BIOS : Jurnal Informatika dan Sains
SEAN INSTITUTE VOL : 1 NO: 01 TAHUN 2023 E-ISSN: XXXX-XXXX

TEXT MESSAGE COMPRESSION ANALYSIS USING THE LZ77 ALGORITHM

Arjon Samuel Sitio
Program Studi Teknik Informatika, STMIK Pelita Nusantara, Indonesia
Email: samuel@gmail.com

ABSTRACT
Data compression is a process for converting an input data stream (original data) into another data stream in the form of output or other (compressed) streams that have a smaller size. One of the main functions of data compression is to reduce the file size by replacing characters that are generally 8 bits in size with shorter codes. In data compression, many algorithms can be used to process input into the desired output, so it must be considered aspects such as compression ratio, space-saving, and compression speed of each algorithm.
Keywords: Compression, analysis, LZ77

1. INTRODUCTION

Along with the development of large-capacity storage media, people no longer encounter problems if they have large files. The more so if the file that we have is an image file or document or text. However, sometimes the large file size is annoying if we have to manage the storage media that we have for various data. Especially if we will send the file electronically, of course, the file capacity becomes a problem in itself. Lz77 compression algorithm is a text compression algorithm that can reduce the size of the text by dividing the same text components, so that the text size will be more concise. The lz77 compression algorithm compresses it by replacing a portion of the data with a reference to equalize the data that has been passed by the encoder and decoder.

This algorithm uses a "sliding window") which consists of search buffer and look-ahead buffer. A search buffer is used as a dictionary, while a look-ahead buffer is a buffer that contains a string to be compressed. Search buffer is a buffer that has been through the compression stage, but is used as a dictionary. The use of search buffer and look-ahead buffer can be adjusted according to the ability of the system. Namely by determining the length of each array of the search buffer and look-ahead buffer so that the lz77 compression algorithm is very suitable for use in the car. In addition to compressing text the lz77 algorithm can also be used to compress files because lz77 can compress binary or bytes, so this algorithm is very dynamic and practical.

2. METHODS

Compression

Basically any data is actually a series of bits 0 and 1. What distinguishes between a particular data with other data is the size of the series of bits and how the 0 and 1 are placed in the series of bits. For example data in the form of audio and video, in audio data a series of certain bits represents one tone, whereas in video data a series of bits represents a video stream, where in the video stream there is an image in one pixel. The more complex the data, the smaller the sequence of bits needed, so that the overall size is also greater. [1]–[3]

In storing computer data transfers, in addition to the contents of the data the meters that are no less important are the measurements. Often the data stored in a

storage media is very large so it requires more space and is inefficient. Especially if the data will be sent, the larger the size, the time required for delivery will be longer. For this reason, data compression is needed to reduce the size of data without changing the content or information contained in the data.

There are many theories and methods for data compression. One theory that is quite simple is to use Huffman coding (Huffman coding). In encoding the Huffman code, the concept of binary tree data structures is used. Huffman's code theory itself is not just one, but there are several variations, optimizations, and combinations thereof. In computer science data compression is the art of representing digital data sources and others in a more compact form. Another term for data compression is data compression. [4]–[7]

Lz77 Compression Algorithm

The lz77 compression algorithm is a lossless data compression method that will be applied in this research. This application aims to compress and decompress data specifically on the Android platform. The LZ77 algorithm was introduced in 1977 by Abraham Lempel and Jacob Ziv. LZ77 algorithm is a basic algorithm that has been developed by many people. For example, LZW, LZSS, and LZMA. The LZ77 algorithm itself is often called LZ1. This algorithm is also called the "sliding window" algorithm because it compresses data by moving the buffer where the symbols are located each time one or more symbols are compressed. There are two buffers needed for this compression algorithm, namely search buffer and look- ahead buffer. Both of these buffers play an important role in compressing a file. The reason for choosing this algorithm is that this algorithm provides the best performance for several file types compared to its modification algorithms. [8]–[10]

The LZ77 compression algorithm compresses it by replacing the data portion with a reference to equalize the data that has been passed by the encoder and decoder. This algorithm uses a "sliding window" consisting of a search buffer and a look-ahead buffer. A search buffer is used as a dictionary, while a look- ahead buffer is a buffer that contains a string to be compressed. Search buffer is a buffer that has been through the compression stage, but is used as a dictionary.

The working principle of the LZ77 algorithm searches for the same longest set of symbols at the beginning of the look-ahead buffer of the search buffer and issues a pointer to that similarity. In general, an LZ77 token has three parts: offset, length, and the next symbol in the look-ahead buffer. Offsets and lengths are pointers that indicate position and number of matches. If a match is found, LZ77 adds a pointer with a symbol behind it. If there are no matches, then a null pointer and unique symbol are generated. Then the sliding window is shifted as much as the compressed symbol based on the pointer.

3. RESULTS AND DISCUSSION

File Compression

In compressing a message has four stages, namely:

1. Design a Search Buffer and Look-ahead Buffer
   This method searches one (or more) symbols in the same look-ahead buffer as the symbols in the search buffer. The search is carried out from right to left. In practice, the comparison of the size of the search buffer with the look-ahead buffer is roughly a few thousand to 10. This algorithm searches for the same longest set of symbols at the beginning of the look-ahead buffer of the search buffer and issues a pointer to that similarity. In general, an lz77 token has three parts: offset, length, and the next symbol in the look-ahead buffer. Offsets and lengths are pointers that indicate position and number of matches. If a match is found, lz77 adds a pointer with a symbol behind it. If there are no matches, then

a null pointer and unique symbol are generated. Then the sliding window is shifted as much as the compressed symbol based on the pointer. Following is the illustration of the lz77 compression method: Suppose the word you want to compress is "beberapabebek"

Table 1 Simulation of Lz77 against strings

5	4	3	2	1							O
					b	c	b	c	r	apa	

2. Looking for the Same Character

Suppose the word to be compressed is "several digits", then the character will look for the same characters in the following table:

Table 2 Simulation of Lz77 against strings

4	3	2	1					(OUTPUT)
				b	e	r	apabebek	(0,0,b)
			b	e	b	a	pabebek	(0,0,e)
		b	e	b	e	p	abebek	(2,2,r)
E	b	e	r	a	p	e	b ek	(0,0,a)
B	e	r	a	p	a	b	Ek	(0,0,p)
E	r	a	p	a	b	e	K	(2,1,b)
Search buffer				look-ahead buffer				

3. Turn Results into Binary

To change the results from table 3 to binary numbers, it can be seen in the following table:

Table 3 results become binary

(output)	**Output Is Binary**		
	OFSET	**engthy**	**Karakter**
(0,0,b)	000	000	01100010
(0,0,e)	000	000	01100101
(2,2,r)	010	010	01110010
(0,0,a)	000	000	01100001
(0,0,p)	000	000	01110000
(2,1,b)	010	001	01100010
(6,1,b)	110	001	01100010
(2,1,k)	010	001	01101011

To distinguish which bits indicate position and length with symbols, are given a sign by adding 1 bit. Bit 0 indicates that the next 8 bits of the perogram are symbols, while bit 1 indicates that the next 6 bits contain offset and symbol length followed by 8 bits containing the symbol. Then from table 3 the results will be found as follows:

0 01100010 0 01100101 1 010 010 01110010 0 01100001 0 01110000 1 010 001 01100010 1 110 001 01100010 1 010 001 01101011

4. Getting the Character Compression
After getting the results that have been binary, the binary numbers are separated into 8 bits, so that it produces compressed characters.
0 01100010 0 01100101 1 010 010 01110010 0 01100001 0 01110000 1 010 001 01100010 1 110 001 01100010 1 010 001 01101011
Numbers are separated into 8 bits.
00110001 00011001 01101001 00111001 00011000 01001110 00010100 01 011000 10111000 10110001 01010001 01101011
The binary above is converted into characters, i.e. gets the decimal number in the following table:

Table 4 results of decimal numbers of binary numbers

49	25	105	57	24
78	20	88	184	177
81	107			

Then the characters produced from table 4 are as follows:

Table 5 results of characters from decimal numbers

1	⊦	i	9	↑
N	¶	X	.	±
Q	k			

So the characters in table 5 are the messages to be sent. Getting the Value of Compression Ratio The ratio or in comparison terms is a difference between the new size and the old size, so that the numbers and the difference between the two sizes are known. In compression it is almost certain that the size of the data bits before being compressed to the size of the bits after being compressed is different or not the same, it is very necessary to make the value of this ratio or comparison. Below is the formula for finding the ratio value in percent:
1. Change the Message to Binary
The message received will be changed to a binary character, in the following way:

Table 6 messages received

1	⊦	i	9	↑
N	¶	X	.	±
Q	k			

From the characters in table 6, it will be changed beforehand to the decimal number, which is in the following table:

Table 7 results in decimal numbers of characters

49	25	105	57	24
78	20	88	184	177
81	107			

With the decimal numbers in table 7, binary numbers can be determined easily, which is as follows:
00110001 00011001 01101001 00111001 00011000 01001110

00010100 01 011000 10111000 10110001 01010001 01101011

3. Separating Offset, Length and Character

To separate the offset, length, and character with a binary that has been obtained, it can be determined by separating the one-bit sign, which is as follows:

00110001 00011001 01101001 00111001 00011000 01001110
00010100 01011000 10111000 10110001 01010001 01101011

The results of the tag separation are as follows:
0 01100010 0 01100101 1 010 010 01110010 0 01100001 0 01110000 1 010 001 01100010 1 110 001 01100010 1 010 001 01101011

If the first bit is found 0, then the next 8 bits are characters and if the first bit is number 1, then the next 6 bits are binary of offset and length, the next 8 bits are characters.

Table 8. Binary of offset, length and symbol into characters

OFFSET	value	code	(output)
0	0	1100010	(0,0,b)
0	0	1100101	(0,0,e)
10	10	1110010	(2,2,r)
0	0	1100001	(0,0,a)
0	0	1110000	(0,0,p)
10	1	1100010	(2,1,b)
110	1	1100010	(6,1,b)
10	1	1101011	(2,1,k)

4. Take the Message

How to retrieve the message, then the compressed message will be returned to the original message (the original message), with the following steps:

Table 9 returns messages

(0,0,b)	b
(0,0,e)	be
(2,2,r)	beber
(0,0,a)	bebera
(0,0,p)	beberap
(2,1,b)	beberapab
(6,1,b)	beberapabeb
(2,1,k)	beberapabebek

So the message can be returned with the process in table 9, so that the message can return to its original character or before it is compressed "several ducks", and the message does not experience the slightest disability or damage. The lz77 algorithm is one of the message compression algorithms that can return the message to the original or before it was compressed.

4. CONCLUSION

The lZ77 compression algorithm process as one method of compressing text messages is by compressing messages using search buffer and look-ahead buffer by applying the lz77 algorithm

REFERENCE

R. Krasmala, A. Budimansyah, and U. T. Lenggana, "Kompresi Citra Dengan Menggabungkan Metode Discrete Cosine Transform (DCT) dan Algoritma Huffman," *J. Online Inform.*, 2017.

I. Hajar and Y. Prayudi, "Kompresi Teks Pada Layanan SMS Menggunakan Metode Lempel Ziv Welch (LZW)," in *KNSI*, 2009.

A. Wibowo, "Kompresi data menggunakan metode huffman," *Semantik*, 2012.

R. G. Gallager, "Variations on a Theme by Huffman," *IEEE Trans. Inf. Theory*, 1978.

M. Sharma, "Compression Using Huffman Coding," *IJCSNS Int. J. Comput. Sci. Netw. Secur.*, 2010.

A. Shahbahrami, R. Bahrampour, M. S. Rostami, and Mostafa Ayoubi, "Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards," *Int. J. Comput. Sci. Eng. Appl.*, 2011.

J. Adámek, "Huffman Codes," in *Foundations of Coding*, 2011.

T. Gunardi and R. Munir, "Implementasi Algoritma Kompresi Lz77 Pada Smartphone Blackberry," *Konf. Nas. Inform. – KNIF*, 2011.

G. Pria Utama, A. Firdaus Achmad, Siswanto, and Feriadi, "Pengamanan Data Dengan Menggunakan Algoritma Kriptografi Aes, Rc4 Dan Kompresi Lz77 Berbasis Java Pada Badan

Karantina Pertanian," *Semin. Nas. Telekomun. dan Inform. Aditya Firdaus A. Semin. Nas. Telekomun. dan Inform.*, 2016.

A. F. A. Siswanto, Feriadi, Gunawan Pria Utama, "Pengamanan Data Dengan Menggunakan Algoritma Kriptografi Aes , Rc4 Dan Kompresi Lz77," *Semin. Nas. Telekomun. dan Inform. (SELESIK 2016)*, 2016.