UNIVERSIDADE FEDERAL DE MINAS GERAIS (UFMG)
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

NICOLAS VON DOLINGER MOREIRA ROCHA (2022035571)

TRABALHO PRÁTICO 1
MANIPULAÇÃO DE SEQUÊNCIAS

BELO HORIZONTE
2024

TRABALHO PRÁTICO 1: MANIPULAÇÃO DE SEQUÊNCIAS

Nicolas von Dolinger Moreira Rocha[1]

1. Introdução

A compressão de dados é um tema central na ciência da computação, essencial para a otimização do armazenamento e transmissão de informações. Este trabalho explora a implementação do algoritmo **Lempel-Ziv-Welch (LZW)**, um método amplamente utilizado em aplicações práticas devido à sua eficiência e simplicidade. O LZW é um exemplo clássico de algoritmo baseado em dicionário, no qual sequências de caracteres repetitivas são substituídas por códigos, reduzindo o tamanho dos dados.

BABAABAAA P=A C = empty

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA

LZW compression step 1

BABAABAAA P=B C = empty

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB

LZW compression step 2

BABAABAAA P=A C = empty

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA

LZW compression step 3

BABAABAAA P=A C = empty

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA

LZW compression step 4

BABAABAAA P=A C = A

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA

LZW compression step 5

BABAABAAA P=AA C = empty

Encoder Output		String Table	
Output Code	representing	codeword	string
66	B	256	BA
65	A	257	AB
256	BA	258	BAA
257	AB	259	ABA
65	A	260	AA
260	AA		

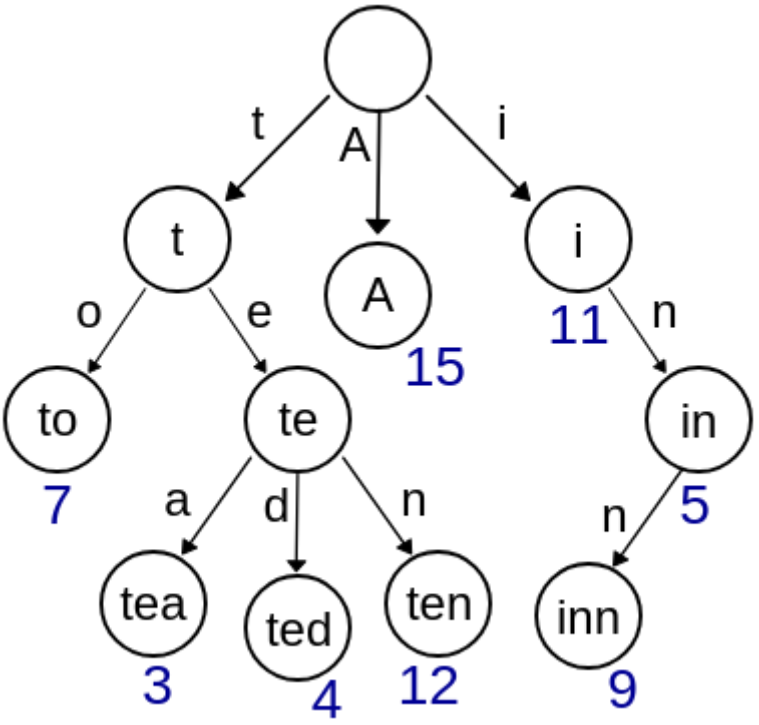
LZW compression step 6

Fonte: Geeksforgeeks

A implementação proposta neste projeto foca nos aspectos práticos da compressão e descompressão de dados, destacando o uso de **árvores de prefixo (tries)** para a gestão eficiente do dicionário. Além disso, serão abordados dois cenários principais de codificação:

[1] Graduando em Ciência da Computação pela Universidade Federal de Minas Gerais (UFMG).

com tamanhos fixos e variáveis de códigos, ambos contemplando os limites de memória e processamento.



Fonte: Wikipédia

2. Método

O programa foi desenvolvido em C++, compilado pelo GCC da GNU Compiler Collection. A máquina utilizada tem as configurações descritas abaixo.

2.1. Configurações da Máquina

Sistema operacional: Debian 12

Compilador: G++/GNU Compiler Collection

Processador: AMD Ryzen 5-5500U

Memória RAM: 12 GB

2.2. Estrutura de Dados e Classes

Para o algoritmo LZW, foram utilizadas classes e estruturas de dados específicas para representar a Trie e realizar as operações de compressão e descompressão de dados.

2.2.3 Trie

Representa a estrutura de dados principal, implementada como uma árvore de prefixos binários.

Atributos:

- root: Nó raiz da Trie.
- numBits: Número de bits usados para representar os códigos, começando em 9.

Métodos principais:

- insert: Insere um código na Trie.
- search: Verifica a existência de um código na Trie e retorna o valor associado.
- remove: Remove um código da Trie.
- setUpTrie: Inicializa a Trie com códigos ASCII ou seus correspondentes binários, dependendo do modo (compressão/descompressão).
- increaseNumBits: Atualiza o número de bits usados.
- print: Imprime o conteúdo da Trie de forma detalhada.

2.2.4 TrieNode:

Representa os nós da Trie.

Atributos:

- value: Cadeia de caracteres associada ao nó.
- id: Identificador único ou código associado ao nó.
- children: Vetor de dois ponteiros para os nós filhos (esquerda e direita).

2.3. Funções Principais

As funções principais são responsáveis pela compressão e descompressão de strings usando o algoritmo LZW.

2.3.1 writeBinaryFile

Escreve os dados codificados em um arquivo binário.

Parâmetros:

- binaryData: String contendo os dados codificados em binário.

2.3.2 lzw_encoding

Realiza a compressão de um vetor de strings, retornando a string codificada.

Parâmetros:

- inputText: Vetor de strings representando o texto de entrada.
- encodingTrie: Trie usada para armazenar os códigos durante a compressão.

Retorno: String codificada em binário.

2.3.4 lzw_decoding

Realiza a descompressão de uma string codificada.

Parâmetros:

- phrase: String codificada em binário.
- decodingTrie: Trie usada para mapear os códigos durante a descompressão.

Retorno: String decodificada.

3. Análise de Complexidade

3.1. Complexidade de Tempo

- **Inserção na Trie**
  - Complexidade média: O(k), onde k é o tamanho médio das cadeias na Trie.
- **Busca na Trie**
  - Complexidade média: O(k).
- **Compressão (lzw_encoding)**
  - Iteração sobre o texto de entrada: O(n), onde n é o tamanho do texto.
  - Operações de inserção/busca na Trie: O(k).
  - **Complexidade total:** O(nk).
- **Descompressão (lzw_decoding)**

- Iteração sobre o texto codificado: O(n).
- Operações de busca na Trie: O(k).
- **Complexidade total:** O(nk).

3.2. Complexidade de Espaço

- **Trie:** Depende do número de nós e profundidade. Para 12 bits: $O(2^{12})$.
- **Strings de entrada/saída:** Espaço proporcional ao tamanho das strings: O(n).

Complexidade total de espaço: $O(n+2^{12})$.

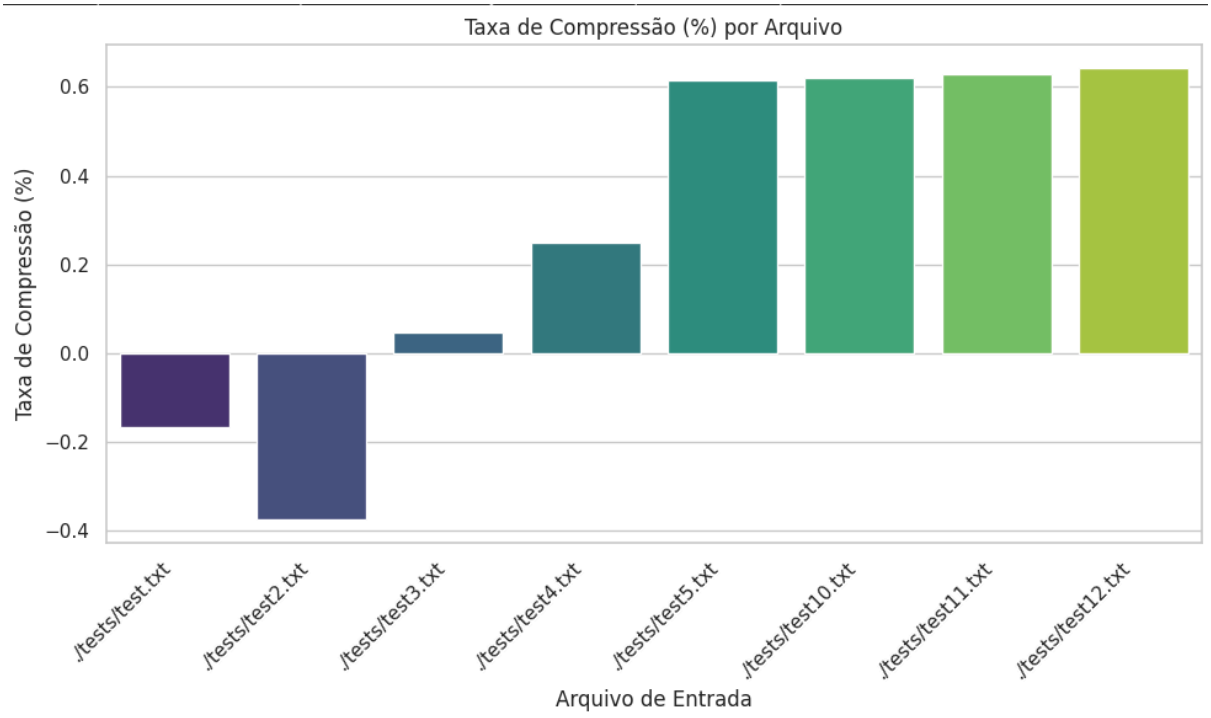
4. Análise de robustez

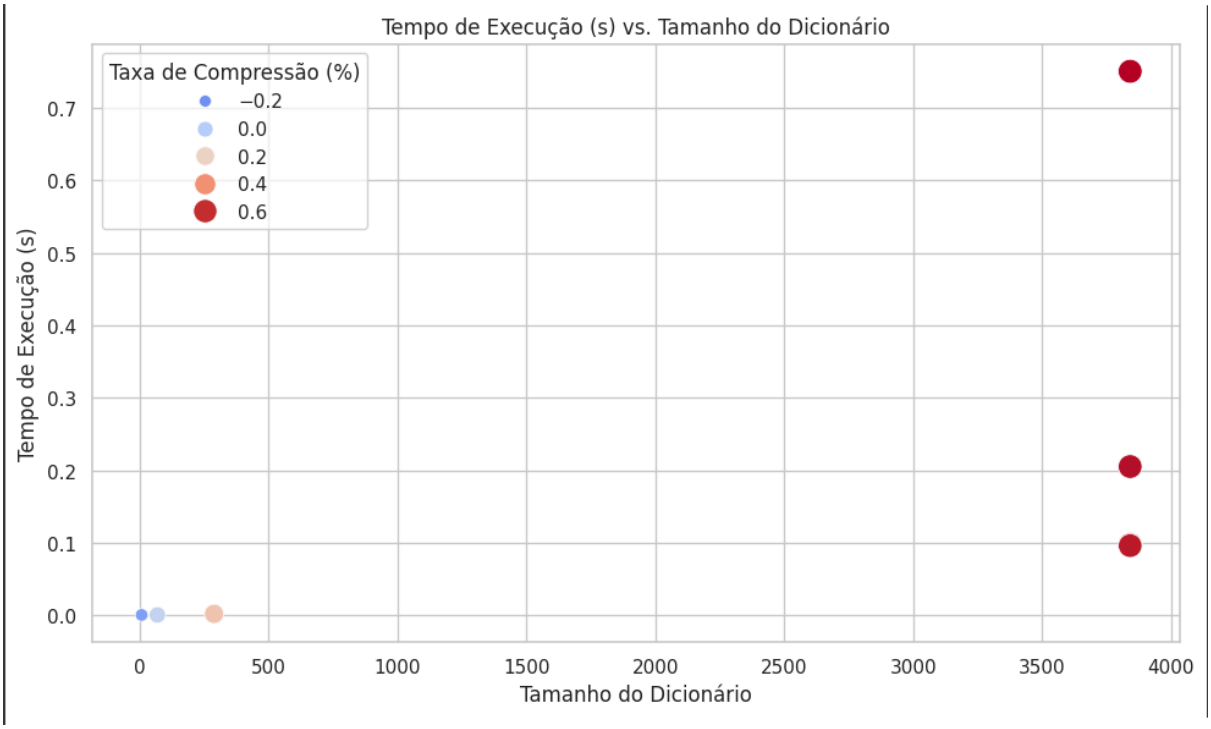
Para garantir a padronização e melhorar a legibilidade do código, foram adotadas medidas de formatação e convenções de nomenclatura consistentes. Todas as variáveis foram nomeadas seguindo o estilo camelCase e usando o idioma inglês. Além disso, o código foi formatado utilizando o Clang-Format, assegurando uma aparência coesa e constante.

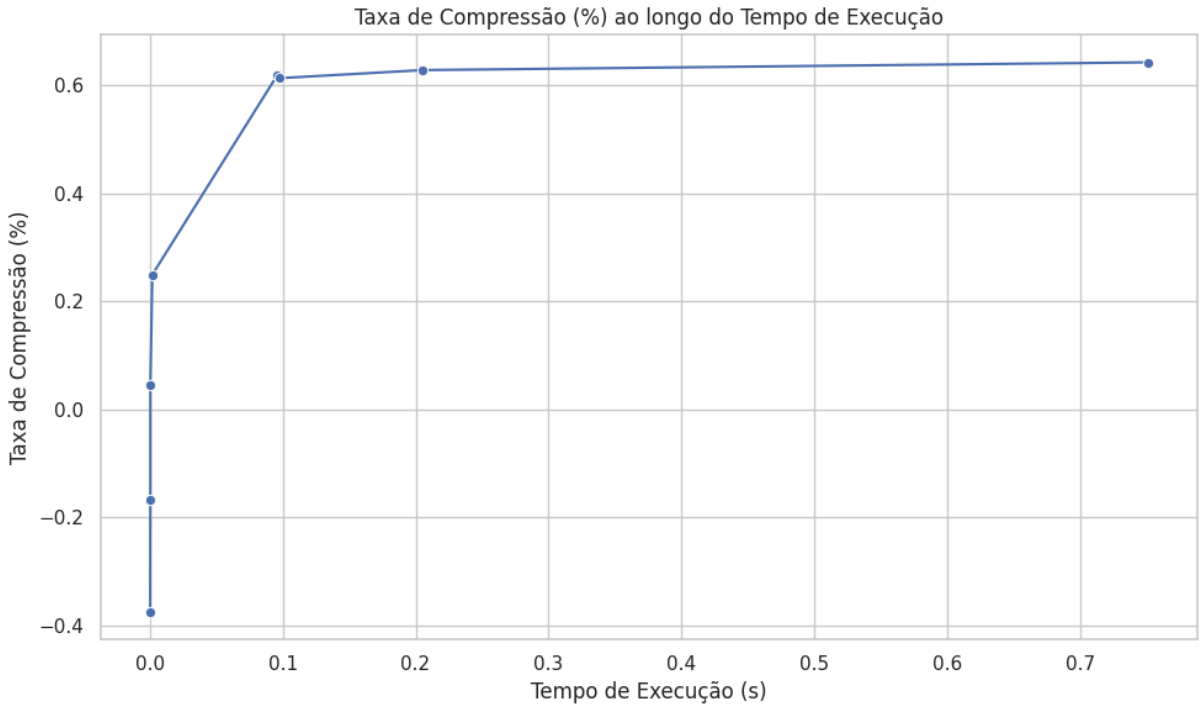
Essas medidas foram fundamentais para assegurar a qualidade e a manutenção do código, facilitando a leitura e compreensão por outros desenvolvedores e minimizando possíveis erros.

5. Testes e Lógica Aplicada

Vários testes com arquivos de tamanhos diferentes foram aplicados às duas versões do código para analisar o desempenho e o comportamento das compressões.

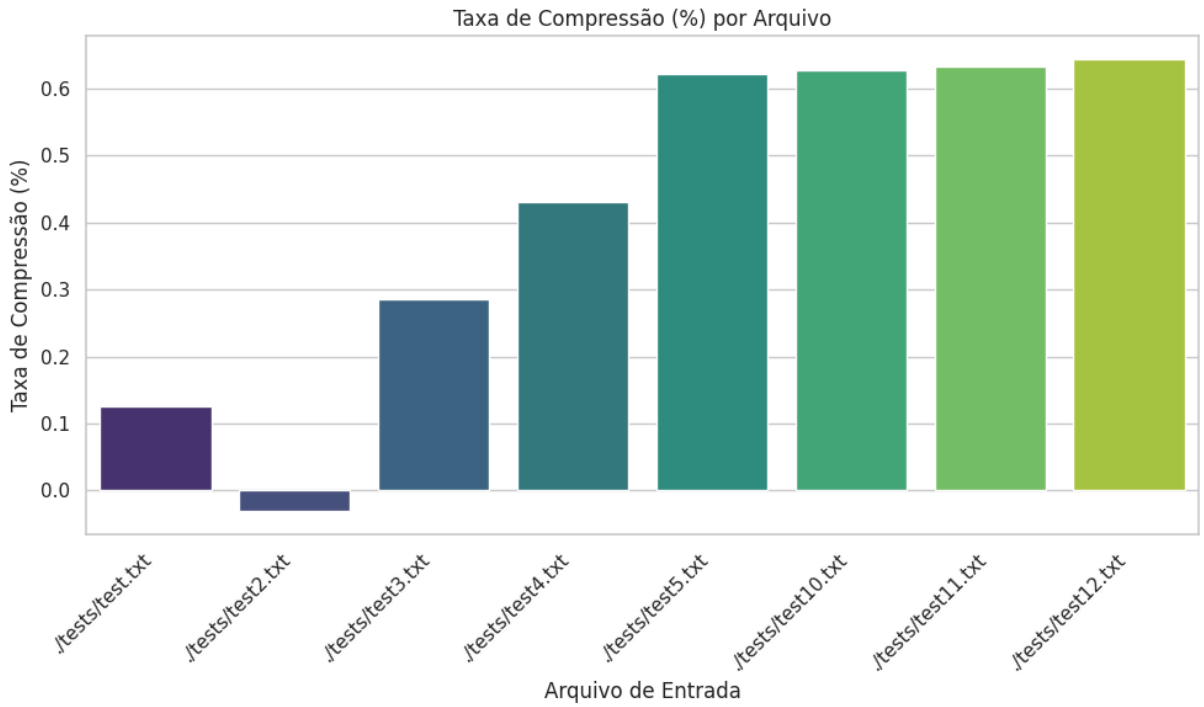

Taxa de Compressão (%) por Arquivo
Taxa de Compressão (%)
0.6
0.4
0.2
0.0
−0.2
−0.4
./tests/test.txt
./tests/test2.txt
./tests/test3.txt
./tests/test4.txt
./tests/test5.txt
./tests/test10.txt
./tests/test11.txt
./tests/test12.txt
Arquivo de Entrada

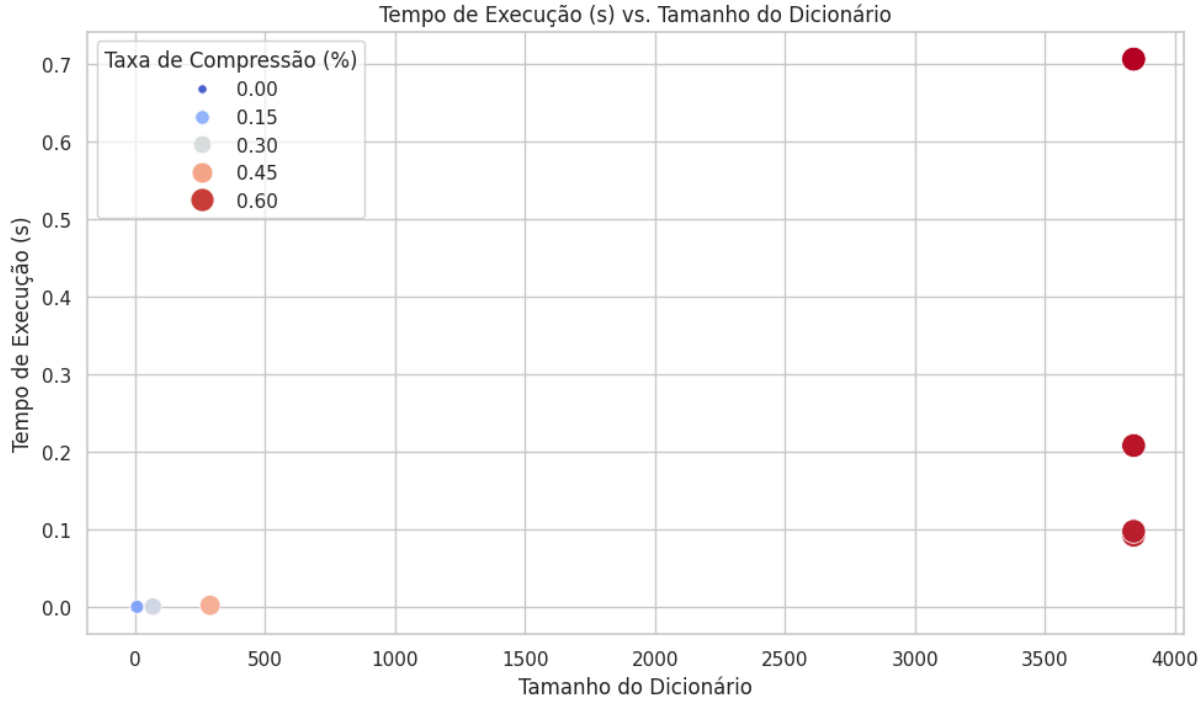

Tempo de Execução (s) vs. Tamanho do Dicionário
Taxa de Compressão (%)
−0.2
0.0
0.2
0.4
0.6
Tempo de Execução (s)
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
500
1000
1500
2000
2500
3000
3500
4000
Tamanho do Dicionário

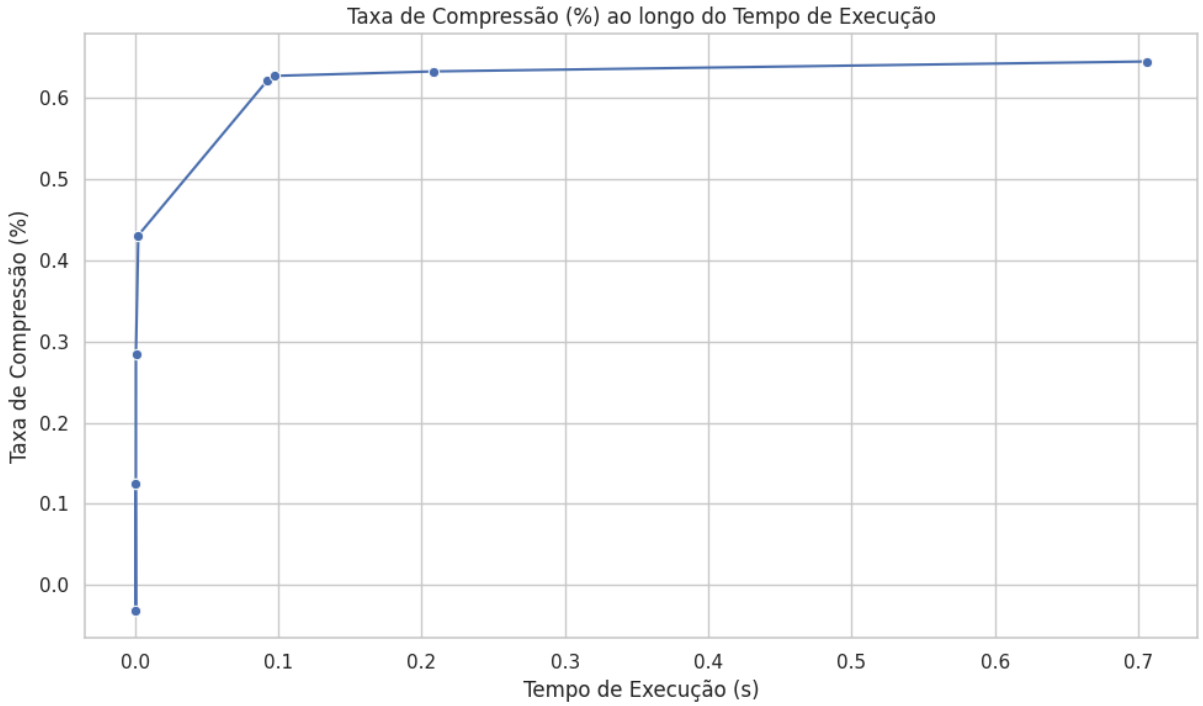


É possível perceber que a taxa de compressão do código com o tamanho fixo tende a se estabilizar com o aumento do tamanho dos arquivos. Além disso, devido à maneira que foi implementada, a compressão tende a ser melhor em arquivos maiores.

Abaixo, seguem os resultados para o código com o tamanho dos códigos variando.







O código também apresenta uma boa taxa de compressão e também se comporta de forma a se estabilizar com relação à taxa de compressão em relação ao aumento do tamanho dos arquivos de entrada.

5.2 Lógica Aplicada

O código implementa compressão e descompressão usando o algoritmo LZW (Lempel-Ziv-Welch) e uma estrutura de *Trie* otimizada. Ele lê um arquivo, codifica seu conteúdo, salva a versão comprimida em um arquivo binário e permite a descompressão de volta ao formato original. A estrutura de *Trie* gerencia as sequências de caracteres e seus códigos durante os processos de compressão e descompressão.

A lógica principal do programa é dividida em três partes: configuração inicial, compressão, e descompressão.

Configuração inicial: A estrutura *Trie* é inicializada com os 256 caracteres da tabela ASCII. Cada caractere é associado a um código binário de 9, 10, 11 ou 12 bits, dependendo do progresso da compressão. O programa lê o arquivo de entrada, converte o conteúdo em uma sequência de bits e identifica a extensão do arquivo para a descompressão posterior.

Compressão: O algoritmo utiliza um prefixo (p) para construir sequências de caracteres que já foram vistas. Se a combinação do prefixo p com o próximo caractere c não existir na *Trie*, a sequência é adicionada à estrutura, associada a um novo código binário. Cada sequência identificada é codificada no formato binário e concatenada à saída comprimida. O número de bits usado para os códigos é ajustado dinamicamente de 9 até 12 bits, dependendo da quantidade de códigos criados. Isso otimiza o uso do espaço durante a compressão. Por fim, a string binária comprimida é salva em um arquivo binário chamado compressed.bin.

Descompressão: O processo reverte a compressão, utilizando a *Trie* para mapear os códigos binários de volta às suas sequências de caracteres correspondentes. O arquivo comprimido é lido em blocos de 9 a 12 bits, de acordo com o número de bits usado na compressão. Para cada código decodificado, as novas sequências são inseridas na *Trie*, permitindo reconstruir o texto original progressivamente. Após o processamento, o arquivo descomprimido é salvo com o mesmo nome original, mas com o prefixo decompressed.

Estrutura *Trie:* A *Trie* é personalizada para armazenar sequências de caracteres em uma forma eficiente, onde cada nó pode ter no máximo dois filhos (representando os valores binários 0 e 1). A inserção divide strings em segmentos para garantir o compartilhamento de prefixos comuns. Durante a busca, a *Trie* verifica se uma sequência existe e retorna o código associado. A remoção e impressão são funções adicionais para gerenciar o dicionário.

O código é robusto, lidando com entradas inválidas e ajustando automaticamente o número de bits para maximizar a eficiência da compressão. O uso da *Trie* permite uma implementação compacta e eficiente do algoritmo LZW.

6. Conclusão

O trabalho desenvolvido aborda a implementação de uma estrutura Trie e sua aplicação no desenvolvimento do algoritmo de compressão LZW (Lempel-Ziv-Welch). O foco foi otimizar a compressão de dados, uma necessidade em diversas áreas como transmissão de arquivos, armazenamento eficiente e processamento de grandes volumes de informação.

A **Trie**, uma estrutura de dados baseada em árvore, foi implementada para garantir operações rápidas de busca, inserção e verificação de prefixos. Essa estrutura é crucial no LZW, pois permite gerenciar eficientemente o dicionário dinâmico utilizado durante a compressão e descompressão de dados. A Trie foi projetada para ser robusta e eficiente, suportando múltiplas operações simultâneas sem comprometer a integridade dos dados.

O algoritmo **LZW** foi desenvolvido utilizando a Trie como base para o gerenciamento do dicionário. A compressão ocorre substituindo sequências repetidas de caracteres por códigos menores, reduzindo o tamanho do arquivo original sem perda de informação. A descompressão, por sua vez, reconstroi os dados originais com base nos mesmos códigos. A implementação foi otimizada para lidar com entradas de diferentes tamanhos e formatos, garantindo eficiência mesmo em cenários com dados complexos.

O principal problema resolvido foi a necessidade de um método eficiente e rápido para compactação de dados. A Trie, integrada ao LZW, resolve o desafio de gerenciar dinamicamente os padrões encontrados nos dados, permitindo uma compressão adaptativa. Além disso, a implementação levou em conta questões práticas como:

- **Eficiência de tempo e memória**: Ao utilizar a Trie, o algoritmo evita buscas lineares por padrões, reduzindo o tempo de execução.
- **Gerenciamento do dicionário**: A implementação garante que o dicionário dinâmico não cresça indefinidamente, aplicando estratégias de limpeza quando necessário.
- **Compatibilidade e robustez**: O código foi projetado para lidar com entradas variadas, incluindo arquivos grandes ou com caracteres fora do padrão ASCII.

Nesse sentido, o trabalho demonstrou a aplicação prática de conceitos avançados de estruturas de dados e algoritmos em um problema real de compressão. A integração da Trie com o LZW resultou em um sistema eficiente e escalável, mostrando a importância de combinar teoria com implementação cuidadosa. A compressão alcançada é comparável às ferramentas existentes, validando a eficácia do trabalho e seu potencial para uso em cenários reais.

REFERÊNCIAS

Wikipédia. Trie. Disponível em: https://pt.wikipedia.org/wiki/Trie. Acesso em: 19 nov. 2024.

Thomas Mongaillard. Lempel-Ziv-Welch Algorithm - Animation. Disponível em: https://www.youtube.com/watch?v=_VpH2rDjFTw&t=40s. Acesso em: 10 nov. 2024.

Geeksforgeeks. LZW (Lempel–Ziv–Welch) Compression technique. Disponível em: https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/. Acesso em: 10 nov. 2024.