**Relatório**

**Cauã Ribas, Nilson Andrade**

Universidade do Vale do Itajaí - Univali

Escola do Mar, Ciência e Tecnologia

Ciência da Computação

(cauaribas, nilson.neto) @edu.univali.br

**Estruturas de Dados**

Avaliação 02 - Codificação de Huffman

**Marcos Cesar Cardoso Carrard**

10/10/2023

**1. Introdução:**

Este relatório descreve a implementação de um programa usando a Linguagem de Alto Nível C/C++, este código implementa um programa recebe um texto qualquer, comprime esse texto usando o Algoritmo de Huffman e, depois, aplica a descompressão pelo mesmo algoritmo apresentando o texto original inserido.

**2. Programa:**

**2.1. Enunciado:**

O objetivo deste trabalho é escrever um programa capaz de receber um texto qualquer, comprimir esse texto usando o Algoritmo de Huffman e, depois, aplicar a descompressão pelo mesmo algoritmo apresentado no texto original inserido. A forma de entrada do texto pode ser por digitação ou por leitura de um arquivo (a critério do usuário). Já o resultado do processo de conversão deverá ser apresentado em tela apenas (com um vetor de char) caso o trabalho seja feito individual ou em dupla.

**2.2. Implementação da Funções:**

**Inclusão de Bibliotecas:**

O código inclui as bibliotecas <iostream> e <fstream> para entrada/saída padrão e manipulação de arquivos. Tambem faz uso de um arquivo hpp para auxílio na criação do código contendo as structs e funções, das listas unicamente encadeadas e árvores binárias.

**Definição de Estruturas e Funções:**

O código define um conjunto de funções e estruturas para manipular árvores de Huffman, listas encadeadas e operações relacionadas à compressão e descompressão. Ele define a estrutura “NoArvore”, que representa um nó de uma árvore de Huffman. Usa a estrutura LUE, que parece ser uma lista encadeada especializada.

**Função “construirArvoreHuffman”:**

Esta função constrói uma árvore de Huffman com base na frequência dos caracteres no texto de entrada. Ela inicializa uma tabela de frequências para cada caractere possível. Conta a frequência de cada caractere no texto. Insere os caracteres e suas frequências na lista encadeada. Em seguida, agrupa os nós menores até que reste apenas um nó na lista, que será a raiz da árvore de Huffman.

**Função “comprimirTexto”:**

Esta função recebe um nó da árvore de Huffman, um caractere e uma string vazia (caminho) para armazenar o código binário do caractere. Ela verifica se a raiz é nula, se é uma folha (caractere) ou se deve continuar a busca na subárvore esquerda (0) ou direita (1). Ela constrói o caminho binário recursivamente até encontrar o caractere desejado ou terminar a busca.

**Função “gerarMapeamentoBits”:**

Esta função gera o mapeamento de bits (codificação) com base na árvore de Huffman. Itera pelos caracteres do texto original e usa a função “comprimirTexto” para obter o código binário de cada caractere. Concatena os códigos binários em uma única string, que representa o texto codificado.

**Função “descomprimirTexto”:**

Esta função descomprime o texto codificado usando a árvore de Huffman.

Ela recebe a raiz da árvore, o texto codificado e realiza a busca na árvore para encontrar os caracteres originais. À medida que os bits são lidos, a função percorre a árvore, buscando caracteres e construindo o texto descomprimido.

**Função “main”:**

O programa principal começa definindo uma lista encadeada “lista” para armazenar os nós da árvore de Huffman. Dentro de um loop “do-while”, o programa exibe um menu com as opções de inserir um texto, ler um arquivo ou sair. Se o usuário escolher inserir um texto, ele solicita o texto, constrói a árvore de Huffman, gera o mapeamento de bits, exibe o texto codificado e o descomprime. Se o usuário escolher ler um arquivo, ele solicita o nome do arquivo, lê o conteúdo do arquivo, realiza as mesmas etapas de compressão/descompressão e exibe os resultados. O loop continua até o usuário escolher sair.

**2.3. Código em C++:**

#include <iostream>

#include <fstream>

#include "auxiliar.hpp"

using namespace std;

template <typename T>

void construirArvoreHuffman(LUE <T> &lista, string texto){

int tabelaFrequencias[256] = {0}; // Inicializa uma tabela de frequências para cada caractere possível (0-255)

// Contar a frequência de cada caractere no texto

for(char c : texto){

tabelaFrequencias[c]++; // A cada iteração, incrementa a contagem de frequência do caractere 'c' na tabela

}

// Inserir na LUE

for(int i = 0; i < 256; i++){

if(tabelaFrequencias[i] > 0){

NoArvore <T> \*raiz = new NoArvore <T>;

raiz->infoA = char(i); // Atribui o caractere correspondente ao valor inteiro 'i' ao campo 'infoA'

raiz->frequencia = tabelaFrequencias[i];

inserirLUE(lista, raiz);

}

}

// Agrupar Nós

while(lista.comeco && lista.comeco->elo){ // Percorre a lista enquanto não estiver vazia e houver mais de um nó na lista

NoArvore <T> \*menorIndex1 = lista.comeco->info;

NoArvore <T> \*menorIndex2 = lista.comeco->elo->info;

NoLUE <T> \*aux1 = lista.comeco;

NoLUE <T> \*aux2 = lista.comeco->elo;

if(lista.comeco->elo->elo){ // Verifica se há pelo menos dois elementos na lista

lista.comeco = lista.comeco->elo->elo; // Se houver dois elementos, avança o ponteiro em duas posições

}else{

lista.comeco = NULL; // Se não houver pelo menos dois elementos, define o ponteiro como NULL para sair do loop

}

// Deleta os dois menores elementos da lista

delete aux1;

delete aux2;

// Cria um novo nó que combina os dois elementos menores

NoArvore <T> \*novaRaiz = new NoArvore <T>;

novaRaiz->infoA = '\0';

novaRaiz->frequencia = menorIndex1->frequencia + menorIndex2->frequencia;

novaRaiz->esq = menorIndex1;

novaRaiz->dir = menorIndex2;

// Insere o novo nó de volta na lista

inserirLUE(lista, novaRaiz);

}

}

template <typename T>

bool comprimirTexto(NoArvore <T> \*raiz, char c, string &caminho){

if(raiz == NULL){ // Se a raiz for nula, não há caminho a seguir

caminho = ""; // Definimos o caminho como vazio e retornamos falso.

return false;

}

if(raiz->infoA != '\0'){ // Se for uma folha (caractere), verifica se é o caractere desejado

if(raiz->infoA == c)

return true;

return false;

}

if(comprimirTexto(raiz->esq, c, caminho)){ // Recursivamente busca à esquerda (0)

caminho = "0" + caminho;

return true;

}

if(comprimirTexto(raiz->dir, c, caminho)){ // Recursivamente busca à esquerda (0)

caminho = "1" + caminho;

return true;

}

return false;

}

template <typename T>

void gerarMapeamentoBits(LUE <T> lista, string &texto, string &codificado){ // codificado = mapa

NoArvore <T> \*arvore = lista.comeco->info; // Obtém a raiz da árvore de Huffman.

codificado = "";

for (char c : texto){ // Itera pelos caracteres do texto original.

string binario = "";

comprimirTexto(arvore, c, binario);

codificado += binario;

}

}

template <typename T>

string descomprimirTexto(NoArvore <T> \*raiz, const string &textoComprimido){

NoArvore <T> \*arvore = raiz;

string textoDescomprimido;

for(char bit : textoComprimido){ // Itera pelos bits do texto codificado.

if(bit == '0'){

arvore = arvore->esq;

}else if(bit == '1'){

arvore = arvore->dir;

}

if(arvore->esq == NULL && arvore->dir == NULL){

textoDescomprimido += arvore->infoA;

arvore = raiz; // Reinicia a busca na raiz da árvore

}

}

return textoDescomprimido;

}

int main(){

LUE <char> lista;

inicializarLUE(lista);

int inputOpcao;

string texto;

do{

system("cls");

cout << "====================================================";

cout << "\n\t\tHuffman-Coding-Program";

cout << "\n\n1. Inserir um texto: ";

cout << "\n2. Ler um arquivo: ";

cout << "\n3. Sair";

cout << "\n\n====================================================\n";

cout << "Opcao: ";

cin >> inputOpcao;

if(inputOpcao == 1){

cout << "Digite o texto de entrada: ";

cin.ignore();

getline(cin, texto);

// Construir a Arvore de Huffman

construirArvoreHuffman(lista, texto);

// Gerar a Sequencia de Bits

string codificado;

gerarMapeamentoBits(lista, texto, codificado);

cout << "\nSequencia de bits: ";

cout << "\n" << codificado << endl;

// Descomprimir texto original

string textoDescomprimido = descomprimirTexto(lista.comeco->info, codificado);

cout << "\nTexto descomprimido: \n" << textoDescomprimido << endl;

texto = ""; // Esvazia a string

cout << "\nPrecione Enter para continuar..." << endl;

cin.get();

}

else if(inputOpcao == 2){

string name;

string linha;

fstream arquivo;

cout << "Por favor digite o nome do seu arquivo: ";

cin.ignore();

getline(cin, name);

arquivo.open(name, fstream :: in | fstream :: app);

if(arquivo.is\_open()){

while(getline(arquivo, linha)){

texto += linha;

}

}

// Construir a Arvore de Huffman

construirArvoreHuffman(lista, texto);

// Gerar a Sequencia de Bits

string codificado;

gerarMapeamentoBits(lista, texto, codificado);

cout << "\nSequencia de bits: ";

cout << "\n" << codificado << endl;

// Descomprimir texto original

string textoDescomprimido = descomprimirTexto(lista.comeco->info, codificado);

cout << "\nTexto descomprimido: \n" << textoDescomprimido;

texto = ""; // Esvazia a string

cout << "\nPrecione Enter para continuar..." << endl;

cin.get();

}

else if(inputOpcao == 3){

cout << "Saindo...";

liberarLUE(lista);

liberarArvore(lista.comeco->info);

}

else{

cout << "Opcao invalida!";

cout << "\nPrecione Enter para continuar..." << endl;

cin.ignore();

cin.get();

}

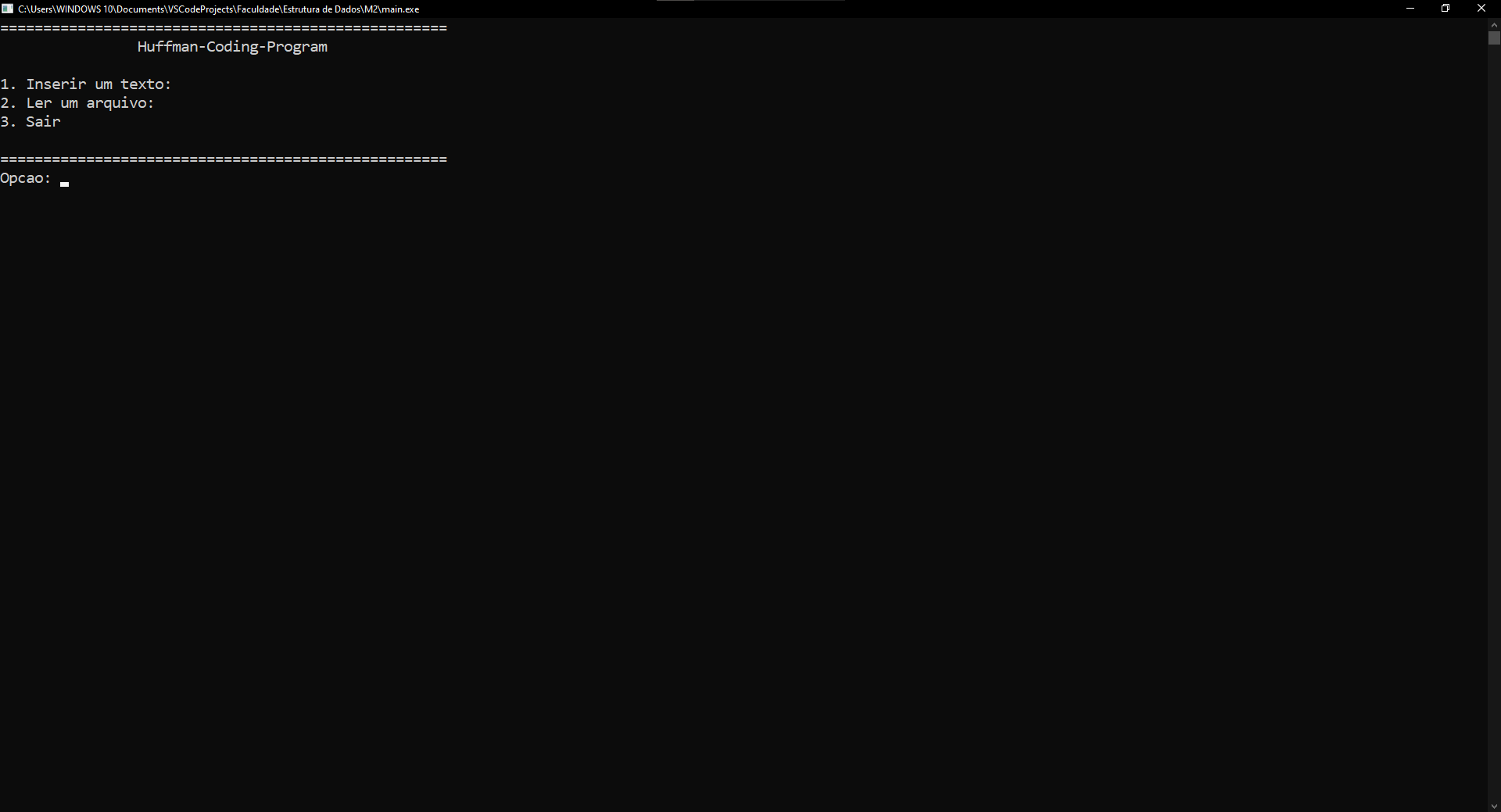
}while(inputOpcao != 3);

return 0;

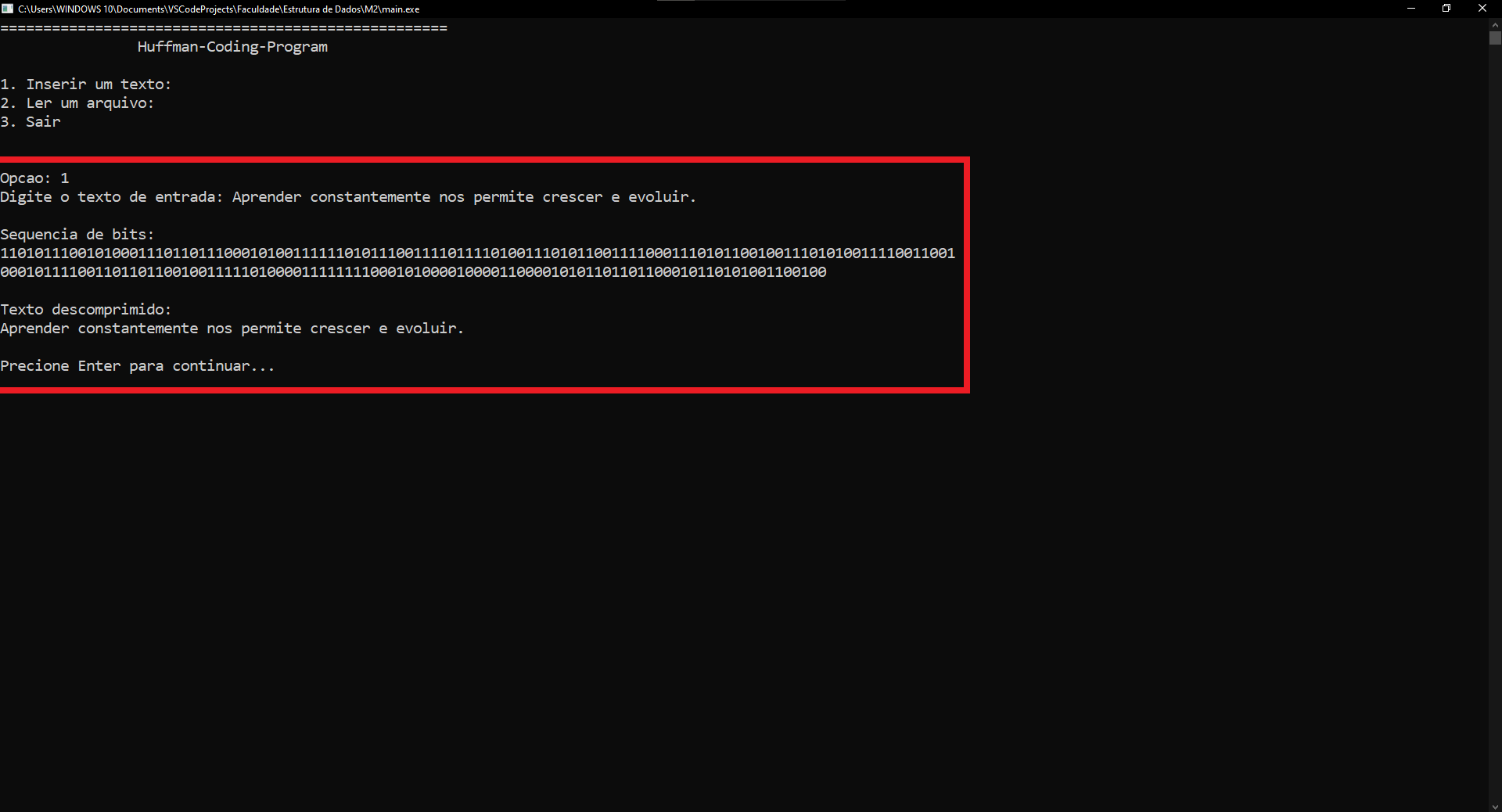
}

**3. Execução do Código:**

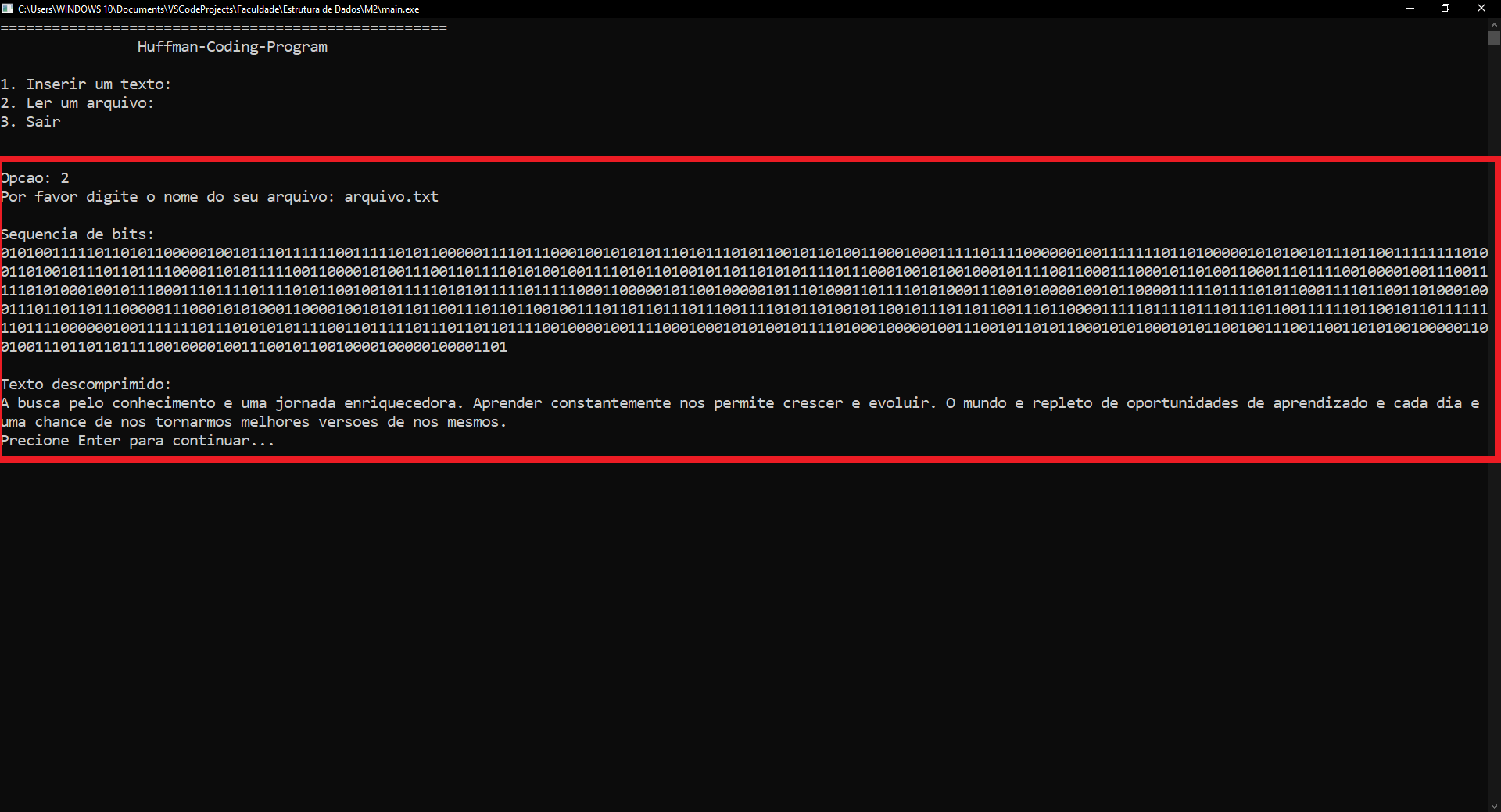
Após executar o código, o usuário deve escolher uma das opções do menu, inserir texto, ler um arquivo, ou sair do programa.



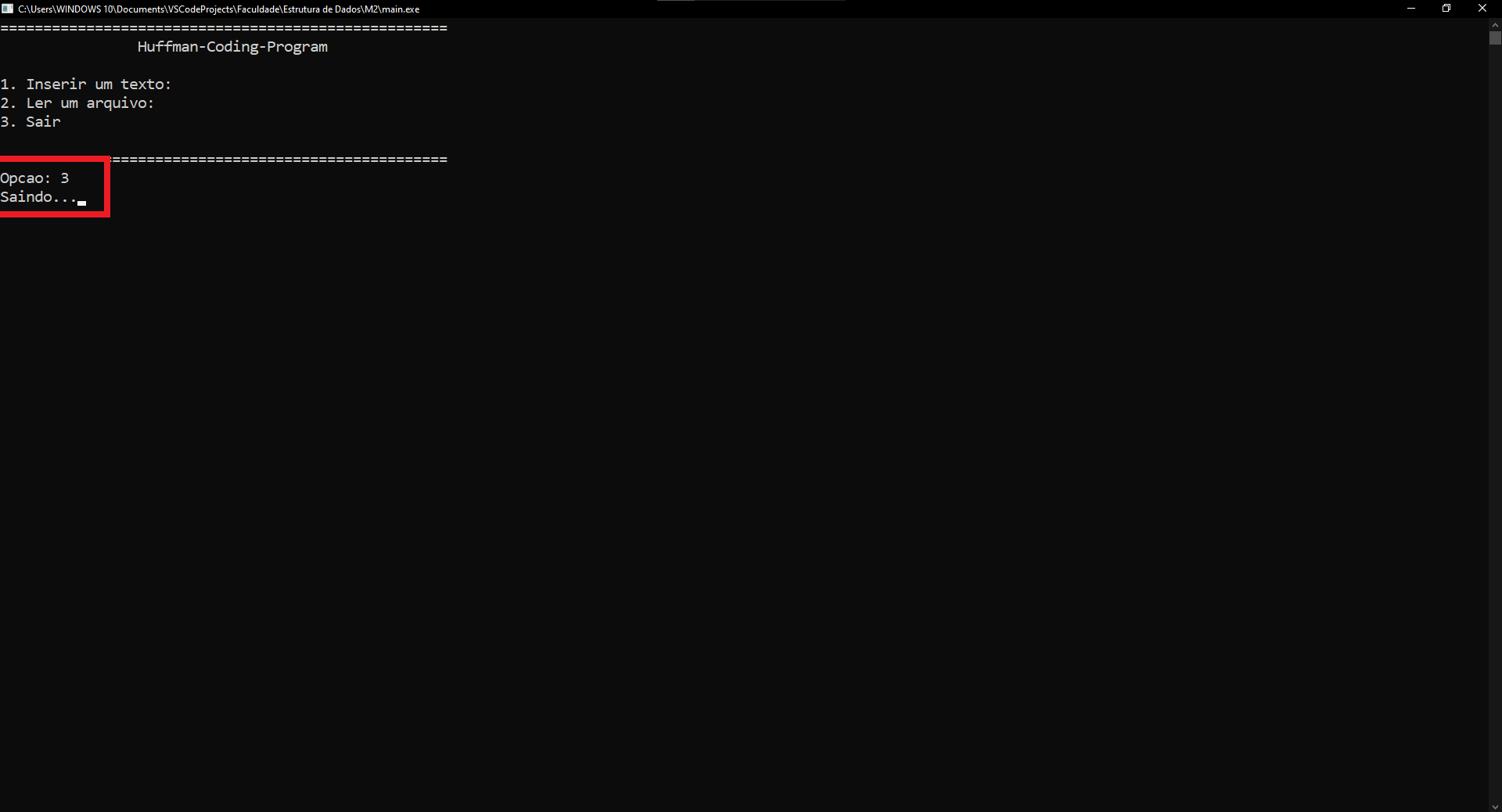
Se o usuário escolher a opção de inserir texto, o programa solicita a entrada de um texto, o programa comprime o texto e mostra na tela a sequência de bits, e por fim mostra o texto descomprimido.



Se o usuário escolher a opção de ler arquivo, o programa solicita a entrada do nome do arquivo com sua extensão (Ex: arquivo.txt), o programa comprime o texto do arquivo e mostra na tela a sequência de bits, e por fim mostra o texto descomprimido.



Se o usuário escolher a opção sair aqui, o programa libera a memória alocado para os nós da lista e as raízes da árvore.



**4. Conclusão:**

Portanto, compreendemos que o código apresentado demonstra de forma prática a implementação do algoritmo de Huffman, uma técnica de codificação eficaz e amplamente utilizada na área de ciência da computação. A capacidade de compactar dados de maneira eficiente é fundamental em diversas aplicações, tornando a codificação de Huffman uma ferramenta valiosa no processamento e transmissão de informações.

**Bibliografia:**

CS UFSCA EDU - Disponível em:

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

HUFFMAN CODING - Disponível em:

https://huffman-coding-online.vercel.app/

DCODE Disponível em:

https://www.dcode.fr/huffman-tree-compression

PLANETCALC - Disponível em:

https://planetcalc.com/2481/