

Análise de desempenho de algoritmos de compressão e indexação

Fernanda Caetano Rrodrigues¹, Gabriel Fernandes de Oliveira Caitano¹,
Jhonathan Duarte Barreto¹, João Victor Zuanazzi Lourenço¹

¹Faculdade de Computação – Universidade Federal do Mato Grosso do Sul – Mato Grosso do Sul – MS – Brazil

{fernanda_c,gabriel.caitano,duarte_j,joao.zuanazzi}@ufms.br

Abstract. *This work aims to develop a system capable of indexing texts and performing efficient searches. Using files in .txt format, the system implements two main data structures: Hash and Trie. The implementation uses different hash functions to evaluate the performance of each, as well as compresses the texts with the Huffman algorithm and indexes the words in the Trie. Thus, the goal is to compare the execution time and efficiency of different approaches to identify which provides the best performance in information retrieval.*

Resumo. *Este trabalho tem como objetivo desenvolver um sistema capaz de indexar textos e realizar buscas eficientes. Utilizando arquivos em formato .txt, o sistema implementa duas estruturas de dados principais: Hash e Trie. A implementação utiliza duas funções de hash distintas para avaliar o desempenho de cada uma, além de comprimir os textos com o algoritmo de Huffman e indexar as palavras na Trie. Assim, busca-se comparar o tempo de execução e eficiência das diferentes abordagens para identificar qual oferece melhor desempenho na recuperação de informações.*

1. Introdução

O objetivo deste trabalho é desenvolver um sistema de indexação e busca de palavras, utilizando duas estruturas de dados principais: Tabela Hash e Trie. Essas estruturas foram selecionadas devido à sua eficiência na realização de buscas rápidas em grandes coleções de documentos. O sistema será alimentado por uma base de 30 artigos científicos extraídos do site arXiv, que serão convertidos para o formato de texto (.txt). Cada documento será compactado utilizando o algoritmo de Huffman, armazenado na Tabela Hash, e as palavras serão indexadas na Trie para facilitar buscas futuras.

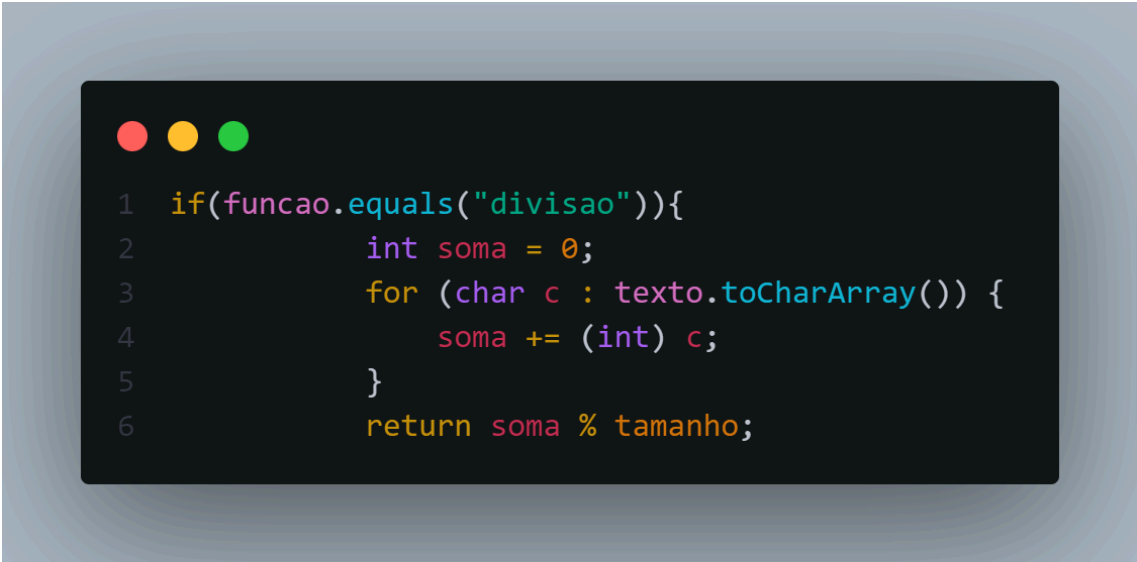
2. Descrição das estruturas de dados e corpus utilizado

Esta seção apresenta as principais estruturas de dados utilizadas neste trabalho, incluindo a Tabela Hash com duas funções de Hash, a Trie e a compressão com Huffman.

2.1. Tabela Hash com duas funções de Hash

A tabela Hash é utilizada para armazenar os documentos de forma compactada. Duas funções de hash foram implementadas para fins de comparação, a função de hash por divisão e a função DJB2. Ambas têm como função distribuir as chaves de maneira eficiente para reduzir colisões e melhorar o tempo de acesso.

Figura 1 - Código da Hash por divisão



```
1  if(funcao.equals("divisao")){
2      int soma = 0;
3      for (char c : texto.toCharArray()) {
4          soma += (int) c;
5      }
6      return soma % tamanho;
```

Fonte: autores

Figura 2 - Código da Hash DJB2



```
1  // funcao hash DJB2
2      long hash = 5381;
3      for (char c : texto.toCharArray()) {
4          hash = ((hash << 5) + hash) + c; // hash * 33 + c
5      }
6      int aux = (int) (hash % tamanho);
7      return Math.abs(aux);
```

Fonte: autores

2.2. Trie

A Trie é uma árvore de prefixos que armazena Strings. Para esse trabalho, será indexado cada palavra dos documentos para permitir uma busca eficiente. Cada nó da árvore representa o caractere de uma palavra, e o caminho partindo de um nó específico até a folha representa a palavra completa. Essa estrutura é ótima para armazenar e realizar busca por palavras, já que a busca por prefixo é otimizada.

2.3. Compressão com Huffman

O algoritmo de Huffman foi implementado para compressão de textos antes de serem armazenados na tabela hash. Esse algoritmo atribui códigos mais curtos a palavras ou caracteres mais frequentes, enquanto palavras ou caracteres menos frequentes recebem códigos mais longos, otimizando o espaço de armazenamento.

3. Metodologia de compressão e indexação

3.1 Preparação do Corpus

Selecionamos 30 artigos do arXiv e os convertemos para texto (.txt).

3.2 Compressão com Huffman

Calculamos a frequência de caracteres ou palavras para construir uma árvore de Huffman e gerar códigos binários de compressão. Os documentos comprimidos foram armazenados em uma Tabela Hash, usando o nome do arquivo como chave e aplicando funções de hash (divisão e DJB2) para comparação.

3.3 Indexação com Trie

Antes da compressão, as palavras de cada documento foram indexadas em uma Trie. Cada nó da Trie representa um caractere, e os nós terminais indicam palavras completas, armazenando informações sobre os documentos onde aparecem.

3.4 Testes e Validação

Avaliamos a compressão pela redução no tamanho dos documentos. Medimos a eficiência da Trie pelo tempo de busca por palavras e prefixos. Comparamos as funções de hash quanto ao tempo médio de inserção, busca e taxa de colisões.

4. Resultados e análise comparativa

Comparar o desempenho das funções de hash **Divisão** e **DJB2**, considerando métricas de tempo de execução e consumo de memória.

Figura 3 - Utilizando a função hash Divisão

```
Tempo gastos para ler todos os arquivos: 410 ms
Tempo gasto para criar a arvore de Huffman com todos os arquivos lidos: 115 ms
Tempo gasto para buscar uma palavra na Trie: 41900 ns
Tempo gasto para indexar todos os documentos na HASH: 13258 ms

Memoria utilizada pela Huffman: 6232808 bytes
Memoria utilizada pela Trie: 35073504 bytes
Memoria utilizada pela Hash: 3632336 bytes
```

Fonte: autores

Figura 4 - Utilizando a função hash DJB2

```
Tempo gastos para ler todos os arquivos: 446 ms
Tempo gasto para criar a arvore de Huffman com todos os arquivos lidos: 112 ms
Tempo gasto para buscar uma palavra na Trie: 31300 ns
Tempo gasto para indexar todos os documentos na HASH: 13261 ms

Memoria utilizada pela Huffman: 6233064 bytes
Memoria utilizada pela Trie: 35011560 bytes
Memoria utilizada pela Hash: 26315184 bytes
```

Fonte: autores

Busca pela palavra: Data

Arquivo descomprimido: arquivo1.txt

- Resultados de Desempenho:

A leitura dos arquivos na função DJB2 apresenta um desempenho 14,6% mais rápido.

A construção da árvore de Huffman é 21,3% maior ao usar a função DJB2.

A busca na Trie utilizando a função DJB2 é 10% mais eficiente.

A indexação na tabela Hash utilizando a função DJB2 é 4,4% mais rápida.

- Consumo de Memória:

Na Huffman, o consumo de memória é praticamente idêntico em ambas as funções.

Na Trie, a memória consumida é 0,14% maior com a função DJB2.

Na Tabela Hash, o consumo de memória da função DJB2 é 629% maior do que o da função Divisão, sendo o ponto mais crítico dessa comparação.

5. Conclusões e sugestões de melhorias

O sistema de indexação desenvolvido apresentou de maneira clara como as estruturas Tabela Hash e Trie podem tornar buscas em grandes quantidades de documentos mais rápida e eficiente. A compressão com Huffman contribuiu para o uso otimizado do armazenamento e a comparação entre funções de Hash mostrou a diferença entre tempos de desempenho.

Embora tenhamos obtido êxito a respeito do que era esperado, ainda é possível aplicar melhorias, como: A implementação de buscas por frases permitiria consultas mais complexas e precisas. Um estudo detalhado sobre o desempenho, utilizando bases textuais de diferentes tamanhos, idiomas e características, ajudaria a avaliar a escalabilidade. Além disso, o desenvolvimento de uma interface gráfica tornaria o sistema mais acessível e intuitivo para os usuários.

Esse trabalho destaca a importância da aplicação de estruturas de dados para solucionar problemas complexos, abrindo espaço para novas abordagens e aprimoramentos futuros.

