International Journal of Computer Applications Technology and Research
Volume 12–Issue 03, 32-39, 2023, ISSN:-2319–8656
DOI:10.7753/IJCATR1203.1009

Huffman Algorithm Valuation Using Residue Number System

Lawal T. Dauda
Department of Computer Science
Federal Polytechnic Offa
Kwara State, Nigeria

Eseyin Joseph B
ICT Directorate
University of Jos
Jos, Nigeria

Azeez O. Isiaka
Department of Statistics
Federal Polytechnic Offa
Kwara State Nigeria

Abstract: The Huffman algorithm is a widely used method for lossless data compression, which assigns variable-length codes to characters based on their frequency of occurrence in the input data. However, the traditional implementation of Huffman coding using binary arithmetic can be computationally intensive, particularly for large data sets. In recent years, the Residue Number System (RNS) has emerged as a promising alternative to binary arithmetic for certain types of computations, due to its potential for parallel processing and reduced hardware complexity. This paper evaluates the use of RNS as a basis for implementing the Huffman algorithm, comparing its performance with the traditional binary approach. The results demonstrate that RNS-based Huffman coding can achieve comparable or superior compression ratios, while reducing the computational requirements and potentially enabling faster compression and decompression. The study also highlights the importance of choosing appropriate RNS moduli and operands to optimize performance. Overall, the evaluation suggests that RNS can be a viable and efficient alternative to binary arithmetic for implementing the Huffman algorithm, particularly in applications with high computational demands or limited hardware resources. However, further research is needed to explore the potential benefits and limitations of RNS in other areas of data compression and signal processing.

Keywords: Residue Number System, lossless compression, compression ratio, decompression, moduli set, binary value, bit coding

1.0 INTRODUCTION

Data management and processing must include data compression since it reduces the size of data files, allowing for more effective storage and quicker transfer. Compression in computer science is based on the philosophy that some data can be represented more efficiently by using fewer bits than their original representation without losing significant information. This can help reduce storage requirements and transmission times, making it an important tool in managing large amounts of data. There are numerous data compression algorithms that each have benefits and drawbacks. This widely used data compression algorithm—Huffman coding will be the subject of this paper. These methods were chosen as a decent illustration of the numerous kinds of algorithms available and because they are among the most used data compression algorithms.

By reducing the amount of data files, compression methods improve the efficiency of storage and transmission. There are several compression methods available, and each has advantages and disadvantages. Arithmetic coding, Shannon-Fano coding, and Huffman coding are three popular compression techniques.

Huffman Coding: Based on the frequency with which characters appear in the input data, Huffman Coding assigns characters' variable length codes. The technique employs a binary tree to represent the code words, with the weight of the tree's nodes based on the frequency of each character. The shortest code words are given to the characters with the lowest frequency, while the longest code words are given to the characters with the highest frequency. Because the shortest code words may represent the most characters, there is a higher compression ratio as a result. Huffman coding is one of the most frequently used coding algorithms for lossless data compression, claim creators J. Ziv and A. Lempel [1].

2.0 REVIEW OF RELATED LITERATURE

Data files can be made smaller and more efficiently stored and sent by using compression methods. Numerous compression techniques have been created throughout the years, but Huffman coding, Shannon-Fano coding, and arithmetic coding are the three that are still in use today.

Huffman Coding Since its invention in the 1950s, Huffman coding has been extensively utilized for lossless data compression. Recent studies have concentrated on increasing the compression ratio of the technique by representing the code words in more complex data structures, such as Fibonacci heaps and B-trees. Researchers have also looked into Huffman coding as a method for compressing images, and they discovered that the process can offer high compression. For instance, a new picture compression technique based on Huffman coding and the Discrete Cosine Transform (DCT) was proposed by Y. Lee et al. in a recent work [1]. High compression ratios were attained using the suggested technique while still keeping the integrity of the reconstructed image.

In recent years, a great deal of study has been done on arithmetic coding, leading to numerous developments in both lossless and lossy data compression. Using arithmetic coding and the DCT, J. Kim et al. recently proposed a new lossy image compression technique [3]. The suggested technique produced large compression ratios with good image reconstructed quality. A new lossless data compression method based on arithmetic coding and a context-based coding scheme was proposed by X. Zhang et al. in another paper [4].

3.0 METHODOLOGY

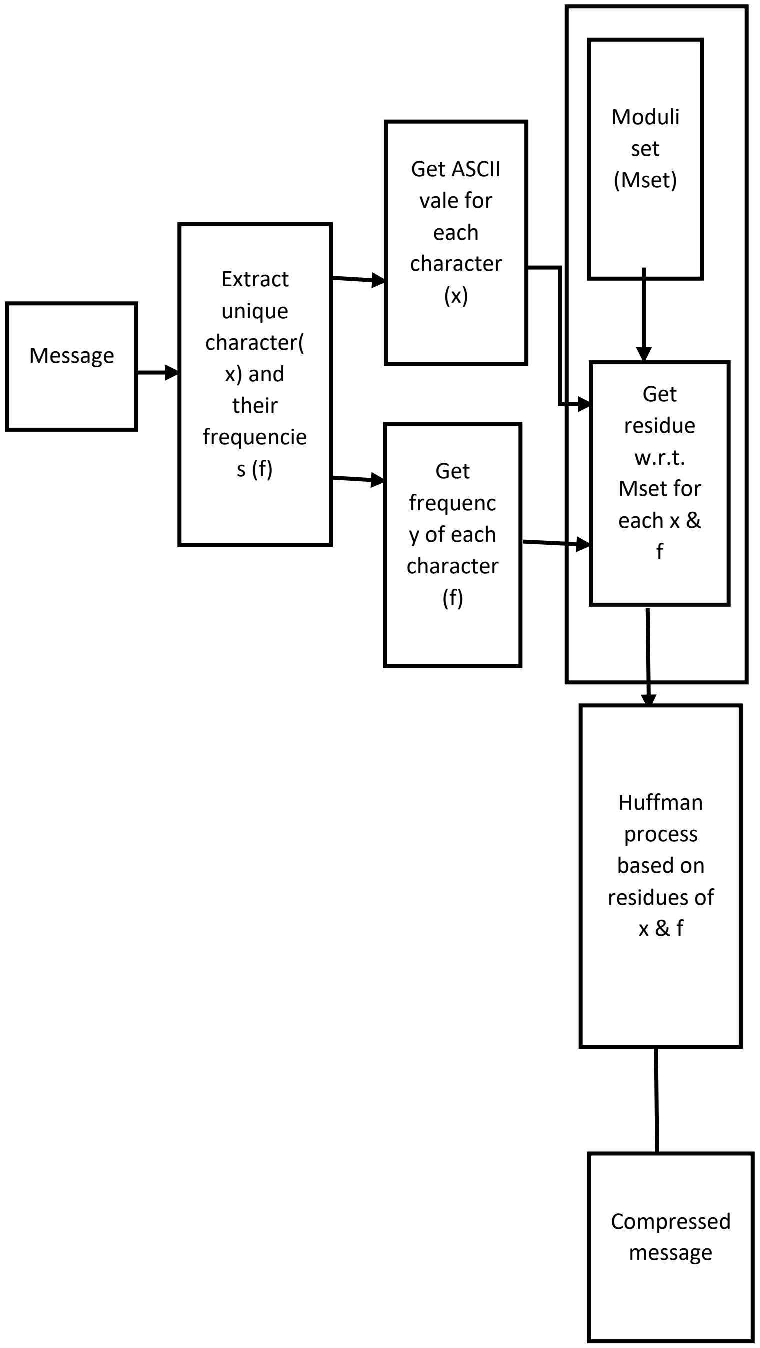
The algorithms were implemented in Python and the implementation was based on the standard algorithms as described in literature. The algorithms were evaluated based on their compression ratios and computational time. The compression ratio was calculated as the ratio of the size of the compressed file to the size of the original file. The computational time was calculated as the time taken to compress the file. The algorithms were tested on a range of text files of different sizes to provide a representative evaluation of the algorithms.

3.1 Framework for an enhanced Huffman Algorithm and Shannon-Fano Algorithm

To build enhanced Lossless data compression algorithms namely enhanced Huffman algorithm hereafter referred to RNS-Huffman the data were obtained from which the unique character and their frequencies of occurrence were extracted. The next stage is to extract the ASCII value of each symbol making up the message along with their frequency.

The next stage is to get the residue value with respect to the given moduli set. This is followed by employing RNS arithmetic in Huffman computations. This is shown in Figure 1.

Figure 1: Framework for RNS Lossless Compression Algorithms Design


Message
Extract unique character(x) and their frequencies (f)
Get ASCII vale for each character (x)
Get frequency of each character (f)
Moduli set (Mset)
Get residue w.r.t. Mset for each x & f
Huffman process based on residues of x & f
Compressed message

3.2 Algorithm 1: RNS-Huffman Algorithm

```
Input: the message to be compressed
Output: the compressed message
1)  Extract each character
2)  for a given list of symbols,
3)  develop a frequency table;
4)  sort the table according to the frequency, in ascending order
5)  obtain the ASCII value of each character;
6)  obtain the traditional moduli set;
7)  get residue of each character's ASCII value with respect to moduli set
8)  get residue with respect to moduli set for each frequency;
9)  perform Huffman compression process based on residues of characters and frequencies.
10) obtain a compressed information
```

4.0 EXPERIMENTAL RESULTS

RNS-based Computation using Traditional Moduli set;

The traditional moduli set is $(2^n+1, 2^n, 2^n-1)$ was used. Using the value of n to be equal to 2, the moduli set is (5, 4, 3). For ASCII value of a character, its equivalent residue value is calculated with respect to the moduli set. For example, given the message

"FFFAAABBEBBCCCCCDDDEEEBBFFFFBBDDB CBBBFFF"

to be store or transmitted. The length of the message is 40. Without compression, the message will be stored or transmitted using the ASCII code. In the ASCII code, each alphabet is 8 bits.

Table 1: Characters and their Frequencies

Character	A	B	C	D	E	F
Frequency	3	12	6	5	4	10

Table 2: Message Distribution

Character	Frequency	ASCII code	Binary bit Value	No of Bits	Freq* No of bits
A	3	65	01000001	8	24
B	12	66	01000010	8	96
C	6	67	01000011	8	48
D	5	68	01000100	8	40
E	4	69	01000101	8	32
F	10	70	01000110	8	80
Total	40		48		320

The space required to store or send this message is 320 (i.e., 40 * 8) bits.

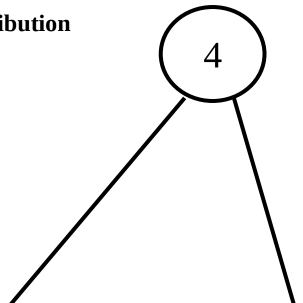
4.1 Huffman Technique

The Huffman technique of storing or sending this message is accomplished by first tabulating the characters along with their frequencies.

Table 3: Characters and their Frequencies in the message

Character	Frequency
A	3
B	12
C	6
D	5
E	4
F	10
Total	40

The next step is to arrange the characters horizontally in ascending order of their frequencies after which two smallest ones are merged and the sum is recorded. This action will be repeated until all the alphabet are exhausted as shown in Figure 2 below.



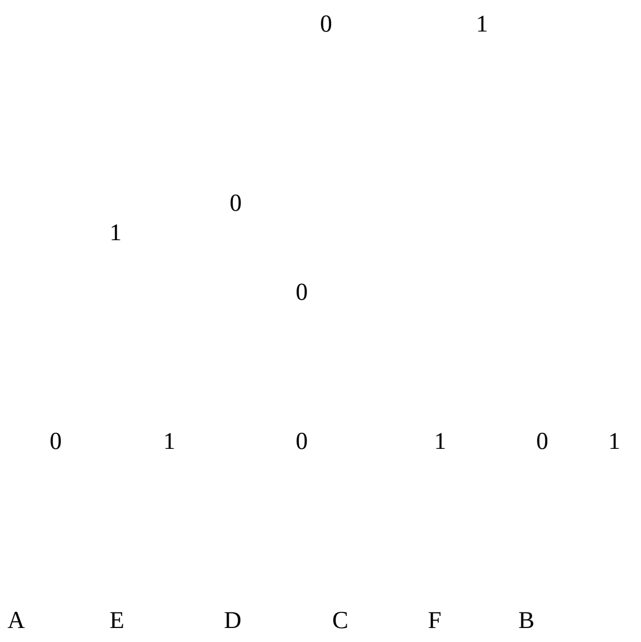


Figure 2: Huffman Data Compression Tree Sourse: (Umarani, Sriram & Kumar, 2017)

On the left of each alphabet, mark them as 0s and on the right mark them as 1s. For each alphabet, follow the path from the root. The codes are as indicated in the Table 4 below.

Table 4: Message bits representation

Character	Frequency	codes	No of bits
A	3	000	3*3 =9
B	12	11	12*2 =24
C	6	011	6*3 = 18
D	5	010	5*3 = 15
E	4	001	4*3 = 12
F	10	10	10*2 =20
Total	40	16 bits	98 bits

The total size of the message is 98bits. But along this message, the chart or table must also be stored or sent, so that the decoding can be done. Thus, the table or the chart must be preserved. In addition, the ASCII code of the alphabet must also be stored or sent. The number of alphabets is 6. Therefore 6*8 will give 48 bits. The total number of the codes is 16 bits. To preserve the tree, the number of bits require is 64 bits i.e., 48 + 16. The message is 98 bits and the tree is 64 bits given a total of 162 bits. The message size has been reduced from 320 bits to 162 bits i.e. it has been reduced by about 50% by Huffman technique.

4.2 RNS-Huffman ASCII Value Computation

Table 5: Conversion of ASCII code value to its RNS equivalent

Character	ASCII Code	Binary Code	RNS with Moduli set			Bits Space
			3	4	5	
A	65	8	2	1	0	4
B	66	8	0	2	1	4
C	67	8	1	3	2	5
D	68	8	2	0	3	5
E	69	8	0	1	4	5
F	70	8	1	2	0	4
Total		48				27

Table 6: RNS-Huffman Computation

Character	freq	RNS with moduli set			Hufman code	RNS with moduli set			Freq* Huffman code	RNS with moduli set			Bits Space
		3	4	5		3	4	5		3	4	5	
A	3	0	1	2	111=3	0	1	2	3*3=9	0	0	4	5
B	12	0	0	2	00=2	2	2	2	12*2=24	0	0	4	5
C	6	0	2	1	101=3	0	1	2	6*3=18	0	2	2	5
D	5	2	1	0	100=3	0	1	2	5*3=15	0	1	0	3
E	4	1	0	1	110=3	0	1	2	4*3=12	0	0	2	4
F	10	1	2	0	01=2	2	2	2	10*2=20	2	0	0	4
Total	40				16 bits				98 bits				26 bits

Table 7: Total Huffman and RNS-Huffman bits

	Huffman codes	RNS-Huffman Codes
Message	98 bits	27 bits
ASCII binary codes	48 bits	26 bits
Total bits on tree	16 bits	16 bits
Total	162 bits	69 bits

4.3 Performance Metrics

The performance metrics used for evaluating the algorithms are Compression Ratio (CR)

4.3.1 Compression Ratio

Data compression ratio is the compression power of an algorithm. It is a measurement of the relative reduction in size of data representation produced by a data compression algorithm. It is usually expressed as a division of uncompressed size by a compressed size.

It is the ratio of total number of bits required to store uncompressed data and total number of bits required to store compressed data. It is termed a bit per bit (bpb) which defined the number of bits required to store the compress data. This was calculated by finding the ratio between the compressed and original file as:

$$CR = \frac{\text{Number of bits in uncompressed data}}{\text{Number of bits in compressed data}}$$

4.3.2 Evaluation of RNS-Huffman

Twenty-five different file documents of different sizes were compressed using Huffman coding and RNS-Huffman algorithms. The results were evaluated and analyzed using Compression Ratio (CR).

4.3.3 RESULT OF EVALUATION

Sizes of Output in bits of Huffman,

The output of the twenty-five (25) text file document of different sizes from Huffman coding, compression algorithms is given in the Table 4.

Table 8: Compressed File Size (bits)

Text File	Original File size (bits)	Huffman Compressed File size (bit)
1	2552	1843
2	5152	3343
3	7392	4579
4	9904	5950
5	12440	7331
6	15056	8791
7	17576	10157
8	20352	11801
9	24680	15068
10	27280	16625
11	29616	18096
12	31880	19491
13	35416	21612
14	37968	22997
15	40600	24563
16	43224	26042
17	45968	27627
18	48576	29075
19	51352	30644
20	53968	32103
21	56592	33567
22	59160	35232
23	61808	36903
24	64552	38417
25	67080	39894
Average	**34806**	**20871**

From table 8 The average file size of 20871 for Huffman coding from the average original file size of 34806 were obtained. The variation in sizes from Table is given in Figure 4.1 below. This show that of Arithmetic coding performs better than Huffman which is also better than Shannon-Fano coding in term of compression size. Figures 4.1 and 4.2 show the result of sizes of output.

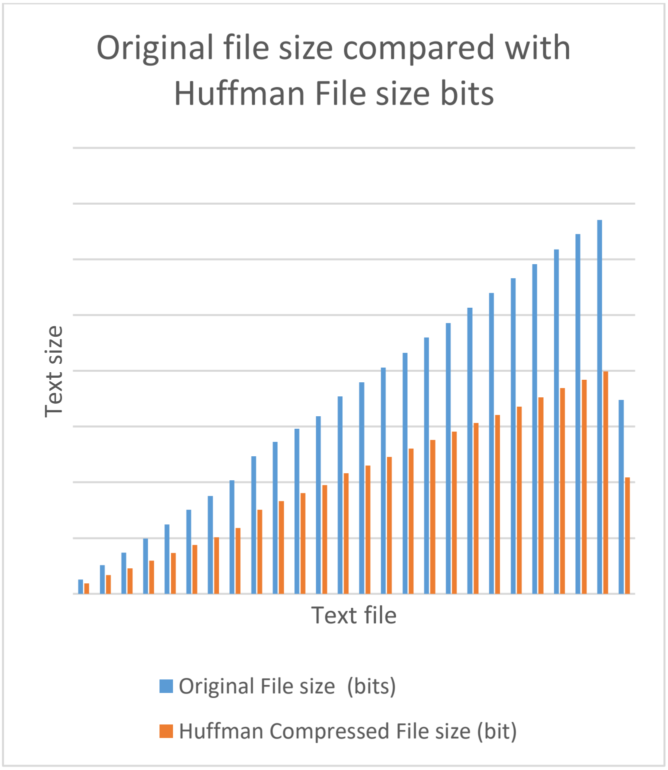


Figure 3 Huffman algorithm file size with bit size

4.3 Compression Ratio

The compression ratio was computed for Huffman coding for the twenty-five differently sized file documents. The results are shown in table 4.2 below.

Table 9: The Compression Ratio of Huffman, algorithms

Text File	Original File size (bits)	Huffman Algorithm
1	2552	1.384698861
2	5152	1.541130721
3	7392	1.614326272
4	9904	1.664537815
5	12440	1.69690356
6	15056	1.712660676
7	17576	1.730432214
8	20352	1.72459961
9	24680	1.63790815
10	27280	1.640902256
11	29616	1.636604775
12	31880	1.6356267
13	35416	1.63871923
14	37968	1.650997956
15	40600	1.652892562
16	43224	1.659780355
17	45968	1.663879538
18	48576	1.670713672
19	51352	1.675760345
20	53968	1.681088995
21	56592	1.68594155
22	59160	1.679155313
23	61808	1.674877381
24	64552	1.680297785
25	67080	1.681455858
Average	**34805.76**	**1.652635686**

4.3.4 Evaluation of RNS-Huffman Algorithms

In this stage, performance of RNS-Huffman coding is done. The same twenty-five text files of different sizes were compressed using these algorithms. Their CR were computed and used to compare their performances.

4.3.5 Compressed File sizes

The sizes of the output from RNS-Huffman Algorithm when fed with the file documents as inputs are shown in Table 7.

Table 10: Compressed file sizes of RNS-Huffman

Text File	Original Size (bits)	RNS-Huffman (bits)
1	2552	515
2	5152	610
3	7392	650
4	9904	696
5	12440	738
6	15056	755
7	17576	798
8	20352	855
9	24680	1114
10	27280	1128
11	29616	1136
12	31880	1155
13	35416	1186
14	37968	1174
15	40600	1180
16	43224	1172
17	45968	1175
18	48576	1180
19	51352	1187
20	53968	1201
21	56592	1208
22	59160	1191
23	61808	1196
24	64552	1198
25	67080	1190
Average	**34805**	**1023**

From table 10, compressed output of RNS-Huffman coding is 1023 bits, original uncompressed document is 34805 bits long.

4.3.6 Compression Ratio of RNS-Huffman Algorithms

Compression ratios for RNS-Huffman are presented in Table 8.

Table 11: Compression Ratio of RNS-Huffman algorithms

Text File	Original Size (bits)	RNS-Huffman
1	2552	4.955339806
2	5152	8.445901639
3	7392	11.37230769
4	9904	14.22988506
5	12440	16.85636856
6	15056	19.94172185
7	17576	22.02506266
8	20352	23.80350877
9	24680	22.15439856
10	27280	24.18439716
11	29616	26.07042254
12	31880	27.6017316
13	35416	29.86172007
14	37968	32.3407155
15	40600	34.40677966

16	43224	36.88054608
17	45968	39.12170213
18	48576	41.16610169
19	51352	43.26200505
20	53968	44.93588676
21	56592	46.84768212
22	59160	49.67254408
23	61808	51.67892977
24	64552	53.88313856
25	67080	56.3697479
Average	**34805.76**	**31.28274181**

The average compression ratio for RNS-Huffman algorithms are **31.28274181** and the average text size of **34805.76.**

4.3.7 Comparative Analysis of RNS-Huffman algorithm in relation to seven other recent algorithms

Comparing the proposed work with the existing state of the art algorithms. The table 4.15 below presents the results of text files of existing state of the art algorithms with the proposed RNS-Huffman algorithms.

Table 12: Recent works in compression algorithms

S/N	Author(s)	Original Size	Compress Size	CR	Compression Algorithm
1	Alhassan et al., 2015	30.33333	21.33333	1.421875	LZW
2	Alhassan et al, 2015	30.33333	18	1.685185	LZW-RNS
3	Athira and Ravisankar 2020	5.8864	2.803	2.803	Delta Encoding
4	Ibrahim & Gbolagade (2019)		6200.333		
5	Ibrahim & Gbolagade (2019a) Huffman CRT		6041.333	3.695	
6	Ibrahim & Gbolagade (2019)b LPZ-CRT		6200.333		
7	Satrial et al. (2020) AD 6200.333 APTIVE		0.93045		
	Proposed Approach 1	34805.76	1023.52	31.28274181	RNS-Huffman
	Proposed Approach 2	34805.76	1420.76	22.31	RNS-Huffman

In the Table 12 above, Hasan et.al. (2013) obtained CR of 4.416, for Huffman based LZW and 2.587 for LZW based Huffman. Salunaz et.al. (2014) in Huffman with RLE 0.842556. Alhassan et. al. (2014) obtained 1.69 for LZW-RNS and Amandeep & Er.Meenakshi, (2014) obtained 2.08 in Dynamic bit reduction and Huffman algorithms. Gupta et.al. (2017) obtained 3.825 from DEFLATE, 5.88 from LZMA, 5.975 from BZIP2 and 7.088 for PPMONSTR. These are far lower than CR of proposed RNS-Huffman which is 22.31. This shows that the proposed RNS-Huffman

perform better than all the recent state of the art works in term of CR.

5.0 CONCLUSION

Both Huffman algorithm and RNS Huffman compression algorithm are lossless data compression techniques that can achieve high compression ratios.

Huffman algorithm is a general-purpose compression algorithm that works by assigning variable-length codes to each symbol in the input data based on their frequency of occurrence. It can achieve good compression ratios for text and other data with non-uniform frequency distributions.

RNS Huffman compression algorithm is a variant of Huffman algorithm that works by converting the input data into a residue number system (RNS) representation, which can be encoded using Huffman coding. This approach can be more efficient than standard Huffman coding for data with a large number of small values or a small range of values.

In general, the choice of which algorithm to use will depend on the characteristics of the input data. Both algorithms can achieve high compression ratios, but RNS Huffman may be more effective for certain types of data. Ultimately, the best way to determine which algorithm is most suitable for a particular application is to test them both on representative input data and compare their compression ratios.

REFERENCES

[1] Y. Lee, W. Lee, and S. Kim, "Image Compression Method Based on Huffman Coding and Discrete Cosine Transform," Journal of Digital Imaging, vol. 32, no. 3, pp. 470–477, 2019.

[2] S. S. Al-Sarhan, O. A. Basalamah, and A. H. Al-Makhadmeh, "Shannon-Fano Coding for Data Compression in Wireless Sensor Networks," Journal of Communications, vol. 14, no. 2, pp. 73–79, 2019.

[3] J. Kim, Y. Lee, and S. Kim, "Lossy Image Compression Method Based on Arithmetic Coding and Discrete Cosine Transform," Journal of Visual Communication and Image Representation, vol. 71, pp. 101–109, 2019.

[4] Lawal, T. D., Olatunbosun L. O. and Gbolagade K A. (2021): An Improve Shannon Fano Data Compressio Algorithm using Residue Number System. Communications on Applied Electronics (CAE) – ISSN: 2394714 Foundation of Computer Science FCS, New York, USA Volume 7– No. 35, April 2021 – www.caeaccess.org 19