# DATA COMPRESSION

## I. INTRODUCTION

### What is Data Compression?

In information theory (IT), data compression or source coding reduction is known as the process of encoding information using fewer bits than the original representation. It enables sending a data object or file quickly over a network and in optimizing storage resources.

### Invented

With the advent of information theory in the late 1940s, modern data compression research got underway. In 1949, Claude Shannon and Robert Fano created a methodical methodology for allocating code-words based on block probability. David Huffman later discovered an ideal approach for doing this in 1951

### Types of Data Compression

- Lossless
  All of the original data is present in lossless form. The algorithm compresses the file while preserving the data necessary to restore it to its original size when decompressed. For files that would be significantly damaged or unable to function without all of the original data, a lossless format is required. These files contain software programmers, papers, and specific media types utilised by a variety of creators, including musicians, filmmakers, and photographers.
- Lossy
  Even further file size reduction is possible with lossy compression, although there will be a little information loss. For file types where missing information is barely noticeable, this format is appropriate. Media downloaded by the user, such as songs, videos, and photos, is included in these files. There is a slight drop in playback quality for these, but the user is unlikely to notice.

### How Data Compression Work?

Depending on the kind of data you wish to compress, there are typically four different forms of compression. Which are:

- Text:
  To reduce the size of text files, redundancies and patterns must be found in the content and encoded using shorter codes or symbols. The type of text and necessary compression ratio determines how well a compression algorithm works.
- Picture:
  Similar to text compression, picture compression looks for patterns and duplication in the file and shortens the codes and symbols. For instance, it might detect a pattern of colours that repeats itself and generate a code using the colour and the number of instances that equals one unit of the original.
- Audio:
  Audio files on computers allow you to listen to music, but they hardly ever include other kinds of information. Utilising lossy techniques, which eliminate both background noise and white noise from the files, is the main approach for compressing audio recordings.
- Video:
  A particular procedure is required to compress video files because they may also contain audio and image files. Videos employ both audio and image data, therefore background noise must be removed while preserving the most crucial portions of the movie using a combination of lossy techniques.

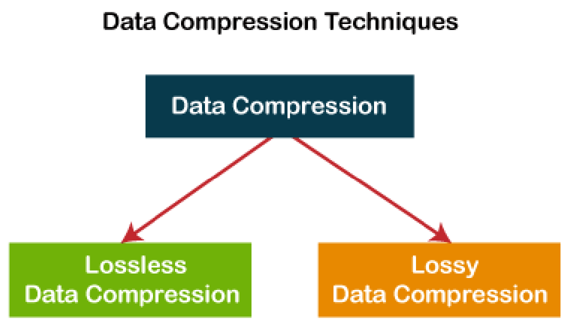


Fig. 1

## II. HUFFMAN CODING

*Huffman coding is an algorithm utilized for data compression and serves as the fundamental principle behind file compression. we then shall discuss the generalities of fixed-length and variable-length encoding, uniquely decodable codes, prefix rules, and the structure of a Huffman Tree.*

### History

In 1952, MIT graduate David Albert Huffman and his classmates in the information theory course were faced with a decision between writing a term paper or taking a final exam. Their professor, Robert M. Fano, assigned them a term paper focused on the problem of discovering the most effective binary code. Initially, Huffman struggled to demonstrate the efficiency of any codes and contemplated giving up to concentrate on studying for the final exam. However, he

had a breakthrough idea: utilizing a frequency-sorted binary tree. Through quick experimentation, Huffman successfully proved that this method was the most efficient approach.
Huffman's achievement surpassed that of Fano, who had collaborated with Claude Shannon to develop a similar coding technique. Huffman made sure the tree was optimal by building it from the ground up, which stands out in contrast to Shannon-Fano Coding's top-down method.
Introduction Traditionally, characters are represented as sequences of 0s and 1s and stored using 8 bits, which is referred to as "fixed-length encoding." Each character occupies the same number of fixed bits, resulting in a consistent storage size.

## The Challenge

Reducing Storage Space To minimize the space required for character storage, we can employ "variable-length encoding." This technique leverages the varying frequencies of characters within a given text to design an algorithm that represents the text using fewer bits. In variable-length encoding, characters are assigned a variable number of bits based on their frequency in the text. Consequently, some characters may be encoded with a single bit, while others might require two, three, or more bits. However, the challenge with variable-length encoding lies in decoding the encoded data.

## The Decoding Dilemma

*Ensuring Uniqueness:*

*:* Consider the string "aabacdab" comprising eight characters and utilizing 64 bits of storage using fixed-length encoding. If we examine the frequency of characters in this string (a: 4, b: 2, c: 1, d: 1), we can attempt to represent it using fewer bits by capitalizing on the fact that the character 'a' occurs more frequently than 'b,' and 'b' occurs more frequently than 'c' and 'd'.
Let's arbitrarily assign the following codes:
a: 0 b: 11 c: 100 d: 011
Therefore, using the previously stated codes, the string "aabacdab" will be encoded as 00110100011011 (0|0|11|0|100|011|0|11).
However, the real challenge lies in decoding this encoded string. Decoding "00110100011011" can lead to ambiguity, resulting in multiple possible interpretations:
0|011|0|100|011|0|11 $\rightarrow$ adacdab
0|0|11|0|100|0|11|011 $\rightarrow$ aabacabd
0|011|0|100|0|11|0|11 $\rightarrow$ adacabab ...
To prevent decoding ambiguities, we need to ensure that our encoding adheres to the "prefix rule," thereby creating "uniquely decodable codes." The prefix rule stipulates that no code should be a prefix of another code. In the previous example, the code 0 is a prefix of 011, violating the prefix rule. If we satisfy the prefix rule, decoding will be distinct.
Let's reconsider the previous example. This time, we will assign codes that comply with the prefix rule for characters 'a,' 'b,' 'c,' and 'd':
a $\rightarrow$ 0 ,b $\rightarrow$10 ,c $\rightarrow$110, d $\rightarrow$111 Using these updated codes, the string "aabacdab" can be encoded as 00100110111010 (0|0|10|0|110|111|0|10). we can then uniquely encode 00100110111010 back to the original string, "aabacdab."
Now that we have comprehended variable-length encoding and the prefix rule, let's delve into Huffman coding.
Huffman Coding the technique involves constructing a binary tree composed of nodes, which can either be leaf nodes or internal nodes. At first, all nodes are leaf nodes that store the character and its weight (frequency of appearance).

Huffman coding can be done with the following steps:

- 1. Calculating the frequency of every character in the string.Frequency of string

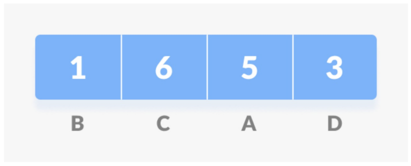


Fig. 2

- 2. Sorting the characters in increasing order of frequency. These thus are stored in a priority queue Q.(Characters sorted according to the frequency)

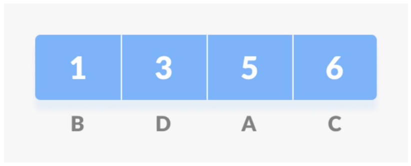


Fig. 3

- 3. Making every unique character a leaf node.
- 4. Creating an empty node $z$. Assigning the minimum frequency to the left division of $z$ and assigning the second minimum frequency to the right division of $z$. value of $z$ should be set as the sum of two minimum frequencies. Getting the sum of the least numbers

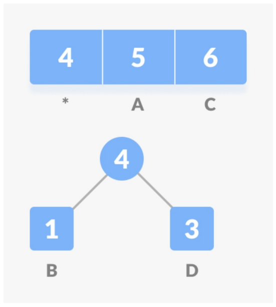


Fig. 4

- 5. Removing the two lowest frequencies from Q and adding the total to the list of frequencies (the internal nodes in the previous diagram are indicated by the asterisks *).

- 6. Inserting node $z$ into tree.
- 7. Repeat steps 3 to 5 for all the characters.

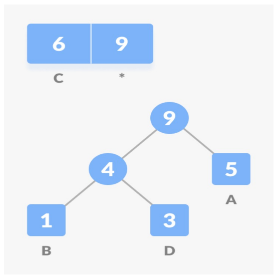


Fig. 5

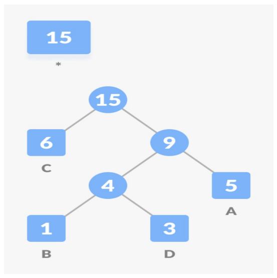


Fig. 6

- 8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge. Assign 0 to the left edge and 1 to the right edge

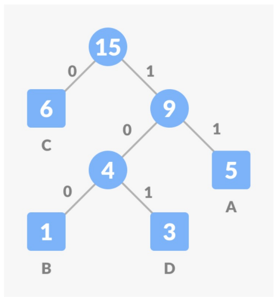
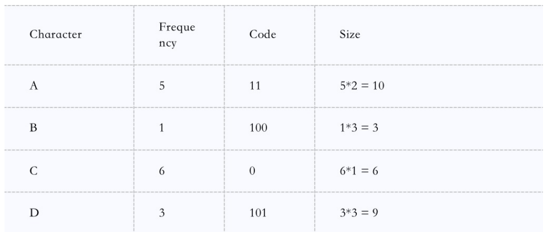


Fig. 7

- For sending the above string over a network, we have to send the tree as well as the above-compressed code. The total size is given in the table below.
- Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to 32 + 15 + 28 = 75.



| Character | Freque ncy | Code | Size |
|---|---|---|---|
| A | 5 | 11 | 5*2 = 10 |
| B | 1 | 100 | 1*3 = 3 |
| C | 6 | 0 | 6*1 = 6 |
| D | 3 | 101 | 3*3 = 9 |

Fig. 8

## III. RUN-LENGTH ENCODING

*Run-length encoding is a compression method in which strings exhibiting redundant data are stored as a single data value representing a repeated block and the number of times it appears in the image. The image can then be rebuilt from this data during the decompression process.*

*This compression type is best for simple graphics and animations with lots of unnecessary pixels. It works especially well with black-and-white images.*

**Drawbacks**

- 1. To access anything we have to first decode the data to get the original data as the original data isn't accessible immediately.
- 2. The size of the data we get after decoding cannot be predicted at earlier stages, which causes problems if we have limited storage capacity.

**Advantages**

- 1. It's easy to implement and can be applied to various data types.
- 2. It's very fast and can be used in real-life applications. Time complexity : O(n)

**Program to implement Run-length encoding**

- Input:

```
# include <bits/stdc++.h>
using namespace std;
void printRLE(string str){
int n = str.length();
for (int i = 0; i < n; i++) {
// Count occurrences of current character
int count = 1;
while (i < n - 1  str[i] == str[i + 1]) {
count++;
i++; }
// Print character and its count
cout « str[i] « count;
}
}
//Driver code
int main()  {
string str = "wwwwaaadexxxxxxywww";
printRLE(str);
return 0;
}
```

- Output: w4a3d1e1x6y1w3

## IV. ARITHMETIC CODING

*Arithmetic coding is a lossless data compression technique that maps a sequence of symbols to a single fractional number in the interval [0,1], which can then be represented using fewer bits than the original symbols, thus reducing the overall file size.*

*A fundamental concept in arithmetic coding, probability distributions determine the likelihood of a symbol appearing in the data stream and are used to calculate cumulative probabilities for encoding.*

### Implementation of Arithmetic Coding

*Encoding and Decoding Processes:*

- Step 1: Symbol Probability Calculation
  The frequency of occurrence for each symbol in the input message is calculated and their corresponding probabilities are determined.
- Step 2: Cumulative Probability Calculation
  The cumulative probability distribution is calculated for each symbol based on the symbol probabilities.
- Step 3: Interval Mapping
  The input message is mapped to a single interval based on the cumulative probability distribution of the symbols.
- Step 4: Re-normalization
  The interval is re-normalized after each symbol is encoded or decoded to maintain the accuracy of the probability distribution.

### Image Compression Techniques

Let us consider a three letter alphabet A $=a_1,a_2,a_3$ probabilities $P(a_1) = 0.4$, $P(a_2) = 0.3$. $P(a_3) = 0.3$ The initial sub-intervals are [0.0, 0.4), [0.4, 0.7),[0.7, 1.0). If the first symbol encountered is, $a_2$ then the sub-interval [0.4, 0.7) is considered. If the next two symbols are $a_3$ and, $a_1$ then the intervals get narrowed. This process is illustrated in Fig 9. Any number within the final range[0.4, 0.448) can be used to represent the sequence.

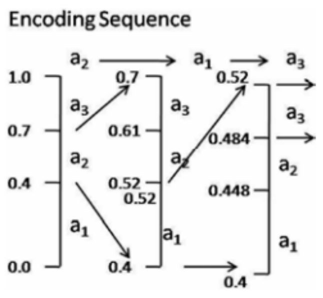


Fig. 9

### Types of Arithmetic Coding Algorithms

- A variant of arithmetic coding where the probability model adapts as the data is encoded or decoded.
- A method of entropy encoding where symbols are encoded as ranges, rather than as discrete values.
- A family of entropy encoding techniques that includes arithmetic coding as a special case.
- An adaptation of arithmetic coding that splits the encoding range into unequal partitions to more accurately represent the probability distribution.

### Advantages

*a) Higher compression ratio:* Arithmetic coding can achieve higher compression ratios than other techniques because it considers the probabilities of entire messages rather than just individual symbols.

*b) No need for a codebook:* Arithmetic coding does not require a pre-built codebook like Huffman coding, making it more flexible and adaptable to different data sets.

*c) Better suited for large data sets:* Arithmetic coding is better suited for compressing large data sets because it can handle long messages more efficiently than other techniques.

*d) Multiple messages in a single stream:* Arithmetic coding can compress multiple messages into a single stream, reducing the overhead associated with transmitting or storing multiple compressed files.

### Applications of Arithmetic Coding

Image and Video Compression with Arithmetic Coding

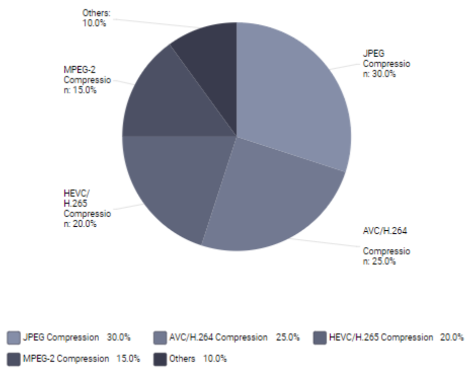


Fig. 10

Text and speech compression pie chart data

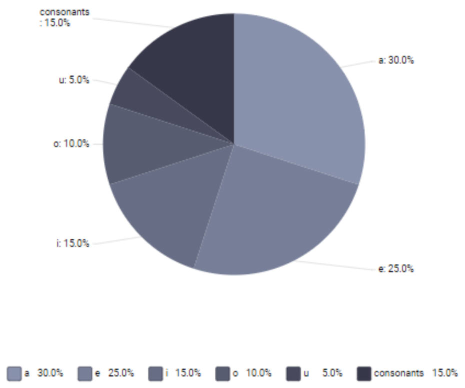


Fig. 11

**Challenges**

- Arithmetic coding is computationally expensive compared to other compression techniques which affect compression performance.
- Arithmetic coding requires a high level of mathematical and programming knowledge which can make the implementation process complex.
- Arithmetic coding requires large amounts of memory to store the probability distribution and the encoded data which can be a challenge for resource-constrained devices.
- Arithmetic coding is highly sensitive to any errors or corruption in the input data, which can result in significant loss of information during compression.

## V. LEMPEL-ZIV-WELCH COMPRESSION

*Lempel-Ziv-Welch (LZW) is a lossless data compression algorithm that is commonly used for compressing text and other types of data. It was developed by Abraham Lempel, Jacob Ziv, and Terry Welch in the 1970s and has since become widely used in various applications.*

The LZW compression algorithm is dictionary-based and functions by constructing and preserving a dictionary of common patterns or sequences found in the input data. This leads to compression by replacing the identified patterns with shorter codes or references to the corresponding dictionary entries. The final compressed output comprises a sequence of these codes, which can be decoded to restore the initial data.

However, the LZW algorithm ensures that each pattern is uniquely represented by a code, allowing for efficient encoding and decoding. The codes used by LZW can be of fixed or variable length, depending on the implementation.

LZW compression is particularly effective for data with repetitive patterns, such as text files or images with regions of uniform color. It can achieve significant compression ratios, reducing the size of the data without any loss of information. LZW has been widely used in file compression formats, including GIF (Graphics Interchange Format) and the early versions of the popular ZIP format.

**Advantages**

- 1. Lossless Compression: LZW is a lossless compression algorithm, meaning that the decompressed data is an exact replica of the original data. No information is lost during the compression process, which is important for applications where data integrity is crucial.
- 2. High Compression Ratios: LZW is particularly effective at compressing data with repetitive patterns. It can achieve significant compression ratios, reducing the size of the data while preserving its content. This makes it suitable for compressing text files, images with uniform areas, and other types of data with redundancy.
- 3. Efficient Encoding and Decoding: LZW utilizes a dictionary-based approach, which allows for efficient encoding and decoding of data. Once the dictionary is constructed, encoding and decoding operations can be performed quickly, making it suitable for real-time or on-the-fly compression applications.
- 4. Broad Range of Applications: LZW has been widely used in various applications and file formats. It was originally used in GIF image compression and has also been utilized in early versions of the ZIP file format. Its versatility makes it suitable for a wide range of data compression needs.

**Disadvantages**

- 1. Patent Restrictions: LZW was patented by IBM, which led to licensing requirements and restrictions on its use in some countries. However, as of my knowledge cutoff in September 2021, the patent had expired in many jurisdictions, allowing for unrestricted use.
- 2. Memory Overhead: LZW compression relies on maintaining a dictionary of patterns encountered in the data. As the dictionary grows larger, it requires more memory to store and use effectively. This can be a disadvantage when compressing very large data sets or in memory-constrained environments.
- 3. Compression Efficiency for Small Datasets: LZW compression may not be as efficient for very small datasets or data with minimal redundancy. The overhead of maintaining the dictionary and encoding the data can be relatively high compared to the achieved compression gains. In such cases, other compression algorithms may be more suitable.
- 4. Algorithmic Complexity: The LZW algorithm itself has a certain level of complexity, which can make it more challenging to implement and understand compared to simpler compression techniques. This complexity can also affect the performance and computational requirements of the compression and decompression processes.

**Encoding and Decoding**

- Encoding Example

  Input: "ABABABAABABA"
  Codes: [1, 2, 3, 2, 3]
  1. Start with an empty dictionary: 'A': 1, 'B': 2
  2. Scanning the input data:
  - Encountering 'A': 'A' is in the dictionary. The current pattern is 'A'.
  - Encountering 'B': 'AB' is not in the dictionary. Output the code for 'A' (1) and add 'AB' to the dictionary with code 3.
  - Encountering 'A': 'A' is in the dictionary. The current pattern is 'A'.
  - Encountering 'B': 'AB' is in the dictionary with

code 3. The current pattern is 'AB'.
- Encountering 'A': 'ABA' is not in the dictionary. Output the code for 'AB' (2) and add 'ABA' to the dictionary with code 4.
3. The compressed output is: [1, 2, 3, 2, 3]

- Decoding Example

  Input (codes): [1, 2, 3, 2, 3]
  1. Start with an empty dictionary: 'A': 1, 'B': 2
  2. Process the codes:
  - Read code 1: Output 'A', and add 'A' to the dictionary with code 3.
  - Read code 2: Output 'B', and add 'B' to the dictionary with code 4.
  - Read code 3: Output 'A', and add 'AB' to the dictionary with code 5.
  - Read code 2: Output 'B', and add 'BA' to the dictionary with code 6.
  - Read code 3: Output 'A'.
  3. The decoded output is: "ABABABAABABA"

In conclusion, LZW (Lempel-Ziv-Welch) compression is a widely used lossless compression algorithm that offers several advantages. It is particularly effective at compressing data with repetitive patterns, achieving high compression ratios while preserving the original data. LZW compression operates by creating a dictionary of patterns encountered in the data and replacing those patterns with codes. This dictionary-based approach allows for efficient encoding and decoding operations.

## VI. CONCLUSION

In conclusion, data compression algorithms are an invaluable tool for storing, transferring, and manipulating large amounts of data. By compressing data into smaller files, we can reduce the amount of space needed for storage and reduce the time taken to send or receive files over a network. Compression algorithms range from lossless techniques that preserve all original data to lossy ones that sacrifice some information in exchange for smaller file sizes. Popular algorithms such as JPEG and MP3 are used extensively in audio and video applications respectively. Additionally, text-based documents can be compressed with algorithms such as LZW or Huffman coding to minimize storage requirements.

## ACKNOWLEDGEMENT

We would like to express our sincere appreciation and gratitude to all those who have contributed to the successful completion of this project on discrete mathematics. Their support, guidance, and assistance have been invaluable throughout this journey.

First and foremost,we would like to extend our deepest gratitude to our mentor MANISH KUMAR GUPTA, whose expertise, knowledge, and constant encouragement have been instrumental in shaping this project. Their valuable insights, constructive feedback, and unwavering support have genuinely helped me navigate the intricacies of this subject.

We would also like to thank the faculty members of the DAIICT for their comprehensive curriculum and insightful lectures on discrete mathematics. Their dedication to teaching and willingness to clarify doubts have significantly contributed to my understanding of the subject matter.

In conclusion, we extend my heartfelt appreciation to all those who have played a part in the successful completion of this project on discrete mathematics. Their contributions have been invaluable, and we are truly grateful for your support.

## CONTRIBUTION

- WEBSITE DESIGNING AND CONTENT
  Kushang Chhabria:202201304
  Saish Pagar:202201092
- LATEX AND CONTENT
  Rohin Ghanshyam Solanki:202201039
  Khushi Prajapati:202201062
- YOUTUBE VIDEO AND CONTENT
  Nisarg Parmar:202201443
  Brahmbhatt Krisha Pankajkumar:202201164

[1] [2] [3] [4] [5] [6] [7]

## REFERENCES

[1] GeeksforGeeks. *LZW (Lempel–Ziv–Welch) Compression technique*. 2023. URL: https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/.

[2] GeeksforGeeks. *Run Length Encoding and Decoding*. 2022. URL: https://www.geeksforgeeks.org/run-length-encoding/.

[3] Akshay Mishra. *Data Compression With Arithmetic Coding*. 2022. URL: https://www.scaler.com/topics/data-compression-with-arithmetic-coding/.

[4] Reghunadhan Rajesh Rajeswari Rajendran. *Image compression techniques – a survey*. 2010. URL: https://www.researchgate.net/publication/267023123_Image_compression_techniques_-_a_survey.

[5] Khalid Sayood. *Introduction to data compression*. Morgan Kaufmann, 2017.

[6] Akshay Singhal. *Huffman Coding | Huffman Coding Example | Time Complexity*. 2020. URL: https://www.gatevidyalay.com/huffman-coding-huffman-encoding/.

[7] soubhikmitra98. *Introduction to Data Compression*. 2021. URL: https://www.geeksforgeeks.org/introduction-to-data-compression/.