO
13: end

Based on the above pseudocode then flow graph LZMA algorithm used can be seen in Picture 5.



Picture 5 Flow Graph Algoritma LZMA

Picture 5 of which have been described above, cyclomatic complexity can be calculated as follows:

V(G) = E – N +2
V(G) = 17 –13 +2
V(G) = 6

where:

E = the number of edges in the flow graph
N = the number of nodes in the flow graph

So, cyclomatic complexity to Picture 4.8 is 6. Based on the cyclomatic complexity, then the path chosen 6 comprising

Path 1 = 1-2-3-13
Path 2 = 1-2-3-4-12-3-13
Path 3 = 1-2-3-4-5-6-7-6-8-10-11-4-12-3-13
Path 4 = 1-2-3-4-5-6-8-10-11-4-12-3-13
Path 5 = 1-2-3-4-5-9-10-11-4-12-3-13
Path 6 = 1-2-3-4-5-9-10-11-12-3-13

To test the above base path used graph matrix. Graph matrix is a rectangular matrix which has the same size as the number of nodes in the flow graph. The graph matrix on LZMA algorithm flow graph above is as follows:

N	1	2	3	4	5	6	7	8	9	10	11	12	13	Sum-1
1		1												0
2			1											0
3				1									1	1
4					1							1		1
5						1			1					1
6							1	1						1
7						1								0
8										1				0
9										1				0
10											1			0
11				1								1		1
12			1											0
13														0
													Jumlah +1	6

2.3.3 Testing Data Of The Tested

In the process of the test data, the testing process will be done through a two-part test, namely:

Testing simply by doing the compression and decompression process without the file transfer process.

Testing by the file transfer process. Where in this file transfer phase will be done during the compression process the data transmitted. This testing process will be conducted to avi video file type. From the results of this test will be taken in the form of data compression results file size, time compression and decompression, and the time and bandwidth required size of the application to perform the compression process at the time of the transfer process occurs.

In a test on the video file, the application is tested with the video file size of 673 MB and 4.94 GB. This measurement is performed to determine how long it takes the application to be able to perform the compression process and the decompression process and the value of the ratio obtained from the compression process. Compression ratio calculation can be done by comparing the size of the compressed file to its original size. Then the calculation of the ratio for each file being tested can be seen as follows:

Ratio = $\frac{250812134}{673430026}$ x 100% = 37,24%

Ratio = $\frac{1324443054}{4947349933}$ x 100% = 26,77%

Here are the results of the test data by performing compression and decompression process in the absence of file transfer processes with applications that are built. Results can be seen in Table 17.

Table 17 Results of Testing Process Compression and Decompression At file

No	Nama File	Tipe File	Ukuran (byte)		Waktu (ms)		Rasio
			File Asli	*File* hasil kompresi	kompresi	dekompresi	
1	big_buck_bunny_360p	avi	673430026	250812134	00:20:08	00:01:07	37,24%
2	Elephants_Dream_360p	avi	4947349933	1324443054	02:05:29	00:07:08	26,77%

At Table 17 described the results of testing using 2 data with different capacities to produce a different ratio values. For the first test data obtained value ratio of 37.24% of the test results, which means that after the compressed data size becomes 37.24% of the data prior to compression. As for the second test data obtained value ratio of 26.77% of the test results, which means that after the compressed data size becomes 26.77% of the data prior to compression.

In addition to the testing of the compression and

decompression process, testing will be conducted on the transfer process. Here are the results of the test data by file transfer process as well as comparing the time it takes the file transfer process in the transfer process using the time it takes the file transfer process without the application of compression results and process the original file transfer. Results can be seen in Table 18.

Table 18 And Comparative Testing Results The time required to process File Transfer

No	Nama File	Ukuran (byte)	Waktu (ms)	
		File Asli	Kirim setelah terjadinya kompresi	Kirim Tanpa Terjadinya Kompresi
1	big_buck_bunny_360p	673430026	00:00:25	00:00:59
2	Elephants_Dream_360p	4947349933	00:05:53	00:08:27

At Table 18 described the results of the testing process and shows the difference in transfer time required in performing the transfer process. It can be concluded that the use of a faster time on the file transfer process that has gone through the compression process than on the file transfer process without the compression first.

3. SUMMARY

After doing the analysis and testing of the system is built, it can be concluded as follows:
File transfer process compression results faster than on the file transfer process without the compression process beforehand.
The compression process by using algorithms and algorithms LZMA LZSS to avi video files of this type generates a file with a size of less than half the original file size, so it can save the use of storage media.

4. BIBLIOGRAPHY

[1] Nurkamid, Mukhamad, *Analisa Keefektifan Jaringan Local Area Network (Intranet) Universitas Muria Kudus*, Universitas Muria Kudus, Kudus, 2011.
[2] Bell, T.C., Cleary, J.G and Witten, I.H , *Text Compression*, Prentice-Hall. Englewood, USA, 1990.
[3] Hu, Ningning, *Network Aware Data Transmission With Compression*, In The 4th Annual CMU Symposium on Computer System, 2001.
[4] Solomon, David., *Data Compression, The Complete Reference. 4th edition*, Springer, Inc : United States of America, 2004.
[5] Salomon, D. & Motta, G., *Handbook Of Data Compression*. 5th *edition*, Springer, Inc : United States Of America, 2010.
[6] Nelson, Mark., *The Data Compression Book*, IDG Books Worldwide, Inc : United States of America, 2000.