Applying Lempel–Ziv–Welch

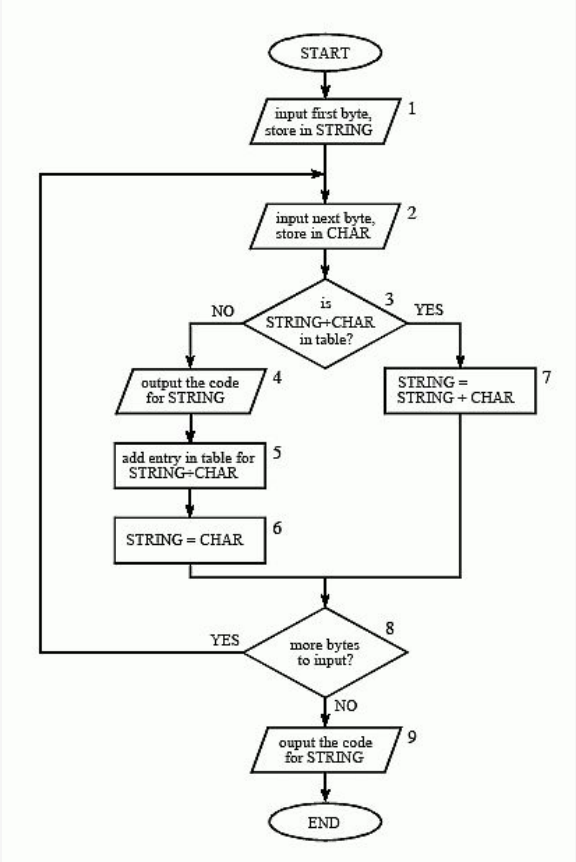
Compression to PDF Files: A Case Study

Information Theory Project Y

Prepared by:
Dana Donevska
Dejan Kelečević
Marko Stošić

May 2024

Table of contents



LZW introduction

Developed by Abraham Lempel, Jacob Ziv, and Terry Welch in 1984.

Advantages

1. **Lossless Compression Algorithm**: Ensures no data is lost during compression.
2. **Simple Application**: Simple to implement and does not require prior knowledge of data statistics.
3. **Efficient for Repetitive Data**: Ideal for files with repeated patterns, such as text.

Disadvantages

1. Requires an initial dictionary.
2. Dictionary size can affect compression efficiency.

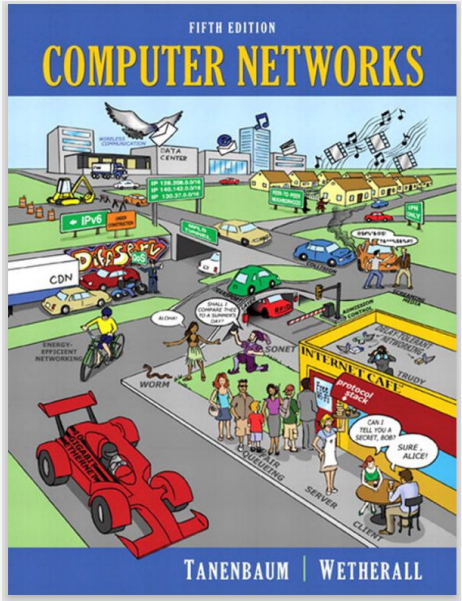
LZW compression is widely used in formats such as GIF, TIFF, and PDF for effectively reducing file sizes while maintaining data integrity. Implemented in Unix compress command-line utility.

PDF file compression

To highlight the importance of compression and to achieve a better compression rate, we conducted an analysis on "Computer Networks, Fifth Edition" by Tanenbaum and Wetherall, a book consisting of 962 pages.

We developed a program featuring the LZW compression algorithm. This method identifies repeated patterns in the input data and replaces them with single codes. The **dictionary**, or code table, is dynamically updated as new patterns are discovered.

The compressed data is represented as a sequence of these codes, which can later be decoded to retrieve the original data.



Our program

On the left, we see the initial stages of the program where the LZW algorithm is prepared and executed. This includes the setup and execution of the compression process.

On the right, we observe the compress method at work, illustrating the dynamic creation of the dictionary during the compression process.

```java
String pdfFilePath = "..\\files\\CNknjiga.pdf"; // Path to input PDF file
String outputFilePath = "..\\files\\CNknjiga_compressed.lzw"; // Path to output compr

try {
    // Step 1: Read the PDF file into a byte array
    byte[] bytes = readPDFIntoByteArray(pdfFilePath);

    // Step2: Write the byte array to a temporary file for compression
    String tempFilePath = "temp_pdf_bytes.bin";
    Files.write(Paths.get(tempFilePath), bytes);

    // Step 3: Compress the binary data using LZW compression
    try (FileInputStream inputFileStream = new FileInputStream(tempFilePath);
         FileOutputStream outputFileStream = new FileOutputStream(outputFilePath)) {

        App lzw = new App(inputFileStream, outputFileStream);
        lzw.compress();
        System.out.println(x:"Compression completed successfully.");
    }
```

```java
// Method to compress the input data using LZW algorithm
public void compress() throws IOException {
    int next_code = 256;  // Next code value to be used
    for (int i = 0; i < TABLE_SIZE; i++) {
        code_value[i] = EOF; // Initialize code values
    }

    int current_code = input.read(); // Read the first byte
    if (current_code == EOF) {
        return; // Empty input
    }

    int input_byte;
    while ((input_byte = input.read()) != EOF) {
        int index = findMatch(current_code, input_byte);
        if (code_value[index] != EOF) {
            current_code = code_value[index];
        } else {
            outputCode(current_code); // Output the current code
            if (next_code <= MAX_CODE) {
                code_value[index] = (short) next_code++; // Add n
                prefix_code[index] = (short) current_code;
                append_character[index] = (short) input_byte;
            }
            current_code = input_byte;
        }
    }
    outputCode(current_code); // Output the last code
    outputCode(MAX_VALUE); // Output end of file marker
    flushBuffer(); // Flush remaining bits in buffer
}
```

Compression rate results

```
// Step 4: Obtain the sizes of the PDF file
//and other document formats
// ...
// Step 5: Compare the sizes and calculate
//compression ratios
// ...
// Step 6: Print the results
// ...
```

Steps 4, 5, and 6 are straightforward. We compared the size of the file compressed with this algorithm to 6 other commonly used file formats.

After compression, the file was saved in the files folder, where the other formats of the same book are previously stored.

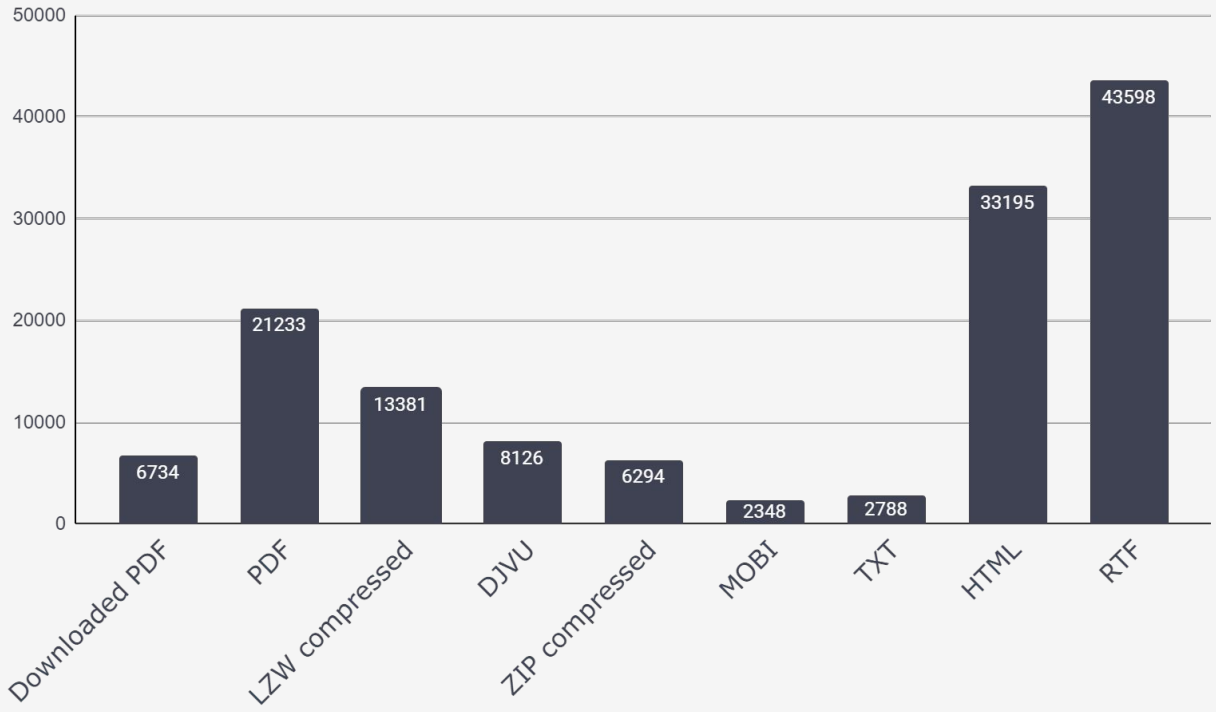
All the code, including the book files and this presentation, can be found at: *https://github.com/DanaDonev/LZW-compression-on-pdf-files.git.*

```
Compression completed successfully.
Compression ratio (PDF to Downloaded PDF): 0.3172455971872924
Compression ratio (PDF to Lempel-Ziv): 0.63019983843885
Compression ratio (PDF to DJVu): 0.38270711561281123
Compression ratio (PDF to ZIP): 0.2963974660021775
Compression ratio (PDF to MOBI): 0.11056042622044648
Compression ratio (PDF to TXT): 0.13130976068794895
Compression ratio (PDF to HTML): 1.5633994589085916
Compression ratio (PDF to RTF): 2.053396452231948
```

The LZW compression algorithm achieved a compression ratio of 0.63, reducing the file size to **63% of the original**.

In this case, LZW is more efficient than HTML and RTF but less effective than ZIP, MOBI, and DJVu formats. Unlike MOBI and DJVu, which can use lossy compression methods, LZW ensures no data loss during the compression process.

File size comparison



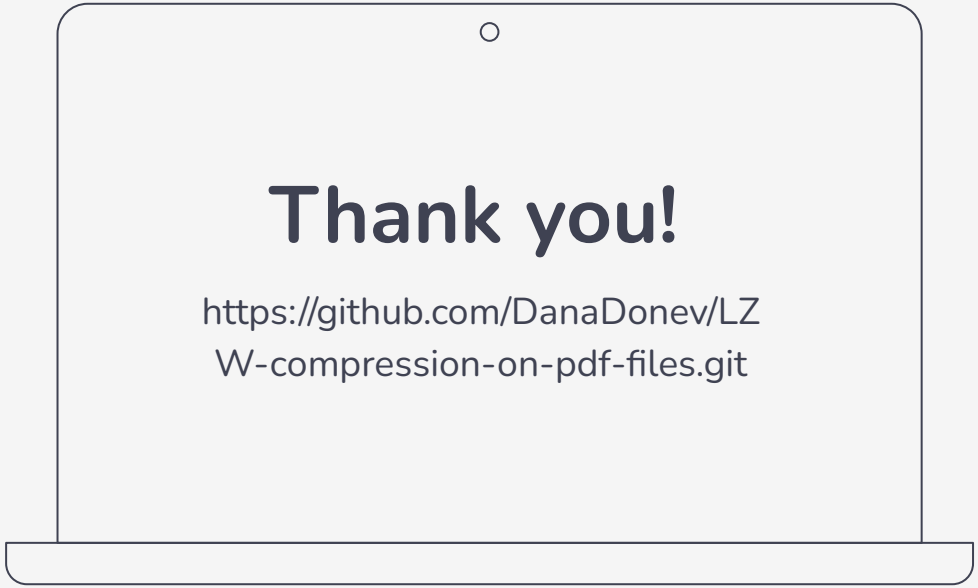
Best results

MOBI format, designed for eBook readers and optimized for text-heavy documents.

TXT format, which eliminates all formatting and metadata, also performed exceptionally well, making it ideal for only textual content.

Worst results

HTML and RTF formats resulted in ~1.5x and ~2x larger file sizes compared to the original PDF.


Thank you!
https://github.com/DanaDonev/LZW-compression-on-pdf-files.git