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

Gustavo Yujii Silva Kadooka

Universidade Estadual Paulista - Câmpus Bauru

2025

Sumário

Introdução - O que é compressão?

A compressão de dados é uma técnica essencial para otimizar o uso de recursos computacionais, **reduzindo o tamanho dos arquivos** e **acelerando a transmissão de dados**.
Com a crescente quantidade de dados gerados, é fundamental usar algoritmos eficientes para garantir que o armazenamento e a transferência ocorram de forma rápida e econômica.

Problema

Como os diferentes algoritmos clássicos de compressão de dados (Huffman, LZ77, LZW, GZIP) se comparam em termos de eficiência de compressão e tempo de execução, e qual é o impacto dessas variáveis em aplicações práticas?

Hipótese

A hipótese inicial é que os algoritmos apresentam diferenças significativas em termos de tempo de execução e taxa de compressão dependendo do formato de arquivo.

Delimitação

O estudo se concentra em algoritmos de compressão sem perdas (*lossless*) aplicados a arquivos de texto (`.txt`), imagens (`.bmp`) e áudio (`.wav`).

unesp

Por que `.txt`?

O formato de arquivo `.txt` contém apenas o texto em si, sem qualquer estrutura adicional ou informações de controle, como cabeçalhos, metadados ou formatação.

- **Sem cabeçalho**: Diferentemente de outros formatos de arquivo, como `.csv` ou `.json`, que podem incluir linhas de cabeçalho ou metadados no início do arquivo, o `.txt` não contém informações adicionais.
- **Armazenamento simples:** O conteúdo do arquivo `.txt` é simplesmente uma sequência de caracteres armazenados sequencialmente, o que facilita sua leitura e manipulação.
- **Limitações:** Não é adequado para representar dados mais estruturados, como tabelas ou documentos com informações adicionais.

Por que `.wav`?

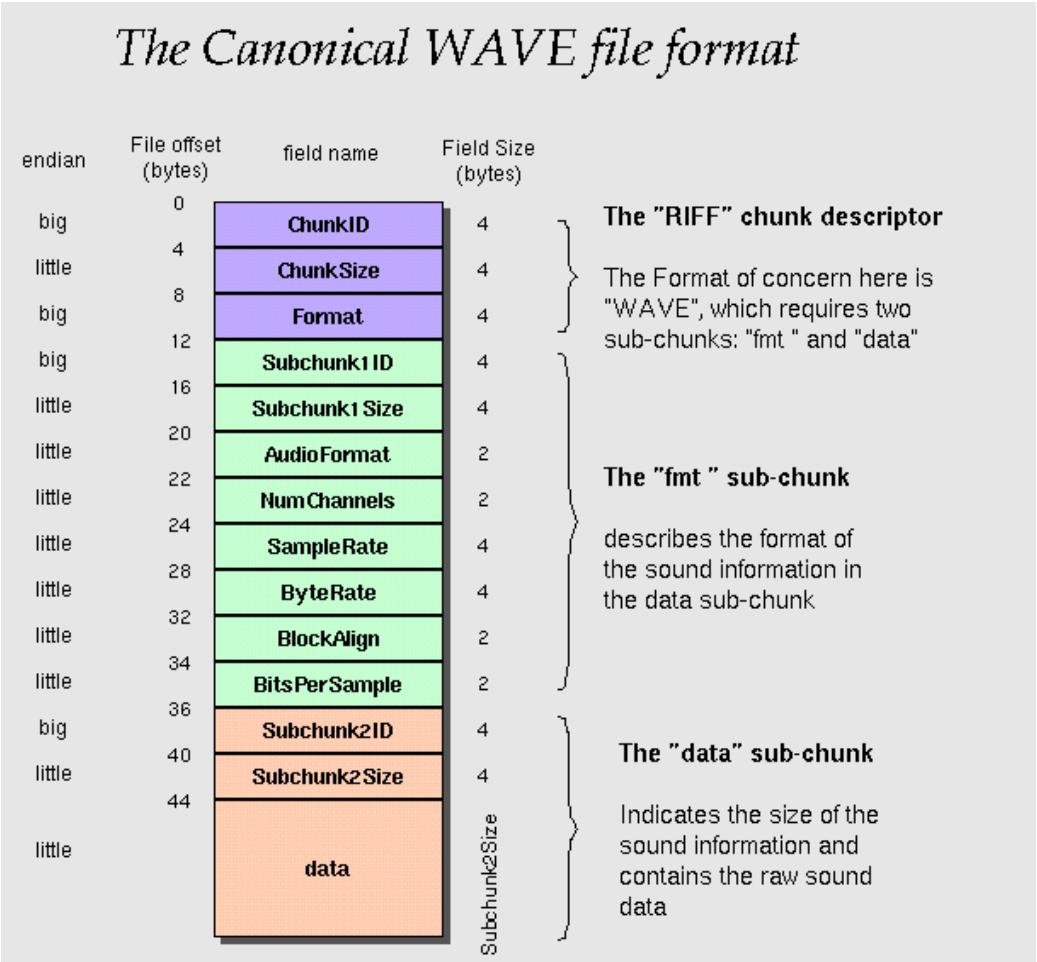
O `.wav` (*Waveform Audio File Format*) é um formato de áudio que geralmente não realiza compressão de dados, sendo considerado um formato de áudio sem perdas (*lossless*).

- Armazena dados de áudio em formato bruto.
- Preserva a qualidade original do áudio sem nenhuma perda de informações.
- Frequentemente usado para gravações de alta qualidade e produção de áudio profissional.
- Embora o formato em si não utilize compressão, ele pode ser comprimido usando algoritmos como FLAC ou ALAC para redução de tamanho.

unesp

Header do `.wav`

Figura 1: *Header* do `.wav`



Fonte: Retirada de Craig Sapp n.d.

Por que `.bmp`?

O formato `.bmp` (*Bitmap*) é um formato de imagem simples que armazena a imagem em forma de mapa de bits, geralmente sem compressão.

- Cada pixel da imagem é armazenado individualmente, sem qualquer tipo de compressão de dados.
- Permite representar imagens em alta qualidade, sem perdas de detalhes ou distorções.
- O arquivo `.bmp` pode ser muito grande, já que não realiza compressão para reduzir o tamanho do arquivo.
- Embora o formato `.bmp` seja sem perdas, ele tende a ser menos eficiente em termos de tamanho de arquivo em comparação com outros formatos como `.png` (também *lossless*, mas com compressão).

Header do `.bmp`

Figura 2: *Header* do `.bmp`

Basic BMP File Format			
Name		**Size**	**Description**
Header		14 bytes	Windows Structure: BITMAPFILEHEADER
	Signature	2 bytes	'BM'
	FileSize	4 bytes	File size in bytes
	reserved	4 bytes	unused (=0)
	DataOffset	4 bytes	File offset to Raster Data
InfoHeader		40 bytes	Windows Structure: BITMAPINFOHEADER
	Size	4 bytes	Size of InfoHeader =40
	Width	4 bytes	Bitmap Width
	Height	4 bytes	Bitmap Height
	Planes	2 bytes	Number of Planes (=1)
	BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 (?) 24 = 24bit RGB. NumColors = 16M
	Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
	ImageSize	4 bytes	(compressed) Size of Image It is valid to set this =0 if Compression = 0
	XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter
	YpixelsPerM	4 bytes	vertical resolution: Pixels/meter
	ColorsUsed	4 bytes	Number of actually used colors
	ColorsImportant	4 bytes	Number of important colors 0 = all
ColorTable		4 * NumColors bytes	present only if Info.BitsPerPixel <= 8 colors should be ordered by importance
	Red	1 byte	Red intensity
	Green	1 byte	Green intensity
	Blue	1 byte	Blue intensity
	reserved	1 byte	unused (=0)
	repeated NumColors times		
Raster Data		Info.ImageSize bytes	The pixel data

Fonte: Retirada de Gdansk University of Technology n.d.

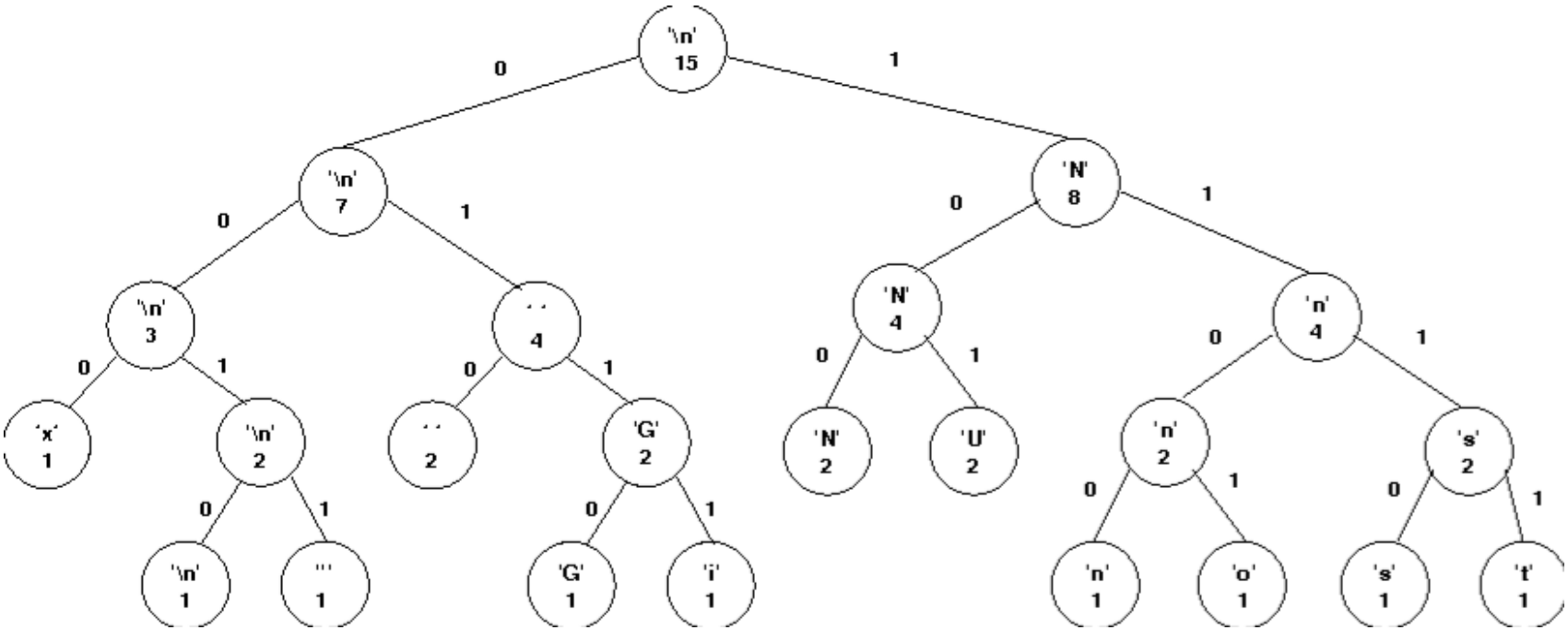
Entendendo Huffman

Técnica de compressão de dados sem perdas (*lossless*) amplamente utilizada. Ele é baseado na construção de uma árvore binária, onde os símbolos com maior frequência de ocorrência são codificados com códigos mais curtos, enquanto os símbolos menos frequentes recebem códigos mais longos.

- Contar as repetições.
- Inserir ordenado numa lista.
- Criar a árvore de Huffman.
- Criar um dicionário.
- Codificar
- Comprimir e Descomprimir.

Entendendo Huffman

Figura 3: Árvore de Huffman



Fonte: Retirada de Islene Calciolari Garcia 2007

Entendendo LZ77 - Lempel-Ziv 77

O LZ77 é um algoritmo de compressão sem perdas que utiliza um dicionário para substituir sequências repetidas de caracteres. Ele foi desenvolvido por Abraham Lempel e Jacob Ziv em 1977 e é amplamente utilizado em algoritmos de compressão como ZIP e GZIP.

- O algoritmo utiliza uma janela deslizante para identificar padrões repetidos.
- O dicionário contém as sequências de caracteres já vistas, e, quando uma sequência é repetida, é armazenada uma referência para a posição e o comprimento da sequência anterior.
- A compressão é feita substituindo sequências repetidas por um par de números: um ponteiro para a posição da sequência anterior e o comprimento da sequência.

Entendendo LZW - Lempel-Ziv-Welch

O LZW é uma variação do algoritmo LZ77, desenvolvido por Terry Welch em 1984. Ele também é um algoritmo de compressão sem perdas, mas se diferencia pela forma como gerencia o dicionário de sequências.

- Ao invés de utilizar uma janela deslizante, o LZW constrói um dicionário dinâmico durante o processo de compressão.
- Cada sequência de caracteres é associada a um índice no dicionário, e as sequências repetidas são substituídas pelo índice correspondente.
- O LZW é amplamente utilizado em formatos de compressão como GIF e TIFF.

unesp

Entendendo GZIP

O GZIP é uma ferramenta e formato de compressão de dados que combina o algoritmo LZ77 com a codificação de Huffman para compressão adicional.

- O GZIP utiliza a técnica de compressão LZ77 para remover sequências repetidas de caracteres.
- Em seguida, aplica a codificação de Huffman para otimizar a representação dos símbolos restantes, atribuindo códigos menores aos símbolos mais frequentes.
- O GZIP é amplamente utilizado para comprimir arquivos em sistemas Unix e é frequentemente usado para a compressão de arquivos de texto, como logs e arquivos HTML.

Por que os algoritmos clássicos?

Algoritmos mais recentes utilizam dos algoritmos clássicos

- **Brotli**: desenvolvido pelo Google, usado principalmente para a compressão de conteúdo da web. Combina técnicas, incluindo o LZ77 e o Huffman, assim como o uso de árvores de Huffman dinâmicas e modelagem estatística
- **Zstandard**: desenvolvido pelo Facebook para compressão em tempo real. Combina técnicas de compressão, incluindo o LZ77 e o Huffman.
- **LZMA**: LZMA é o algoritmo por trás do formato de arquivo 7z usado pelo 7-Zip. Ele é uma versão aprimorada do LZ77 e usa uma codificação de Huffman para compressão adicional.

unesp

Justificativa

Justifica-se pela necessidade de compreender de forma aprofundada como diferentes algoritmos clássicos de compressão se comportam em termos de taxa de compressão e tempo de execução, analisando sua aplicabilidade em diferentes cenários.
A escolha do algoritmo de compressão adequado afeta diretamente o desempenho de sistemas de armazenamento e comunicação de dados, impactando desde a velocidade de transferência até o uso de recursos em nuvem.

Contribuição

Este estudo oferece uma análise comparativa, ajudando na escolha do algoritmo ideal para diferentes cenários de uso baseados no formato de arquivo.

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.

Como analisar a eficiência?

Taxa de compressão

$$T = \frac{T_{\text{original}} - T_{\text{comprimido}}}{T_{\text{original}}}$$

Taxa de compressão média

$$T_{\text{média}} = \frac{1}{n} \sum_{i=1}^{n} \frac{T_{\text{original},i} - T_{\text{comprimido},i}}{T_{\text{original},i}}$$

Como analisar a eficiência?

Gráficos

- **Barras**:
  - Eixo X: representa o **tipo de arquivo** (`.txt`, `.bmp`, `.wav`)
  - Eixo Y: a **taxa de compressão média (%)**.

  Cada grupo de barras representa um algoritmo de compressão (Huffman, LZ77, LZW, GZIP).
- **Boxplots**:
  - Eixo X: representa o **tipo de arquivo** (`.txt`, `.bmp`, `.wav`)
  - Eixo Y: a **taxa de compressão média (%)**.
- **Dispersão** (*Scatter Plot*):
  - Eixo X: **tamanho do arquivo (KB)**.
  - Eixo Y: **taxa de compressão média (%)**.
- **Linha**:
  - Eixo X: **tamanho do arquivo (KB)**.
  - Eixo Y: **taxa de compressão média (%)**.

Como analisar a eficiência?

Com os gráficos de dispersão e de linha, podemos traçar uma **Regressão Linear** para ver se é aplicável.

$$Y = \beta_0 + \beta_1 \cdot X + \epsilon$$

Coeficiente de correlação de Pearson

$$r = \frac{n\sum xy - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}}$$

- r = 1: Correlação positiva perfeita.
- r = -1: Correlação negativa perfeita.
- r = 0: Nenhuma correlação linear.
- 0 ¡ r ¡ 1: Correlação positiva.
- -1 ¡ r ¡ 0: Correlação negativa.

Como analisar a eficiência?

Teste de normalidade de Shapiro-Wilk

- Hipotese nula (H_0): Os dados seguem uma distribuição normal.
- Hipotese alternativa (H_1): Os dados não seguem uma distribuição normal.

Se o valor-p (p-value) do teste for menor que o nível de significância (geralmente 0,05), você rejeita a hipótese nula e conclui que os dados não são normais.

Teste t de Student

- O teste t de Student compara as médias de dois grupos independentes.
- Se o valor-p do teste t for menor que 0,05, rejeitamos a hipótese nula (não há diferença significativa entre os grupos) e concluímos que há uma diferença significativa nas taxas de compressão.

Como será a implementação?

Back-End

A implementação do *back-end* será feita utilizando a linguagem `C++`.
A análise de eficiência e geração de gráficos será feita com a linguagem `Python`.

Front-End

A implementação do *front-end* será feita utilizando a biblioteca `GTK`.

Cronograma

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

Figura 4: **Fonte:** Desenvolvido pelo autor.

Referências I

- Craig Sapp (n.d.). *WAVE PCM soundfile format*. Instituição: Stanford University. Acessado em: 06 abr. 2025. URL: http://soundfile.sapp.org/doc/WaveFormat/.
- Gdansk University of Technology (n.d.). *VGA File Format Structure in BMP*. Instituição: Gdansk University of Technology. Acessado em: 06 abr. 2025. URL: http://www.ue.eti.pg.gda.pl/fpgalab/zadania.spartan3/zad_vga_struktura_pliku_bmp_en.html.
- Islene Calciolari Garcia (2007). *Laboratório 4 - Compactação de Arquivos e Algoritmo de Huffman*. Instituição: Universidade Estadual de Campinas. Acessado em: 06 abr. 2025. URL: https://www.ic.unicamp.br/~islene/mc202/lab4/lab4.html.

unesp