GE 103
Data Compression

Jignesh Agrawal[#1], Manik Gupta[#2], Dhananjay Goel[#3], Akarshi Roy Choudhury[#4]
[1]*2021EEB1181,* 2021eeb1181@iitrpr.ac.in
[2]*2021EEB1187,* 2021eeb1187@iitrpr.ac.in
[3]*2021EEB1165,* 2021eeb1165@iitrpr.ac.in
[4]*2021EEB1149,* 2021eeb1149@iitrpr.ac.in

***Abstract* – Data compression refers to the process of condensing data, in order to minimize transmission time and occupy minimum storage space. It can be classified as lossy and lossless. This project focuses on the execution and comparison of three different algorithms of lossless data compression, namely the naïve approach (our approach), the Huffman approach, and the Shannon-Fano approach. The logical inference asserting that Huffman coding is the most efficient technique; has been verified by comparing the comparison ratios of the three methods.**

***Keywords* – data compression, Shannon theory, Huffman coding, Shannon Fano algorithm, naïve approach**

I. INTRODUCTION

With the advancement of technology and breathtaking developments in software as well as hardware systems, a huge number of clusters of data are circulated over the internet each second. Although communicating information has become quite effortless and facile, however, it has given rise to the crucial issue of data storage. All information can't be easily transmitted over the network, majorly due to size constraints, thus, bringing about the need of making the data compact. The notion of data compression involves obtaining a very large packet of information from the sender, compressing it into a smaller chunk, followed by efficient transmission from one device to another, and finally decompressing the chunk, back into the larger packet, before being presented to the receiver. Another advantage of data compression is that it minimizes data redundancy to a great extent, thereby, saving storage space. To summarize, it can be said that data compression is basically the encryption of data into a lesser number of bits as compared to the original data file, in order to occupy lesser storage space as well as minimize transmission time.

II. LITERATURE REVIEW

This section of the report provides an insight into the historical research works based on Shannon's theory, types of compression, Shannon-Fano algorithm, and Huffman coding.

A. Shannon Theory

In statistical physics, entropy has been defined as the measure of randomness or chaos in a system. Shannon carried forward this idea of disorderliness and proposed a theory, which is widely used in implementing the concept of data compression. A system is assumed to exist in various states and at a particular instant, one can obtain a probability distribution over those states. Let S be the set of all possible states and p(s) be the probability of a given state s∈S. Then, the entropy of the system is given by the formulae:

$$H(S) = \sum_{s \in S} p(s)\, log_2 \frac{1}{p(s)}$$

If the probabilities are more biased, then the entropy will be higher. On the other hand, if the probabilities are more even, then the entropy will be lower. Shannon modified this notion by replacing 'states' with 'messages', and further defined the self-information i(s) of a message as the number of bits required to encode a 'message'.

$$i(s) = \; log_2 \frac{1}{p(s)}$$

Messages having lower probability tend to contain lesser information.
By analyzing the formulae, it can be stated that the greater the evenness in the distribution, the lesser the entropy of the system. This can further be extended to the notion that compression becomes easier with decreasing entropy.[1]

B. Types of Data Compression

Data compression can be classified into two categories:

1) Lossy Data Compression:

It involves the reduction of the number of bits by removing trivial information, thus reducing the file quality. This method is undertaken while compressing multimedia files such as images, audio, and video. JPEG and MP3 formats are examples of lossy data compression.[2]

2) Lossless Data Compression:

In this type, no data is lost during compression. The redundant data is encoded and then decompressed to obtain the exact original data. It is used to compress files, in which the loss of even a single bit of data isn't affordable, for instance, in text files, spreadsheets, etc. ZIP format is a well-known example of a lossless data compression algorithm. [3]

C. Prefix Codes

The codes in which no assigned code is a prefix for another assigned code are called prefix codes. For example, {a = 1, b = 001, c= 01, d =000}. Every character encrypted by prefix code can be uniquely decrypted. [4]

D. Shannon Fano Algorithm

Used for lossless data compression, this algorithm is a variable-length coding technique, which is based on code generation for each character depending on their frequency in data. Firstly, the probability distribution of characters is tabulated in a sorted manner. Post sorting, the table is divided into two halves such that the difference between the sum of occurrences of both the halves is the least. Then, the left half is assigned 0, while the right half is assigned 1, or vice versa. This procedure is repeated until prefix codes for individual characters are obtained. [8]

E. Huffman Coding

It is one of the most efficient variable length coding techniques used for carrying out lossless data compression. The fundamental of the Huffman algorithm lies in the statistical data for the frequency distribution of characters in the English alphabet. Based on this, the layout of a 'binary' tree is built, in which the characters with lower frequencies are placed near the roots of the tree, whereas the ones with higher frequencies are placed in the vicinity of the apex, the highest one being the tree-generator. Initially, the two characters with the lowest frequencies are placed at the root level, followed by their summation, and then the adjacent positioning of the character with a slightly higher frequency. This process continues, till the one with the highest frequency gets placed at the top. The left and right branches are then labeled as '0' and '1' respectively or even vice versa. The binary prefix code for each character is then generated by traversing along the branches, which finally culminate at the character itself. The total number of bits of encoded file can then be calculated by summation of products of frequencies and prefix codes. [6] Each ASCII character is 8 bits long. The Huffman method facilitates the allocation of shorter binary codes to all the letters, thus minimizing redundancy and successfully compressing the given data to a large extent. [7]

F. Compression Ratio

It is the ratio of the difference between uncompressed file size and compressed file size to the original, i.e., uncompressed file size. [8]

III. OBJECTIVE

Our project aims at compressing text files in a lossless manner – first by a self-designed naïve approach, then by the Shannon Fano approach, and then lastly by the popular Huffman approach. Further, we showcase a comparison between the three and prove that the most efficient method is Huffman coding.

A. Naïve Approach

In this approach, words with maximum frequencies are assigned strings such as 'AA', 'BB', etc., which are tabulated in a descending order beforehand. Consequently, the size of all such words boils down to merely 2 characters. So, if the word is of 5 letters (say), then earlier, the storage space occupied would have been 40 bits times its frequency, however, now, it reduces to 16 bits times the frequency. The reason for encrypting the words in the form of letter doublets is that, usually, words in English do not begin with such strings, so the possibility of overlap is negligible, and our encrypted file can easily be decrypted. In our code, we have encoded the 10 words having maximum frequencies i.e., the codes range from 'AA' to 'JJ'. To sum up, the naïve approach basically compresses files by reducing the 'word' length.

B. *Shannon Fano's approach*

The process undertaken is explained through a flowchart as follows:
Frequency distribution dictionary of characters is found → Sorting the dictionary (min to max) → Reversing the sorted dictionary (max to min) → Dividing it into two halves such that there is the least difference between the sum of elements of the two halves → Assigning '0' to the left half and '1' to the right half → Repeating the process for each half recursively
This gives the codewords for each character in the end that are used to encrypt the data and compress it.

C. *Huffman Approach*

The approach is explained below using a flow chart:
Frequency distribution dictionary of characters is found → Sorting the dictionary → Constructing Huffman tree → Encrypting file using the codewords obtained from Huffman tree → Comparing encrypted file with the original file → Decrypting the encrypted (Huffman) file
The file size obtained will certainly be smaller than the original one, owing to the fact that larger frequencies were multiplied with smaller binary codes and likewise, smaller frequencies with larger binary codes.

D. *Why Huffman approach is considered the most optimal solution to data compression?*

Huffman's approach is the most optimal solution as the length of the code of a character is directly proportional to the probability of its occurrence in the text and thus it encodes all the characters despite taking the least space needed. As Prefix codes are given to the characters, there is no chance of ambiguity while decoding and we obtain the file after encryption as it is without any data loss.

IV. CONCLUSIONS

The compression techniques are run on three types of files, and then the compression ratios are calculated for each case. A comparison between all the cases leads to the determination of the most efficient method.

A. File 1 – The Adventures of Sherlock Holmes

The original file size was 50638416 bits. The data obtained post-compression is as follows:

1) *Naïve Approach* – The compressed file size is 49174960 bits and the compression ratio is 2.89.
2) *Shannon Fano Approach* – The compressed file size is 28818847 bits and the compression ratio is 43.09.
3) *Huffman Approach* – The compressed file size is 28760793 bits and the compression ratio is 43.2.

B. File 2 – Randomly Generated File containing random letters with very low entropy (unevenness)

The original file size was 8000000 bits. The data obtained post-compression is as follows:

1) *Naïve Approach* – The compressed file size is 7974520 bits and the compression ratio is 0.3185.
2) *Shannon Fano Approach* – The compressed file size is 6699396 bits and compression ratio is 16.25755.
3) *Huffman Approach* – The compressed file size is 6696475 bits and the compression ratio is 16.29406.

C. File 3 – Very Even File containing sequences of 'A', 'B' and ' '

The original file size was 8000000 bits. The data obtained post-compression is as follows:

1) *Naïve Approach* – The compressed file size is 3214296 bits and the compression ratio is 59.8213.
2) *Shannon Fano Approach* – The compressed file size is 1562500 bits and compression ratio is 80.46875.
3) *Huffman Approach* – The compressed file size is 1562500 bits and the compression ratio is 80.46875.

The naïve approach is found to work efficiently on a short piece of text, the reason being that only 10 words with the highest frequencies are encoded. The compression ratios of the Huffman approach and Shannon Fano approach are very close; however, the Huffman approach has a slight edge in every case.
Thus, the Huffman approach is the most efficient method of lossless data compression. The results are consistent with the logical deductions.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to our course instructor and mentor, Dr. Sudarshan Iyengar, and Mr. Sushil Ghildiyal for guiding us throughout the entire project. We are thankful to the computer department of the Indian Institute of Technology, Ropar, for providing us with the facilities of computer labs. We extend our gratitude to Mr. Ashu Kaushik and all the lab assistants and teaching assistants, for supporting us constantly.

REFERENCES

[1] https://www.cs.cmu.edu/~guyb/realworld/compression.pdf
[2] https://www.barracuda.com/glossary/data-compression
[3] https://teachcomputerscience.com/data-compression/
[4] https://home.cse.ust.hk/faculty/golin/COMP271Sp03/Notes/MyL17.pdf
[5] https://www.mbit.edu.in/wp-content/uploads/2020/05/data_compression.pdf
[6] https://ieeexplore.ieee.org/abstract/document/5706721
[7] https://www.researchgate.net/publication/322557949_A_review_of_data_compression_techniques
[8] http://www.ripublication.com/ijaer17/ijaerv12n23_80.pdf
[9] https://ejournal.unsrat.ac.id/index.php/decartesian/article/view/3207/2748