iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IV **Month of publication:** April 2025

DOI: https://doi.org/10.22214/ijraset.2025.69301

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IV Apr 2025- Available at www.ijraset.com

Implementing Huffman Coding for Data Compression

Sandeep Singh
Maharaja Surajmal Institute of Technology, New Delhi, 110045

Abstract: ***This paper provides a thorough comparison between the Quaternary Tree Structure and M-Gram Entropy Variable to Variable Coding variations of the traditional Huffman data compression algorithm. The main goal is to analyze the original Huffman algorithm's binary tree code structure and compare it with the quaternary tree structure used in quaternary tree compression. Furthermore, the paper explores the theoretical foundation and application of the novel M-Gram Entropy Variable to Variable Coding method. By closely examining encoding processes, decoding mechanisms, and compression effectiveness, this work seeks to clarify the unique features and comparative advantages of each technique. This work aims to offer useful insights for data compression researchers and practitioners by illuminating the trade-offs between compression ratio, computational complexity, and adaptation to various data kinds.***
Keywords: Huffman coding; lossless data compression, Binary Tree, Quaternary Tree, Tree structure

I. INTRODUCTION

In today's computing systems, data compression methods are essential for effectively using transmission bandwidth and storage resources. These methods make it possible to represent data in a more condensed form, which speeds up data transmission over networks and reduces the amount of space needed for storage. This research focuses on investigating one such data compression technique, called Huffman coding, which efficiently compresses information using binary trees.

The process of compressing a huge file into a smaller one is known as compression. In order to make it easier to transfer a file that is enormous in size and has several personalities. The way a compression works is by searching the data for repeating patterns and replacing them with a specific sign [14].

In this paper, we used the binary tree approach to achieve notable file size reductions after using Huffman coding for file compression. In order to achieve compression, Huffman coding uses variable-length codes to represent input characters dependent on their frequency. Shorter codes are assigned to more frequently occurring characters. This method speeds up data transmission and retrieval while simultaneously lowering the amount of storage needed.

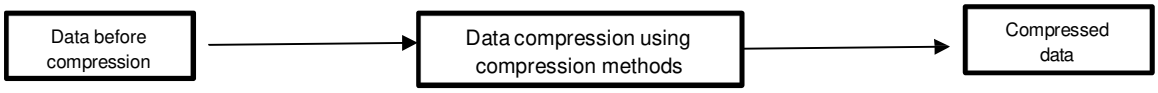


Fig 1. The process of data compression

The efficacy of Huffman coding has been widely investigated and acknowledged in the field of data compression. Huffman coding can achieve compression ratios that are nearly equal to the theoretical limit by taking advantage of the statistical characteristics of the input data. This method has been widely applied in many other contexts, such as text and image compression, where it has proven to be remarkably successful in lowering file sizes without compromising data integrity.

Developed in 1952 by David A. Huffman, Huffman coding is a greedy technique that builds a binary tree iteratively using the frequencies of the input symbols to create an ideal prefix-free code [6-7]. Shorter code words are assigned to more common symbols as the encoding process moves along the tree, producing a condensed representation of the original data. In contrast, the decoding procedure ensures lossless compression and decompression by reconstructing the original data using the same tree structure.

The statistical characteristics of the input data directly affect the efficacy of Huffman coding.

Because lossy compression techniques are utilized in multimedia and image data, where a little amount of information loss is acceptable, they were the main focus of earlier data compression models. This research, on the other hand, focuses on text data compression via the Huffman coding technique. A lossless compression method called Huffman coding makes guarantee that the compressed data can be precisely restored to the original.

With the use of binary trees, this approach implements Huffman coding and achieves notable gains in compression ratio over earlier models. Although the quaternary tree structure is examined and explored in this study as a different technique, its practical implementation is not sought after because of its increased complexity. Because of its effectiveness and ease of use, the binary tree approach is recommended for text data compression. This research delivers a higher compression ratio than previous models, which frequently produced lower compression ratios, by concentrating on binary trees. Binary trees are the best option for text data compression because of their speed and simplicity in Huffman coding, which ensures maximal compression without sacrificing the integrity of the original data. This method shows that, in contrast to earlier models that were largely created for lossy compression in multimedia applications, concentrating on text data might result in more effective and efficient compression approaches.

The rest of the paper is organized as follows. Section 2 gives us an overview of what kind of problems can be addressed by this research. Section 3 describes the methodologies used in this project. Section 4 tells us the results and outcomes we got from this research. Section 5 gives us a conclusion of the paper.

II. PROBLEM STATEMENT

In order to efficiently compress files containing text, we intend to implement the Huffman coding technique in this paper using a conventional binary tree approach. The main objective is to create a program that can compress text files by giving characters variable-length codes according to how often they appear.

Finding the frequency of each character in the input text files is the first step in the procedure. We build a Huffman tree a binary tree structure where letters are represented as leaf nodes and inside nodes represent merged characters with combined frequencies using this frequency information. By doing; this, compression is maximized and more often occurring characters are given shorter codes.

In order to efficiently compress text files, we intend to implement the Huffman coding technique in this paper using a conventional binary tree approach. The main objective is to create a program that can compress text files by giving characters variable-length codes according to how often they appear.

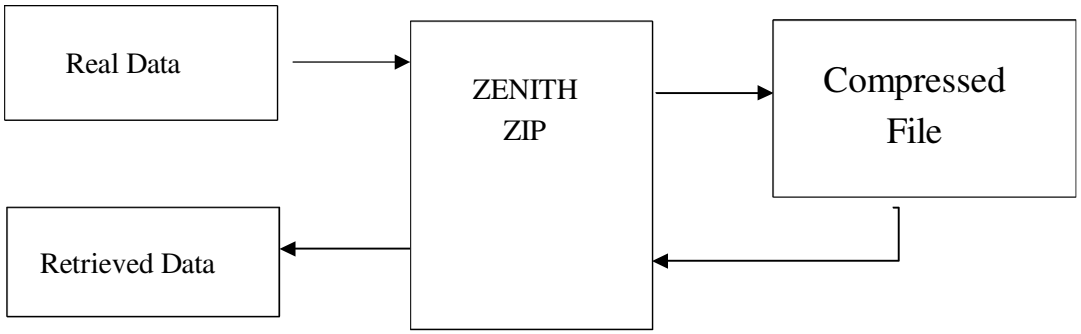


Fig 2. ZenithZip Working

We create Huffman codes for every character by going through the Huffman tree from the root to the corresponding leaf nodes after it has been constructed. After that, the input text file is compressed using these codes, which swap out the original character representations for their matching Huffman codes. Reversibility is guaranteed by implementing an effective decompression technique.

III. DATA COMPRESSION

In computer science and information theory, data compression is a fundamental technique used to reduce the size of data streams or files. Data compression primarily aims to store or transfer data more quickly over networks, either by lowering the amount of storage space needed or by increasing data transmission speed [8-12]. Through this procedure, the original material is encoded in a more condensed manner while maintaining all of its crucial information. Lossless and lossy compression are the two primary categories of data compression. Lossless compression is a technique that allows for a flawless reconstruction of the original data from the compressed version. Stated differently, no data is lost in the process of compression. For text files, executable programs, and other material where maintaining every detail is crucial, this kind of compression is usually utilized. Lossless compression algorithms work by locating and removing redundant information from the data, such as bits that are left unutilized or in repeating patterns.

Huffman coding is one of the most widely used lossless compression algorithms. Based on the frequency of input symbols (like characters in a text file) in the data, Huffman coding applies variable-length codes to them.
Shorter codes are allocated to more common symbols, whereas longer codes are assigned to less common symbols. In order to achieve the best compression ratios, this technique takes advantage of the statistical characteristics of the input data [10].
Lempel-Ziv-Welch (LZW) compression is another well-liked lossless compression method; it is utilized in formats such as GIF and the UNIX compress tool. LZW compression reduces data redundancy by substituting references to previously encountered sequences for repetitive data sequences. Lossy Compression: In contrast, lossy compression permits a certain amount of data loss throughout the compression procedure [10]. Multimedia data, including images, music, and video, are frequently compressed using this kind of technique, where a small quality loss may be acceptable in exchange for larger compression ratios. Lossy compression algorithms reduce the amount of data by eliminating information that is unnecessary or not as visually striking.
Discrete cosine transform (DCT) is one of the most well-known lossy compression techniques; it is used to compress images and audio in formats like JPEG and MP3. With little effect on perceived quality, the DCT converts spatial domain data pixels in the case of images into the frequency domain, where less significant high-frequency components can be quantized or eliminated.

A. Huffman Coding Using Binary Tree

One of the main contributions to the field of lossless data compression is Huffman coding, which was created by David A. Huffman in 1952. The fundamental idea behind it is to give input symbols variable-length codes according on how frequently those symbols occur in the data [14]. By assigning longer codes to less common symbols and shorter codes to more often occurring ones, the method reduces the average number of bits needed to represent the data, hence optimizing compression [1-5]. There are various processes involved in the Huffman coding process. First, each symbol's frequency in the input data is examined. After that, a binary tree called the Huffman tree or Huffman encoding tree is created. In order to create a binary tree with symbols at the leaf nodes, the two symbols with the lowest frequencies are combined into a single parent node iteratively. Each symbol is given a Huffman code after the tree is constructed, which is determined by the path that connects the root node to the matching leaf node. Because these codes don't contain prefixes, decoding is clear. The input data is changed to its matching Huffman codes during the encoding stage, producing a more condensed representation. On the other hand, decoding entails putting the Huffman tree back together and going through the encoded material to find the original symbols. The generation of appropriate prefix codes, flexibility in responding to shifting input data distributions, and comparatively easy implementation are only a few benefits of Huffman coding.

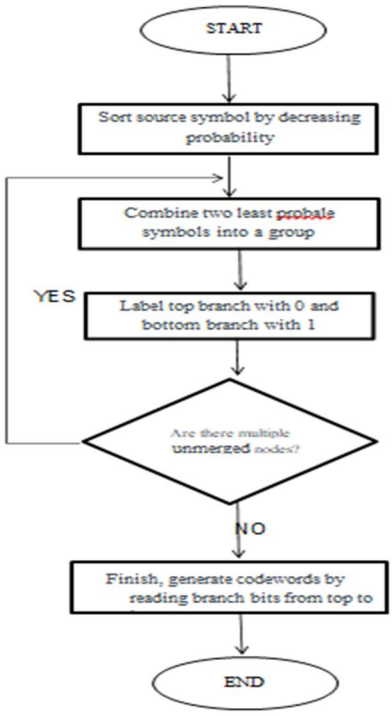


Fig.3 Flowchart of Huffman Tree

Algorithm: Construction of Huffman Tree Data: Word frequencies F: {(w_i, f_i), ...},
Priority queue H: {($node_i$, $score_i$), ...}, sorted by increasing scores, Number of symbols: n
Result: Huffman tree
foreach (w_i, f_i) ∈Fdo
Create node$_i$ with key w_i and score f_i ; Add $node_i$ to H;
end
while$length(H)$> 1 do
L ← empty list of nodes;
S ← 0;
for$i$ ← $0$to$n$do if$H$ = ∅then
break;

else
end end

Pop ($node_i$, $score_i$) from H; Append ($node_i$, $score_i$) to L; Add $score_i$ to S;
Create new node N = ('None', S);
foreach$node$∈ L do
Add $node$ to N's children;
end
Push N to H;
end

B. Huffman Coding Using Quaternary Tree

Binary with varying length in most cases, Huffman coding makes it difficult to find a balance between memory usage and speed. In this case, a quaternary tree is used to get the perfect code word that speeds up the search. This section also covers the details of a few more tree topologies that might offer the code word for data compression. The structure and algorithm of binary and quaternary trees are explained in this section.

A rooted tree is referred to be an m-ary tree if each of its internal vertices has precisely m children. A tree is said to be a full m-ary tree if each internal vertex has exactly m children. When m = 2, a binary tree is an m-ary tree. The rooted tree is said to be ordered when all of its children are arranged. On ordered rooted trees, the progeny of every internal vertex is shown from left to right. When two children are born to an internal vertex of an ordered binary tree (sometimes called just a binary tree), the first child is called the left child and the second kid is called the right child. The tree rooted at the left child (or right child, resp.) of a vertex is its left subtree (or right subtree, resp.) [15].

Ternary trees require more time to traverse than quaternary trees as a result. The current approach creates dictionary code-words for data compression using modified Huffman coding. Whereas the quaternary tree has at most four subtrees and requires at least two bits for a single level of traversing, the standard Huffman tree has at most two subtrees and produces a single bit for a single level of traveling.

A tree T is a connected undirected acyclic graph. It has vertices $V = \{v_0, v_1\ldots\ldots, v_{n-1}\}$ and a set of edges as $E = \{ e_0, e_1\ldots\ldots, e_{n-1}\}$. V is referred to be u's child if u is v's parent. Children who share a parent are referred to as siblings. If a tree's vertex has no children, it is referred to be a leaf.

An internal vertex always has one or more children. The tree T has the unique node R, which is frequently referred to as the root of T. If every vertex in a tree T has two offspring or less, the tree is called a binary tree. A tree T is said to be quaternary if it has, at most, four children with the names LEFT, LEFTMID, RIGHT-MID, and RIGHT [15].

When every one of a tree T's four internal vertex children is present, the tree is referred to as a full quaternary tree. The tree architectures that are binary and quaternary are as shown in (Figs. 4 and 5) correspondingly.

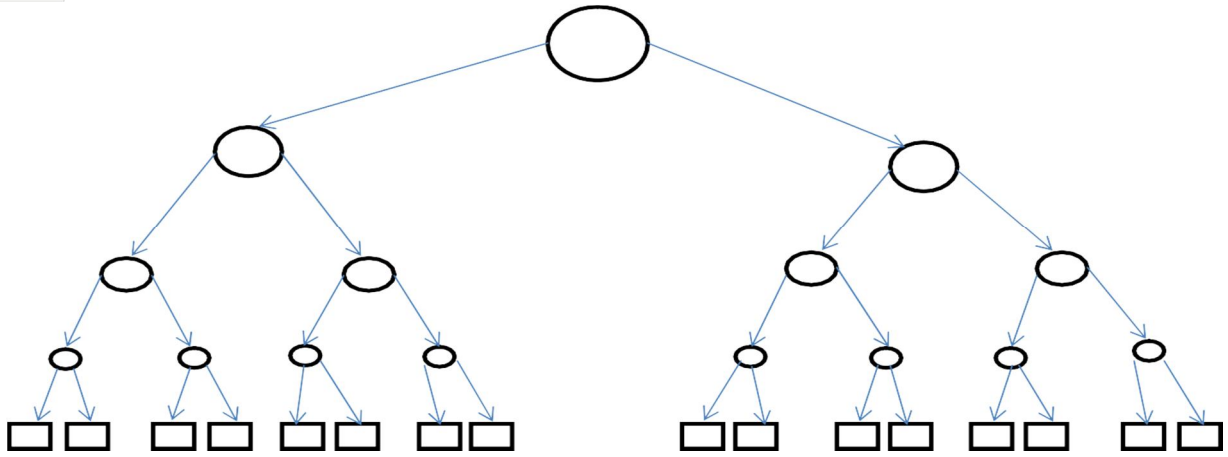


Fig4.Binary Tree

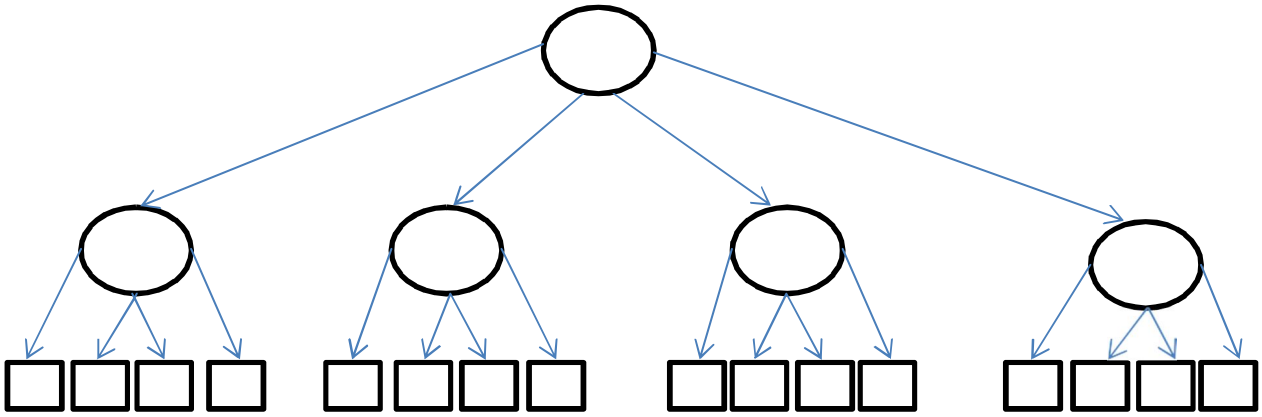


Fig5.Quaternary Tree

Algorithm: Quaternary Huffman Tree Encoding

Q- HUFFMAN (C)

1. Q ← C
2. n ← |Q|
3. i ← n WHILE i > 1
4. allocate a new node z
5. left[z] ← v ← EXTRACT-MIN(Q)
6. left-mid[z] ← w ← EXTRACT-MIN(Q)
7. IF i = 2
8. f[z] ← f[v] + f[w]
9. ELSE IF i = 3
10. right-mid[z] ← x ← EXTRACT-MIN(Q)
11. f[z] ← f[v] + f[w] + f[x]
12. ELSE
13. right-mid[z] ← x ← EXTRACT-MIN(Q)
14. right[z] ← y ← EXTRACT-MIN(Q)
15. f[z] ← f[v] + f[w] + f[x] + f[y]
16. END IF
17. INSERT(Q, z)
18. i ← |Q|

19. END WHILE
20. RETURN EXTRACT-MIN(Q)

IV. RESULT AND DISCUSSION

This paper is centered around using the binary tree method to apply Huffman coding as a data compression solution. We have effectively illustrated the effectiveness of Huffman coding in lowering file sizes while maintaining data integrity through this implementation. We began the effort by thoroughly researching the fundamental ideas of Huffman coding. Next, we used a binary tree data structure to build the Huffman coding technique.

The frequency analysis of the input data, the creation of the Huffman tree, the assignment of Huffman codes to symbols, and the encoding of the data using the produced codes were all crucial phases in this implementation. When compared to their uncompressed counterparts, the compressed files that were produced showed notable size reductions.

The system has several noteworthy advantages, including efficiency and versatility. This approach guarantees the best compression ratios for different kinds of data by dynamically modifying Huffman codes according to the frequency distribution of input symbols. Furthermore, this method is applicable to a wide range of applications across several sectors due to the simplicity of the Huffman coding technique and the efficacy of the binary tree-based implementation.

We conducted trials using real-world data sets from multiple sources, such as written documents to demonstrate the efficacy of this method. These tests' outcomes often
showed significant file size reductions with hardly any loss of data quality. Furthermore, the method demonstrated computational efficiency, enabling quick data compression and decompression.

Apart from its compression powers, this technology is versatile when it comes to integrating with current applications and systems. A wider range of developers and
academics can use Huffman coding since it is easily adaptable to various computer languages and platforms when implemented using a binary tree structure.

The two images are attached to demonstrate the success of this approach visually: one is a sample input file (normal file, Figure 6) before to compression, and the other is the
corresponding compressed file (encoded file, Fig 7). These pictures demonstrate the
substantial decrease in file size that the Huffman coding-based compression technique was able to accomplish.

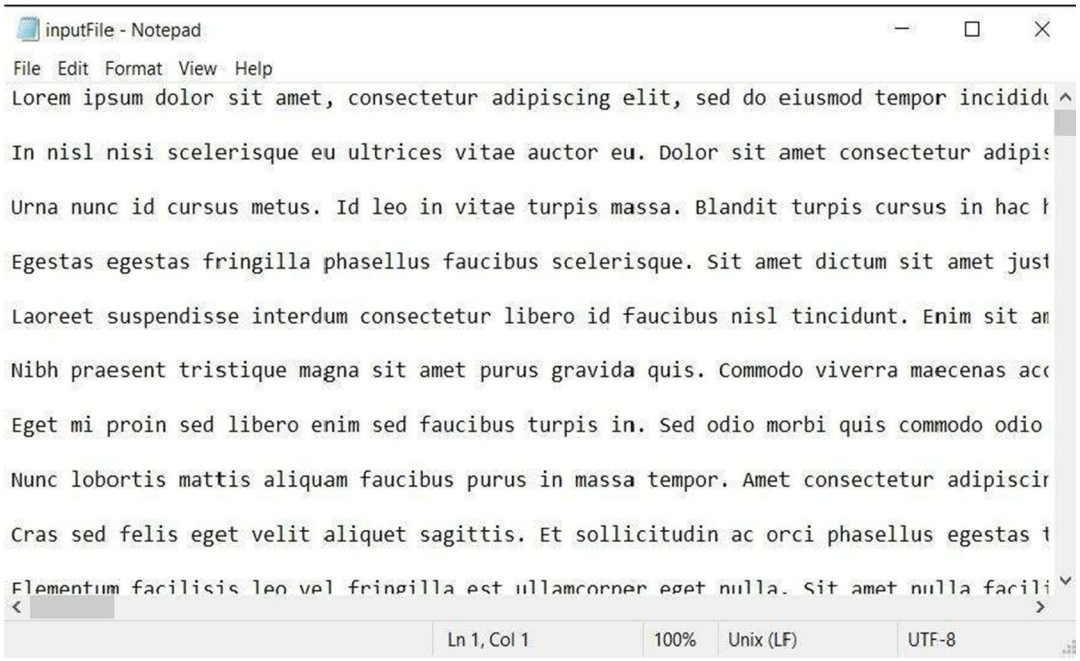


Fig 6. Normal File

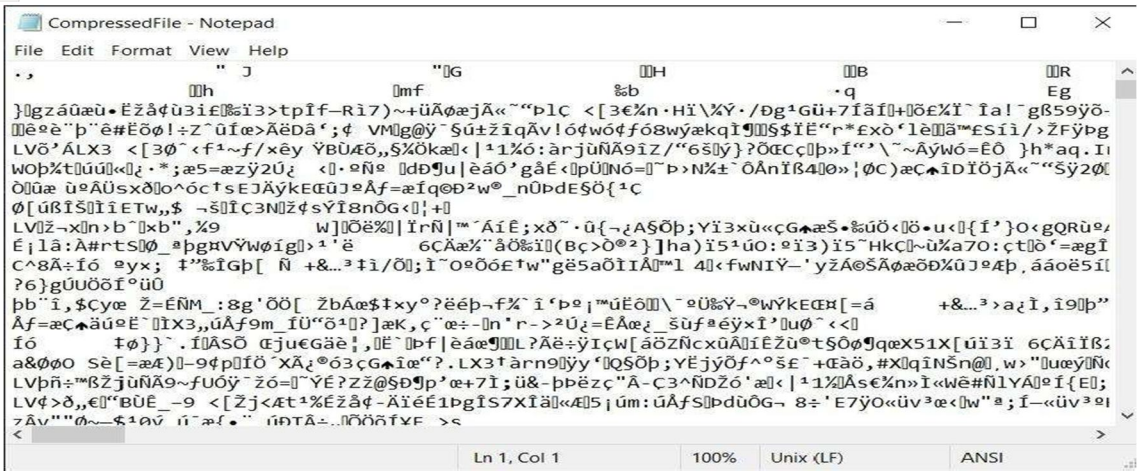


Fig 7. Encoded File

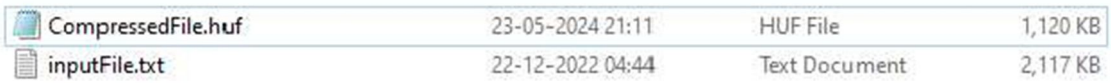


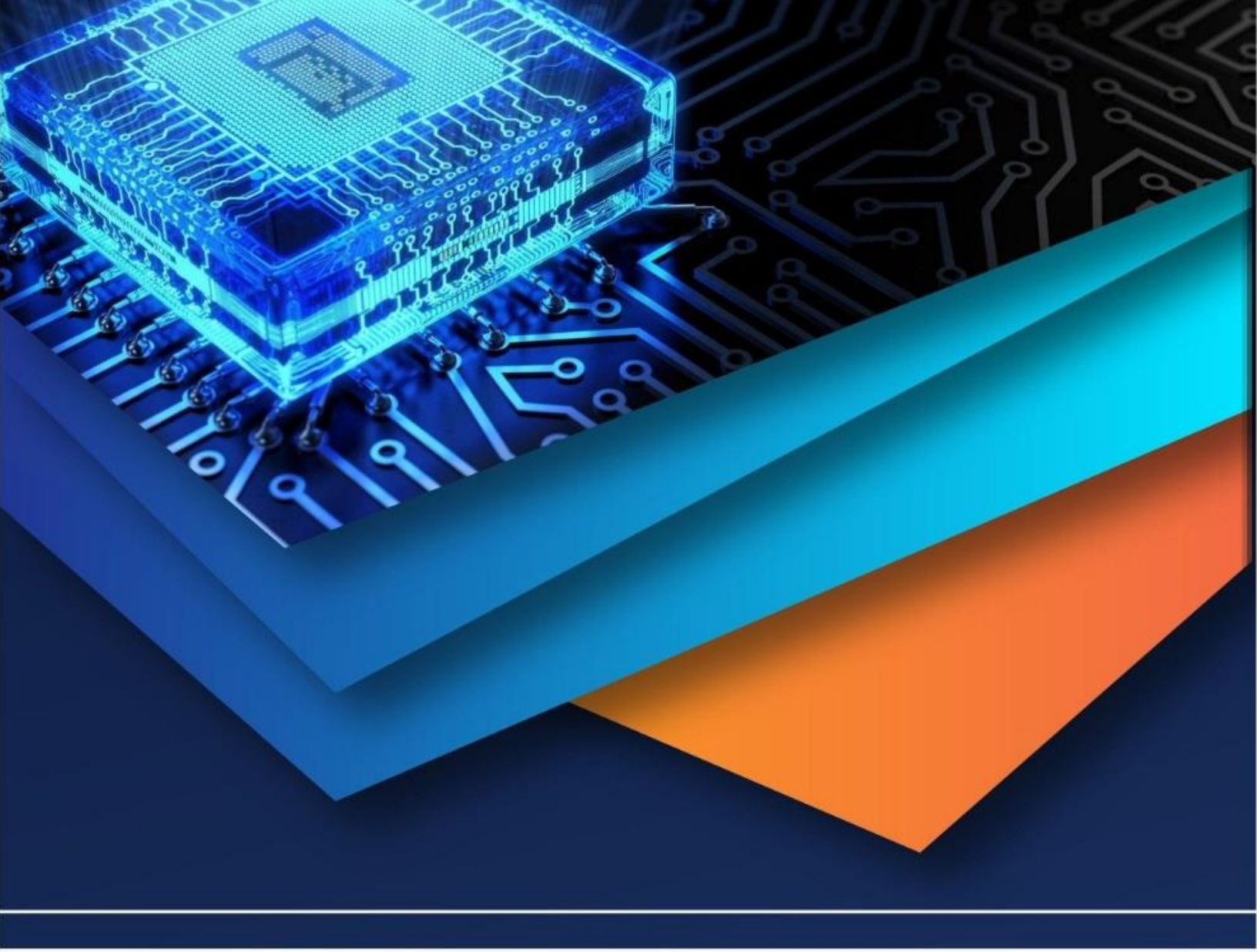
Fig 8. Original and Compressed File size comparison

V. CONCLUSION

To sum up, this research on Huffman coding data compression has shed light on the effectiveness and adaptability of this essential compression method. We have shown through the implementation that Huffman coding may drastically reduce file sizes while maintaining data integrity, making it a practical option for a range of applications that need effective data transmission and storage. This investigation into Huffman coding has demonstrated its flexibility and effectiveness, especially when applied with a binary tree technique.

REFERENCES

[1] Malik, N. Goyat and V. Saroha, "Greedy Algorithm: Huffman Algorithm," International Journal of Advanced Research in Computer Science and Software Engineering, vol. 3, no. 7, pp. 296-303, 2013.
[2] A. S. Sidhu and M. Garg, "Research Paper on Text Data Compression Algorithm using Hybrid Approach," IJCSMC, vol. 3, no. 12, pp. 1-10, 2014.
[3] H. Al-Bahadili and S. M. Hussain, "A Bit-level Text Compression Scheme Based on the ACW Algorithm," International Journal of Automation and Computing, pp. 123- 131, 2010.
[4] I. Akman, H. Bayindir, S. Ozleme, Z. Akin and a. S. Misra, "Lossless Text Compression Technique Using Syllable Based Morphology," International Arab Journal of Information Technology, vol. 8, no. 1, pp. 66-74, 2011.
[5] M. Schindler, "Practical Huffman coding," 1998. [Online]. Available: http://www.compressconsult.com/huffman/.
[6] R.S. Brar and B. Singh, "A survey on different compression techniques and bit reduction Algorithm for compression of text data" International Journal of Advanced Research In Computer Science and Software Engineering (IJARCSSE) Volume 3, Issue 3, March 2013
[7] S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data Compression Methodologies for Lossless Data and Comparison between Algorithms" International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 2, March 2013
[8] S. Shanmugasundaram and R. Lourdusamy, "A Comparative Paper of Text Compression Algorithms" International Journal of Wisdom Based Computing, Vol.1 (3), Dec 2011
[9] S. Kapoor and A. Chopra, "A Review of Lempel Ziv Compression Techniques" IJCST Vol.4,
[10] Issue 2, April-June 2013
[11] S.R. Kodituwakku and U. S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data "Indian Journal of Computer Science & Engineering Vol 1 No 4
[12] R. Kaur and M. Goyal, "An Algorithm for Lossless Text Data Compression" International Journal of Engineering Research & Technology (IJERT), Vol. 2 Issue 7, July - 2013
[13] H. Altarawneh and M. Altarawneh, "Data Compression Techniques on Text Files: A Comparison Paper" International Journal of Computer Applications, Vol 26– No.5, and July 2011
[14] U. Khurana and A. Koul, "Text Compression and Superfast Searching" Thapar Institute of Engineering and Technology, Patiala, Punjab, India-147002
[15] Tito Waluyo Purboyo and Anggunmeka Luhur Prasasti, "A review of data compression techniques," International Journal of Applied Engineering Research, vol. 12, no. 19, pp. 8956-8963, Jan. 2017.
[16] A. Habib, M. J. Islam, and M. S. Rahman, "Quaternary Tree Structure as a Novel Method for Huffman Coding Tree," J. Comput. Sci., vol. 19, no. 9, pp. 1132-1142, 2023. Available: https://doi.org/10.3844/jcssp.2023.1132.1142.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY