Universidade Federal de Minas Gerais
Sistemas de Informação

Thiago Silva Santos

Trabalho Prático 3

Belo Horizonte - MG
2023

1 Introdução

Um dos desafios decorrentes do acúmulo de grandes quantidades de arquivos é a necessidade de encontrar espaço suficiente para armazenar todas as informações. Mesmo com várias opções de armazenamento disponíveis, eventualmente podemos ficar sem espaço para acomodá-las. Neste caso específico, o objetivo de preservar todas as edições do jornal, uma vez que o departamento financeiro deseja reduzir a quantidade de discos rígidos utilizados para armazenar essas edições. Para alcançar essa meta, é necessário empregar o algoritmo de Huffman, que permitirá a compressão dos dados, possibilitando que ocupem menos espaço em discos rígidos.

2 Método

2.1 Estrutura de dados

No desenvolvimento deste programa, são utilizadas diversas estruturas de dados. Para atender ao requisito de contabilizar os caracteres em UTF-8, optamos por utilizar uma árvore AVL. Essa estrutura de dados possui a característica de ter nós únicos, o que nos permite implementar a funcionalidade de contador em cada nó. Sempre que identificamos que um determinado nó está presente na árvore, incrementamos o contador associado a ele. Os nós dessa árvore não diferem dos nós comuns, eles apenas contêm informações sobre si mesmos e suas folhas.

Uma classe auxiliar utilizada durante o projeto é a lista encadeada. Ela é utilizada em alguns casos específicos nos quais há a necessidade de operar uma lista com tamanho variado e para utilizar as funções de ordenação que facilitam a implementação do algoritmo de Huffman. A lista encadeada permite a inserção e remoção de forma simples em qualquer posição, o que é essencial para o processo de compressão dos dados. Além disso, sua estrutura flexível e dinâmica se adequa às necessidades do programa, contribuindo para uma implementação eficaz do algoritmo.

2.2 Classes

A classe Compactor é a responsável por lidar com a compactação e descompactação de arquivos. Ela possui diversas funções auxiliares que contribuem para seu funcionamento eficiente. Uma das funções importantes presentes na classe é a função de busca binária. Essa função é utilizada durante o processo de compactação para buscar rapidamente determinados elementos, como caracteres em uma estrutura de dados específica. Outra função relevante é a conversão de string para byte. Em muitos casos, durante o processo de compactação, é necessário converter strings em sequências de bytes para facilitar a manipulação e armazenamento dos dados. Essa conversão permite que a classe Compactor trabalhe diretamente com os bytes do arquivo, tornando o processo mais eficiente e otimizado.

Além dessas funções auxiliares, a classe Compactor possui métodos principais que executam a compactação e descompactação do arquivo. Esses métodos utilizam o algoritmos de Huffman para comprimir os dados e o inverso para descomprimi-los. A classe

também gerencia a estrutura de dados necessária para armazenar as informações durante o processo, garantindo uma correta compactação e descompactação dos arquivos.

3 Análise de Complexidade

Durante o processo de compressão, várias funções desempenham um papel importante na eficiência do algoritmo. Duas dessas funções são a count_char e o algoritmo de Huffman. Vamos analisar a complexidade dessas funções e do processo de compressão como um todo.

A função count_char é responsável por contar a frequência de ocorrência de cada caracter presente no arquivo a ser comprimido. Essa função utiliza uma árvore AVL para realizar a contagem de forma eficiente. A inserção em uma árvore AVL tem complexidade logarítmica $\Theta(\log n)$, pois a estrutura da árvore é balanceada automaticamente para garantir a altura mínima. Portanto, a função count_char possui complexidade $\Theta(n \log n)$, já que cada caractere precisa ser inserido na árvore AVL para contar sua ocorrência.

O algoritmo de Huffman é usado para determinar a árvore de codificação que será usada na compactação dos dados. Esse algoritmo tem complexidade $O(n^2)$ no pior caso e $\Omega(n)$ no melhor caso, onde n é o número de caracteres distintos presentes no arquivo. No pior caso, a complexidade quadrática ocorre quando todos os caracteres têm frequências diferentes e a árvore de Huffman precisa ser construída a partir disso. No entanto, no melhor caso, onde todos os caracteres têm a mesma frequência, a complexidade é linear.

Quanto à função de compressão em si, sua complexidade é linear $\Theta(n)$, pois ela precisa percorrer cada caractere do arquivo uma única vez e aplicar a codificação correspondente a partir da árvore de Huffman. No entanto, é importante mencionar que o tempo de execução pode ser afetado pela leitura e gravação simultânea no arquivo, o que pode aumentar o tempo total necessário para a compressão.

É importante ressaltar que, embora o algoritmo de Huffman tem uma complexidade assintótica de $O(n^2)$ no pior caso, sua complexidade média é consideravelmente melhor, sendo de $O(nk)$. O desempenho do algoritmo é influenciado pela distribuição das frequências dos caracteres no arquivo a ser comprimido. Caso as frequências sejam mais uniformemente distribuídas, o algoritmo terá um desempenho mais próximo da complexidade linear, enquanto distribuições desbalanceadas podem levar a um pior caso quadrático. Portanto, é fundamental considerar tanto o pior caso quanto o caso médio ao avaliar a eficiência do algoritmo de Huffman.

4 Estratégias de Robustez

Durante o processo de leitura e escrita no algoritmo de compressão e descompressão, foram adotadas estratégias para aumentar a velocidade do sistema. Uma dessas estratégias foi a utilização de um buffer de leitura. O buffer de leitura permite diminuir a quantidade de leituras diretamente do arquivo, otimizando o desempenho do programa.

No entanto, em alguns casos, quando o algoritmo atinge o fim do buffer de leitura, pode ser necessário ler os próximos caracteres. Entretanto, essa operação não é possível no momento, já que o buffer está no fim. Para tratar essa condição, é lançada uma

exceção chamada BufferOverflow, indicando que o buffer atual foi totalmente usado e que é necessário instanciar o próximo buffer em simultâneo com o anterior para continuar a leitura do arquivo. Essa abordagem garante uma transição entre os buffers, permitindo a leitura contínua dos dados.

A descompressão também apresenta desafios semelhantes que precisam ser tratados. Durante o processo de descompactação, é necessário realizar a leitura dos códigos de compressão e decodificá-los para obter os caracteres originais. Nesse caso, também pode ocorrer a necessidade de ler além do buffer atual para recuperar a sequência completa de códigos. Quando essa situação é detectada, é adotada a mesma estratégia de lançar a exceção DecompressBufferOverflow e instanciar o próximo buffer em simultâneo com o anterior, garantindo uma descompactação adequada.

Além das estratégias mencionadas anteriormente, foram implementadas exceções adicionais para lidar com possíveis problemas durante a execução do algoritmo de compressão e descompressão. Uma dessas exceções é a CouldNotOpenFile, que ocorre quando não é possível abrir um arquivo de entrada, seja ele um arquivo texto ou binário. Essa exceção é lançada para indicar que houve uma falha na abertura do arquivo especificado, seja devido a permissões insuficientes, inexistência do arquivo ou outros problemas relacionados. O lançamento dessa exceção permite ao programa tratar a situação de forma adequada, fornecendo uma mensagem de erro informativa ao usuário.

Outra exceção relevante é a FileNotUTF8, que ocorre quando o arquivo de texto fornecido para compressão não está codificado em UTF-8. A exceção é lançada para indicar que a codificação do arquivo não é suportada pelo algoritmo, já que o programa foi projetado para trabalhar com arquivos de texto codificados em UTF-8. Essa exceção permite ao programa detectar e informar ao usuário que a codificação do arquivo não é compatível, evitando problemas e resultados incorretos durante o processo de compressão.

Além disso, uma exceção adicional chamada WithoutArguments é lançada quando uma quantidade incorreta de parâmetros é fornecida para o programa. Essa exceção é útil para indicar que o programa requer uma quantidade específica de parâmetros para funcionar corretamente e que o número fornecido está incorreto. O lançamento dessa exceção permite uma verificação prévia dos argumentos fornecidos e fornece uma mensagem clara de erro, ajudando o usuário a entender e corrigir a situação.

5 Análise Experimental

Para avaliar o desempenho do algoritmo de compressão e descompressão, foram realizados experimentos utilizando 10 arquivos de diferentes tamanhos, variando de aproximadamente 5 MB a 50 MB. O objetivo foi analisar o tempo de execução em relação ao tamanho do arquivo e a taxa de compressão obtida.

Os resultados dos experimentos demonstraram um comportamento linear no tempo de execução tanto para a compactação quanto para a descompactação. Isso significa que quando o tamanho do arquivo dobra, o tempo necessário para executar o processo de compressão ou descompressão também dobra.

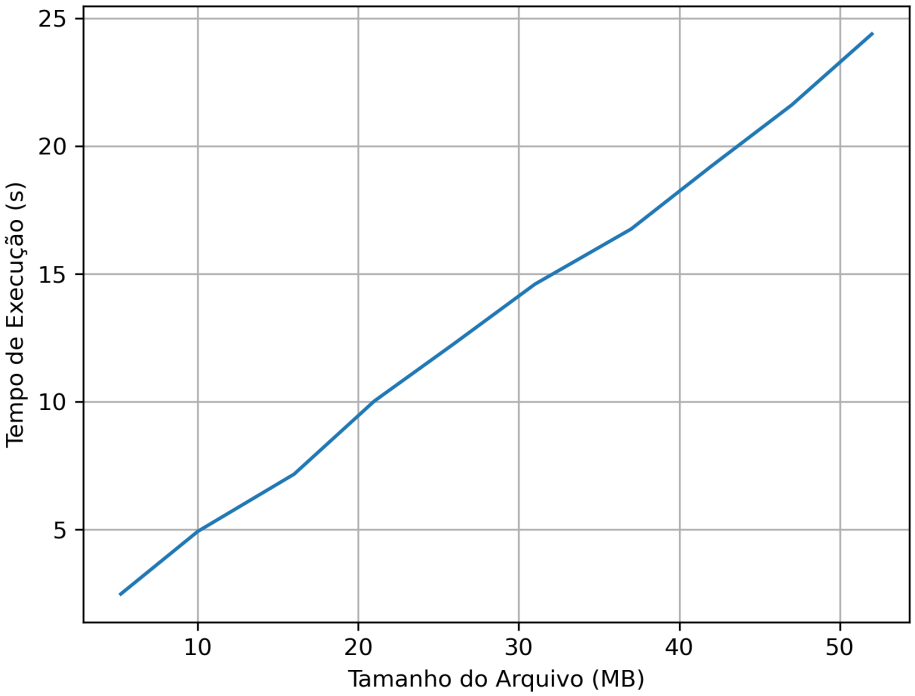

Figura 1: Tempo de execução para compactação

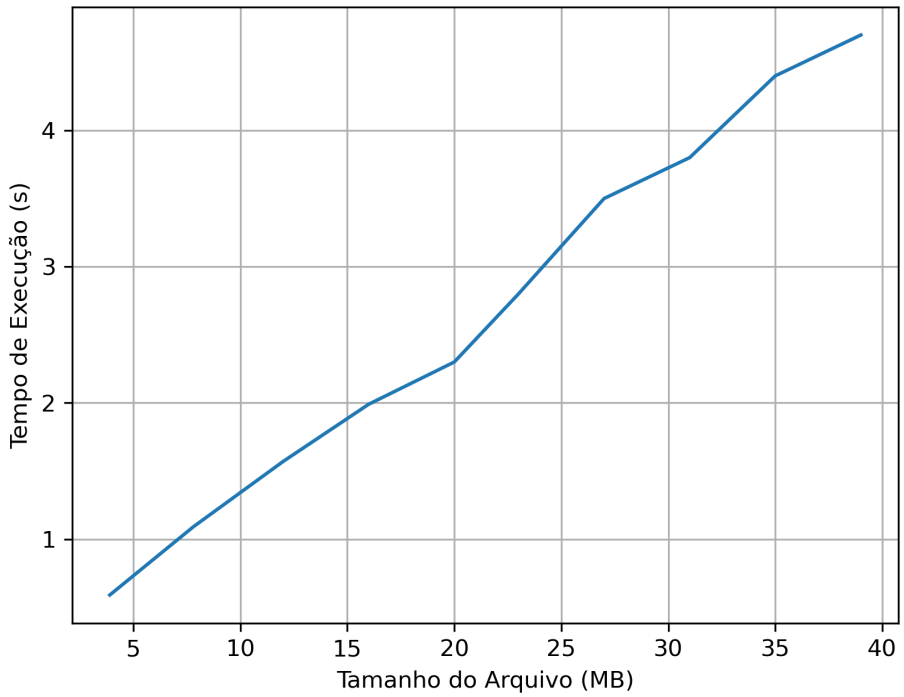

Figura 2: Tempo de execução para descompactação

Em relação à taxa de compressão, os experimentos revelaram que ela ficou em torno de 1,33. No entanto, é importante ressaltar que esse valor pode variar dependendo do conteúdo do arquivo de teste. Os arquivos utilizados nos experimentos foram compostos por caracteres gerados de forma aleatória, o que pode influenciar na taxa de compressão obtida. Arquivos com maior redundância de caracteres tendem a apresentar uma taxa de compressão mais alta, enquanto arquivos com pouca redundância podem ter uma taxa de compressão menor.

6 Conclusões

Neste trabalho, foi desenvolvido um algoritmo de compressão e descompressão utilizando o algoritmo de Huffman. O objetivo era reduzir o espaço ocupado pelos arquivos de um jornal, preservando todas as suas edições. Durante o processo, foram exploradas estruturas de dados como a árvore AVL, implementadas estratégias de robustez e realizada uma análise experimental para avaliar o desempenho do algoritmo.

Ao longo do trabalho, aprendemos que o algoritmo de Huffman é uma técnica eficiente para compressão de dados, reduzindo o tamanho dos arquivos através de uma codificação adaptativa. A utilização de estruturas de dados como árvores AVL permitiu a contagem eficiente dos caracteres e a construção da árvore de Huffman. Também analisamos a complexidade do algoritmo, observando que a função count_char apresenta uma complexidade de $\Theta(n \log n)$ devido ao uso da árvore AVL, enquanto a construção da árvore de Huffman possui uma complexidade de $O(n^2)$ no pior caso e $\Omega(n)$ no melhor caso.

A análise experimental revelou que o tempo de execução do algoritmo apresenta um comportamento linear em relação ao tamanho do arquivo, tanto para a compressão quanto para a descompressão. Esses resultados destacam a eficácia e a capacidade do algoritmo de reduzir o tamanho dos arquivos de forma significativa.

Em suma, este trabalho demonstrou a aplicação do algoritmo de Huffman para a compressão e descompressão de arquivos, visando a redução do espaço ocupado sem comprometer a integridade das informações. Foram exploradas técnicas e estruturas de dados relevantes, bem como realizados experimentos para avaliar o desempenho do algoritmo. Essas aprendizagens fornecem uma base sólida para futuros aprimoramentos e utilização de técnicas de compressão em diferentes contextos.

Bibliografia

Luiz Chaimowicz e Raquel Prates. **Slide Estrutura de Dados**: Listas Lineares. Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Luiz Chaimowicz e Raquel Prates. **Slide Estrutura de Dados**: Ordenação: Quicksort. Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

Luiz Chaimowicz e Raquel Prates. **Slide Estrutura de Dados**: Árvore AV. Slides virtuais da disciplina de estruturas de dados. Disponibilizado via moodle. Departamento de Ciência da Computação. Universidade Federal de Minas Gerais. Belo Horizonte.

CORMEN, T.H. et al. **Introduction to Algorithms**, 3° Edição. The MIT Press, 31 jun. 2009

Construção e Análise de Algoritmos - UFC - Algoritmos Gulosos - Código de Huffman. **Prof. Rudini Sampaio**, 4 jun. 2020. Disponível em: <https://www.youtube.com/watch?v=8AoASTY9OeQ>. Acesso em: 17 jun. 2023

UTF-8. **Wikipedia**, 14 nov. 2021. Disponível em: <https://en.wikipedia.org/wiki/UTF-8>. Acesso em: 18 jun. 2023.

Instruções para compilação e execução

1. O programa pode ser compilado com o comando: **make**

2. Para execução do programa, deve usar seu caminho direto ou Makefile:
   ./bin/huffmanCoding [OPTION] [FILE] [FILE]
   make run ARGS="[OPTION] [FILE] [FILE]"

3. O programa espera uma flag para sua utilização:
   -c comprimir um arquivo texto, esperado dois arquivos
   -d descomprimir um arquivo binário, espera dois arquivos
   primeiro arquivo é de entrada e o segundo arquivo é de saída