

# Explorando algoritmos de compressão de dados: teoria, implementação e desempenho

Gustavo Yujii Silva Kadooka

Universidade Estadual Paulista - Câmpus Bauru

2025

1 Resumo da proposta

2 Progresso atual

## Objetivo geral

Analisar comparativamente a eficiência dos algoritmos de compressão de dados clássicos (Huffman, LZ77, LZW, GZIP), focando na taxa de compressão e tempo de execução.

## Objetivos específicos

- Descrever os princípios teóricos dos algoritmos selecionados.
- Implementar os algoritmos e testar em diferentes tipos de arquivos.
- Fornecer recomendações para a escolha do algoritmo mais adequado.

Figura 1: Cronograma atualizado

Etapas	ABR	MAIO	JUN	JUL	AGO	SET	OUT
Levantamento bibliográfico	X	X	X	X	X	X	X
Desenvolvimento <i>back-end</i>	X	X	X	X			
Desenvolvimento <i>front-end</i>		X	X	X	X		
Testes e análise de desempenho			X	X	X		
Escrita da monografia		X	X	X	X	X	X

**Fonte:** Desenvolvido pelo autor.

# Atividades Concluídas (abril – maio)

- Levantamento bibliográfico concluído em abril.
- Início da implementação do *back-end* (API de manipulação de arquivos WAV e BMP e algoritmos Huffman e LZ77).
- Início do front-end com GTK (interface básica).
- Escrita parcial da monografia (introdução e fundamentação teórica).

# Levantamento bibliográfico

Algumas das referências bibliográficas utilizadas até o momento são:

## Artigos Clássicos

Hartley 1928 Shannon 1948 Huffman 1952 Ziv and Lempel 1977  
Welch 1984

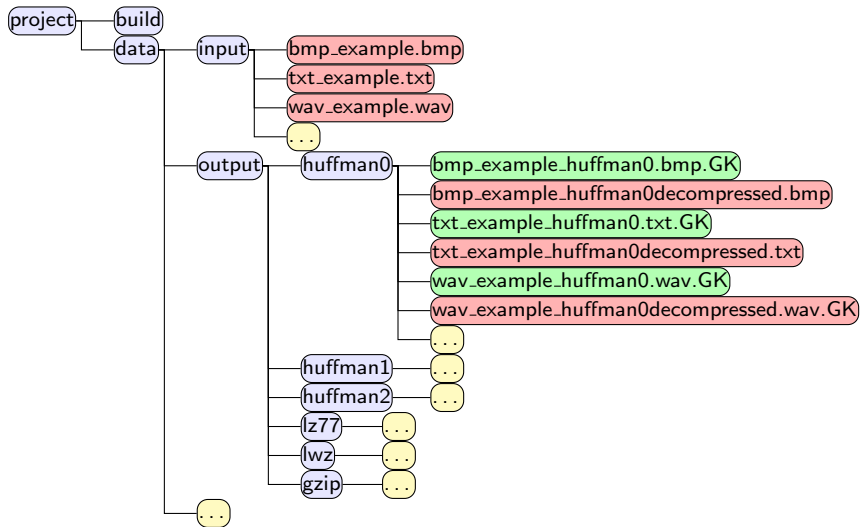
## Novos algoritmos de compressão

Alakuijala et al. 2016 Collet 2016 DeepMind 2016

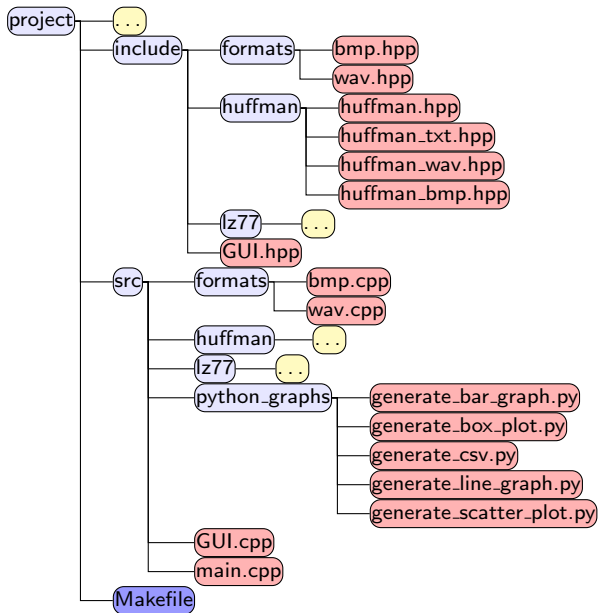
## Outros

Salomon and Motta 2007 Deutsch 1996

# Back-End

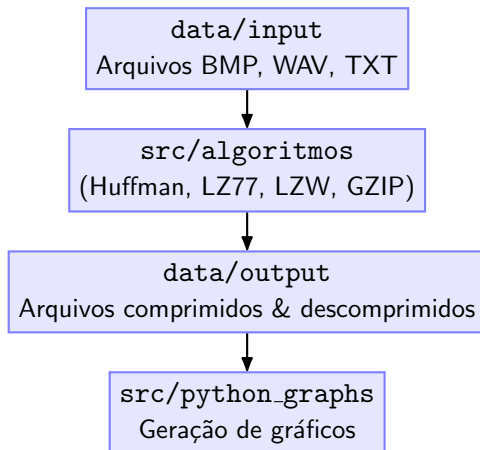


# Back-End





# Fluxo Geral do Projeto



**Fonte:** Diagrama elaborado pelo autor com base na estrutura de diretórios do projeto.

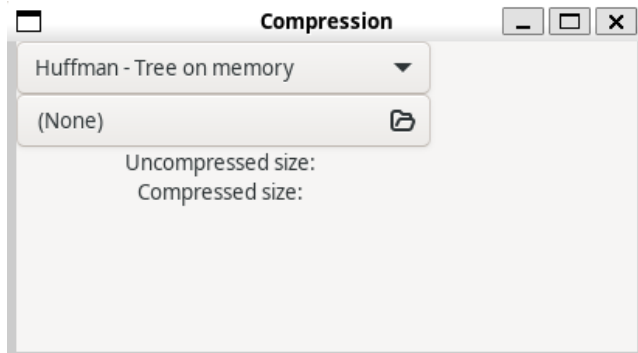
# Back-End – API wav.h

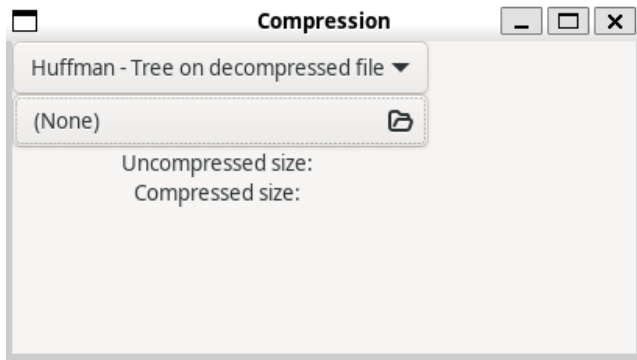
```
1 class wav {
2 public:
3     struct RIFF_chunk_descriptor {
4         char ID[4];
5         uint32_t size;
6         char format[4];
7     } riff_chunk_descriptor;
8     struct FMT_subchunk {
9         char ID[4];
10        ...
11    } fmt_subchunk;
12    struct DATA_subchunk {
13        ...
14    } data_subchunk;
15    uint8_t* data;
16
17    int read(std::string filename);
18    void print();
19 };
20
```

# Back-End – API wav.c

```
1 int wav::read(std::string filename) {
2     FILE *file = fopen(filename.c_str(), "rb");
3     if(!file) {
4         return -1;
5     }
6     if(fread(&riff_chunk_descriptor, sizeof(
7 riff_chunk_descriptor), 1, file) != 1) {
8         fclose(file);
9         return -1;
10    }
11    if(fread(&fmt_subchunk, sizeof(fmt_subchunk), 1, file) !=
12    1) { ... }
13    if(fread(&data_subchunk, sizeof(data_subchunk), 1, file)
14    != 1) { ... }
15    data = new uint8_t[data_subchunk.size];
16    if(!data) { ... }
17    if(fread(data, data_subchunk.size, 1, file) != 1) { ... }
18    fclose(file);
19    return 0;
20 }
```







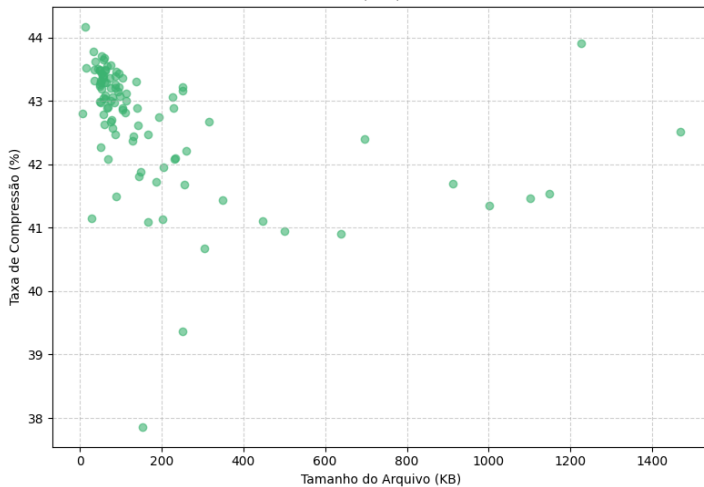
# Testes e análise de desempenho

Arquivo	Original (B)	Comprimido (B)	Compressão (%)
txt_example5	457730	269554	41,12%
txt_example19	88005	49819	43,40%
wav_example4	10543996	9748029	7,56%
bmp_example2	53747850	53045595	1,31%
txt_example9	1503949	864639	42,50%
txt_example92	37189	21018	43,48%
txt_example2	6311	3610	42,80%
txt_example45	80413	46081	42,69%
txt_example33	68654	39212	42,90%
txt_example70	49444	28055	43,26%

Table 1: Exemplo de .csv usando o algoritmo de Huffman com árvore na memória

# Testes e análise de desempenho

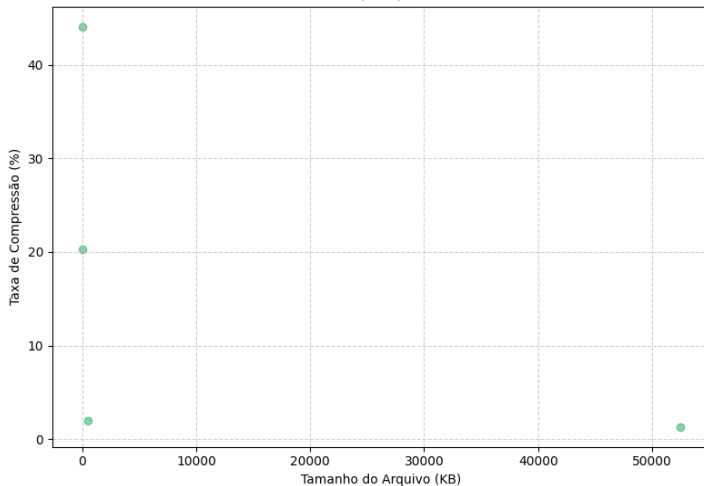
Gráfico de Dispersão: Tamanho do Arquivo vs Taxa de Compressão (TXT)





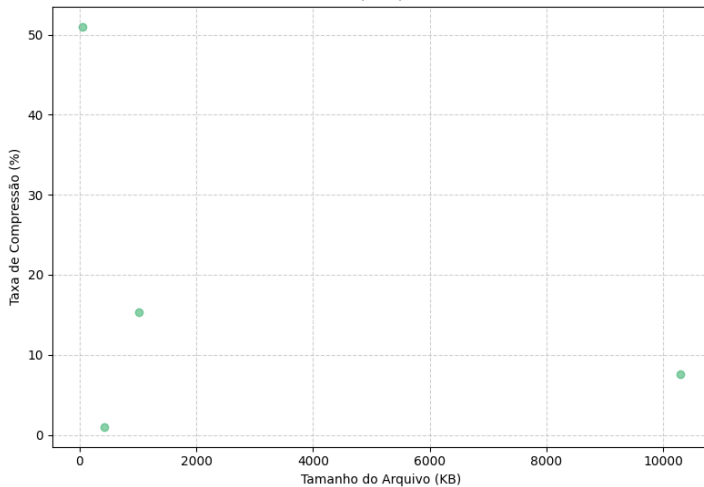
# Testes e análise de desempenho

Gráfico de Dispersão: Tamanho do Arquivo vs Taxa de Compressão (BMP)

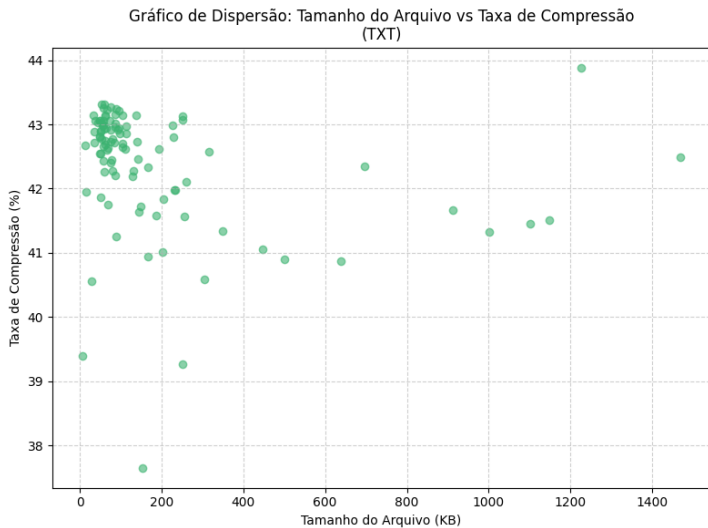


# Testes e análise de desempenho

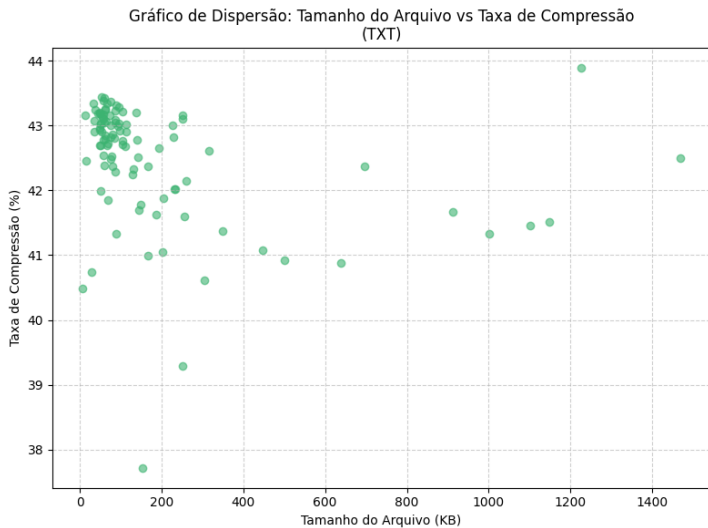
Gráfico de Dispersão: Tamanho do Arquivo vs Taxa de Compressão (WAV)



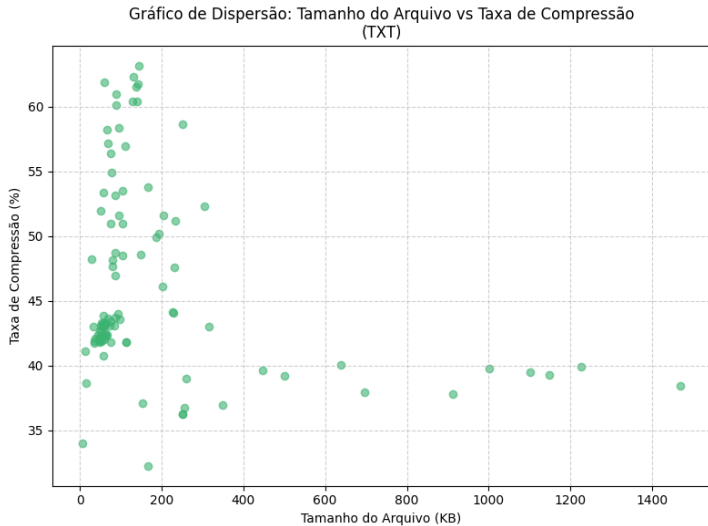
# Testes e análise de desempenho



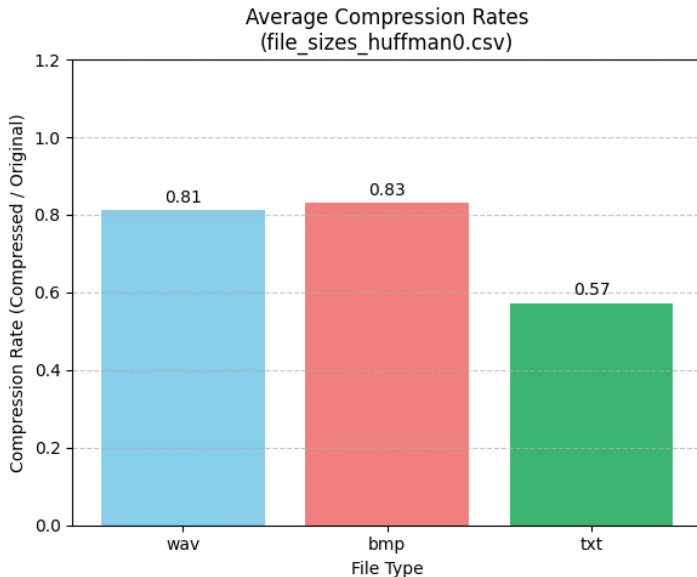
# Testes e análise de desempenho



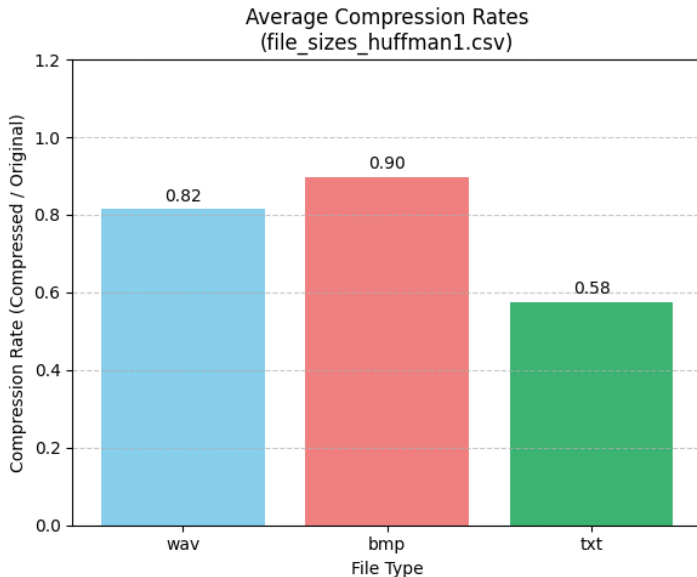
# Testes e análise de desempenho



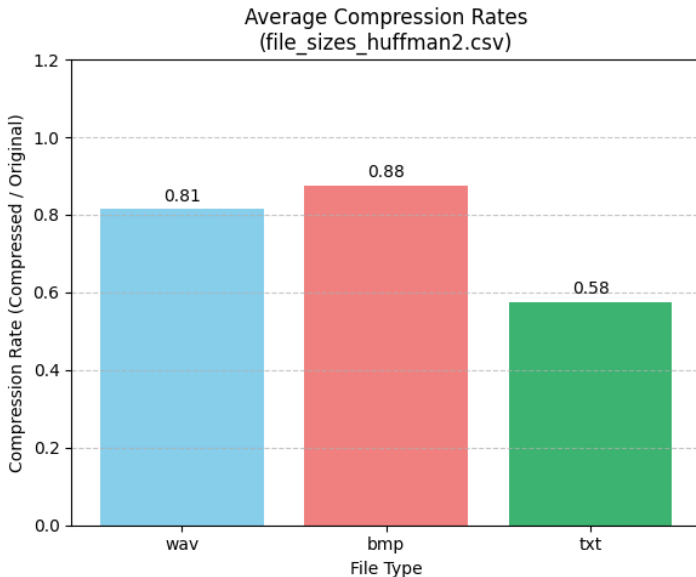
# Testes e análise de desempenho



# Testes e análise de desempenho

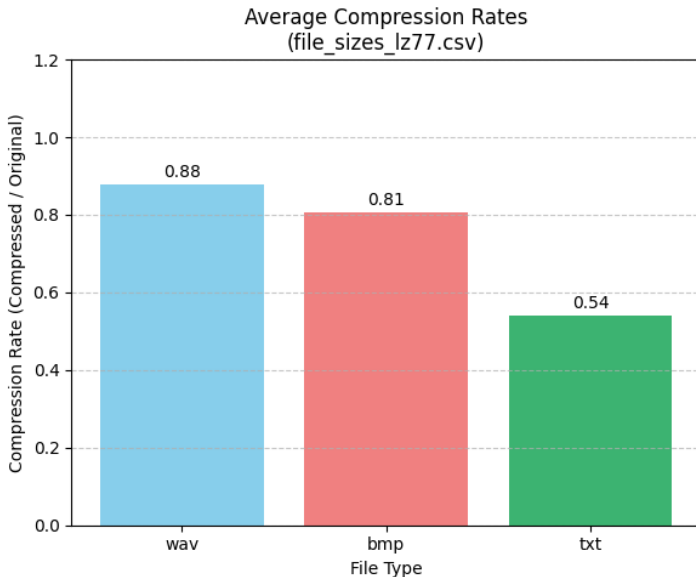


# Testes e análise de desempenho





# Testes e análise de desempenho



**UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"**  
**FACULDADE DE CIÊNCIAS - CAMPUS BAURU**  
**DEPARTAMENTO DE COMPUTAÇÃO**  
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

« □ » « ☰ » « ≡ » « ≡ » ≡ || ≡ ↺ 🔍 ↻

GUSTAVO YUJII SILVA KADOOKA

**EXPLORANDO ALGORITMOS DE COMPRESSÃO DE DADOS:  
TEORIA, IMPLEMENTAÇÃO E DESEMPENHO**

BAURU  
Novembro/2025

GUSTAVO YUJII SILVA KADOOKA

« □ » « ☰ » « ≡ » « ≡ » ≡ ≡ 🔍 ↺

# **EXPLORANDO ALGORITMOS DE COMPRESSÃO DE DADOS: TEORIA, IMPLEMENTAÇÃO E DESEMPENHO**

Trabalho de Conclusão de Curso em  
Ciência da Computação apresentado  
ao Departamento de Computação da  
Universidade Estadual Paulista “Júlio de  
Mesquita Filho”, Faculdade de Ciências,  
Campus Bauru.

Orientador: Profa. Dra. Andréa Carla  
Gonçalves Vianna



K11e

Kadooka, Gustavo Yujii Silva

Explorando Algoritmos de Compressão de Dados: Teoria,  
Implementação e Desempenho / Gustavo Yujii Silva Kadooka. --  
Bauru, 2025

60 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da  
Computação) - Universidade Estadual Paulista (UNESP), Faculdade  
de Ciências, Bauru

Orientadora: Andréa Carla Gonçalves Vianna

1. Compressão de dados. 2. Teoria da informação. 3. Compressão  
sem perdas – Huffman, LZ77, LZW, GZIP. I. Título.

Gustavo Yujii Silva Kadooka

# Explorando Algoritmos de Compressão de Dados: Teoria, Implementação e Desempenho

Trabalho de Conclusão de Curso em Ciência da Computação apresentado ao Departamento de Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Banca Examinadora

---

**Profa. Dra. Andréa Carla Gonçalves  
Vianna**

Orientador

Universidade Estadual Paulista “Júlio de  
Mesquita Filho”

Faculdade de Ciências

Departamento de Ciência da Computação

---

**Professor Convidado 1**

Universidade Estadual Paulista “Júlio de  
Mesquita Filho”

Faculdade de Ciências

Departamento de Ciência da Computação

---

**Professor Convidado 2**

Universidade Estadual Paulista “Júlio de  
Mesquita Filho”

Faculdade de Ciências

Departamento de Ciência da Computação

Bauru, 11 de novembro de 2025.



*À minha amada família, fonte inesgotável de inspiração e força.*

# Agradecimentos

Agradeço à Universidade Estadual Paulista “Júlio de Mesquita Filho” – Faculdade de Ciências, Campus de Bauru –, por ter sido o espaço onde pude me desenvolver como estudante, profissional e indivíduo. Sou grato a todos os professores e funcionários da instituição pelo conhecimento compartilhado e pelo apoio ao longo da graduação.

À minha orientadora, Profa. Dra. Andréa Carla Gonçalves Vianna, pela disponibilidade, paciência diante das dificuldades e por acreditar no meu potencial durante o desenvolvimento deste trabalho.

Aos meus colegas de curso, que estiveram presentes ao longo desta jornada, dividindo aprendizados, desafios e conquistas. O convívio e a troca de experiências tornaram a caminhada mais leve e significativa.

À minha irmã Gabriely, pelas palavras de incentivo e carinho em todos os momentos, mesmo nos mais difíceis.

E, com o meu mais profundo e sincero agradecimento, menciono especialmente meus pais – Flávia e Silvio –, por todo o apoio emocional, pelos ensinamentos de vida, pela confiança depositada em mim e por sempre me encorajarem a seguir em frente. Esta conquista é, acima de tudo, de vocês.

# Resumo

Este trabalho apresenta um estudo abrangente, a implementação prática e a análise de desempenho de algoritmos clássicos de compressão de dados, a saber: Huffman, LZ77, LZW e GZIP. São explorados os conceitos teóricos fundamentais de cada algoritmo, seguidos pela implementação prática utilizando a linguagem de programação C++. As implementações dos algoritmos lidam com os formatos de arquivo de texto simples (.txt), imagens bitmap (.bmp) e arquivos de áudio PCM (.wav), demonstrando versatilidade e aplicabilidade em diferentes domínios de dados. Para facilitar a interação do usuário, foi desenvolvida uma interface gráfica utilizando a biblioteca GTK, permitindo a seleção dos algoritmos de compressão e o arquivo de entrada. Ademais, para realizar uma análise comparativa rigorosa, foram utilizados scripts em Python para processar os dados experimentais e gerar representações gráficas dos principais indicadores de desempenho, como a taxa de compressão e o tempo de execução para diferentes tipos de arquivos. Os resultados experimentais indicam que não há um algoritmo que se destaque universalmente, pois cada um apresenta vantagens específicas conforme o tipo de arquivo e as características dos dados. Isso ressalta a importância da escolha do método de compressão adequado às necessidades específicas da aplicação. O projeto contribui tanto com uma ferramenta prática para compressão de dados quanto com um recurso educacional que auxilia na compreensão dos algoritmos clássicos e seu comportamento em cenários reais.

**Palavras-chave:** Compressão de dados. Algoritmos de compressão. Huffman. LZ77. LZW. GZIP. C++. Python. GTK.



# Abstract

This work presents a comprehensive study, implementation, and performance analysis of classical data compression algorithms, namely Huffman, LZ77, LZW, and GZIP. The theoretical concepts behind each algorithm are explored in depth, followed by practical implementations using C++ programming language. The implementations support plain text (.txt), bitmap images (.bmp), and PCM audio files (.wav), demonstrating versatility and applicability in different data domains. To facilitate user interaction, a graphical user interface (GUI) was developed with the GTK library, enabling selection of compression algorithms, and input files. Furthermore, to conduct a rigorous comparative analysis, Python scripts were employed to process experimental data and generate graphical representations of key performance metrics such as compression ratio and execution time for different file types. The experimental results indicate that no single algorithm universally outperforms others; rather, each one exhibits specific advantages depending on the file type and data characteristics. This highlights the importance of choosing the appropriate compression method tailored to the application's needs. The project contributes as a practical tool for data compression and an educational resource that aids in understanding fundamental compression algorithms and their behavior in real-world scenarios.

**Keywords:** Data compression, compression algorithms, Huffman coding, LZ77, LZW, GZIP, C++, Python, GTK.

## Lista de ilustrações

Figura 1 – Entropia de uma moeda em função de  $p$  . . . . . 20



## Lista de tabelas



# Lista de abreviaturas e siglas

LZ77	Lempel-Ziv 1977 (algoritmo de compressão)
LZW	Lempel-Ziv-Welch (algoritmo de compressão)
GZIP	GNU zip (algoritmo de compressão)
GTK	GIMP Toolkit (biblioteca gráfica para interface)

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
<b>1.1</b>	<b>Problema</b>	<b>13</b>
<b>1.2</b>	<b>Objetivos</b>	<b>13</b>
1.2.1	Objetivo geral	13
1.2.2	Objetivos específicos	14
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>15</b>
<b>2.1</b>	<b>Teoria da Informação</b>	<b>15</b>
2.1.1	Quantificação da Informação de Hartley	15
2.1.2	Conteúdo informacional de Shannon	18
2.1.3	Entropia	19
2.1.4	Redundância	20
<b>2.2</b>	<b>Compressão de dados</b>	<b>22</b>
<b>2.3</b>	<b>Algoritmos clássicos de compressão sem perdas</b>	<b>22</b>
<b>3</b>	<b>IMPLEMENTAÇÃO</b>	<b>23</b>
<b>4</b>	<b>ANÁLISE COMPARATIVA</b>	<b>24</b>
<b>5</b>	<b>CONCLUSÃO</b>	<b>25</b>
	<b>REFERÊNCIAS</b>	<b>26</b>

# 1 Introdução

A compressão de dados é uma técnica essencial na ciência da computação, utilizada para reduzir o tamanho dos arquivos, otimizando o uso de recursos e acelerando a transmissão de dados – aspectos cruciais na era digital em que vivemos.

O desenvolvimento da área teve início na década de 1950, com os primeiros métodos voltados à otimização do espaço de armazenamento. Entre estes métodos, destaca-se o algoritmo de Huffman (HUFFMAN, 1952), que emprega uma codificação de prefixo variável, atribuindo códigos menores aos símbolos mais frequentes e maiores aos menos frequentes. Essa abordagem se consolidou como um dos pilares da área e é utilizada em formatos como ZIP e GZIP.

Posteriormente, novos algoritmos surgiram. Um dos mais significativos é o LZ77, proposto por Abraham Lempel e Jacob Ziv (ZIV; LEMPEL, 1977), que adota uma estratégia baseada em dicionários, substituindo sequências repetidas por referências a posições anteriores. A partir desta ideia, o LZW vem à tona aprimorando o LZ77 ao construir dinamicamente um dicionário de *strings* durante a compressão (WELCH, 1984), sendo amplamente utilizado em formatos como GIF e TIFF.

Outro algoritmo relevante é o GZIP, amplamente adotado em sistemas Unix e aplicações web. Ele combina a eficiência da codificação de Huffman com o LZ77, alcançando boas taxas de compressão sem perdas. Sua popularidade decorre da combinação de alta eficácia com a facilidade de descompressão.

Apesar do surgimento de novas técnicas, os algoritmos de Huffman, LZ77, LZW e GZIP continuam a formar a espinha dorsal dos métodos clássicos de compressão de dados e são amplamente utilizados em diversas aplicações.

Nos últimos anos, surgiram algoritmos mais modernos, como o Brotli – desenvolvido pela Google – que combina codificação de Huffman e transformações de fluxo de dados para obter taxas de compressão superiores às do GZIP (ALAKUIJALA et al., 2016). Outro exemplo é o Zstandard (Zstd), criado pelo Facebook, que alia velocidade e compressão eficiente, sendo adequado tanto para compressão em tempo real quanto para grandes volumes de dados (COLLET, 2016).

Além disto, algoritmos baseados em aprendizado de máquina têm ganhado destaque, principalmente na compressão de imagens e áudio. Redes neurais como GANs (para imagens) e modelos como o WaveNet (para áudio) (DeepMind, 2016) vêm sendo explorados para melhorar a qualidade das compressões. No campo da compressão de vídeo, codecs como HEVC (H.265) e VVC vêm sendo desenvolvidos

para oferecer compressões mais eficazes em vídeos de alta definição.

A compressão de dados desempenha, portanto, um papel estratégico em diversas áreas tecnológicas, impactando diretamente sistemas de armazenamento, transmissão de dados, streaming, redes e computação em nuvem. A análise de desempenho dos algoritmos, considerando métricas como taxa de compressão e tempo de execução, é essencial para a escolha do método mais adequado a cada situação.

Neste contexto, apresenta-se um estudo aprofundado sobre os algoritmos clássicos de compressão Huffman, LZ77, LZW e GZIP. São abordados os fundamentos teóricos de cada método, suas implementações práticas em linguagem C++ com suporte a arquivos nos formatos `.txt`, `.bmp` e `.wav`, e uma interface gráfica desenvolvida com GTK. A avaliação experimental é realizada por meio de comparações de desempenho, com auxílio de scripts em Python para geração de gráficos e análise dos resultados. O objetivo é compreender o comportamento dos algoritmos em diferentes tipos de dados, destacando suas vantagens, limitações e aplicações mais adequadas.

O restante deste trabalho está organizado da seguinte forma: o Capítulo 2 apresenta os fundamentos teóricos sobre compressão de dados, abordando a Teoria da Informação e os algoritmos Huffman, LZ77, LZW e GZIP. O Capítulo 3 descreve a implementação dos algoritmos, incluindo aspectos técnicos e ferramentas utilizadas. No Capítulo 4, é realizada uma análise comparativa com base nos testes realizados, considerando taxa de compressão e tempo de execução. Por fim, o Capítulo 5 apresenta as conclusões do trabalho e sugestões para pesquisas futuras.

## 1.1 Problema

Como os diferentes algoritmos clássicos de compressão de dados (Huffman, LZ77, LZW e GZIP) se comparam em termos de eficiência de compressão e tempo de execução, e qual o impacto dessas métricas em aplicações práticas que lidam com texto, imagem e áudio?

## 1.2 Objetivos

### 1.2.1 Objetivo geral

Investigar e comparar o desempenho de algoritmos clássicos de compressão de dados sem perdas, a saber: Huffman, LZ77, LZW e GZIP, por meio da implementação prática dessas técnicas, aplicando-as a diferentes tipos de arquivos e analisando sua eficiência em termos da taxa de compressão e tempo de execução.

### 1.2.2 Objetivos específicos

- Estudar os fundamentos teóricos dos algoritmos de compressão sem perdas selecionados, compreendendo o seu funcionamento, estrutura e aplicação histórica.
- Implementar, em linguagem C++, os algoritmos Huffman, LZ77, LZW e GZIP, com foco na compressão e descompressão de arquivos de texto (.txt), imagem bitmap (.bmp) e áudio PCM (.wav).
- Desenvolver uma interface gráfica com a biblioteca GTK, que permita ao usuário escolher o algoritmo e o algoritmo de entrada de forma interativa.
- Criar scripts em Python para a análise comparativa, organizando e visualizando os resultados experimentais por meio de gráficos e tabelas.
- Avaliar os pontos fortes e fracos de cada algoritmo, considerando o tipo de dado, a estrutura do arquivo, o tempo de processamento e a taxa de compressão.
- Identificar o algoritmo mais adequado a cada cenário específico, com base nos resultados experimentais obtidos.



## 2 Fundamentação teórica

Este capítulo apresenta os fundamentos teóricos que embasam o desenvolvimento deste trabalho. Para compreender o funcionamento e as diferenças entre os algoritmos de compressão de dados abordados, é necessário introduzir alguns conceitos essenciais.

Primeiramente, será apresentada a teoria da informação, a qual fornece o embasamento matemático para a compressão de dados sem perdas, por meio de conceitos como entropia e redundância. Em seguida, serão discutidos os princípios gerais da compressão de dados, com foco nas técnicas sem perdas, suas aplicações e métricas de desempenho. Por fim, serão detalhados os algoritmos clássicos de compressão sem perdas utilizados neste trabalho: Huffman, LZ77, LZW e GZIP, incluindo suas características, funcionamento e aplicações práticas.

A estrutura deste capítulo está organizada da seguinte forma:

- **Seção 2.1** – Apresenta os principais conceitos da Teoria da Informação, com ênfase na entropia de Shannon e seu papel na compressão de dados;
- **Seção 2.2** – Discute os fundamentos da compressão de dados sem perdas, destacando seus objetivos, métricas e distinções em relação à compressão com perdas;
- **Seção 2.3** – Descreve detalhadamente os algoritmos Huffman, LZ77, LZW e GZIP, explicando seus princípios de funcionamento, vantagens, limitações e casos de uso.

### 2.1 Teoria da Informação

A Teoria da Informação, proposta por Claude Shannon em 1948, constitui o alicerce teórico para diversos processos de codificação e compressão de dados na ciência da computação e engenharia de comunicações. Seu objetivo central é quantificar a informação contida em uma mensagem e estabelecer limites teóricos para a eficiência da transmissão e armazenamento de dados (SHANNON, 1948).

#### 2.1.1 Quantificação da Informação de Hartley

Em 1928, Ralph V. L. Hartley publicou o artigo *Transmission of Information* (HARTLEY, 1928), no qual propôs uma medida quantitativa para a informação baseada

unicamente em aspectos físicos e objetivos do processo de comunicação. Sua motivação era eliminar fatores subjetivos, como interpretação ou significado, e estabelecer uma definição útil para aplicações em engenharia.

Hartley introduz seu raciocínio a partir da observação de sistemas de comunicação reais, como o sistema de cabo telegráfico submarino operado manualmente. Nesse sistema, o operador utiliza uma chave com três posições, por exemplo, tensão positiva, tensão nula e tensão negativa, cada uma representando um símbolo físico distinto. A mensagem enviada é uma sequência dessas seleções, e o receptor interpreta os sinais registrados por um oscilógrafo em uma fita fotosensível.

O ponto fundamental é que, em cada instante de envio, o operador faz uma escolha consciente entre múltiplos símbolos possíveis. À medida que as seleções ocorrem, mais possibilidades de mensagens são descartadas, o que equivale a dizer que a informação vai se tornando mais específica.

Hartley argumenta que a quantidade de informação transmitida deve depender do número de símbolos disponíveis a cada escolha e do número total de escolhas feitas. Por exemplo, se há  $s = 3$  símbolos possíveis por escolha e  $n$  escolhas são feitas, o número total de sequências distintas possíveis é:

$$N = s^n$$

Esse valor representa a quantidade de mensagens distintas que o sistema é capaz de transmitir fisicamente. No entanto, usar diretamente esse número como medida de informação não é prático, pois ele cresce exponencialmente com o número de seleções. Para obter uma medida linear e proporcional ao número de escolhas, Hartley propõe tomar o logaritmo desse valor:

$$H = \log_b N = \log_b(s^n) = n \log_b s$$

Essa expressão garante que:

- A medida de informação seja proporcional ao número de escolhas  $n$ ;
- Duas mensagens independentes tenham informação total igual à soma das informações individuais (propriedade de aditividade);
- O valor seja expresso em uma unidade prática — como bits, ao usar base  $b = 2$ .

Exemplos:

- **Exemplo 0:** Considere um sistema em que cada seleção permite escolher entre três símbolos distintos (por exemplo,  $s = 3$ ). Se forem feitas 5 seleções ( $n = 5$ ), o número total de mensagens possíveis é:

$$N = 3^5 = 243$$

A quantidade de informação total transmitida será:

$$H = \log_2 243 \approx 7,925 \text{ bits}$$

Ou, usando a fórmula direta:

$$H = 5 \log_2 3 \approx 5 \times 1,585 \approx 7,925 \text{ bits}$$

Isso significa que, em média, cada seleção transmite aproximadamente 1,585 bits de informação.

- **Exemplo 1:** Suponha um sistema onde o transmissor pode escolher entre 2 símbolos diferentes (como 0 e 1, em um código binário). Uma única escolha transmite:

$$I = \log_2 2 = 1 \text{ bit}$$

Com 8 escolhas, tem-se:

$$H = 8 \log_2 2 = 8 \text{ bits}$$

- **Exemplo 2:** Considere agora um sistema com 4 símbolos distintos (por exemplo, A, B, C, D). Cada escolha transmite:

$$I = \log_2 4 = 2 \text{ bits}$$

Após 5 seleções, a quantidade total de informação transmitida é:

$$H = 5 \log_2 4 = 10 \text{ bits}$$

- **Exemplo 3:** Em um sistema decimal com 10 símbolos (de 0 a 9), uma única seleção transmite:

$$I = \log_2 10 \approx 3,32 \text{ bits}$$

Três seleções transmitiriam aproximadamente:

$$H = 3 \log_2 10 \approx 9,97 \text{ bits}$$

Esses exemplos ilustram que, quanto maior o número de símbolos possíveis por escolha, maior é o conteúdo de informação por escolha individual. A grande sacada de Hartley foi perceber que a informação é proporcional ao número de seleções, e que o logaritmo do número de possibilidades elimina a explosão exponencial, fornecendo uma medida prática e linear.

## 2.1.2 Conteúdo informacional de Shannon

A abordagem de Hartley, no entanto, assume que todos os símbolos têm igual probabilidade de ocorrência, o que nem sempre é verdadeiro na prática. Para resolver essa limitação, Shannon generalizou o conceito e definiu o conteúdo informacional<sup>1</sup> associada a um evento  $x$  com probabilidade  $P(x)$ :

$$I(x) = -\log_b P(x)$$

A ideia principal é que o valor informacional de uma mensagem depende do grau em que a mesma é surpreendente. Se um evento altamente provável ocorrer, a mensagem carrega pouca informação; por outro lado, se um evento improvável acontece, diz-se que carrega muita informação.

Por definição, se  $P(x) = 0$  então  $I(x) = 0$ , ou seja, se o evento for impossível, o valor informacional é nulo.

Considere dois eventos distintos:

- Evento  $A$ : "Vai chover amanhã", com probabilidade  $P(A) = 0,9$ ;
- Evento  $B$ : "Vai nevar amanhã", com probabilidade  $P(B) = 0,01$ .

Aplicando a fórmula do conteúdo informacional com base 2:

$$I(A) = -\log_2(0,9) \approx 0,152 \text{ bits}$$

$$I(B) = -\log_2(0,01) \approx 6,644 \text{ bits}$$

Isso significa que a ocorrência de  $A$  transmite pouca informação, pois é esperada. Já a ocorrência de  $B$  transmite uma quantidade significativamente maior de informação, pois representa um evento altamente improvável — portanto, mais surpreendente.

Esse exemplo mostra que a informação está relacionada com a incerteza: quanto menor a probabilidade de um evento, maior o seu conteúdo informacional.

<sup>1</sup> Em inglês, information content, surprisal ou self-information

### 2.1.3 Entropia

A entropia de uma fonte, então, é o valor esperado do conteúdo de informação de seus símbolos:

$$H(X) = - \sum_{i=1}^n P(x_i) \log_b P(x_i)$$

Essa formulação considera a estrutura estatística dos dados, permitindo medir a incerteza média da fonte — e, conseqüentemente, o limite teórico mínimo para compressão sem perdas.

Considere uma variável aleatória  $X$  que representa o lançamento de uma moeda, onde os possíveis resultados são:

- $x_1 = \text{cara}$ , com probabilidade  $P(x_1) = p$
- $x_2 = \text{coroa}$ , com probabilidade  $P(x_2) = 1 - p$

A entropia da variável  $X$  é:

$$H(X) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

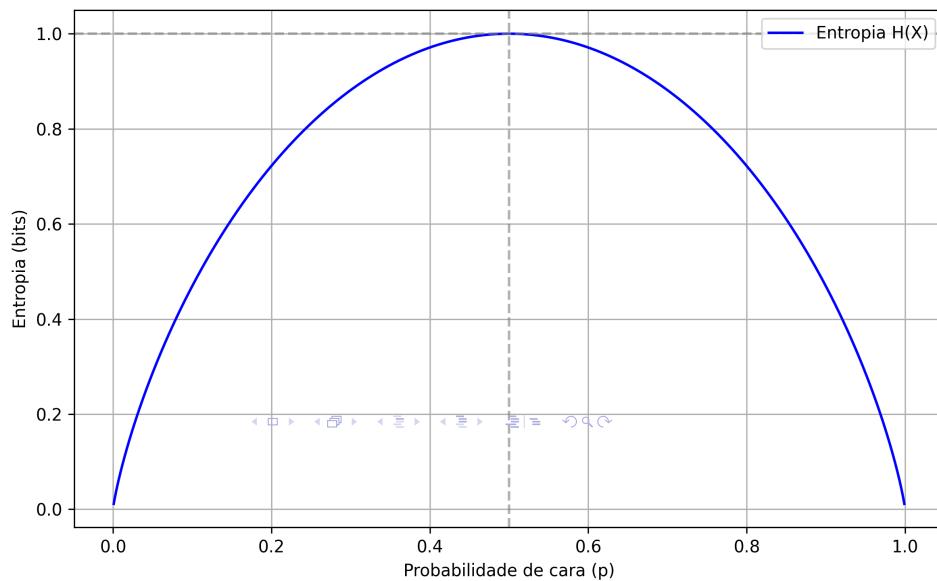
Essa função atinge seu valor máximo quando  $p = 0,5$ , ou seja, quando a moeda é justa e os dois eventos são igualmente prováveis. Nesse caso:

$$H(X) = -0,5 \log_2 0,5 - 0,5 \log_2 0,5 = -2 \times 0,5 \times (-1) = 1 \text{ bit}$$

Esse valor representa o máximo de incerteza possível para uma variável binária: não se sabe nada sobre qual será o resultado. Por outro lado, se a moeda estiver viciada e sempre cair do mesmo lado (por exemplo,  $p = 0$  ou  $p = 1$ ), então:

$$H(X) = -1 \cdot \log_2 1 = 0 \text{ bits}$$

Ou seja, não há incerteza — o resultado é completamente previsível, e, portanto, não há informação nova sendo transmitida.

Figura 1 – Entropia de uma moeda em função de  $p$ 

Fonte: Elaborada pelo autor

#### 2.1.4 Redundância

A redundância representa a diferença entre o comprimento médio real de codificação e a entropia da fonte. Em outras palavras, trata-se da quantidade de informação excedente que pode estar presente nos dados, muitas vezes introduzida por razões como segurança, integridade ou simplicidade de codificação. Quanto maior for essa redundância estatística, maior será o potencial de compressão sem perdas.

A relação entre a entropia  $H(X)$  de uma fonte e o comprimento médio  $L$  de um código pode ser expressa por:

$$R = L - H(X)$$

onde  $R$  é a redundância. Um código é considerado eficiente quando sua média de bits por símbolo se aproxima da entropia da fonte, ou seja, quando  $R \rightarrow 0$ .

O principal objetivo dos algoritmos de compressão de dados é justamente reduzir essa redundância presente na representação original da informação. Ao identificar e eliminar padrões repetitivos, estruturas previsíveis ou codificações desnecessárias, é possível representar os dados com uma quantidade menor de bits, mantendo integralmente a informação. Assim, a compressão atua aproximando o código do limite teórico mínimo imposto pela entropia.

Considere uma fonte com três símbolos e as seguintes frequências:

Símbolo	Frequência	Probabilidade
A	50	0,5
B	30	0,3
C	20	0,2

A entropia dessa fonte, em bits por símbolo, é:

$$H(X) = -(0,5 \log_2 0,5 + 0,3 \log_2 0,3 + 0,2 \log_2 0,2) \approx 1,485 \text{ bits/símbolo}$$

Após aplicar o algoritmo de Huffman, suponha que os símbolos sejam codificados da seguinte forma:

- A  $\rightarrow$  1 bit
- B  $\rightarrow$  2 bits
- C  $\rightarrow$  2 bits

O comprimento médio da codificação é:

$$L = (0,5 \times 1) + (0,3 \times 2) + (0,2 \times 2) = 1,5 \text{ bits/símbolo}$$

Portanto, a redundância é:

$$R = L - H(X) = 1,5 - 1,485 = 0,015 \text{ bits/símbolo}$$

Esse pequeno valor de  $R$  indica que o código de Huffman se aproxima do limite teórico de compressão, demonstrando alta eficiência.

É importante distinguir entre entropia por símbolo e por bit. A entropia calculada a partir da distribuição de probabilidades de uma fonte discreta refere-se, por definição, à quantidade média de informação por símbolo. No processo de compressão:

- A entropia total da informação permanece constante (nenhuma informação é perdida);
- A entropia por símbolo diminui, pois menos bits são usados por símbolo;
- A entropia por bit aumenta, indicando que cada bit carrega mais informação útil.

Esse comportamento evidencia a eficácia da compressão, que reduz o volume de dados sem comprometer o conteúdo informacional original.

## 2.2 Compressão de dados

## 2.3 Algoritmos clássicos de compressão sem perdas



# 3 Implementação



## 4 Análise comparativa



## 5 Conclusão



# Referências

ALAKUIJALA, J.; SZABADKA, Z.; VANDEVENNE, L.; FARRUGGIA, R.; KLIUCHNIKOV, E.; SZABADKAI, I.; ASSCHE, Z. V. Brotli compressed data format. In: INTERNATIONAL WORLD WIDE WEB CONFERENCES STEERING COMMITTEE. *Proceedings of the 25th International Conference on World Wide Web*. [S.l.], 2016. p. 1593–1603.

COLLET, Y. *Zstandard Compression Algorithm*. 2016. Disponível em: <<https://facebook.github.io/zstd/>>.

DeepMind. *WaveNet: A generative model for raw audio*. 2016. Disponível em: <<https://deepmind.google/research/breakthroughs/wavenet/>>.

HARTLEY, R. V. L. Transmission of information. *Bell System Technical Journal*, v. 7, n. 3, p. 535–563, 1928.

HUFFMAN, D. A. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, Institute of Radio Engineers, v. 40, n. 9, p. 1098–1101, 1952.

SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal*, v. 27, n. 3, p. 379–423, 1948.

WELCH, T. A. A technique for high-performance data compression. *Computer*, IEEE, v. 17, n. 6, p. 8–19, 1984.






ZIV, J.; LEMPEL, A. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IEEE, v. 23, n. 3, p. 337–343, 1977.

# Progresso de Junho (em andamento)






- Continuidade do desenvolvimento dos algoritmos (LZW e GZIP).
- Testes de compressão com arquivos .txt, .bmp e .wav.
- Geração de gráficos com Python.
- Consolidação de resultados parciais para análise.

# O que vem a seguir (julho em diante)

- Finalizar implementação do back-end e front-end.
- Análise estatística dos dados com Python.
- Escrita dos resultados e conclusão da monografia.

-  Alakuijala, Jyrki et al. (2016). “Brotli compressed data format”. In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, pp. 1593–1603. DOI: 10.1145/2872427.2883049.
-  Collet, Yann (2016). *Zstandard Compression Algorithm*. URL: <https://facebook.github.io/zstd/>.
-  DeepMind (2016). *WaveNet: A generative model for raw audio*. URL: <https://deepmind.google/research/breakthroughs/wavenet/>.
-  Deutsch, Peter (1996). *GZIP file format specification version 4.3*. URL: <https://tools.ietf.org/html/rfc1952>.
-  Hartley, Ralph V. L. (1928). “Transmission of Information”. In: *Bell System Technical Journal* 7.3, pp. 535–563. DOI: 10.1002/j.1538-7305.1928.tb01236.x.

# Referências II

-  Huffman, David A. (1952). "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9, pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.
-  Salomon, David and Giovanni Motta (2007). *Data Compression: The Complete Reference*. 4th. Springer. ISBN: 978-1-84628-602-5.
-  Shannon, Claude E. (1948). "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27.3, pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
-  Welch, Terry A (1984). "A technique for high-performance data compression". In: *Computer* 17.6, pp. 8–19. DOI: 10.1109/MC.1984.1659158.
-  Ziv, Jacob and Abraham Lempel (1977). "A universal algorithm for sequential data compression". In: *IEEE Transactions on Information Theory* 23.3, pp. 337–343. DOI: 10.1109/TIT.1977.1055714.