Journal of NAMP 67, 1 (2024) 9-20

The Nigerian Association of Mathematical Physics

Journal homepage: https://nampjournals.org.ng

Comparative Analysis and Performance Evaluation of Huffman and Shannon Fano Data Compression Algorithms using the Reduction of Storage size of a given Data String.

[1]Princewill Aigbe, [2]Emmanuel Nwelih

Department of Computer Science, College of Computing Western Delta University,Oghara, Nigeria.

Department of Computer Science, University of Benin, Benin City, Nigeria

https://doi.org/10.60787/jnamp-v67i1-337

ARTICLE INFO

Article history:
Received xxxxx
Revised xxxxx
Accepted xxxxx
Available online xxxxx

Keywords:
Deduplication,
Algorithm,
Compression,
Data encoding,
Average code length.

ABSTRACT

Even with bigger storage disks, it is very necessary to explore opportunities to maximize the potential capacity of the required storage disks. The act of transforming data such that it uses less memory space is known as data compression. Huffman and Shannon Fano encoding algorithms can compress different forms of data such as images, text, audio, and video. This paper centres on a comparative evaluation of the stated data encoding algorithms in terms of code word generation operations, average code length per symbol, CRP, compression factor, and time complexity analysis with a sample data string. The evaluation culminated in the computation and a pictorial representation of the running time of the two data encoding algorithms. The Huffman encoding algorithm takes a shorter time than Shannon Fano encoding algorithm. However, the two compressions have similar average code length per symbol and operate within the same compression ratio and factor performance

1. Introduction

Even as disk capacities continue to increase, data storage vendors are constantly seeking methods by which their customers can cram ever-expanding mountains of data into storage devices. Even with bigger storage disks, it is very necessary to explore opportunities to maximize the potential capacity of the required storage disks. One way in which a massive amount of data can be reduced in size before storage is data deduplication. Data deduplication is a commonality factoring data reduction technique that involves redundant copies of data being removed from a system.

*Corresponding author: Princewill Aigbe
E-mail address: princewill.aigbe@wdu.edu.ng

It is applied in both data backup and network data schemes, and enables the storage of a unique model of data within a database and broader information system.

Data deduplication works by examining and then comparing incoming data pieces with already stored data. If any specific data is already present, the applied data deduplication technique remove the new data and replace it with a reference to the data already in place. Storage vendors rely on data deduplication technique to make better use of storage space, and the goal here is increased storage efficiency (Pujar and Kadlasker, 2017). With a proper deduplication technique, businesses can effectively store more data than their overall storage capacity might suggest. For example, with the application of a proper deduplication technique, a business can get a 4:1 reduction benefit, meaning it would be possible to store 60 TB of data on a 15 TB data storage device (Muhammad, 2018).
Different techniques can be used to carried out data deduplication. The various techniques are shown in Figure 1.1.

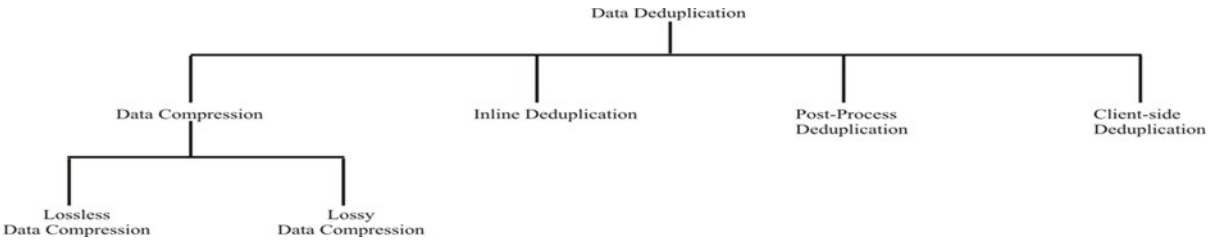

Figure 1: Data deduplication techniques

Text or string compression can be done by eliminating unnecessary characters, embedding a repeat character to specify repeated characters, and substituting a smaller bit string for a commonly occurring bit string. Data compression can cut a text file to 50%, or to a percentage still smaller than its original size (Ahmad and Mahmoud, 2019).
A particular data compression operation is either lossy or lossless mechanism. In lossless compression mechanism, bits reduction is done by identifying and eliminating statistical redundancy, and no data or information is lost in the process. Lossless compression is possible because most real-world data exhibits statistical redundancy. For example, an image may have areas of colour that do not change over several pixels. In this mechanism, compressed data can be reversed. A typical example of lossless compression is Huffman data encoding algorithm. Lossy compression mechanism reduces bits by removing unnecessary or less important information resulting in some form of data loss. A data compression operation can also be carried by a device (Tanvi *et al.*, 2019). A device that performs data compression is referred to as an encoder, and one that performs the reversal of the process (decompression) as a decoder. This means that a given data compression algorithm has two forms (encoding and decoding).

Massive data files are transferred from one computer to another, and even between computer networks with ease and minimal bandwidth every second. It is also known that a given storage device with a smaller memory space can accommodate a data file with possible much larger memory size. This is possible because the large data files can have their memory sizes reduced to the barest minimum before transmission. Therefore, the research study is motivated to carry out a vivid study of the selected data compression algorithms in order to understand the procedures or processes involve in the reduction of a specified data file size.
Given a data string $\{s_i\}$ with frequencies $f(s_i)$, and that the research interest is to find a set of binary code words $C = \{c(s_1), c(s_2), \ldots c(s_n)\}$ such that the average number of bits used to represent the data string $\{s\}$ is minimized. Mathematically, the data string reduction operation is represented by the equation in expression 1:

$$B(C) = \sum_{i=1}^{n} f(si) \mid c(si) \mid$$

Where $B(C)$ is the reduced set binary codewords, that is, bits representation of the data string $_i$s Therefore, paper centres on the study of given data encoding compression algorithm, that is, Huffman encoding or compression algorithm that generates binary bits to accommodate or store the data string s $_i$ in a reduced memory storage space.

2. Methodology

The steps in the two selected algorithms, that is, Huffman and Shannon Fano data encoding algorithms, are carefully analysed for the purpose of showing the operations of these algorithms at each stage. Tree data structure is used to demonstrate pictorially the different stages of data compression process with a sample data string. Tables are used to show the code word and code length generated per symbol or character of the sample data string in the compression process. These tables help determine the average code length representation of the sample data string by the two selected data encoding algorithms, and this is done using a simple established mathematical expression. The average code length per symbol of the sample data string gives an insight of the compression ratio performance (CRP) of the two selected data encoding algorithms. The tree data structure generated at each stage of the analysis are used to derived the time complexity or running time of the two data encoding algorithms. The difference in running time of the two algorithms are established using their average time complexities. The performance difference of the two encoding algorithms is reached using the criteria of average code length per symbol or character, compression ratio performance (CRP), and average running time.

3. Data Encoding Methods

An effective mechanism or technique of achieving data deduplication is data compression. Data compression is the process of encoding information using fewer bits than the original representation. A particular data compression operation is either lossy or lossless mechanism. In lossless compression mechanism, bits reduction is done by identifying and eliminating statistical redundancy, and no data or information is lost in the process. Lossless compression is possible because most real-world data exhibits statistical redundancy. For example, an image may have areas of colour that do not change over several pixels. In this mechanism, compressed data can be reversed. Lossy compression mechanism reduces bits by removing unnecessary or less important information resulting in some form of data loss. A data compression operation can also be carried by a device (Tanvi *et al.,* 2019). A device that performs data compression is referred to as an encoder, and one that performs the reversal of the process (decompression) as a decoder. This means that a given data compression algorithm has two forms (encoding and decoding). For a given file or document of specific memory size to take less memory space, some form of encoding must be performed on the file or document. Encoding methods are of two types, and these are fixed length encoding and variable length encoding.

i. In fixed length encoding, length of code for a character or symbol is fixed.

The fixed length encoding method has the following advantages and disadvantages:

Advantages:

a. Decoding of files or documents encoded with fixed length encoding method is faster as there is no need to determine the position of the codes.

b. It is easy to encode and decode to compressed file or document

Disadvantages:

a. Inefficient. as a result of wastage of space.

b. Uses more bits

ii. In variable length encoding, code representation for a given character or symbol is of variable length. The variable length encoding method has the following advantage and disadvantage:

Advantages:

a. An improvement over fixed length encoding method as a result space efficiency.
b. Decoding of compressed file or document is with zero error.

Disadvantages:

a. Decoding of files or documents encoded with this method is more difficult since as a result of chances of ambiguity.
b. Harder to encode and decode file or document.

Huffman and the Shannon Fano encoding algorithms are two famous methods of variable length encoding for lossless data compression.

4. Huffman Compression Algorithm

Huffman encoding algorithm is a very popular algorithm for encoding data. Huffman encoding is a greedy algorithm, reducing the average access time of codes as much as possible. This method generates variable-length bit sequences called codes in such a way that the most frequently occurring character has the shortest code length. This is an optimal way to minimize the average access time of characters. It provides prefix codes and hence ensures lossless data compression and prevents ambiguity (Manjeet, 2017). The following steps illustrate the Huffman encoding algorithm for the compression of string data: dddd yyy w r qqqqqq

Step 1:

The sample data string to be encoded for compression is dddd yyy w r qqqqqq

Step 2:

Calculating the frequency of every character in the data string including spaces. Hence frequencies = {'d':4, ‘y:3, 'sp':4, 'w':1, 'r':1, ‘q:'6}. Space character is denoted as sp.

Step 3:

Create nodes of all the unique characters and their frequency. This is shown in Figure 2.

Figure 2: Data string characters represented by nodes

Step 4:

At every step, select 2 nodes of least frequency and combine them to form node x of frequency equal to sum of selected nodes. This is shown in Figures 3a through 3d.

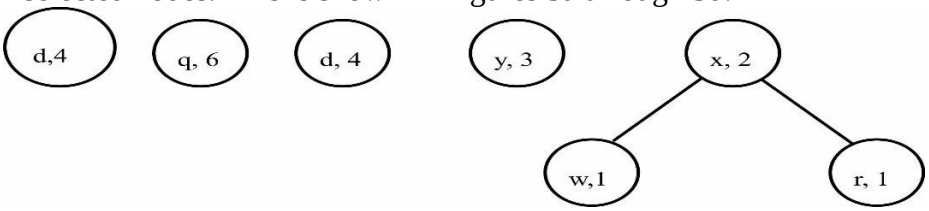

Figure 3a: Creation of node x,2

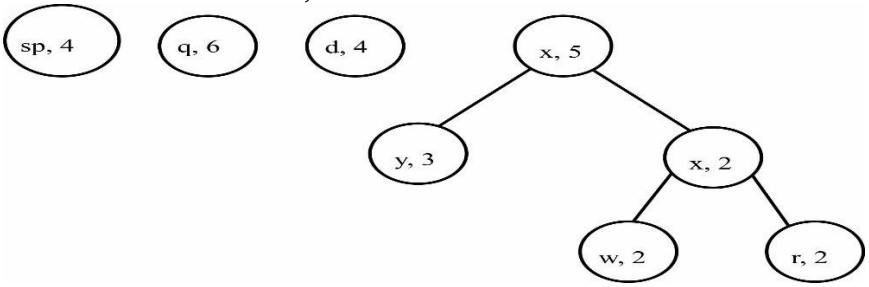

Figure 3b: Creation of node x,5

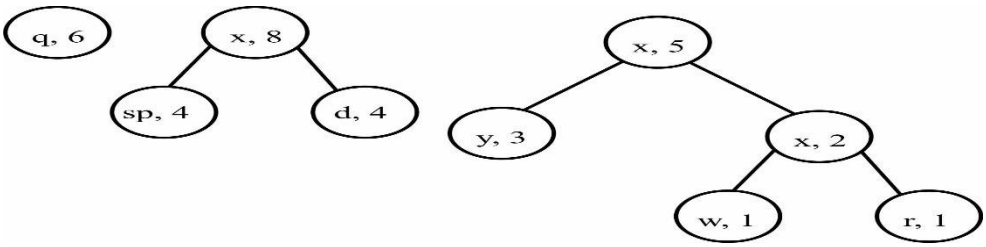

Figure 3c: Creation of node x,8

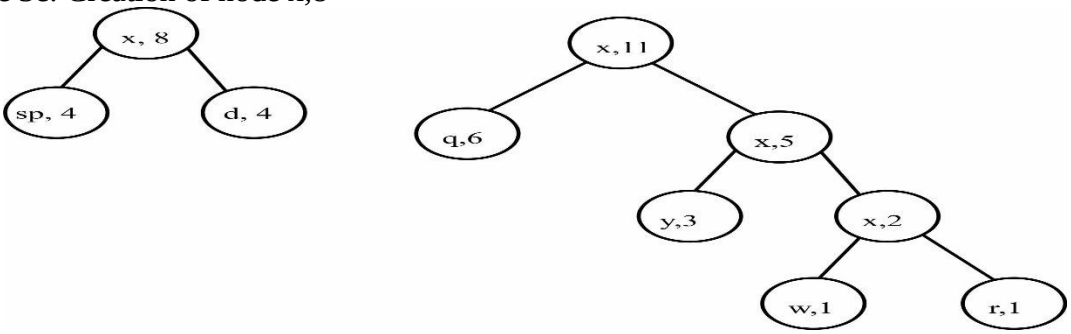

Figure 3d: Creation of node x,11

Step 5:

At this step the final tree is obtained with the root node x,19 s shown in Figure 4. The tree can now be traversed to assign codes.

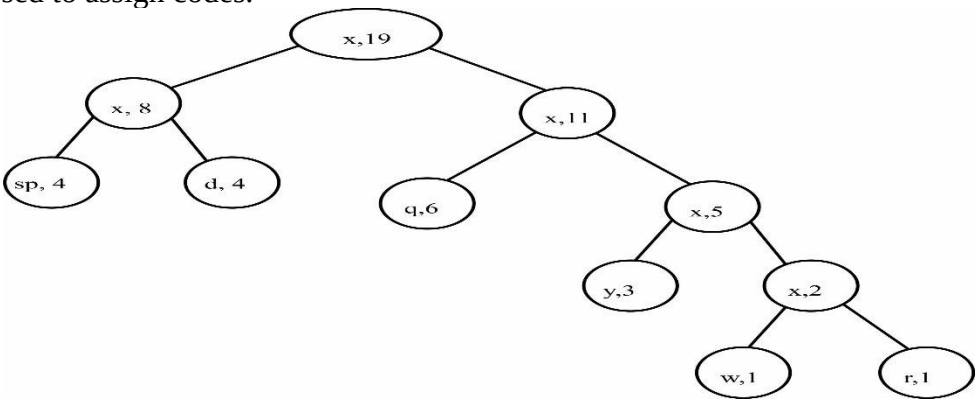

Figure 4: Creation of a tree representation of the data string

Step 6:

Start traversing from the root node. Every path to the left of a node is assigned 0 and that to the right is assigned 1. The traversed tree with the assigned codes is shown in Figure 5.

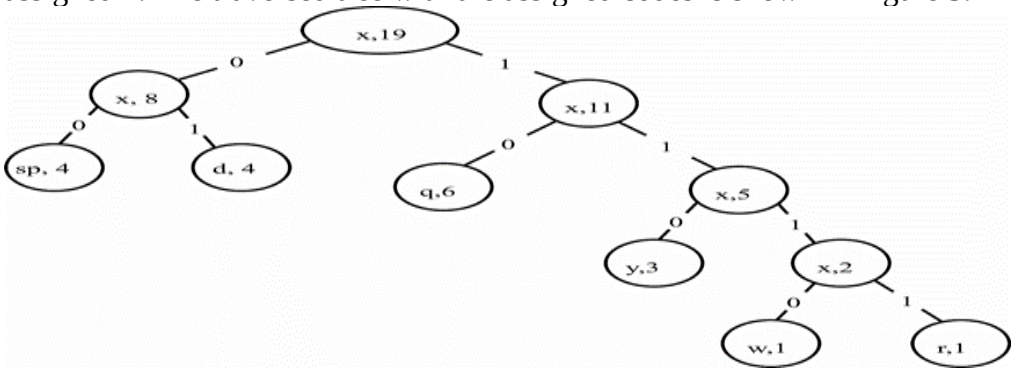

Figure 5: Traversed tree

Step 7:
To generate the code word corresponding to a given character y, traverse the tree in Figure 5 starting from the node root to leaf node y. The code word will be the numbers encountered on the path. Hence for character y in the data string, the generated code word or length is 110. Generally, the code generated for the sample data string is shown in Table 1.

Table 1: Code Generated using Huffman Encoding Method

Symbol	Code word	Code Length
w	1110	4
r	1111	4
y	110	3
d	01	2
sp (space)	00	2
q	10	2

4.1 Huffman Compression Algorithm Data Size Reduction Process

Character is 1 byte long = 8 bits. In the given sample, the data string "dddd yyy w r qqqqqq" requires: 8x19 = 152 bits, where 19 is the total number of symbols in the data string. Since 0 and 1 take only 1 bit each, with Huffman Encoding method, the data string requires code length for: d = 01 = 2 bit; y = 110 = 3 bits; q = 10 = 2 bits; w = 1110 = 4 bits
r = 1111 = 4 bits; sp = 00 =2 bits.
The total number of bits required to represent the compressed sample data string in memory is given by the expression as follows:

$$Total\ Bits = \sum_{i=1}^{n} (Frequency\ of\ ith\ symbol * Code\ length\ of\ ith\ symbol) \quad (2)$$

Using expression 2 for the sample data string, the total bits is computed as follows:
Total bits = (d*2) + (y*3) + (w*4) + (r*4) + (sp*2) + (q*2)
Total bits = (4x2) + (3*3) + (1*4) + (1*4) + (4*2) + (6*2) = 45 bits.
The average code length to represent the compressed equivalent of the given data string, and is given by the expression 3 as follows:

Average code length = Total no. of symbols / Total no. of bits (3)

Using expression 3 for the sample data string, the average code length is computed as follows:
Average code length = 45/19 = 2.38. This means that the average code length is 2.38 bits per symbol as compared to 8 bits per symbol before encoding or compression. The compression ratio using the Huffman encoding algorithm for the compression of the given data string, can be computed with the expression given as follows:

Compression Ratio = Total no. of Symbols (input) / Total no. of Bits (output) (4)
= 45 / 19
Compression Ratio = 2.38

This value shows that the rate of compression of uncompressed data string to the compressed data string is approximately 3 to 1. This is the compression ratio performance (CRP) of the data compression algorithm. This is equivalent to the average code length of 2.38 bits per symbol as compared to 8 bits per symbol before encoding or compression.
Similarly, the compression factor of Huffman data compression algorithm is computed as the inverse of the compression ratio, and this is given in expression 5.

*Compression Factor = (Total no. of Bits (output) / Total no. of Symbols (input)) * 100* (5)

Using expression 5 for the sample data string, the compression factor is computed as follows:
*Compression factor = (19 / 45) * 100*
= 0.42 * 100

= 42%. This value represents the percentage compression of the sample data string using Huffman data compression algorithm.

4.2 Time Complexity analysis of Huffman Encoding Method

Given n symbols to be represented by n nodes in a tree, the construction of a tree data structure for the n nodes takes O(n). Every iteration takes out two nodes and insert one node therefore, for n nodes, it takes n - 1 iterations to make the tree (Ardilus and Hertog, 2018). Using heap to store the weight of each tree, each iteration requires 2*(n - 1) times, that is, the time taken to determine the extraction of code of a given symbol with minimum frequency from the priority queue with the time complexity of $O(\log_2 n)$, and this is average case of the algorithm, and to fetch code for all the nodes, the tree must be traversed in O(n) time (Kodituwakku and Amarasinghe, 2019). Therefore, the worst case time complexity of the Huffman encoding method for data compression is: $O(n) + O(\log_2 n) = O(n \log_2 n)$.

5. Shannon Fano Compression Algorithm

Shannon Fano Algorithm is an entropy encoding method used for lossless data compression. It uses the probabilities of occurrence of a character and assigns a unique variable-length code to each of characters (Mohammed and Ibrahim, 2018). If x is a character, the probability of x is given by the following expression:

Probability(x) = Frequency(x) / sum of frequencies. (6)

Given the data string: dddd yyy w r qqqqqq, the following steps illustrate the Shannon Fano encoding algorithm for the compression of string data.

Step 1:

The probability computation of each symbol or character that makes up the data string using expression (3) is shown in Table 2.

Table 2: Probability Computation

S/n	Symbol	Frequency	Probability
1.	D	4	4/19 = 0.21
2.	Y	3	3/19 = 0.16
3.	W	1	1/19 = 0.05
4.	R	1	1/19 = 0.05
5.	Q	6	6/19 = 0.32
6.	Sp (space)	4	4/19 = 0.21
7.	Sum of Frequencies = 19		

The symbols or characters that constitute the data string are sorted in increasing order of frequency as shown Figure 6.

r,0.05	w,0.05	y,0.16	d,0.21	sp,0.21	Q,0.32

Figure 6: Sorted symbols in terms of frequency

Step 2:

Divide the symbols or characters that make up the data string into two halves (left half and right half) such that the sum of frequencies is as close as possible as shown in Figure 7.

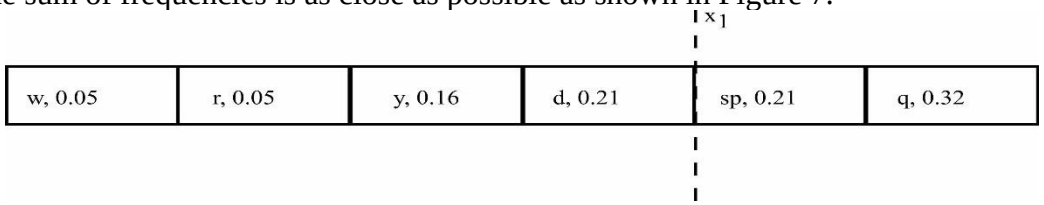

Figure7: Division of data string into two halves

Step 3:
Repeat the step 2 for each half until individual elements are left (Ashok *et al.*, 2017). The repetitions are shown in Figure 8a through 8d.

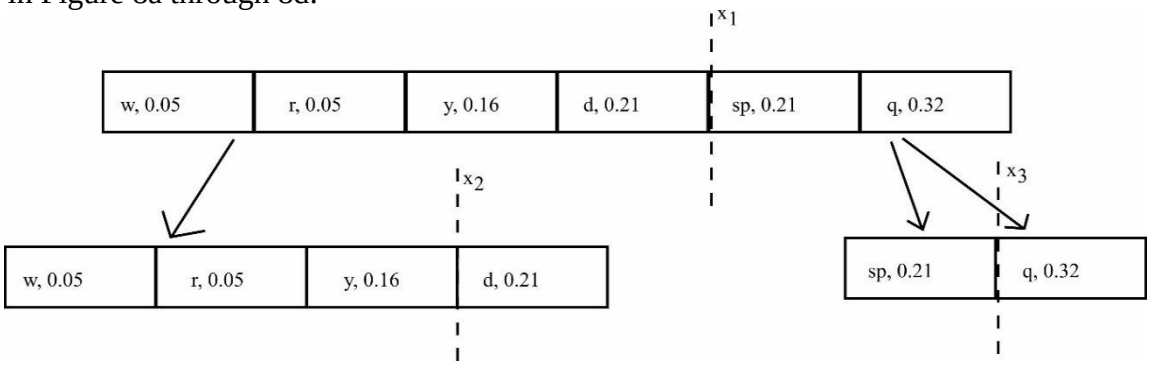

Figure 8a: Division of x_1 halves

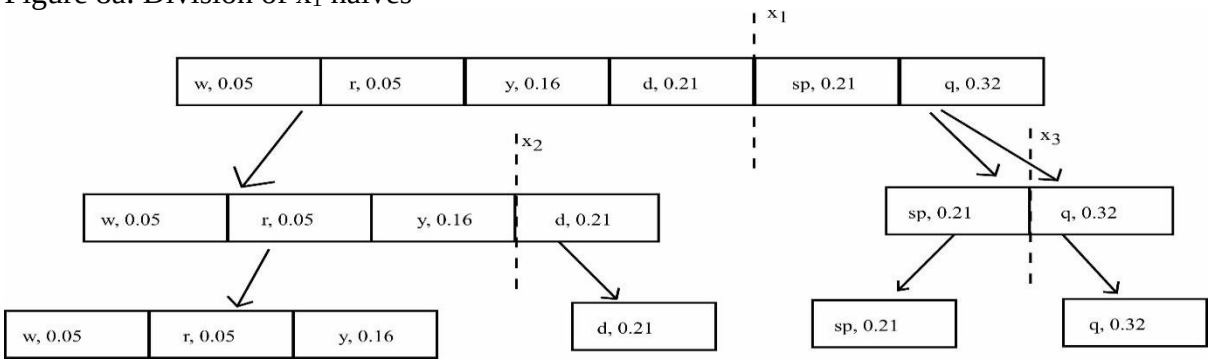

Figure 8b: Division of x_2 and x_3 halves

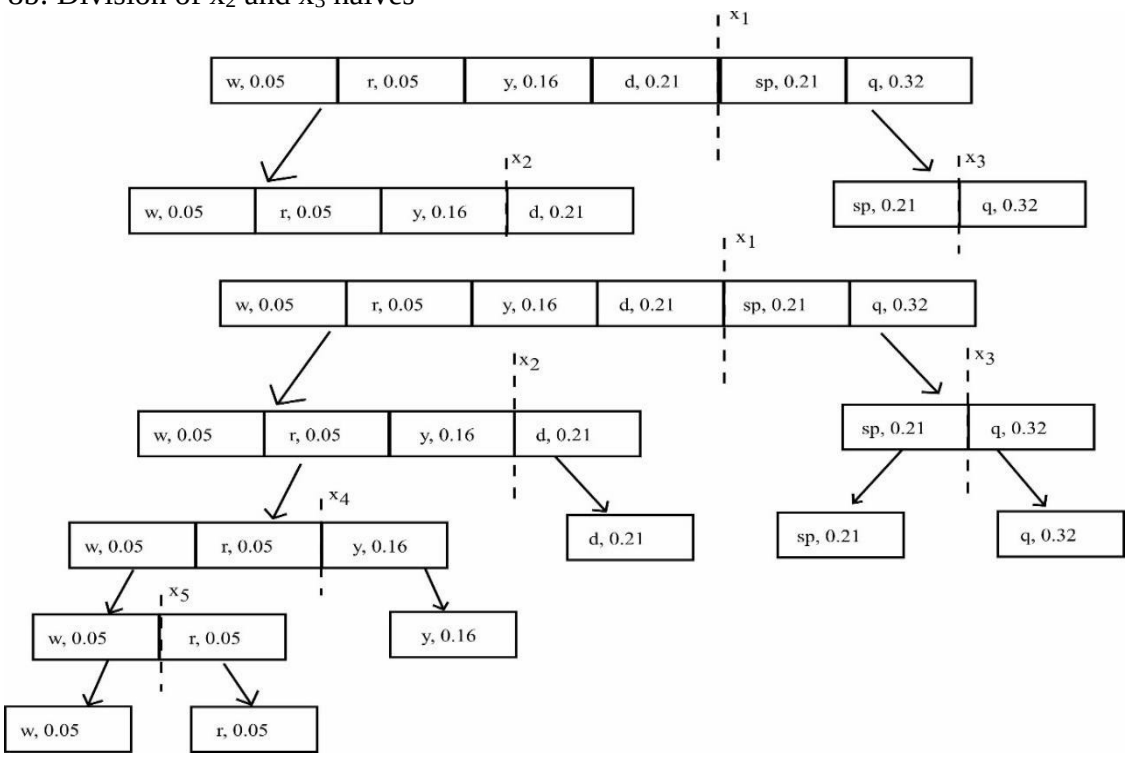

Figure 8c: Division of x4 and x5 halves

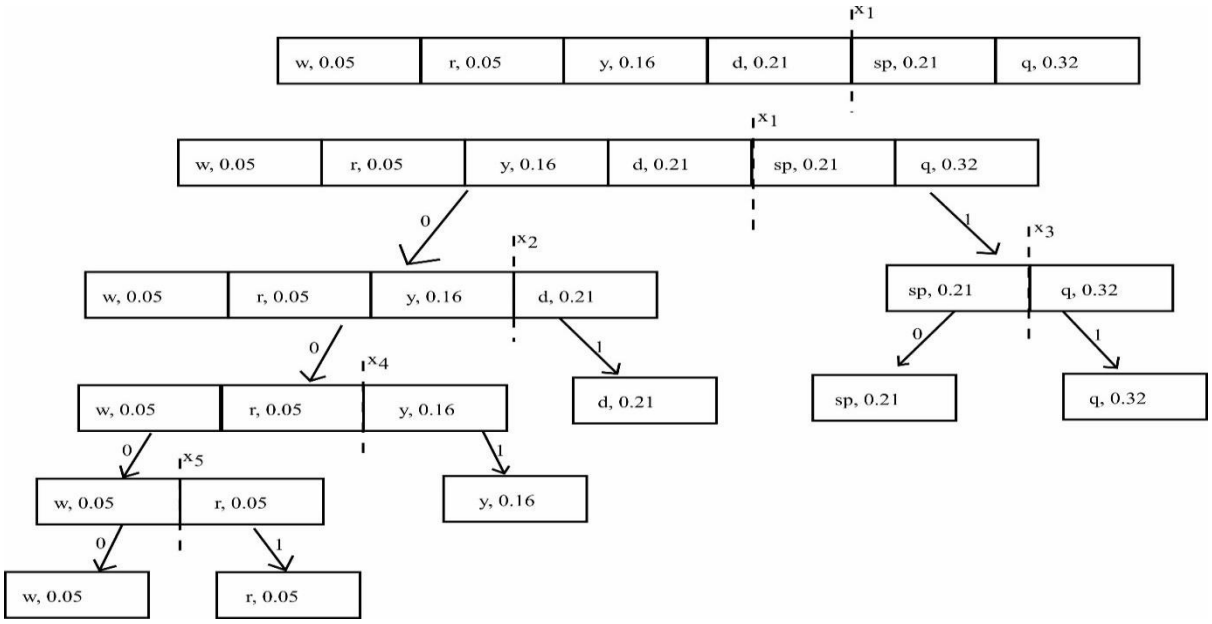

Figure 8d: Final tree representation of the sample data string

The generated code for the sample data string by traversing the final tree representation is shown in Table 3.

Table 3: Code Generated using Shannon Fano Encoding method

Symbol	Code word	Code Length
w	0000	4
r	0001	4
y	001	3
d	01	2
Sp (space)	10	2
q	11	2

5.1 Shannon Fano Compression Algorithm Data Size Reduction Process

A character is 1 byte long = 8 bits (VidyaSayer and Victor, 2018). In the given sample, the data string "dddd yyy w r qqqqqq" requires: 8x19 = 152 bits, where 19 is the total number of symbols in the data string. Since 0 and 1 take only 1 bit each, with Shannon Fano Encoding method, the data string requires code length for:

w = 1110 = 4 bits; r = 1111 = 4 bits; y = 110 = 3 bits; sp = 00 = 2 bits; d = 01 = 2 bit;
q = 10 = 2 bits

The total number of bits required to represent the compressed sample data string in memory is given by the expression (2) as follows:

$$Total\ Bits = \sum_{i=1}^{n} (Frequency\ of\ ith\ symbol * Code\ length\ of\ ith\ symbol) \quad)$$

Using expression (2) for the sample data string, the total bits is computed as follows:

Total bits = (d*2) + (y*3) + (w*4) + (r*4) + (sp*4) + (q*2)
Total bits = (4x2) + (3*3) + (1*4) + (1*4) + (4*2) + (6*2) = 45 bits

The average code length to represent the compressed equivalent of the given data string, and is computed using expression (3).

Average code length = Total no. of symbols / Total no. of bits

Using expression (3) for the sample data string, the average code length is computed as follows:

Average code length = 45/19 = 2.38. This means that the average code length is 2.38 bits per symbol as compared to 8 bits per symbol before encoding or compression.

The compression ratio of the Shannon Fano encoding algorithm for the compression of the given data string, can be computed with expression (4) as follows:

Compression Ratio = Total no. of Symbols (input) / Total no. of Bits (output) = 45 / 19

Compression Ratio = 2.38

This value shows that the rate of compression of uncompressed data string to the compressed data string is approximately 3 to 1. This is the compression ratio performance (CRP) of the Shannon Fano data compression algorithm. This is equivalent to the average code length of 2.38 bits per symbol as compared to 8 bits per symbol before encoding or compression.

Similarly, the compression factor of Shannon Fano data compression algorithm is computed as the inverse of the compression ratio, and this is given in expression (5).

*Compression Factor = (Total no. of Bits (output)/Total no. of Symbols (input)) * 100*

Using expression (5) for the sample data string, the compression factor is computed as follows:

Compression factor = 19 / 45

= 0.42

= 42%. This value represents the percentage compression of the sample data string using Shannon Fano data compression algorithm.

5.2 Time Complexity analysis of Shannon Fano Algorithm

The individual symbols that constitute the sample data string may be arranged in the increasing order of frequency or decreasing order of probability are repeatedly partitioned (Ahamad *et al.*, 2019). The partitions carried out each time may be extremely unbalanced (Mahdi *et al.*, 2016), for instance, if the probabilities are 0.06, 0.1, 0.11, 0.2, and 0.4, the recurrence relation is:

$T(n) = T(n – 1) + O(n)$

$T(n) = O(n^2)$); This is the worst case of the Shannon Fano encoding algorithm for data compression. If the partition can be divided in such a way that their sizes are nearly equal [Sullivan et al., (2015, for instance, if the probabilities are 0.4, 0.7, 0.7, 0.5, the recurrence relation is: $T(n) = 2*T(n/2) + O(n)$

$T(n) = O(n*\log_2 n)$; this is the average and best cases the Shannon Fano encoding algorithm for data compression.

6. Comparison of the Selected Data Compression Algorithms

The selected encoding algorithms, that is, Huffman and Shannon Fano, for data compression are compared using the following criteria:

a. Average code length per symbol

The average code length per symbol is computed using expression (2) for the sample data string given as: "dddd yyy w r qqqqqq" is 2.38 for the two algorithms. This means that the two data encoding algorithms take approximately 2.38 bit codes to represent one symbol or character for the sample data string considered.

b. **Compression ratio performance (CRP)**

The compression ratio performance (CRP) of the two data encoding algorithms is computed using expression (4). The value obtained is:

Compression Ratio = 45 / 19 = 2.38

This value shows that the rate of compression of uncompressed data string to the compressed data string is approximately 3 to 1. This is the compression ratio performance (CRP) which is equivalent or similar for the two data compression algorithm.

c. Compression factor

The compression factor of the two selected data compression algorithms for the sample data string considered is computed using expression (5) as follows:

*Compression Factor = (Total no. of Bits (output)/Total no. of Symbols (input)) * 100*

Compression factor = 19 / 45 = 0.42 42% This value represents the percentage compression of the sample data string using the both Huffman and Shannon Fano data compression algorithms. The two data compression algorithms exhibit similar compression factor.

d. Running time:

The analysis of the running time of the two encoding algorithms for data compression, that is, Huffman and Shannon Fano, shows that they take different running time to generate codes for input symbols. The time complexity of the two data encoding algorithms considered is shown in Table 4.

Table 4: Complexity of Huffman and Shannon Fano Data Encoding Algorithms

Data encoding Algorithm	**Time Complexity**		
	Best Time	**Average Time**	**Worse Time**
Huffman	$O(Log_2 n)$	$O(Log_2 n)$	$O(n*Log_2 n)$
Shannon Fanon	$O(n*Log_2 n)$	$O(n*Log_2 n)$	$O(n^2)$

The running time to generate codes for different input size using the two data encoding algorithms average time complexity is shown in Table 5.

Table 5: Average Running Time of Huffman and Shannon Fano Data Encoding Algorithms

S/N	**Input Size (n)**	**Huffman Encoding Algorithm: t(n) = Log_2 n**	**Shannon Fano Encoding Algorithm: t(n) = n*Log_2 n**
1	5	2.3219	11.6095
2	10	3.3219	33.2190
3	15	3.9065	58.5975
4	20	4.3219	86.4380
5	25	4.6039	115.9750
6	30	4.9069	147.2070

The computation in Table 5 shows that the Huffman encoding algorithm takes a smaller or shorter time in seconds to generate codes compared to the Shannon Fano encoding algorithm for the same input size using their average time complexity. The performance behavior is further shown in Figure 11:

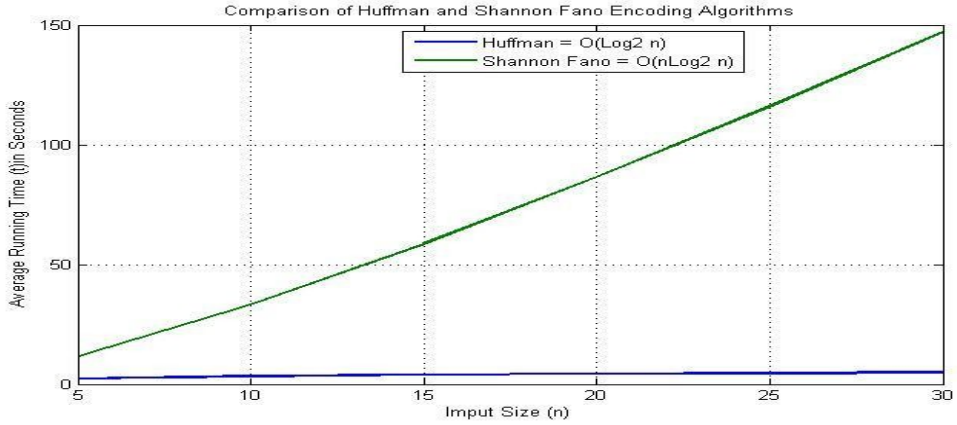

Figure 11: Average running time of Huffman and Shannon Fano encoding Algorithms

The graph in Figure 11 shows pictorially that given the same size of data of a specified amount of memory space to be compressed, Huffman encoding algorithm compresses the data in a superior time than the Shannon Fano encoding algorithm.

7. Findings

The analysis and evaluation performance of the two selected data encoding algorithms with the data string of the same memory size for compression results in the following findings:

i. The Huffman encoding algorithm takes a shorter time than Shannon Fano encoding algorithm.
ii. The Huffman and Shannon Fano encoding algorithms have the same average code length per symbol.
iii. The Huffman and Shannon Fano encoding algorithms operates within the same range of compression ratio performance and compression factor.

8. Conclusion

Algorithmic analysis of the Huffman and Shannon Fano data encoding algorithms for compression reveals that the two encoding algorithms have the same characteristic behaviour in terms of the average code generated per symbol or character as well as compression ratio performance (CRP) and compression factor based on the sample data string used for investigation. Evaluation of the two encoding algorithms based on time complexity shows that the Huffman encoding algorithm has a superior average running time over the Shannon Fano encoding algorithm when subjected to the compression of sample data string of the same size.

9. References

[1]. Ahmad, O., Mohammed, O. and Mahmoud, K. (2019). Comparative Study between LM-DH Technique and Huffman Coding Technique. International Journal of Applied Engineering Research. 36(20), 123 -145.

[2]. Ardiles, S. A. and Hertog, N (2018). Development of Word-Based Text Compression Algorithm for Indonesian Language Document. International Conference on Information and Communication Technology (ICoICT), Jakarta, Indonesia.

[3]. Ashok, K., Babu, V., and Satish, K. (2017). Implementation of Data Compression Using Huffman Coding. International Conference on Methods and Models in Computer Science, Mumbai, India.

[4]. Kodituwakku, S. R. and Amarasinghe, S. U. (2019). Comparison of Lossless Data, Compression Algorithms for Text. Indian Journal of Computer Science and Engineering, 21(14), 023 – 034.

[5]. Mahdi, O. A., Mohammed, M. A. and Mohamed, A. J. (2016). Implementing a Novel Approach an Convert Audio Compression to Text Coding via Hybrid Technique. International Journal of Computer Science Issues, 9 (6), 053 – 059.

[6]. Manjeet, K. (2017). Lossless Text Data Compression Algorithm Using Modified Huffman Algorithm. International Journal of Advanced Research in Computer Science and Software Engineering, 41(23), 045 – 057.

[7]. Mohammad, H. (2018). A Survey of Data Compression Algorithms and their Applications. Applications of Advanced Algorithms Journal, 5(3), 201 – 215.

[8]. Mohammed, A. and Ibrahim, M. M. (2018). Comparative Study between Various Algorithms of Data Compression Techniques. International Journal of Computer Science and Network Security (IJCSNS), 23(35), 134 – 145.

[9]. Pujar, J. H. and Kadlaskar, L. M. (2017). A New Lossless Method of Image Compression and Decompression Using Huffman Coding Techniques. Journal of Theoretical and Applied Information Technology, 5 (1), 018 – 023.

[10]. Sullivan, G. J., Ohm, J. R., Han, W. J. and Wiegand, T. (2015). Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology, 22 (12), 1649 – 1668.

[11]. Tanvi, P., Kruti, D., Judith, A. and Poonam, C. (2019). Survey of Text Compression Algorithms. International Journal of Engineering Research and Technology (IJERT), 34(24), 124 – 144.

[12]. VidyaSagar, M. and Victor, J. S. (2018). Modified Run Length Encoding Scheme for High Data Compression Rate. International Journal of Advanced Research in Computer Engineering and Technology (IJARCET). 45(38): 145 - 158.