British Journal of Applied Science & Technology
16(2): 1-10, 2016, Article no.BJAST.25712
ISSN: 2231-0843, NLM ID: 101664541

SCIENCEDOMAIN *international*
www.sciencedomain.org

Plain Text Encoding/Decoding Technique Using a Combination of Huffman and Run-Length Algorithms

Ahmed Ibrahim[1*]

[1]Department of Computer Sciences, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Kingdom of Saudi Arabia.

Author's contribution

This work was carried out completely by the author himself. All aspects of this work, namely, study design, statistical analysis, literature search and review, manuscript writing, etc. were done by author AI.

Article Information

DOI: 10.9734/BJAST/2016/25712
Editor(s):
(1) Samir Kumar Bandyopadhyay, Department of Computer Science and Engineering, University of Calcutta, India.
Reviewers:
(1) Ferda Ernawan, Universiti Malaysia Pahang, Malaysia.
(2) Anonymous, University of Malakand, Pakistan.
(3) Usha Mehta, Institute of Technology, Nirma University, India.
Complete Peer review History: http://sciencedomain.org/review-history/14609

Original Research Article

Received 16th March 2016
Accepted 3rd May 2016
Published 13th May 2016

ABSTRACT

This work is devoted to study the effect of applying a hybrid encoding/decoding algorithm to textual data. The sole purpose is to analyze the effect on the size as well as the complexity of the output encoded data. The proposed combination is that of Huffman and Run-Length algorithms. This study focuses on the sequence of applying the two algorithms to see if it has an effect on the output data or not, and the impact of input data format on the result. Results show that the data format and the sequence in which the algorithms are applied actually affect the output. Moreover, it is shown why these two algorithms were chosen and each of them contribute to the overall result.

Keywords: Encoding techniques; huffman and run-length algorithms; compression techniques; encoding; decoding.

**Corresponding author: E-mail: azibrahim@pnu.edu.sa;*

1. INTRODUCTION AND RELATED WORK

In recent days dealing with data no matter of their forms like plain text or binary data is a concrete problem. Data can be stored in local devices or resources or transmitted across networks. So, what if transmission path is unsafe or the storage devices can be hacked? The common answer is that data will be insecure. From this point of view the trouble can be figured. The major problems are the safety and size of data. In fact, the problem could not be solved unless having an algorithm or technique that could facilitate the process of maintaining data by scaling down the size and representing the data in some other secure form without losing the original data. So using cryptography will help solving the mentioned problem and accomplishing the main goal. The domain of this paper is to encode/decode the input data using Huffman and Run-Length techniques.

Encoding is a field of study which deals with the secret transmission "transforms data into another format" of messages/data between two end-users. It uses schemes that are publicly available so it can be reversed easily and it is for maintaining data usability [1].

In fact, one of the most significant problems facing the digital data in general is the size of data and its protection. All types of data should be stored, sent, and received in a secure form with little size as much as it can be. Many algorithms are presented to solve this problem.

To be more specific, converting plain text to encrypted or cipher text using a unique encryption key is called encryption [2]. Which means encryption/decryption of data is only possible with the corresponding key. The encryption/decryption issue will not be discussed during this job.

One of the simplest techniques for lossless data compression is Run-length encoding (RLE). It reduces strings by recurring characters to a single character. That means the run of characters "plain text source" is replaced with the number of the same characters and a single character which is the RLE principle. The RLE performance is based on a sequence of identical values of the input data [3-4].

Some other technique for lossless data compression is Huffman encoding. The Huffman algorithm is the one of earliest data-compression and encryption algorithms. It developed by David A. Huffman in 1979. The Huffman algorithm yields a variable and fixed-length binary code depending on the probabilities of each symbol of a source alphabet. The proof [5] that Huffman code can be surprisingly difficult to cryptanalyze motivated us make a decision of using Huffman encoding as a second main step of changing the form of original plain text and as a private-key "closed key" of encoding/decoding. The final step is to cipher the output data obtained by applying Huffman algorithm using RLE algorithm. As well, the RLE algorithm will be applied to reduce the size obtained data.

Our technique process can be summarized as follows: the original plain text will be encoded as ASCII code in binary form, then represented as a binary code using Huffman encoding. Finally, the RLE will check the repeating string of characters to produce non-repeated data with new cipher form. That will be done because of the properties of Huffman and RLE algorithms that have been discussed.

The work of Rezaul (et. al) [6] shows an efficient decoding technique for Huffman codes and presents a novel data structure for Huffman coding in which in addition to sending symbols in order of their appearance in the Huffman tree one needs to send codes of all circular leaf nodes, the number of which is always bounded above by half the number of symbols.

A new devised algorithm to hide text in an image using Huffman encoding and 2D Wavelet transform is presented by Saddaf Rubab (et. Al) [2]. The paper [2] discusses the Huffman Algorithm as a two-part process. The first part is an encoding process. The process starts by set of symbols/letters and their respective frequencies in ascending or descending order. Each symbol/letter with its frequency is a leaf node at the start. Selecting two symbols with smallest frequencies is the next step. The process will continue by adding their frequencies and assign it to parent node, until only one node remains which is called the root node. The first process will finish by assigning binary 0's and 1's to all the nodes. The second part is the decoding process which starts by creating a Huffman Table, which is used to decode symbols/letters in original data using the generated codes.

In [7] a new approach of run length encoding (RLE) is proposed to compress discrete cosine transform (DCT) coefficients of time domain ECG

signals. Energy compaction property of DCT facilitates the process of length encoding by accumulating the correlative coefficients into separate segments. Thus the high probability of redundancies in consecutive coefficients facilitates the use of RLE. To increase the CR, two stages of RLE are performed on the quantized DCT coefficients. Then a binary equivalent of RLE values are obtained by applying Huffman coding. The conclusion shows that the performance of a compression scheme can vary with the characteristic of the input data set. So, for the same bit rate, the RLE based compression scheme achieves different distortion indices for different databases [7].

The paper [8] proposes a new algorithm for data compression, called j-bit encoding (JBE). An experiment by using 5 types of files with 50 different sizes for each type was conducted, 5 combination algorithms has been tested and compared. The proposed algorithm gives better compression ratio when inserted between move to front transform (MTF) and arithmetic coding (ARI).

Compression Using Huffman encoding article [9] discussed the various techniques available for Lossless compression. The analysis of Huffman algorithm and comparison with various Lossless compression techniques as Arithmetic, LZW and Run Length Encoding has been stated. Comparison of common algorithms with Huffman Encoding has been performed from different points of view as the basis of their use in different applications and their advantages and disadvantages. The main conclusion is that the Huffman algorithm is used in JPEG compression. It produces optimal and compact code, but relatively slow. Huffman algorithm is based on a statistical model that adds to overhead. As well, the researcher concluded that arithmetic encoding is really effective for more frequently occurring sequences of pixels with fewer bits and reduces the file size dramatically. RLE is simple to implement and fast to execute. LZW algorithm is more adept to practice for TIFF, GIF and Textual Files.

Comparison of lossless data compression algorithms for text data article [10] tested six lossless data compression algorithms and compares their performance. A set of selected algorithms such as Huffman Encoding, Shannon Fano Algorithm, RLE, LZW, and Adaptive Huffman Algorithm are examined and followed out to evaluate the performance in compressing text data. Although they are tested for ten text files with different file sizes and different contents. The compression behavior depends on the category of the compression algorithm: lossy or lossless. Compression ratio, factor and time, and saving percentage are used to evaluate the performance of discussing lossless algorithms. It is important to mention that the performance depends on the type and the structure of the input source file. The main article's conclusion is that the Shannon Fano algorithm can be viewed as the most efficient algorithm comparing to other discussed algorithms. The reset value of this algorithm exhibit in an acceptable scope and it better results for the files with large size. Besides the article depicts the all discussed algorithms work well, except Run length encoding algorithm. Finally, the LZW algorithm does not work well for large file size.

The procedure of converting images into text format can be done. Lossless Huffman Encoding Technique for Image Compression and Reconstruction Using Binary Trees discussed the way of saving a black and white image, which its pixels of different shades of grey. Each pixel has a number value corresponding to the brightness or darkness of the shade. That stands for the pixel colored with black is 0 and white is 255, and all the numbers in between the black and white colors are shades of grey. So, each pixel is coded as some integer from 0 to 255 [11].

A digital image can be coded with respect to a model using Huffman encoding. It's well recognized that the Huffman's algorithm generates minimum redundancy codes compared to other algorithms. The presented scientific article presents a compression and decompression technique based on Huffman encoding and decoding for scan testing to reduce test data volume, test application time. The main conclusion is that the image compression method is well suited for grey scale (black and white) bitmap images [12].

Breaking a Huffman Code article examines the problem of deciphering a file that has been Huffman coded. The authors find that a Huffman code can be surprisingly difficult to cryptanalyze. The article introduces the analysis of the situation for a three-symbol source alphabet in details [4].

As a matter of fact, an image or audio file can be converted to a form of plain text to use as an input source. For instance, it is possible to convert any image into editable text file with software such as JiNa OCR Image to text.

Furthermore an intermediate programmer can produce an application that will read each pixel in an image. Then save the information about the pixels that been read in text form.

The major contribution of this paper is developing an algorithm that will encode/decode digital data and reduce the input file size.

The sections of this paper are formed as follows: Part I shows the introduction, problem, goal, and related work. Part II represents suggested technique that will be used in the experiment. Part III discusses the experiment on some case studies. Part IV represents the conclusions and future work.

2. METHODOLOGY

2.1 Technique

Foremost of all, the algorithm can be described as the set of well-defined instructions that allows a computer to execute a specific task in a specific order [13] "well-defined procedure". The technique is a procedure to complete a specific task.

Fig. 1 shows the process sequence that manipulates the data presented as a plain text. Fig. 2 depicts the steps of the proposed technique.

The application that allows to implement the suggested Encoding/Decoding technique was developed using NetBeans IDE. The developed application allows the user to pick a text file and represent it in Huffman and/or RLE form. All the obtained data will be saved in Huffman file and/or RLE text file. The information about the input/output text file as title, size, date, time, and used algorithm will be saved in the database to be used later on for comparison.

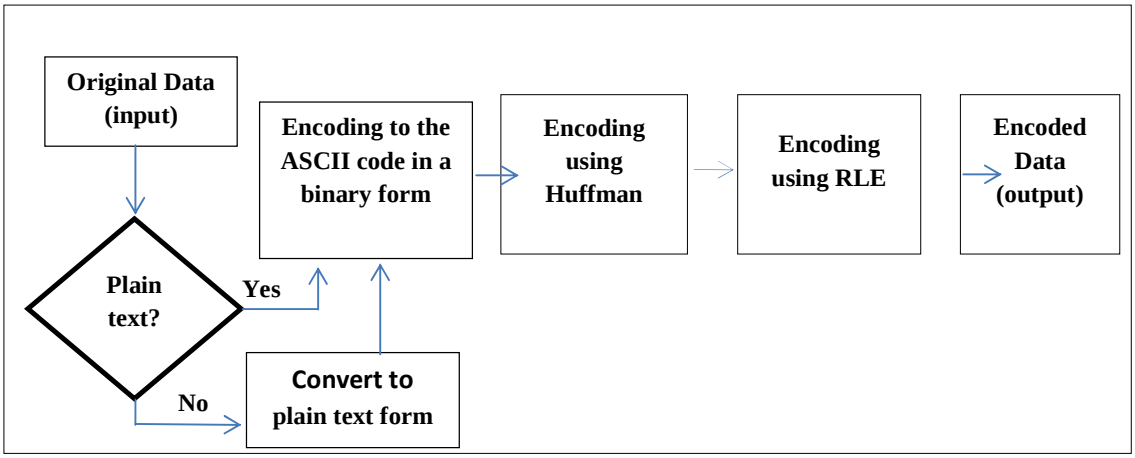

Fig. 1. Process of encoding using huffman and rle techniques

Step 1: Start.
Step 2: Recognize the type of original data "source".
Step 3: If the type of original data is plain text, go to step 5.
Step 4: If the type of original data is non-plain text (e.g. Image), then represent the original data in plain text form.
Step 5: Encode the original plain text to the ASCII code in a binary form.
Step 6: Represent the original plain text in Huffman form.
Step 7: Represent the data that's been encoded using the Huffman (Step 6) in RLE form.
Step 8: Compare the size and form of the original data with the size and form of output data in step 7
Step 8: Calculate the saving percentage %.
Step 9: Stop.

Fig. 2. Encoding/Decoding technique – steps

2.2 Experiments

This section depicts the comparison Table of experiments that were done on different original data. The Tables (Tables 1 and 2) show the original plain text, its size, and output data "in final step". It is quite apparent that the data and its size change throughout the executed process.

The formula used to calculate the shrinkage of the source data as a percentage is [10]:

$$\text{Saving percentage} = \frac{\text{Size before compression} - \text{Size after compression}}{\text{Size before compression}} \% \quad (1)$$

3. RESULTS AND DISCUSSION

As seen in Table 1 the original data is changed to a different form after implementing Huffman and RLE algorithms respectively. Actually, implementing RLE directly on original data will change the input data in RLE form, but using Huffman coding followed by RLE to encode and find the repeated characters "recurring characters to a single character", will change the input "original" data to a form that is characterized with more compression factor.

But what will happen if the suggested sequence process is changed, using the same input as in Table 1; i.e. implementing RLE then Huffman algorithms on the same original data?

As examined in Table 2, the original data is changed to a different form after implementing RLE and Huffman algorithms respectively. It is obvious, that the size of the output data is greater than that of the output data obtained previously in Table 1. From the point of view of encoding/decoding and the output data size, this proves that the suggested sequence process "Huffman-RLE" gives better results. Likewise, Table 1 shows that better outcomes will be achieved if the type of the input data is ASCII code in binary form.

In fact, examining the idea of using the Huffman coding before the RLE algorithm could be explained as follows: The Huffman binary code of a character will not always take the same form of a binary code, it depends on its frequency and position in a word. For example, the Huffman binary code of "o" in the word "room" is 1 and the Huffman binary code of "o" in "room and moon" is 11, whereas the Huffman binary code of "o" in "domain+" is 00. Therefore, the outcomes of Huffman coding is not a static value. In addition the Huffman coded text differs from ASCII encoded text. For example the Huffman coded text of "room" is 110010, and the ASCII encoded text is 01110010011011110110111101101101. So, these two mentioned reasons give the suggested sequence process more validity.

As a consequence of what's been discussed above, the character will have a different binary form using the Huffman algorithm.

3.1 Results

The two Tables presented above show some results that could be summarized as follows:

1. Encoding/Decoding is passed because of changing the original data view as seen in output data view field. Which means the main aim is passed.

2. Encoding the input using Huffman encoding before RLE is implemented and makes the process more reliable.

3. After accomplishing the process:

   3.1 If the input data is a plain text, the size of data grows exponentially. In first experiment, the size of the original data which is in text format is 43 bytes and the size of output data is 138 bytes. This means, the size of original data has increased 341%.

   3.2 If the input data is an ASCII code in binary form, the size of data grows slightly. The second experiment in Table 1 shows that the size of the original data which is in binary format was increased 8%.

4. The reason behind increasing the size of output data is that the Huffman data has no long sequences of frequented 0's and 1's in Table 1 and most of RLE frequencies is 1 in Table 2.

5. Implementing the RLE before Huffman coding of the original data gives bad results as seen in Table 2.

Table 1. Encoding and data size changing using huffman then RLE algorithms respectively

Source data type	Source data view (plain text)	Encoding using Huffman algorithms	Output data view (Encoding) "Implementing Huffman and RLE algorithms respectively"	Encoding/ Decoding (pass or fail)	Original size bytes	Output size bytes	Saving percentage %	Output size (increased/decreased)
The string "Alphabet"	Princess Nourah bint Abdulrahman University	10010110000001111111010 0101010101110001100011111 0011110010110010111011101 0000011111101110010101110 1010001100111101101100010 1100101000010110111111011 0111011000111110010011000 10100001111001011	11201110216081101120111 0111011101110313021304 1 20412011102120111031102 11011506110312011101110 21101110113021204110211 02120111021201110114011 10211051102110311021305 12011202120111011404120 111021	Pass	43	190	-341	↗
ASCII code in binary form Of "Princess Nourah bint Abdulrahman University"	01010000011100 10011010010110 11100110001101 10010101110011 01110011001000 00010011100110 11110111010101 11001001100001 01101000001000 00011000100110 10010110111001 11010000100000 01000001011000 10011001000111	1010111110001101100101 10 1001000110011100100110 10 1000110010001100110111 11 1011000110010000100010 10 1000110110011110100101 11 1101111110011101100101 10 1001000110001011110111 11 1011111010011101100110 11 1000101010010011100011 01 1001111010010111100100 10 1001111010010001110111 11 1010101010010001100101 10 1000100110011010100011 01 1000110010010110100010 11	11101110513021102120111 02110112011302120312011 20211011101130212011302 12021106110213021201140 11301110111011302110212 04110112011105110612031 10212011102110112011302 13011104110611051101120 31102120211031301110111 01120112031302110212041 10112011104120112011101 12041101120113031106110 11101110111011201130212 01110211011301120212021	Pass	343	372	-8	↗

Source data type	Source data view (plain text)	Encoding using Huffman algorithms	Output data view (Encoding) "Implementing Huffman and RLE algorithms respectively"	Encoding/ Decoding (pass or fail)	Original size bytes	Output size bytes	Saving percentage %	Output size (increased/decreased)
	01010110110001 11001001100001 01101000011011 01011000010110 11100010000001 01010101101110 01101001011101 10011001010111 00100111001101 10100101110100 01111001	10000110	10111011302110213021201 12011102110113011103140 2110					

↗ - increase in data size and ↘ - decrease in data size

Table 2. Encoding and data size changing using RLE and Huffman algorithms respectively

Source data type	Source data view (plain text)	Encoding using RLE algorithms	Output data view (Encoding) "Implementing RLE and Huffman algorithms respectively"	Encoding/Decoding (pass or fail)	Original size bytes	Output size bytes	Saving Percentage %	Output size (increased/decreased)
The string "Alphabet"	Princess Nourah bint Abdulrahman University	1P1r1i1n1c1e2s1 1N1o1u1r1a1h1 1b1i1n1t1 1A1b1d1u1l1r1a1h1m1a1n1 1U1n1i1v1e1r1s1i1t1y	01111111011101010111100100001 10011011011111100011111001 01101001000111111010010011 10101101001010101010110111001 01111001000010011010110110 0 10011100101100010101000110 1 10001110101101001010101101 1 01011010010000101101100000 1 00001111001110000011011101 1 10101111100111100100110100 101	Pass	43	272	-532	↗
ASCII code in binary form Of "Princess Nourah bint Abdulrahman University"	010100000111001 001101001011011 100110001101100 101011100110111 001100100000010 011100110111101 110101011100100 110000101101000 001000000110001 001101001011011 100111010000100 000010000010110 001001100100011	10111011503120112021101120 1 11021103120213021102120111 0 11103120211031202120116011 2 03120211041103110111011103 1 20112021401110211011501160 2 13011202110112011102310 320 31101140116011501110213011 2 02120113031101110111021102 1 30312011202140111021101140 2 11021101110214011102110313 0 11601110111011101110231031 2 02110112011103110212021201 1 10111031201120312021102110 1	01000010001100101011010111 1 00011110111001000111100001 0 11100101101011110111011011 0 11100101110111100001000010 1 10101111011100101101011110 1 11011110001100111000111101 1 01011101110010110000010110 10010000100001011010111100 0 11110111011000100001011100 1 00011001010001100111011101 1 01100011101110010001111000 01011100101101011110110100 1 00011000100011001110001100 1	Pass	344	714	-107	↗

Source data type	Source data view (plain text)	Encoding using RLE algorithms	Output data view (Encoding) "Implementing RLE and Huffman algorithms respectively"	Encoding/Decoding (pass or fail)	Original size bytes	Output size bytes	Saving Percentage %	Output size (increased/decreased)
	101010110110001 110010011000010 110100001101101 011000010110111 000100000010101 010110111001101 001011101100110 010101110010011 100110110100101 11010001111001	1201110311011304120 11	0100001011101101100011110 11 1011110001101101101001000 01 0000101110010111011011011 01 0111100011110111011000100 00 1011100100011000101110010 11 1001000010111011000100000 101 1100101101 …					

↗ - increase in data size and ↘ - decrease in data size

4. CONCLUSIONS AND FUTURE WORK

The proposed technique can be utilized to encode and decode data and to increase security level during data communication. The encoded data is surprisingly difficult to cryptanalyze. The outcome is totally different compared with the original data. The decoding cannot be attained without using the Huffman-RLE algorithm sequence. So, as a result, Encoding/decoding is done and passed if the input is an ASCII code in binary form, and the proposed technique increases the size of data. This technique may be improved using the adaptive Huffman encoding technique or by optimizing the Huffman code, which will happen when the probability of each symbol is a negative power of two [14].

COMPETING INTERESTS

Author has declared that no competing interests exist.

REFERENCES

1. Nigam Sangwan. Text encryption with huffman compression. International Journal of Computer Applications (0975–8887). 2012;54(6):29-32.
2. Saddaf Rubab, Younus M. Improved image steganography technique for colored images using huffman encoding with symlet wavelets. IJCSI International Journal of Computer Science Issues. ISSN (Online): 1694-0814. 2012;9(2-1):194-196.
3. Nagarajan A, Alagarsamy K. An enhanced approach in run length enencoding scheme. International Journal of Engineering Trends and Technology. ISSN: 2231-5381. 2011;2(1):43-47.
4. Sarika S, Srilali S. Improved run length enencoding scheme for efficient compression data rate. Int. Journal of Engineering Research and Applications. ISSN: 2248-9622. 2013;3(6):2017-2020.
5. David W. Gillman, Mojdeh Mohtashemi, Ronald L. Rivest. On breaking a huffman code. IEEE - Transactions on Information Theory. 1996;42(3):972-976.
6. Rezaul Alam Chowdhury, Kaykobad M, Irwin King. An efficient decoding technique for Huffman codes. Information Processing Letters. Elsevier Science. 2002;81;305–308.
7. Akhter S, Haque MA. ECG comptression using run length encoding. IEEE - Signal Processing Conference, 18th European. ISSN: 2219-5491. 2010;1645-1649.
8. Made Agus Dwi Suarjaya I. A new algorithm for data compression optimization. (IJACSA) International Journal of Advanced Computer Science and Applications. 2012;3(8):14-17.
9. Mamta Sharma. Compression using huffman encoding. IJCSNS International Journal of Computer Science and Network Security. 2010;10(5):133-141.
10. Kodituwakku SR, Amarasinghe US. Comparison of lossless data compression algorithms for text data. Indian Journal of Computer Science and Engineering. ISSN: 0976-5166. 2001;1(4)416-425.
11. Mridul K. Mathur, Seema Loonker, Dheeraj Saxena. Lossless huffman coding technique for image compression and reconstruction using binary trees. ISSN:2229-6093. 2012;3(1):76-79.
12. Pujar Jagadish H, Kadlaskar, Lohit M. A new lossless method of image compression and decompression using Huffman encoding technique. Journal of Theoretical and Applied Information Technology. 2010;15(1/2):18-23.
13. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. Third Printing, MIT Press; 2002.
14. Al-hmeary BA. Role of run length encoding on increasing huffman effect in text compression. Journal of Kerbala University. ISSN: 18130410. 2008;6(2): 199-204.

Peer-review history:
The peer review history for this paper can be accessed here:
http://sciencedomain.org/review-history/14609