UERJ – CAMPUS ZONA OESTE CIÊNCIA DA COMPUTAÇÃO

RUAN SOARES DA SILVA FONSECA - 1923333070
DANYLO HENRIQUE DA SILVA DOS SANTOS - 2013332021

TRABALHO AV1 – ESTRUTURA DE DADOS 2 PROF. DR. DENIS GONÇALVES COUPLE

RIO DE JANEIRO
2023

RUAN SOARES DA SILVA FONSECA - 1923333070
DANYLO HENRIQUE DA SILVA DOS SANTOS - 2013332021

ESTRUTURA DE DADOS 2
TRABALHO COMPLEMENTAR PARA AV1

Trabalho acadêmico apresentado à disciplina de Estrutura de Dados 2 do Curso de Ciência da Computação como requisito de nota parcial da AV1. Requerido pelo prof. Dr.Denis Golçalves Couple.

RIO DE JANEIRO
2023

SUMÁRIO

1 INTRODUÇÃO

Foi desenvolvido um programa em C++ para armazenar as palavras de diversos arquivos em uma arvore AVL. O programa contem funcionalidades de pesquisar as palavras armazenadas, mostrando a quantidade de ocorrências da palavra para cada arquivo e as linhas das ocorrências. Além disso, há a funcionalidade de deletar uma palavra da arvore e de imprimir todas as palavras da arvore.

Inicialmente é necessário adicionar um arquivo antes de tentar executar as demais funções, caso o contrário o sistema imprimira que é necessário adicionar um arquivo primeiro. Ademais, para a função de buscar e a de deletar palavra o sistema não considera diferenças entre letras maiúsculas e minúsculas.

2 DETALHES DO PROGRAMA

2.1 CÓDIGO DO PROGAMA

A seguir o código para implementação do programa:

```
 #include <iostream>
#include <fstream>
#include <cstring>
#include <cstdlib>
#include <ctype.h>
#include <locale.h>

#pragma warning(disable : 4996)

using namespace std;

struct Arquivo
{
	char nome[50];
	int linha;
	int ocorrencias;
	Arquivo* prox;
};

struct No
{
	char palavra[50];
	Arquivo* arquivos;
	No* esq;
	No* dir;
	int altura;
};

int altura(No* no)
{
	return (no == NULL) ? 0 : no->altura;
}

int max(int a, int b)
{
	return (a > b) ? a : b;
}

char* tornaMaiusculo(char* palavra) {
	for (size_t i = 0; i < strlen(palavra); i++)
		palavra[i] = toupper(palavra[i]);
	return palavra;
}

No* novoNo(char* palavra, char* nomeArquivo, int linha)
{

	palavra = tornaMaiusculo(palavra);
```

```
        No* no = new No;
        strcpy(no->palavra, palavra);
        no->arquivos = new Arquivo;
        strcpy(no->arquivos->nome, nomeArquivo);
        no->arquivos->linha = linha;
        no->arquivos->ocorrencias = 1;
        no->arquivos->prox = NULL;
        no->esq = NULL;
        no->dir = NULL;
        no->altura = 1;
        return no;
}

No* rotacaoDireita(No* y)
{
        No* x = y->esq;
        No* T2 = x->dir;
        x->dir = y;
        y->esq = T2;
        y->altura = max(altura(y->esq), altura(y->dir)) + 1;
        x->altura = max(altura(x->esq), altura(x->dir)) + 1;
        return x;
}

No* rotacaoEsquerda(No* x)
{
        No* y = x->dir;
        No* T2 = y->esq;
        y->esq = x;
        x->dir = T2;
        x->altura = max(altura(x->esq), altura(x->dir)) + 1;
        y->altura = max(altura(y->esq), altura(y->dir)) + 1;
        return y;
}

int balanceamento(No* no)
{
        return (no == NULL) ? 0 : altura(no->esq) - altura(no->dir);
}

Arquivo* buscaArquivo(No* no, char* nomeArquivo)
{
        Arquivo* arq = no->arquivos;
        while (arq != NULL)
        {
                if (strcmp(arq->nome, nomeArquivo) == 0)
                        return arq;
                arq = arq->prox;
        }
        return NULL;
}

void inserirOcorrencia(No* no, char* nomeArquivo, int linha)
{
        Arquivo* arq = buscaArquivo(no, nomeArquivo);
        if (arq == NULL)
        {
                Arquivo* novoArq = new Arquivo;
                strcpy(novoArq->nome, nomeArquivo);
                novoArq->linha = linha;
                novoArq->ocorrencias = 1;
```

```
            novoArq->prox = no->arquivos;
            no->arquivos = novoArq;
        }
        else
        {
            arq->ocorrencias++;
            arq->linha = linha;
        }
}

No* inserir(No* no, char* palavra, char* nomeArquivo, int linha)
{
        palavra = tornaMaiusculo(palavra);

        if (no == NULL)
            return novoNo(palavra, nomeArquivo, linha);
        if (strcmp(palavra, no->palavra) < 0)
            no->esq = inserir(no->esq, palavra, nomeArquivo, linha);
        else if (strcmp(palavra, no->palavra) > 0)
            no->dir = inserir(no->dir, palavra, nomeArquivo, linha);
        else
        {
            inserirOcorrencia(no, nomeArquivo, linha);
        }
        no->altura = max(altura(no->esq), altura(no->dir)) + 1;
        int fatorBalanceamento = balanceamento(no);
        if (fatorBalanceamento > 1 && strcmp(palavra, no->esq->palavra) < 0)
            return rotacaoDireita(no);
        if (fatorBalanceamento < -1 && strcmp(palavra, no->dir->palavra) > 0)
            return rotacaoEsquerda(no);

        if (fatorBalanceamento > 1 && strcmp(palavra, no->esq->palavra) > 0)
        {
            no->esq = rotacaoEsquerda(no->esq);
            return rotacaoDireita(no);
        }

        if (fatorBalanceamento < -1 && strcmp(palavra, no->dir->palavra) < 0)
        {
            no->dir = rotacaoDireita(no->dir);
            return rotacaoEsquerda(no);
        }

        return no;
}

void imprimirOcorrencias(Arquivo* arquivos)
{
        while (arquivos != NULL)
        {
            cout << "\t" << arquivos->nome << ": linha " << arquivos->linha << ",
ocorrencias " << arquivos->ocorrencias << endl;
            arquivos = arquivos->prox;
        }
}

void imprimirArvore(No* raiz)
{
        if (raiz != NULL)
        {
            imprimirArvore(raiz->esq);
            cout << raiz->palavra << ":" << endl;
```

```
            imprimirOcorrencias(raiz->arquivos);
            imprimirArvore(raiz->dir);
      }
}

void processarArquivo(No** raiz, char nomeArquivo[], char voltar[]) {
      if (strcmp(nomeArquivo, voltar) == 0)
            return;


      FILE* arquivo = fopen(nomeArquivo, "r");
      if (arquivo == NULL)
      {
            printf("Arquivo nao encontrado.\n\n");
      }
      else
      {
            int linha = 1;
            char palavra[50];
            while (fscanf(arquivo, "%s", palavra) != EOF)
            {
                  int tamPalavra = strlen(palavra);
                  if (tamPalavra > 1 && (palavra[tamPalavra - 1] == '.' ||
palavra[tamPalavra - 1] == ','))
                  {
                        palavra[tamPalavra - 1] = '\0';
                  }
                  *raiz = inserir(*raiz, palavra, nomeArquivo, linha);
                  if (palavra[tamPalavra - 1] == '.' || palavra[tamPalavra - 1]
== ',')
                  {
                        linha++;
                  }
            }
            fclose(arquivo);
            cout << "Arquivo lido com sucesso!" << endl << endl;

      }
      cout << "Digite o nome do arquivo a ser adicionado ou 0 para voltar ao menu
principal: ";
      cin.getline(nomeArquivo, 50);
      processarArquivo(raiz, nomeArquivo, voltar);

}

void pesquisarPalavra(No* raiz, char* palavra)
{


      No* atual = raiz;
      bool encontrada = false;
      while (atual != NULL)
      {
            if (strcmp(palavra, atual->palavra) == 0)
            {
                  cout << "Palavra encontrada: " << atual->palavra << endl;
                  cout << "Ocorroncias:" << endl;
                  imprimirOcorrencias(atual->arquivos);
                  cout << endl;
                  encontrada = true;
```

```
                        break; // Removendo o break, o codigo continua a busca mesmo
depois de encontrar a palavra
                }
                if (strcmp(palavra, atual->palavra) < 0)
                        atual = atual->esq;
                else
                        atual = atual->dir;
        }
        if (!encontrada)
                cout << "Palavra nao encontrada." << endl << endl;
}

No* encontrarMinimo(No* no)
{
        while (no->esq != NULL)
                no = no->esq;
        return no;
}

No* deletarNo(No* raiz, char* palavra)
{
        if (raiz == NULL)
                return raiz;

        if (strcmp(palavra, raiz->palavra) < 0)
                raiz->esq = deletarNo(raiz->esq, palavra);
        else if (strcmp(palavra, raiz->palavra) > 0)
                raiz->dir = deletarNo(raiz->dir, palavra);
        else
        {
                if (raiz->esq == NULL || raiz->dir == NULL)
                {
                        No* temp = raiz->esq ? raiz->esq : raiz->dir;
                        if (temp == NULL)
                        {
                                temp = raiz;
                                raiz = NULL;
                        }
                        else
                                *raiz = *temp;
                                delete temp;
                                cout << "Palavra deletada com sucesso!" << endl;
                }
                else
                {
                        No* temp = encontrarMinimo(raiz->dir);
                        strcpy(raiz->palavra, temp->palavra);
                        raiz->arquivos = temp->arquivos;
                        raiz->dir = deletarNo(raiz->dir, temp->palavra);
                }
        }

        if (raiz == NULL)
                return raiz;

        raiz->altura = 1 + max(altura(raiz->esq), altura(raiz->dir));

        int fatorBalanceamento = balanceamento(raiz);

        if (fatorBalanceamento > 1 && balanceamento(raiz->esq) >= 0)
                return rotacaoDireita(raiz);
```

```
	if (fatorBalanceamento > 1 && balanceamento(raiz->esq) < 0)
	{
		raiz->esq = rotacaoEsquerda(raiz->esq);
		return rotacaoDireita(raiz);
	}

	if (fatorBalanceamento < -1 && balanceamento(raiz->dir) <= 0)
		return rotacaoEsquerda(raiz);

	if (fatorBalanceamento < -1 && balanceamento(raiz->dir) > 0)
	{
		raiz->dir = rotacaoDireita(raiz->dir);
		return rotacaoEsquerda(raiz);
	}

	return raiz;
}

bool verificaExisteArvore(No* raiz) {
	if (raiz == NULL) {
		system("CLS");
		cout << "NECESSÁRIO ADICIONAR ARQUIVO PRIMEIRO!" << endl;
		return false;
	}
	else
	{
		return true;
	}
}

void limpaTela() {
	system("cls || clear");
}

void imprimeMenu() {
	cout << "---------------------------------------" << endl;
	cout << "Menu:" << endl;
	cout << "1. Buscar palavra" << endl;
	cout << "2. Deletar no da arvore" << endl;
	cout << "3. Adicionar arquivo" << endl;
	cout << "4. Imprimir arvore" << endl;
	cout << "0. Sair" << endl;
	cout << "---------------------------------------" << endl;
	cout << "Digite sua opcao: ";

}

int main()
{
	setlocale(LC_ALL, "Portuguese");
	No* raiz = NULL;
	int opcao = -1;
	char nomeArquivo[50];
	char palavraPesquisada[50];
	char voltar[] = "0";
	do
	{

		imprimeMenu();
		cin >> opcao;

		cout << endl;
```

```
		switch (opcao)
		{
		case 1:
			cin.ignore();

			if (verificaExisteArvore(raiz) == false)
				break;

			do
			{
				cout << "Digite a palavra a ser pesquisada ou 0 para
voltar ao menu principal: ";
				cin.getline(palavraPesquisada, 50);
				tornaMaiusculo(palavraPesquisada);
				pesquisarPalavra(raiz, palavraPesquisada);
			} while (strcmp(palavraPesquisada, voltar) != 0);

			limpaTela();

			break;

		case 2:
			if (verificaExisteArvore(raiz) == false)
				break;

			cout << "Digite a palavra a ser deletada: ";
			cin.ignore();
			cin.getline(palavraPesquisada, 50);
			tornaMaiusculo(palavraPesquisada);
			raiz = deletarNo(raiz, palavraPesquisada);
			break;
		case 3:
			cin.ignore();
			cout << "Digite o nome do arquivo a ser adicionado ou 0 para
voltar ao menu principal: ";
			cin.getline(nomeArquivo, 50);
			processarArquivo(&raiz, nomeArquivo, voltar);
			limpaTela();

			if (raiz != NULL)
				cout << "Arquivo(s) lido com sucesso!";

			break;

		case 4:
			if (verificaExisteArvore(raiz) == false)
				break;

			cout << "Imprimindo a arvore:" << endl;
			imprimirArvore(raiz);
			break;

		case 0:
			cout << "Fim do programa" << endl;
			return 0;
			break;

		default:
			opcao = -1;
			cout << "Opcao invalida. Digite novamente." << endl;
			break;
		}
```

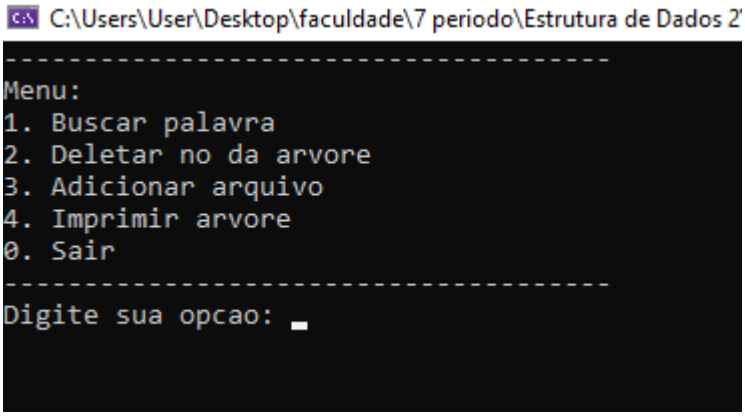
```
            cout << endl;
        } while (opcao != 0);

        return 0;
}
```

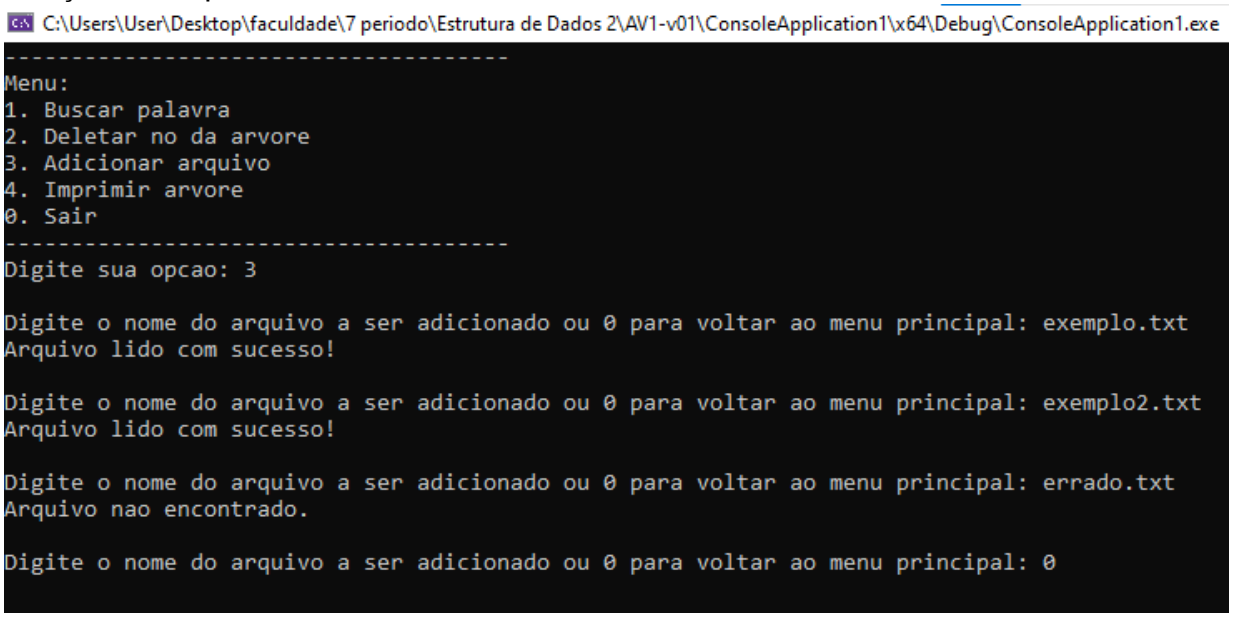
2.2 PRINTS DA EXECUÇÃO DO SISTEMA

A seguir os prints da execução do programa:

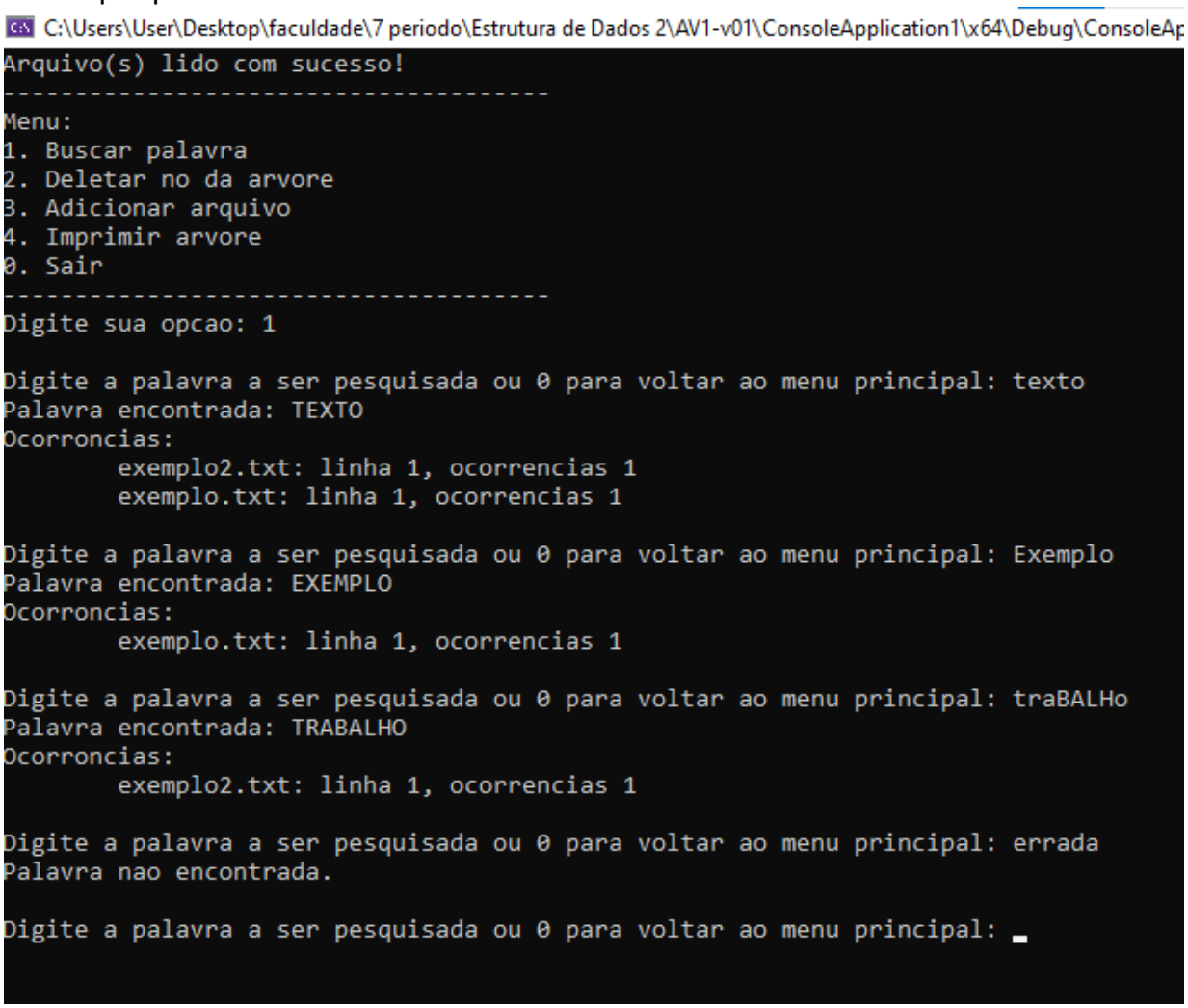
Menu do sistema:



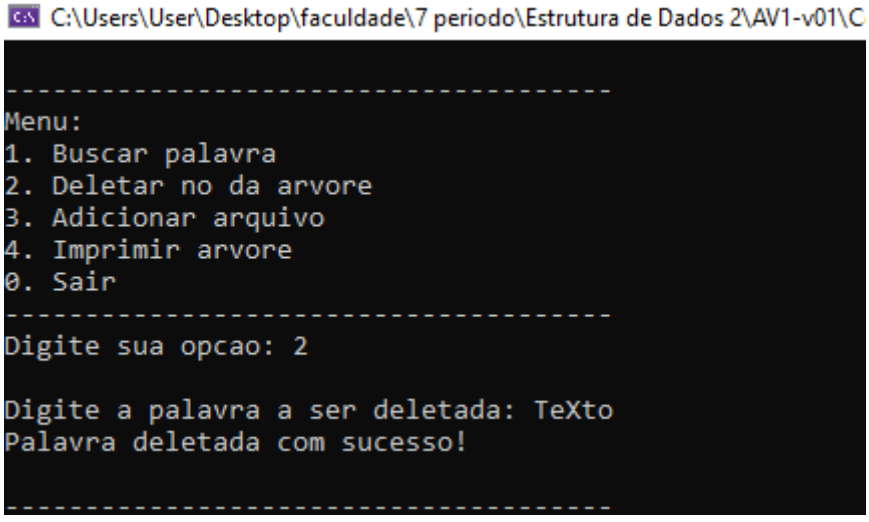
Adição de arquivos:



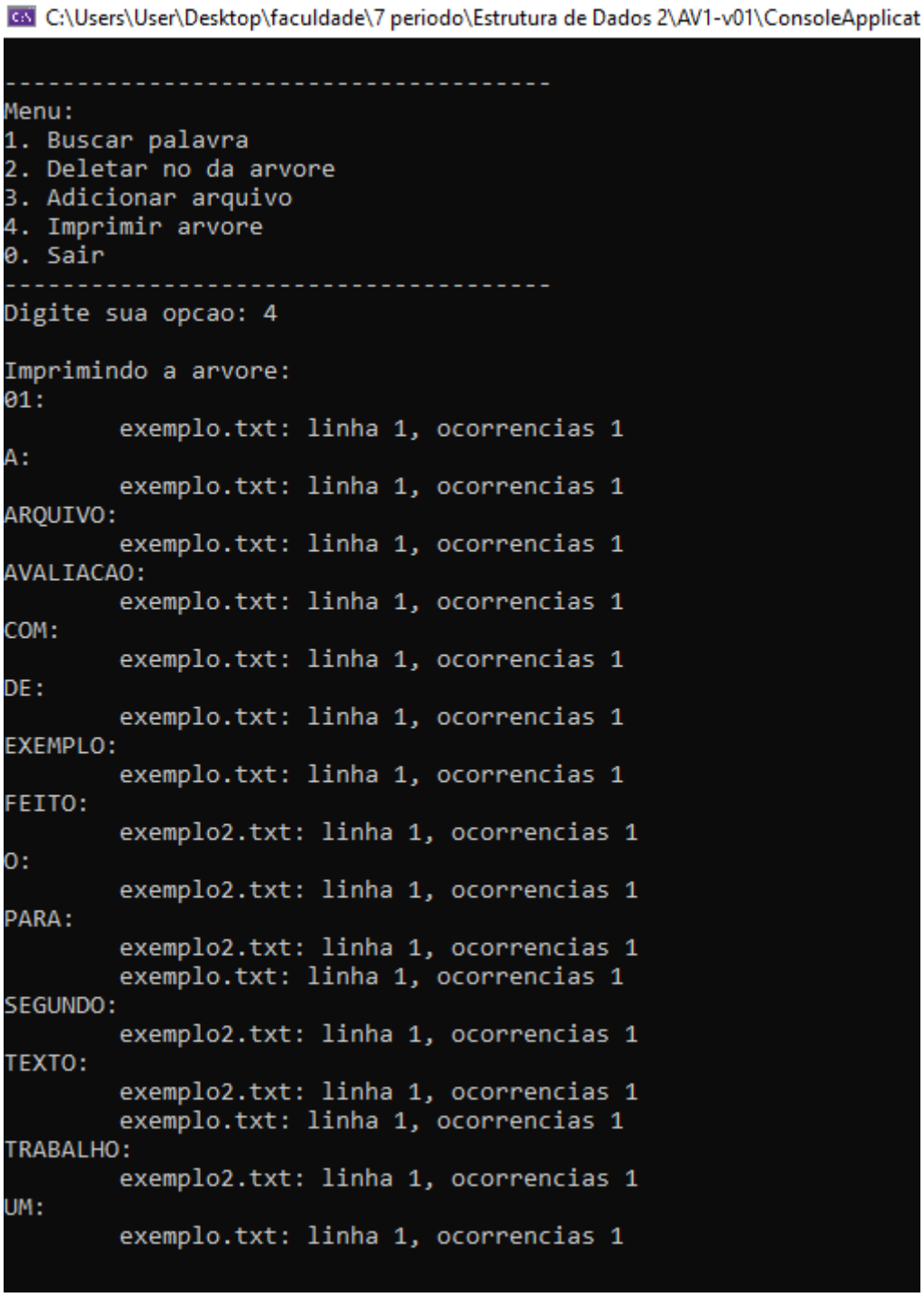
Busca por palavras:



Deletar palavra:



Imprimir arvore:



3 BIBLIOGRAFIA

https://www.programiz.com/dsa/dynamic-programming

https://www.geeksforgeeks.org/greedy-algorithms/

https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamicprogramming.html