http://infor.seaninstitute.org/index.php/infokum/index
JURNAL INFOKUM, Volume 9, No.2, Juni 2021 ISSN : 2302-9706

DUAL TREE HUFFMAN ENCODING IMPLEMENTATION TO REDUCE LOW FREQUENCY BIT CODES

Tommy [1], Rosyidah Siregar [2], Putri Harliana [3], Shahira An-Nissa[4], Herlina Harahap [5], Yessi Fitri Annisa Lubis [6]
[1,2,3,5,6] Universitas Harapan Medan, [4] Universitas Sumatera Utara
tomshirakawa@gmail.com, rosyidah_siregar.unhar@harapan.ac.id, cimoputri@gmail.com, shahira.nissa@gmail.com, herlina_hrp@gmail.com, yessi.annisa@gmail.com

Abstract

Article Info
Received 01 June 2021
Revised 20 June 2021
Accepted 30 June 2021

Huffman encoding is a compression algorithm that uses symbol encoding using a simpler bit substitution based on the frequency of the symbol. A symbol will be represented with much fewer bits if it has a large frequency. Conversely, symbols with less frequency will be encoded with bits of greater length. The bit code for symbols with lower frequencies often has a very long size which sometimes causes the compressed file size to be larger than the original file. This study develops the implementation of two separate Huffman trees by dividing the symbol list into two lists, each of which will form a bit code in parallel. This implementation is able to minimize the length of the symbol bit code, especially in symbols with low frequencies so as to increase the compression ratio for several types of content which is become the weakness of the original Huffman.

Keywords: compression, huffman, dual tree

1. Introduction

Data compression can simply be defined as the activity of reducing the size or capacity of a file. Reducing the size or capacity of the file can be obtained by substituting the bits of the file with other bits that are simpler so that the overall file size becomes smaller. Data compression can also be defined as the art or science of representing information in a more concise form by identifying and using the existing structures in the data [1]. Currently, the development of technology for storage and throughput capacity of existing networks is not matched by the capabilities and speed of data generation, which in turn makes I/O the now anticipated bottleneck [2]. This motivation makes data compression even more important today.

Data compression consists of two approaches or methods, namely lossless compression and lossy compression, where lossless compression makes changes to the original data without losing information from the original data [3], which in other words this type of compression can be transform back to its original form. In lossy compression, data changes are made by changing the structure of the data but still maintaining the essence contained in the information in the original data [3]. This type of compression cannot be transformed back to its original form because the structure or definition of the compressed data is no longer the same as the original data. Lossless compression is usually used on text or binary which requires a reverse decompression process to get the original file [4]. While lossy compression is usually

used in multimedia files such as images, audio and video where the compression results can still represent the information content of the original file so that it does not require a decompression process [5].

Lossless compression conceptually is a process that substitutes the data from the input file, either in the form of bits, bytes, or character sequences into other representations that have a smaller length or size. Lossless compression is very dependent on the recurrence rate of occurrence of the data to be substituted. The higher the frequency of occurrence of the substituted data, the higher the achieved compression ratio. Compression methods such as Huffman Encoding, Shannon-fano encoding, Half-Byte, and several other lossless compression methods use the frequency or probability of occurrence of the data as the basis for the compression process. Several articles discussing these compression methods have shown how frequency and probability greatly affect the compression ratio of compressed files [6-9].

The Huffman Encoding is a method that substitutes data character bytes with replacement bits obtained from the Huffman tree search. The Huffman tree is formed by structuring the characters contained in the data which are considered as nodes. Each node will be combined into a new node based on its probability value. The substitution bit code for each character is then obtained from tracing the nodes in the Huffman tree. Huffman Encoding was originally intended for text-type data, although it is possible to apply it to other media [10-12]. Although Huffman Encoding has enormous potential, this method has drawbacks. Data with low frequency will use very long substitution bits [13]. In theory, the overhead caused by low-frequency data does not have a significant impact considering that bit substitution of high-frequency data can overcome the overhead. However, there are some cases where the substitution bits of high frequency data cannot cover the substitution bits of low frequency data. The length of the substitution bit of the low-frequency data is caused by the high level obtained from the Huffman tree that is formed.

Huffman tree with a very depth level will have an impact on the length of the data substitution bit. To overcome this level of depth, this study tries to divide the data into two separate groups based on the frequency which then each group will form its own Huffman tree. This Huffman tree division can reduce the depth level of the Huffman tree that is formed so that the bit representation of the symbols can be minimized.

2. Method

2.1 Huffman Encoding

Huffman Encoding is an algorithm developed by David Huffman which is used to encode the redundancies contained in the data without losing data quality [14]. In this method, the data bytes are substituted with the bits obtained from the Huffman tree. The Huffman tree is formed by compiling a series of characters from the file to be compressed, where each character will form a node of the Huffman tree. These nodes will then form new nodes based on their frequency. The process of merging nodes will continue to produce a single node that will visually form a Huffman tree structure.

The steps of the Huffman encoding algorithm can be described as follows [13] :

1. Compile a list of frequencies from symbols or characters found in the data. The probabilities obtained are then sorted from the largest to the smallest. A node is formed as a binary tree with obtained probabilities.
2. The two symbols with the lowest probability are aggregated and produce a new probability value.
3. A parent node is created from step 2, and then marks the left branch as child node 1 and the right branch as child node 0. The two child nodes are then removed from the list.
4. The process is repeated to stage 2. If only one node remains in the list, the process ends.

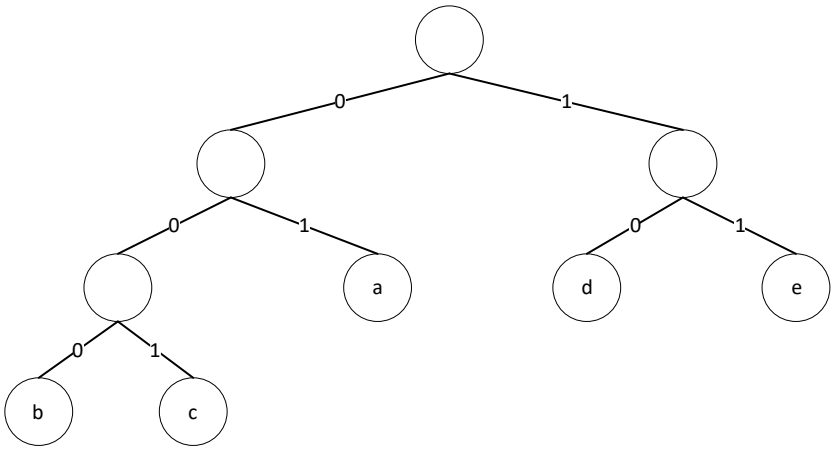


Figure 1. Huffman tree illustration

2.2 Dual Tree Huffman Encoding

Dual tree Huffman encoding is used in this research to reduce the number of nodes from the Huffman tree so that the tree depth level is not too large and the resulting bit substitution is also not too long compared to an ordinary Huffman tree. In general, the stages of this implementation are the same as the usual Huffman stages with the addition of dividing the frequency list into two separate lists based on the order of frequency. The steps of the dual tree Huffman encoding proposed in this study can be described as follows:

1. Compile a frequency list from the symbols contained in the data. Sort the symbols from the largest to the smallest by frequency.
2. Divide the frequency list into two separate lists. Division can use the middle value of the list order or use the ratio of proportions.
3. Build the Huffman tree on each list according to the original Huffman method.
4. Encoding symbols with the bit codes obtained from the Huffman tree.
5. Each encoded symbols will get additional flag bit (0 or 1) to identify which Huffman tree it belongs to.

To clarify the stages of the dual tree Huffman encoding proposed in this study, a simple example of a text "aaaabbbccde" will be described which will then be compressed using the proposed method. At the initial stage, a frequency list will be compiled from the symbols or characters contained in the data. The list of frequencies obtained can be seen in table 1. Next is to divide the list into two lists using the middle value of the symbol frequency list or using equation 1.

$$Splitter = \left\lceil \frac{N}{2} \right\rceil \quad \text{.....(1)}$$

Where N is the number of unique symbols in the list. If each member of the list is represented by (x,y) where x represents the symbol and y represents the frequency and the list is denoted as an array, then the members of the two separate lists can be obtained using equations 2 and 3.

$List_1 = \{ (x,y) \mid (x,y) \in List[0, 1, \dots, Splitter] \}$ .....(2)

$List_2 = \{ (x,y) \mid (x,y) \in List[Splitter + 1, \dots, N] \}$ .....(3)

Table 1. Symbol Frequency List

Symbol	a	b	C	d	e
Frequency	4	3	2	1	1

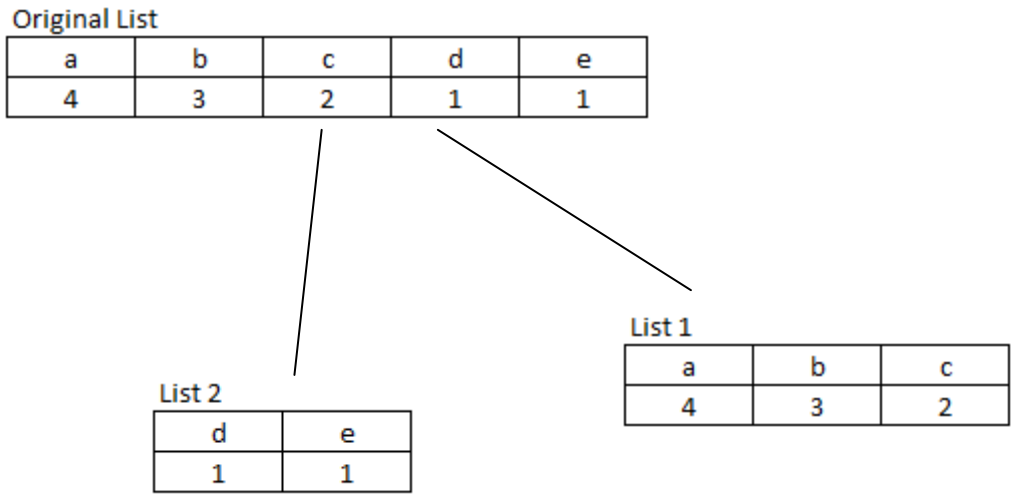
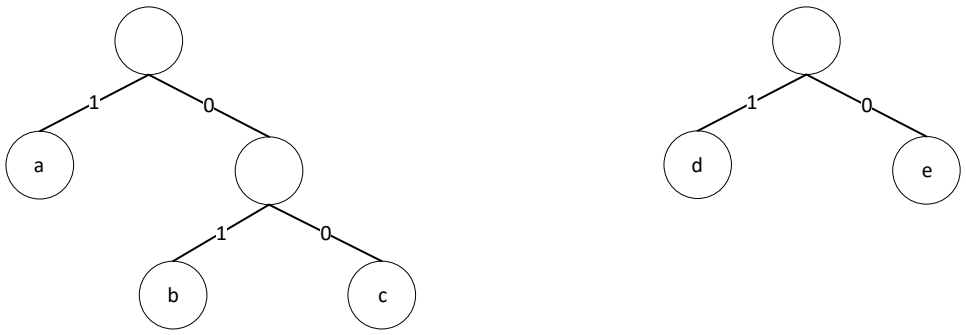


Figure 2. List splitting illustration

The next step is to form a Huffman tree by using the steps of the usual Huffman encoding for each list which can be seen in Figure 3.



(a) (b)

Figure 2. Huffman tree of List 1(a) and List 2(b)

Based on the Huffman tree formed, the lookup table for the encoding process for the symbols contained in the data can be obtained from table 2 and 3.

Table 2. List 1 Lookup Table

SUBSTITUTION BIT	SYMBOL
11	a
101	b
100	c

Table 3. List 2 Lookup Table

SUBSTITUTION BIT	SYMBOL
11	d
10	e

Each substitution bit is added a bit 1 as a prefix to simplify the decoding process. The final stage is encoding each symbol contained in the data by adding a flag bit for each encoded symbol.

The symbol "a" will be encoded with bit 11, with the addition of the flag bit "0" which indicates the substitution bit is obtained from the lookup table list 1 so that the encoding bit is 011.

The symbol "b" will be encoded with bit 101, with the addition of the flag bit "0" which indicates the substitution bit is obtained from the lookup table list 1 so that the encoding bit is 0101.

The symbol "c" will be encoded with bit 100, with the addition of the flag bit "0" which indicates the substitution bit is obtained from the lookup table list 1 so that the encoding bit is 0100.
The symbol "d" will be encoded with bit 11, with the addition of the flag bit "1" which indicates the substitution bit obtained from the lookup table list 2 so that the encoding bit is 111.

The symbol "e" will be encoded with bit 10, with the addition of the flag bit "1" which indicates the substitution bit obtained from the lookup table list 2 so that the encoding bit is 110.

Based on the coding above, the compressed bits obtained are:

01101101101101010101010000100111110 = 38 Bits.

3. Results and Discussion

The dual tree Huffman encoding developed in this research will then be tested to compress several test files that have different size and types. The tested files are selected so the symbols variation and distribution are significantly different from on files to another. The test results can be seen in table 4.

Table 4. Experiment Results

File	Original Size	Compressed Size	Compression Ratio
Alice.txt	152,089	107,365	1.4166
Database.php	4,648	3,815	1.2183
jquery-3.5.1.min.js	89,476	71,042	1.2595
ScienceDirect_citations_1628573734693.txt	9,082	7,420	1.2240
.gtignore	503	497	1.0121
DB_CHAT.SQL	7,463	6,240	1.1960

The test results of compression using Huffman dual tree as shown in table 4 show that this method is able to provide smaller sizes in all test files. With an average compression ratio of 1.2211. The next step is to compare the method developed in this study with the original Huffman encoding as can be seen in table 5.

Table 5. Comparison Results

File	Original HUffman	Dual Tree Huffman	Efficiency
Alice.txt	88,880	107,365	-20.80
Database.php	4,300	3,815	11.28
jquery-3.5.1.min.js	61,080	71,042	-16.31
ScienceDirect_citations_1628573734693.txt	7,027	7,420	-5.59
.gtignore	503	497	1.19
DB_CHAT.SQL	6,468	6,240	3.53

The comparison to the original Huffman as shown in table 5 shows the varying efficiency results. In files with content dominated by alphabetic characters, the original Huffman has better performance than the dual tree Huffman. However, in files with content that has multiple and distributed symbols, the Huffman dual tree proposed in this study has better performance.

4. Conclusions

Dual Tree Huffman is an implementation of two separate Huffman trees on data compression using Huffman encoding. The implementation of dual tree in Huffman encoding is intended to overcome the length of bit substitution in low frequency data. Based on the results of the tests that have been carried out, the implementation of two separate trees shows a compression ratio that is not much different from the original Huffman. However, for files that are dominated by alphabets, the original Huffman still has better performance because the dual tree will have the overhead of storing two separate lookup tables and the additional flag bits used. Even so, the dual tree Huffman has a better performance than the original Huffman, which can be seen from the increase in compression results compared to the original Huffman. In addition, dual tree Huffman has better adaptability to content than the original Huffman, where all the files tested are able to be compressed with an average compression ratio of 1.2211.

Reference

[1] K. Sayood, "Introduction to Data Compression," Morgan Kaufmann, pp. 1-80, 2006.
[2] K. Duwe, J. L¨uttgau, G. Mania, J. Squar, A. Fuchs, M. Kuhn, E. Betke and T. Ludwig, "State of the Art and Future Trends in Data Reduction for High-Performance Computing," Supercomputing Frontiers and Innovations, vol. 7, no. 1, pp. 4-36, 2020.
[3] L. A. Fitriya, T. W. Purboyo and A. L. Prasasti, "A review of data compression techniques,"

International Journal of Applied Engineering Research, vol. 12, no. 19, pp. 8956-8963, 2017.
[4] H. D. Kotha, M. Tummanapally and V. K. Upadhyay, "Review on Lossless Compression Techniques," J. Phys. Conf. Ser., vol. 1228, no. 1, pp. 1-6, 2019.
[5] W. E. Pangesti, G. Widagdo, D. Riana and S. Hadianti, "Implementasi Kompresi Citra Digital Dengan Membandingkan Metode Lossy Dan Lossless Compression Menggunakan Matlab," Jurnal KHATULISTIWA INFORMATIKA, vol. 8, no. 1, pp. 53-58, 2020.
[6] Tommy, R. Siregar, I. Lubis, A. M. Elhanafi, A. M. H. and M. Harahap, "A Simple Compression Scheme Based on ASCII Value Differencing," in IOP Conf. Series: Journal of Physics: Conf. Series, Vol. 1007, Medan, 2018.
[7] Pujianto, Mujito, B. H. Prasetyo and D. Prabowo, "Perbandingan Metode Huffman dan Run Length Encoding Pada Kompresi Dokumen," InfoTekJar : Jurnal Nasional Informatika dan Teknologi Jaringan, vol. 5, no. 1, pp. 216-223, 2020.
[8] C. Lamorahan, B. Pinontoan and N. Nainggolan, "Data Compression Using Shannon-Fano Algorithm," Distributed Computing, vol. 2, pp. 10-17, 2013.
[9] A. Siregar, R. Siregar, D. Handoko, Tommy and A. M. Elhanafi, "Analisis Perbandingan Kompresi Data Teks Dengan Menggunakan Algoritma Diferensiasi Ascii Dan Half-Byte," in SNASTIKOM 2020, Medan, 2020.
[10] Sunardi, S. Alam and S. R. D. R., "Implementasi Aplikasi Kompresi Data dengan Metode Huffman Code," PROSIDING SEMINAR ILMIAH SISTEM INFORMASI DAN TEKNOLOGI INFORMASI, vol. 8, no. 2, pp. 41-26, 2019.
[11] S. Ashida, H. Kakemizu, M. Nagahara and Y. Yamamoto, "Sampled-data audio signal compression with Huffman coding," in SICE 2004 Annual Conference, Sapporo, 2004.
[12] T. Hidayat and M. H. Zakaria, "Comparison of Lossless Compression Schemes for WAV Audio Data 16-Bit Between Huffman and Coding Arithmetic," International Journal of Simulation: Systems, vol. 19, no. 6, pp. 36.1-36.7, 2019.
[13] Erdal and A. Ergüzen, "An Efficient Encoding Algorithm Using Local Path on Huffman Encoding Algorithm for Compression," Applied Science, vol. 9, no. 4, pp. 1-19, 2019.
[14] A. H. David, "A Method for the Construction of Minimum-Redundancy Codes," Proc. IRE, vol. 40, no. 9, p. 1098–1101, 1952.