Trabalho prático 3

Compactação de arquivos texto

Iago da Silva Rodrigues Alves – 2022035881

iagosilva92@ufmg.br

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte - MG – Brasil

1. Introdução

O trabalho é um programa desenvolvido para reduzir o tamanho de arquivos de texto por meio do algoritmo de compressão de Huffman. Esse algoritmo utiliza uma tabela de frequência dos caracteres presentes no texto original para construir uma árvore de Huffman, em que os caracteres mais frequentes são representados por códigos binários mais curtos. A partir dessa árvore, o texto original é codificado em uma sequência de bits mais compacta, permitindo a redução do tamanho do arquivo. Dessa forma, este programa possui duas principais funcionalidades: a compactação de um arquivo de texto e a descompactação do arquivo compactado.

Outrossim, como decisão de implementação decidi após a compactação criar um arquivo que armazene a frequência necessária para construção da árvore para descompactação. Portanto, é imprescindível que para descompactar o usuário certifique-se de que a compactação foi executada corretamente.

2. Método

2.1 Ferramentas

Esse projeto utiliza as seguintes ferramentas e especificações:

- Linguagem: C++11
- Compilação: g++ 11.3.0
- Sistema Operacional: *Ubuntu 22.04*
- GNU Make 4.3

2.2 Tipos Abstratos de Dados

Node.hpp: O TAD Node representa um nó utilizado na construção da árvore de Huffman. Cada nó possui um caractere, uma frequência, e ponteiros para os nós filhos (esquerdo e direito) e para o próximo nó na lista ordenada. Ele possui métodos para obter e definir essas propriedades.

TabelaFrequencia.hpp: O TAD TabelaFrequencia é responsável por gerenciar a tabela de frequência dos caracteres presentes no texto. Essa tabela é representada por um vetor de inteiros, em que cada posição corresponde a um caractere e seu valor corresponde à sua frequência. As principais operações desse TAD são a inicialização da tabela e o preenchimento da tabela.

ListaOrdenada.hpp: O TAD ListaOrdenada representa uma lista encadeada ordenada de nós. Cada nó da lista possui um ponteiro para o próximo nó e métodos para obter e definir o nó seguinte. A lista ordenada é utilizada para construir a árvore de Huffman a partir da tabela de frequência. As principais operações desse TAD são a inserção ordenada de nós, o preenchimento da lista com base na tabela de frequência e a remoção do nó inicial.

Arvore.hpp: O TAD Arvore representa a árvore de Huffman utilizada na compactação e descompactação dos arquivos. A árvore é composta pelos nós citados acima. Esse TAD possui métodos para montar a árvore a partir de uma lista ordenada e destruir a árvore, liberando a memória alocada.

Dicionario.hpp: O TAD Dicionario representa o dicionário de códigos gerado a partir da árvore de Huffman. O dicionário é um vetor de strings, em que cada posição corresponde a um caractere e seu valor corresponde ao código binário associado a esse caractere na árvore. Esse TAD possui métodos para gerar o dicionário a partir da árvore de Huffman e imprimir o dicionário.

Huffman.hpp: O TAD Huffman é responsável pela codificação e decodificação do texto utilizando a árvore de Huffman e o dicionário de códigos. Ele também realiza a compactação e descompactação dos arquivos. Esse TAD possui métodos para codificar o texto original, decodificar o texto codificado, compactar o texto codificado e descompactar o arquivo compactado. Ele também possui métodos para verificar um bit específico em um byte e obter o texto codificado e decodificado.

2.3 Funções

O programa possui várias funções importantes que desempenham um papel fundamental na compactação e descompactação de arquivos. Algumas dessas funções são: criação da Tabela de Frequência, criação da Lista Ordenada, construção da Árvore de Huffman, geração do Dicionário de Códigos, codificação do texto de entrada, compactação e descompactação.

Essas funções interagem entre si para realizar o processo de compactação e descompactação de forma eficiente. A Tabela de Frequência é o ponto de partida, pois ela contém as informações sobre a frequência de ocorrência de cada caractere no texto de entrada. Em seguida, a Lista Ordenada é construída com base nessa tabela, permitindo a organização dos caracteres de acordo com a frequência.

Com a Lista Ordenada, é possível montar a Árvore de Huffman, que é uma estrutura hierárquica onde os caracteres mais frequentes estão mais próximos da raiz. A partir dessa árvore, o Dicionário de Códigos é gerado, atribuindo uma sequência de bits única para cada caractere presente na árvore.

Com o Dicionário de Códigos em mãos, é possível codificar o texto de entrada, substituindo cada caractere pelo seu código correspondente. Essa etapa prepara o texto para a compactação propriamente dita. A função de compactação utiliza o texto codificado e o comprime em um arquivo binário, utilizando técnicas como o uso de bits para representar os caracteres.

Essas funções em conjunto proporcionam o funcionamento completo do programa, permitindo a compactação e descompactação de arquivos de forma eficiente e confiável.

3. Análise de Complexidade

O programa apresenta várias partes com diferentes complexidades de tempo e espaço. A primeira parte é a criação da tabela de frequência, que itera por cada caractere possível no texto, resultando em complexidade de tempo O(1). Em seguida, temos a criação de uma lista encadeada ordenada, onde cada nó representa um caractere distinto no texto, resultando em complexidade de tempo e espaço O(n), onde n é o número de caracteres distintos.

A etapa seguinte é a montagem da árvore de Huffman, que requer a fusão dos nós da lista até obter a árvore final. Essa etapa possui complexidade de tempo e espaço O(n), pois a árvore de Huffman possui n nós. A montagem do dicionário, que associa um código binário a cada caractere, tem complexidade de tempo e espaço O(n), pois é necessário percorrer a árvore para gerar os códigos.

A etapa de compactação envolve escrever cada byte compactado no arquivo, com complexidade de tempo O(m) e complexidade de espaço O(1), pois o espaço ocupado pelo arquivo compactado não depende do tamanho do texto original. A descompactação requer ler cada bit e percorrer a árvore de Huffman para encontrar o caractere correspondente, resultando em complexidade de tempo O(k), onde k é o número de bits no arquivo compactado, e complexidade de espaço O(1).

4. Estratégias de Robustez

O programa implementa mecanismos de programação defensiva e tolerância a falhas a fim de garantir a robustez e confiabilidade do sistema e para isso ele utiliza a biblioteca stdexcept**.** Podemos observar a utilização desses mecanismos nas

funções em que é necessário por exemplo abrir um arquivo para leitura. Por exemplo na função de compactação:

```
void Huffman::compactar(std::string str, TabelaFrequencia tabela, std::string arquivoSaida) {
  std::ofstream arquivo(arquivoSaida, std::ios::binary);
  unsigned char byte = 0;
  int i = 0, j = 7;

  if (!arquivo.is_open()) {
    throw std::runtime_error ("Erro ao abrir/criar o arquivo para compactacao");
  }
```

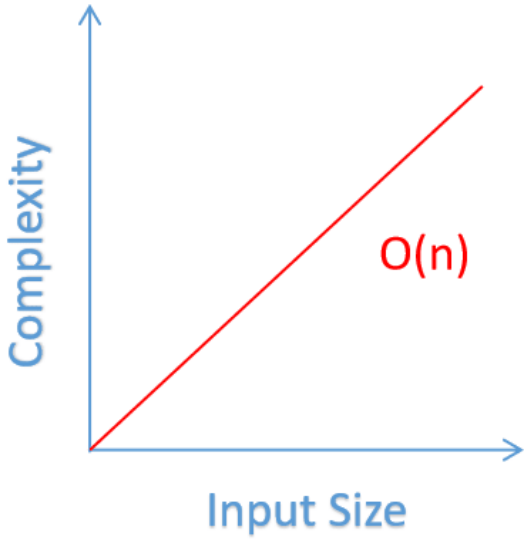
Além dos mecanismos de programação defensiva, o programa implementa também mecanismos de tolerância a falhas. Esses mecanismos são utilizados para lidar com erros que possam ocorrer durante a execução do programa e que não possam ser prevenidos. No caso desse programa, podemos observar a utilização desses mecanismos na função de montagem da árvore. Caso ocorra algum erro durante essas operações, como falha na alocação de memória a função captura a exceção e interrompe o programa.

Outrossim, o programa não utiliza a declaração do namespace std para evitar conflitos de nomes e a poluição do namespace. E também todos os construtores presentes seguem a implementação na forma "member initializer constructor" que traz benefícios como aumento de performance e redução de complexidade de código. Evitando que os membros sejam inicializados na implementação do construtor

5. Análise Experimental

Além da análise de complexidade teórica, outra abordagem importante para avaliar o desempenho de um algoritmo é a análise experimental. No entanto, é importante ressaltar que a análise experimental também tem suas limitações, uma vez que os resultados podem ser afetados por fatores externos, como a arquitetura do computador, o sistema operacional e a carga de trabalho do sistema no momento dos testes. Por isso, é importante realizar vários testes e calcular a média dos resultados para obter uma estimativa mais precisa do desempenho do algoritmo.

Abaixo trago um gráfico que compara o tempo de execução do programa em microssegundos e o tamanho dos arquivos de entrada, tanto para compactação quanto para descompactação.



Ao plotar o gráfico tempo x tamanho, é possível identificar padrões e tendências na curva, que podem indicar a complexidade do algoritmo. A curva claramente apresenta uma inclinação linear, isso sugere que a complexidade é proporcional ao tamanho da entrada. Logo, significa que, à medida que o tamanho da entrada aumenta, o tempo de execução do programa aumenta linearmente.

6. Conclusões

Nesse trabalho desenvolvido para compressão de arquivos de texto utilizando o algoritmo de Huffman, é possível concluir que o programa apresenta um conjunto sólido de funcionalidades e estruturas de dados bem definidas. Em resumo, ele é capaz de realizar a compressão e descompressão de arquivos de forma eficiente, proporcionando uma redução significativa no tamanho dos arquivos sem comprometer a integridade dos dados.

Posto isso, para alcançar esse objetivo, foram aplicados diversos conceitos e técnicas aprendidas em sala de aula, tais como modularização, depuração e a construção de tipos abstratos de dados. Outrossim, o programa foi projetado para garantir a robustez e a tolerância a falhas, contando com mecanismos de programação defensiva e estratégias de tratamento de exceções, fornecendo uma base sólida para futuras extensões e melhorias.

7. Bibliografia

Slides da disciplina de Estrutura de Dados. Disponibilizados via Moddle. 1° semestre 2023.

Algoritmo de Huffman para compressão de dados – Instituto de Matemática e Estatística da Universidade de São Paulo. Disponível em: <https://www.ime.usp.br/~pf/estruturas-de-dados/aulas/huffman.html>
Acesso em: junho de 2023

8. Instruções para compilação e execução

Passo 1: Compilar o programa

- Digite o comando make no terminal

Passo 2: Executar a compactação

- Crie um arquivo texto com o texto a ser compactado
- Crie um arquivo binário vazio

Opções para executar a compactação

1. bin/huffman –c nome_arquivo.txt nome_arquivo.bin
2. make run ARGS=”-c nome_arquivo.txt nome_arquivo.bin”
3. Adicionar o PATH da pasta bin no seu bash e escrever na linha de comando:
    a. huffman –c nome_arquivo.txt nome_arquivo.bin

Passo 3: Executar a descompactação

- Crie um arquivo texto vazio
- Certifique-se de que existe um arquivo binário com o texto compactado

Opções para executar a descompactação

1. bin/huffman –d nome_arquivo.bin nome_arquivo.txt
2. make run ARGS=” –d nome_arquivo.bin nome_arquivo.txt”
3. Adicionar o PATH da pasta bin no seu bash e escrever na linha de comando:
    b. huffman –d nome_arquivo.bin nome_arquivo.txt

Observação: A compactação é essencial para a execução da descompactação, tendo em vista que uma das decisões do projeto foi a criação de um arquivo separadamente para o armazenamento da frequência do primeiro arquivo de entrada, arquivo que é necessário para descompactação.

Passo 4: Limpeza

- Digite o comando make clean no terminal