© 2017 IJSRST | Volume 3 | Issue 8 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
Themed Section: Science and Technology

Comparative Analysis of Sequitur Algorithm with Adaptive Huffman Coding Algorithm on Text File Compression with Exponential Method

Muhammad Fadlan[1], Surya Darma Nasution[2], Fadlina[3], Saidi Ramadan Siregar[4], Pristiwanto[5]

[1]Research Scholar. STMIK Budi Darma, Medan, Indonesia

[2,3,4,5] Department of Computer Engineering, STMIK Budi Darma, Medan, Indonesia

ABSTRACT

This research was conducted to analyze and compare sequitur algorithm with the adaptive Huffman coding algorithm in compressing text file, to find which algorithm more effective and efficient in compressing compressed text file will be compared with an exponential method to be seen the performance of each algorithm based on CR, SS, RC. Text file compression will read the current string in the text file. In the Huffman adapting algorithm the text file coding will be changed into bit form, and the sequitur algorithm will change the file based on the grammar. The test result states that Huffman adapting coding algorithm is no better than sequitur algorithm.

Keywords: Compression, Text file, Adaptive Huffman Coding, Sequitur, Exponential.

I. INTRODUCTION

The more the technology, the more data or files we want to save, or we send, but sometimes the memory capacity that we have not comparable with the data we will save; Therefore the data will be stored compressed first so that its size becomes smaller. If the size of the data can be compressed to be smaller than the original size, then automatically the memory can store more data and be concerning delivery will be faster and save the time required.

At the moment now much software used to handle data compression problems. In the process of data compression, there are some things to be considered, time process (the time that runs when data is compressed and decompressed), ratio (size of data after compression and decompression), completeness (completeness of data after the files are in compression and decompression), space savings (percentage difference in data size after compressed with data size before in compression).

Data compression has an essential role in file storage and distribution systems. Data compression can be used in multimedia fields, text documents, and database records.[1].

The adaptive Huffman coding algorithm is an extension of the Huffman analogy, where dynamically encoded files have a significant impact on the effectiveness of the tree as an encoder. Unlike the adaptive Huffman coding algorithm that works on a character-by-character basis, the sequitur algorithm operates by running the constraint diagrams and usability rules.

Previous research ever conducted by Craig G. Nevill-Manning and Ian H. Witten SEQUITUR solves a new problem by operating gradually. Also, the simple structure of this method allows it to operate in space and time which is linear in the input size. The results of this study can process 50,000 symbols per second and have been applied [2].

II. METHODS AND MATERIAL

2.1 Data Compression

The compression process is the process of reducing the size of data to produce a robust or compact digital representation but can still represent the quantity of information contained in the data. There are two types of compression, lossy compression is capable of losing specific data from the data, and lossless compression of information contained in the data results similar to the information when the data has not been compressed[3][4][5].

2.2 Text File

A text file is a computer file composed of a series of lines of text. The types of text files that fall into the category contain a series of characters without any visual format information. The contents of this category file are usually notes or personal lists, articles, books, and so on. Text files are similar to files produced by word processing programs whose principal content is textual[6].

2.3 Sequitur Algorithm

The sequitur algorithm is an algorithm based on the concept of context-free grammar. In this algorithm is known non-terminal symbol and terminal symbol. Both types of symbols are elements of the production rules (rules used to construct a sentence). A non-terminal symbol is placed on the left and a string of terminal and non-terminal symbols on the right. The non-terminal symbol on the left becomes the name of the string on the right. Sequitur builds its grammar using two principles:

1. Diagram Uniqueness, no pair of adjacent symbols that appear more than one.
2. All rules must be used more than once, and once-used rules must be ignored or eliminated[7].

2.4 Adaptive Huffman Coding

Faller and Gallager first introduced adaptive Huffman coding. Knuth contributed with improvements to his algorithm and produced an algorithm known as the FGK algorithm. Vitter introduced the latest version of Adaptive Huffman Coding. All methods found are define-word schemes determining the mapping of source messages to codewords based on the probability estimation of source messages. The code is adaptive, changing according to its optimal estimate at the time. In this case, Adaptive Huffman Code responds to locality. In a sense, the encoder studies the characteristics of the source. Decoders should learn the similarity with the encoder by updating the Huffman tree so that it is in sync with the encoder[7].

The main idea of compression and decomposition begins with an empty Huffman tree and modifies the symbol being read and processed. Compressor and decompressor modify the tree in the same way so that at any point in the process will use the same code. Although the codes may change in every step. Where the decoder reflects the encoder operation.

Initially, compression begins with an empty Huffman tree. There are no pre-existing symbols. The first symbol is inserted in the stream in a compressed form. This symbol is then added to the tree, if the symbol is found again in the stream (tree) then the frequency increases one, and will modify the tree, the tree is rechecked to see if it is still a Huffman tree (best code). If it does not rearrange trees, and cause code changes[7].

2.5 Exponential Method

In using the Exponential Comparison Method, there are several steps that must be done:

1. Develop alternative decision alternatives.
2. Determine which criteria or comparison of decisions are important to evaluate.
3. Determine the importance of each decision criterion.
4. Assess all alternatives on each criterion.
5. Calculate the score or total value of each alternative.
6. Determining the order of priority decisions is based on the score or the total value of each alternative[8].

The calculation formulation of scores for each alternative in the exponential comparison method is as follows:

$$Total\ Nilai(TN_i) = \sum_{j=1}^{m} (RK_{ij})^{TKK_j}$$

Information :
TNi: Total i-alternate value

RKij: The relative importance of- j criterion on decision choice i
TKKj: The degree of importance of the criteria of the junta decision; TKKj> 0; round
m: Number of decision criteria
n: Number of decision choices
j: 1,2,3, ... m; m: Number of criteria
i: 1,2,3, ..., n; n: Number of alternative options

III. RESULTS AND DISCUSSION

Comparative analysis of sequitur algorithm with adaptive Huffman coding algorithm is done to find out which algorithm is more effective in compressing text file. Here is the process that happens:
Text: gadjahmada

Table I. The size of the string before it is compressed

Character	Frequency	ASCII Decimal	ASCII Binari	Bit	Bit x Frequency
g	1	103	01100111	8	8
a	4	97	01100001	8	32
d	2	100	01100100	8	16
j	1	106	01101010	8	8
h	1	104	01101000	8	8
m	1	109	01101101	8	8
Total Bit					80

3.1 Sequitur Algorithm

Suppose we want to compress the string "gadjahmada" with sequitur algorithm. To find out the string size see table 1.

Table II. Sequitur Algorithm

Proses	*String*	Diagram	Nonterminal	*Rule*	*New String*	*Delete Rule*
1	gadjahmada	ad	A	A=ad	gAjahmAa	

In table 1 the character "gadjahmada" will be checked to see the pairs of characters appearing more than 1 time, the character pairs appearing more than 1 time is "ad", then the character "ad" will be processed, the character "ad" will be inserted into the diagram which generates nonterminal A, and generates a new character "gAjahmAa", check whether there are any new character pairs or not, otherwise the process is complete.

Table III. Total Bit x Frequency

Char	Freq	Dec	Biner	Bit	Bit x Freq
g	1	103	01100111	8	8
A	2	65	01000001	8	16
j	1	106	01101010	8	8
a	2	97	01100001	8	16
h	1	104	01101000	8	8
m	1	109	01101101	8	8
Total Bit x Frequency					64

3.2 Adaptive Huffman Coding

Adaptive Huffman coding algorithm is the development of the Huffman algorithm, suppose we want to compress the string "gadjahmada", then the steps - steps are as follows:

1. Character g is inserted into the tree, generate the frequency value to 1.
2. A character is inserted into the tree and creates a new tree branch, and the frequency value becomes 2.
3. The d character in inserted makes the branch bar below character a, and the frequency value becomes 3.
4. The character j is inserted and create a new branch under the character d, and the g character in the move to the left so that the tree remains a Huffman tree.
5. Character an inserted into the tree, Because previously there a character has been inputted then the character frequency increases from 1 to 2, and the tree is in the fox to remain a Huffman tree and input character h then the frequency value so 6.
6. Input the character m and create a new tree and add the frequency to 7, as shown in Figure 1. continued (g) input a character and added a value to 3 and make the frequency to 8.
7. Input character d and add the frequency value “d” to 2 and the number of frequencies to 9.
8. Input character a and add the frequency value “a” to 4 and change the number of frequencies to 10.

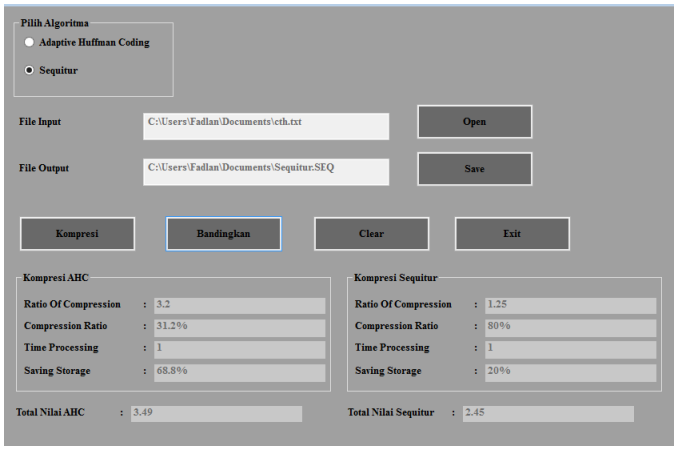

Figure 1. Huffman Tree

The compression result of the string "gadjahmada" produces **100 11 00 101 0101** which amounts to 70 bits.

3.3 Comparison With Exponential Method

To compare the two algorithms, we must define the criteria. The criteria of the two algorithms are as follows:

1. Ratio Of Compression (RC)
   a. Rc Adaptive Huffman Coding=80/25=3,2
   b. R_C Sequitur = 80/64= 1,25

2. Compression Ratio (CR)
   a. CR Adaptive Huffman Coding =70/80 x 100%=31,2%
   b. CR Sequitur = 64/80 x 100% = 80%

3. Space Savings (SS)
   a. SS Adaptive Huffman Coding = 100% - 31,2% = 68,8%
   b. SS Sequiture = 100% - 80% = 20%

Tabel IV. Criteria

Criterion	Weight	AHC	Sequitur
RC	0,25	3,2	1,2
CR	0,25	0,3	0,8
SS	0,5	0,7	0,2
Total Point ∑(N)B			1

Point Algoritma AHC = (3,2)0,25 + (0.3)0,25 + (0,7)0,25
= 2,9

Point Algoritma Sequitur = (1,2)0,25 + (0,8)0,25 + (0,2)0,25
= 2,6

Based on the comparison using the exponential comparison method of both algorithms are obtained results that sequitur algorithm more efficiently used in compressing text files than adaptive Huffman coding algorithm.

4 Experimental Results

In an experiment to compare sequitur algorithm with Adaptive Huffman Coding algorithm done with an application, wherein the application there is a button to input and text file and compress text file based on an algorithm and compare both algorithms, here's how it looks:

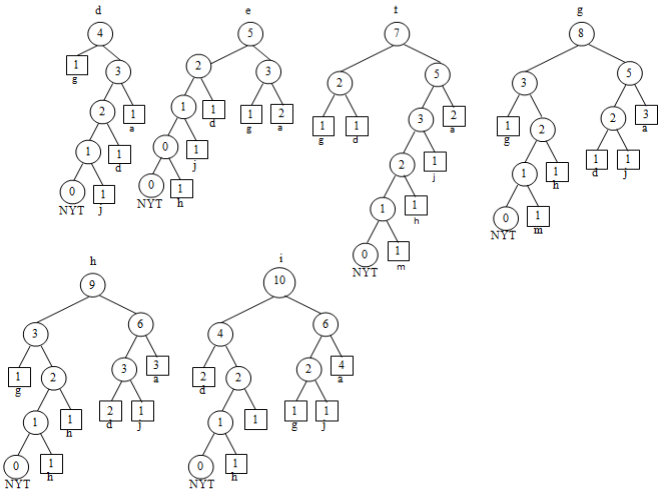

Figure 2. Comparison Process With Application

IV. CONCLUSION

Based on the exponential comparison method the sequitur algorithm is more effective in compressing the text file than the adaptive Huffman coding algorithm. The sequitur algorithm will compress if there is a loop of words in the text file that is entered and will stop processing if the non-terminal symbol used has reached non-terminal Z. Adaptive Huffman coding algorithm compresses Huffman tree shapes that will be changed from binary, in adaptive Huffman tree coding will continue to grow and will stop if all text has been entered.

V. REFERENCES

[1]. S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data Compression Methodologies for Lossless Data and Comparison between Algorithms," vol. 2, no. 2, pp. 142–147, 2013.

[2]. C. G. Nevill-Manning and I. H. Witten, "Identifying Hierarchical Structure in Sequences: A linear-time algorithm," J. Artif. Intell. Res., vol. 7, pp. 67–82, 1997.
[3]. Darma Putra, Pengolahan Citra. Yogyakarta: C.V ANDI OFFSET, 2010.
[4]. S. D. Nasution and Mesran, "Goldbach Codes Algorithm for Text Goldbach Codes Algorithm for Text Compression," vol. 4, no. December, pp. 43–46, 2016.
[5]. S. D. Nasution, G. L. Ginting, M. Syahrizal, and R. Rahim, "Data Security Using Vigenere Cipher and Goldbach Codes Algorithm," Int. J. Eng. Res. Technol., vol. 6, no. 1, pp. 360–363, 2017.
[6]. E. Jubilee, Rahasia Manajemen File. Elex Media Komputindo, 2010.
[7]. D. Salomon and G. Motta, HandBook Of Data Compression. Springer.
[8]. M. S. Prof.Dr. Ir. Marimin, Pengambilan Keputusan Kriteria Majemuk. Jakarta: Grasindo, 2004.