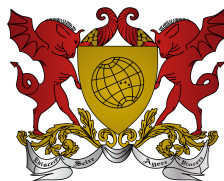




SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO UNIVERSIDADE FEDERAL
DE VIÇOSA · UFV
CAMPUS FLORESTAL



Trabalho 3 - AEDS 1
Dicionário Organizado

Gabriel Santos Ferreira de Pádua [EF04705]
João Pedro Rafael Santos Silva [EF03899]

Florestal - MG
2021



Sumário

1. Introdução	3
2. Organização	3
3. Desenvolvimento	9
3.2 identificação de palavras:	9
3.3 ordenação:	9
3.4 Continuação do código:	9
4. Resultados	10
5. Conclusão	10
6. Referências	11



1. Introdução

Inicialmente tivemos o objetivo de projetar um programa que fosse capaz de montar um dicionário a partir de textos estabelecidos como entrada, o algoritmo é capaz de separar e organizar palavra por palavra, usando como estrutura de dados a lista encadeada, vetores para listas de palavras, manipulação de arquivos (leitura e escrita) e algoritmos de ordenação.

O algoritmo organiza e separa palavras, ao passo em que verifica se elas já existem no dicionário, as coloca em uma lista encadeada de palavras e faz a contagem de suas aparições por linha.

Ao fim da leitura dos arquivos é executado um algoritmo para ordenar as palavras dentro das seções de letras do dicionário para que elas se disponham em ordem alfabética. Estão dispostas em vetores, e por mais que acrescentadas dentro das respectivas letras que iniciam não estão em ordem (palavra-palavra).

Projetado um menu, onde ficará disponíveis funções para o usuário, permitindo-o:

- Entrar com arquivos “.txt” para a criação de um dicionário referente a leitura - permite a entrada de sucessivos arquivos, mantendo a gravação das palavras do arquivo anterior;
- Mostrar na saída padrão todo o dicionário que está na memória.
- Mostrar na saída padrão o dicionário de maneira ordenada.
- Mostrar na saída padrão uma seção de letra do dicionário.
- Mostrar na saída padrão uma seção de letra do dicionário de maneira ordenada.
- Limpar dicionário
- Criação de novo arquivo com nome especificado pelo usuário para gravação dos dados (independentemente um será criado no formato .txt contendo os resultados de saída - output).
- Remover palavra passada por parâmetro.

2. Organização

Consiste em uma pasta principal que contém todos os arquivos necessários para o funcionamento do programa

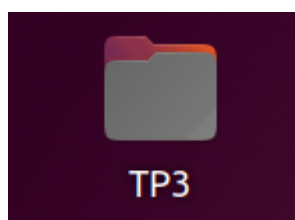


figura 1 - pasta principal



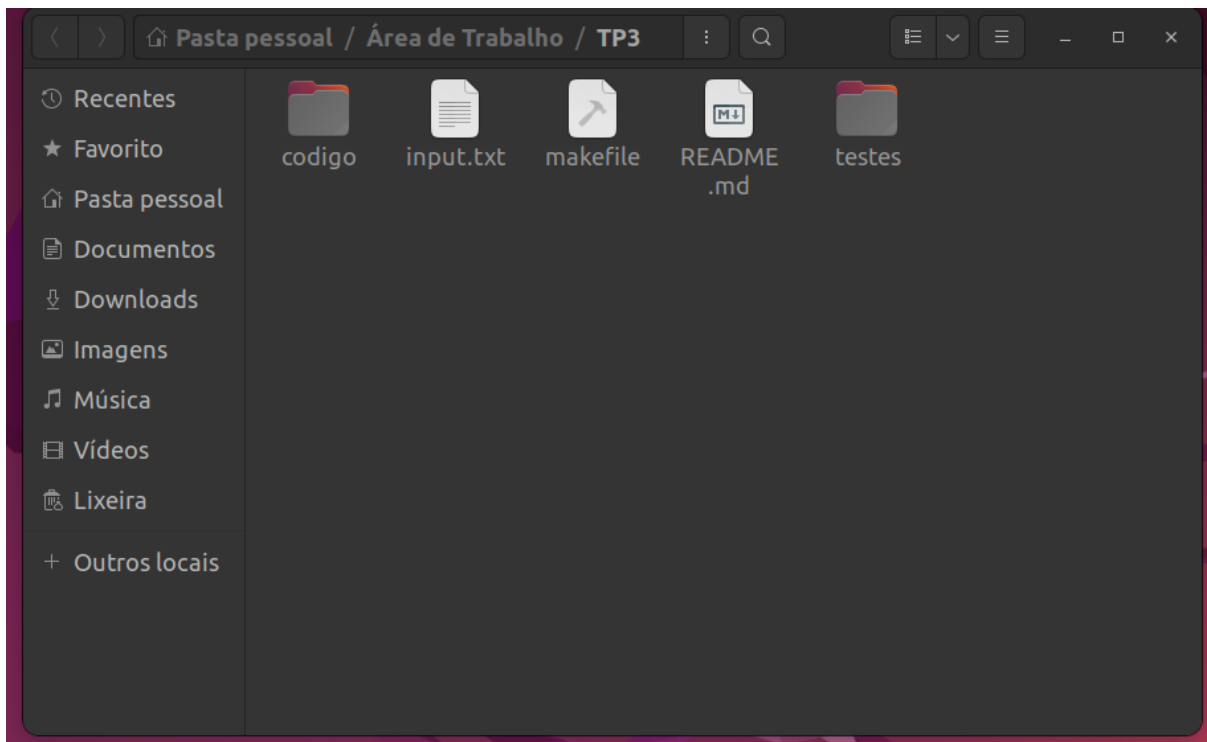


figura 2 - raiz do projeto

O arquivo “**input.txt**” serve como arquivo de texto para testes, e importe ressaltar que é possível a entrada de outros arquivos de texto.

Foi criado um arquivo **Makefile** para facilitar a compilação e execução do programa e teste realizados, comando make disponíveis no terminal (linux):

- “make” ou “make all” : compila o código.
- “make testedicionario”: compila o TAD Dicionario.
- “make testeLista”: compila o TAD Lista.
- “make testepalavra”: compila o TAD Palavra.
- “make testelistapalavras”: compila o TAD ListaPalavras.
- “make testevetorpalavras”: compila o TAD VetorPalavras.
- “make clear”: força a remoção dos arquivos criados: “*.o *.h.gch *.out *.exe”.

Arquivo “**output.txt**” refere-se ao arquivo de saída padrão do programa dicionario, arquivo criado contendo os dados processados.

Arquivo “**README.md**” possui descrições e planejamentos futuros sobre o código.

A pasta “codigo” consiste em quatro módulos separados mas dependentes entre si e o arquivo principal, onde é chamado o módulo principal “dicionario”, e é implementado a entrada e saída de interação com o usuário.



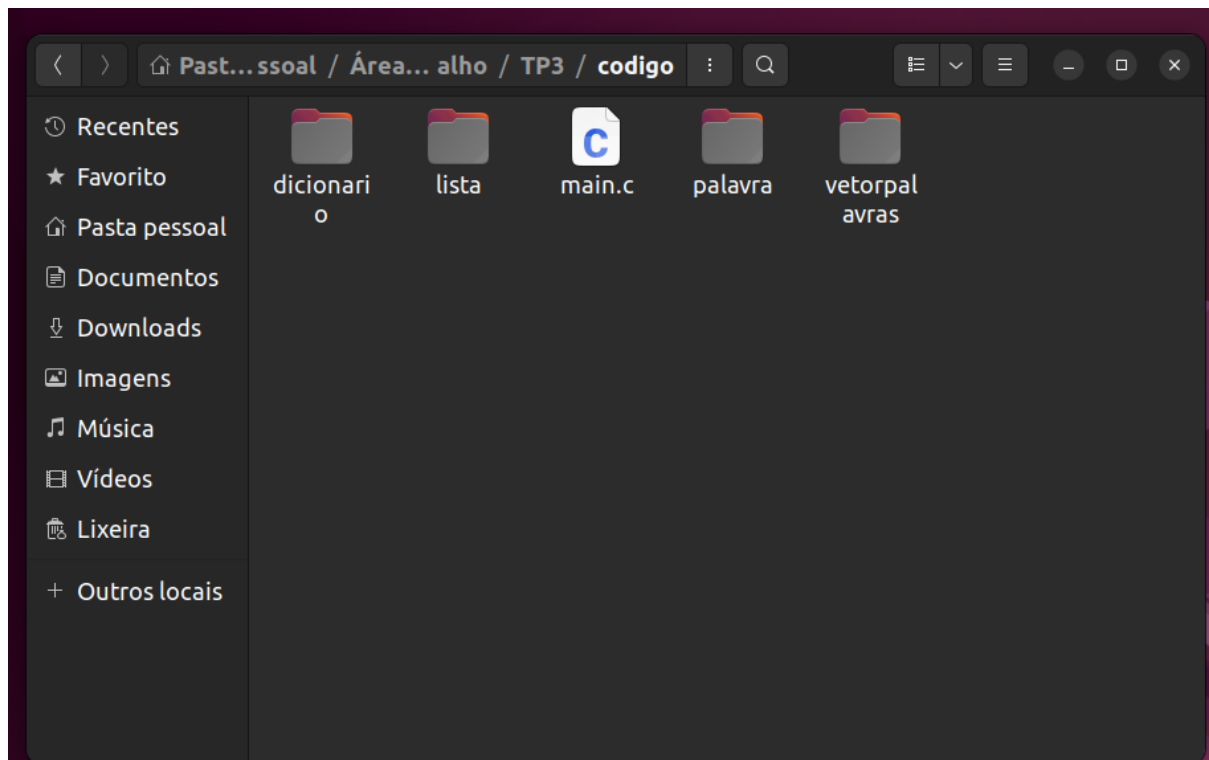


figura 3 - pasta “codigo”

O módulo “lista” contém a implementação de uma lista encadeada simples (Figura 4) de inteiros (Titem) para representar as linhas do arquivo em que as palavras ocorreram.

```
typedef struct sCelula *PsCelula;

typedef struct sCelula
{
    Titem Item;
    PsCelula pProx; /* Apontador pProx; */
} CelulaLista;

typedef struct slista
{
    int nItens;
    PsCelula pPrimeiro;
    PsCelula pUltimo;
} ListaDeOcorrencias;
```

figura 4 - ListaDeOcorrencias



As funções que integram o tipo ListaDeOcorrencias são características de uma lista encadeada.

```
void fazListaVazia(ListaDeOcorrencias **);
CelulaLista *criaCelula(Titem item);
void adicionarCelula(ListaDeOcorrencias *lista,
CelulaLista *item);
void imprimeLista(ListaDeOcorrencias *l, FILE*);
void removeUltimaCelula(ListaDeOcorrencias *l);
void nitems(ListaDeOcorrencias *l, int *pres,
FILE* output);
int listaEhVazia(ListaDeOcorrencias *lista);
```

figura 5 - ListadeOcorrencias

O módulo “palavra” contém a implementação da abstração do TAD Palavra que contém o vetor de caracteres da palavra e a lista de linhas de ocorrência no texto (Figura 6) para representar as linhas do arquivo em que as palavras ocorreram.

```
#include "../lista/lista.h"
typedef char *String;

typedef struct
{
    String string;
    ListaDeOcorrencias *lista;
} Palavra;
```

figura 6 - Palavra

As funções que integram o tipo Palavra são :





```
void criaPalavraVazia(Palavra **palavra);  
void preencheCadeiaDeCaracteres(Palavra *palavra,  
char * string);  
void adicionaOcorrencia (Palavra* palavra, int  
linha);  
String retornaCadeiaDeCaracteres(Palavra  
*palavra);  
void imprimeCadeiaDeCaracteres(Palavra *palavra);  
void imprimePalavra(Palavra *palavra, FILE  
*output);
```

figura 7 - Métodos Palavra

modulo “vetorpalavra” esta implementado o vetor para tratar as palavras pelos algoritmos de ordenação e suas funções

```
#include "../palavra/palavra.h"  
  
typedef struct sCelulaListaPalavra* PsCelulaListaPalavra;  
typedef struct sCelulaListaPalavra {  
  
    Palavra *palavra;  
    PsCelulaListaPalavra prox;  
  
} CelulaListaPalavra;  
  
typedef CelulaListaPalavra* VetorCelulaPalavra;  
  
typedef struct {  
  
    int nItens;  
    VetorCelulaPalavra vetor;  
  
} ListaPalavras;  
  
typedef void (*SortFunction)(int, VetorCelulaPalavra);
```

figura 8 - “vetorpalavra.h”



Em suas funções estão inclusas os algoritmos de ordenação e as demais:

```
2
3 void sort(int n, VetorCelulaPalavra vetor, SortFunction func);
4 void bubbleSort(int n, VetorCelulaPalavra vetor);
5 void insertionSort(int n, VetorCelulaPalavra vetor);
6 void heapsort(int n, VetorCelulaPalavra vetor);
7 void quickSort(int n, VetorCelulaPalavra vetor);
8 void selectionSort(int n, VetorCelulaPalavra vetor);
9 void shellSort(int n, VetorCelulaPalavra vetor);
10
11 void criaNovaListaDePalavrasVazia(ListaPalavras** lista);
12 VetorCelulaPalavra criaCelulaListaPalavras(Palavra *palavra);
13 void inserePalavra(ListaPalavras* lista, VetorCelulaPalavra celula);
14 void popCelulaListaPalavras(ListaPalavras *lista);
15 void removeCelulaListaPalavra(ListaPalavras *lista, String palavra);
16 int verificaPalavraExisteNaLista(ListaPalavras *, String , VetorCelulaPalavra );
17 int numeroDePalavras(ListaPalavras *lista);
18 void imprimelistapalavras(ListaPalavras *lista, FILE* output);
19 int compareString(String s1, String s2);
20 void insertionSort(int n, VetorCelulaPalavra vetor);
```

Figura 9 - Funções “vetorpalavra”

○ ○ ○

```
void inicializaDicionario(Dicionario ** dict);
int constroiDicionario(String filename, Dicionario
*dict);
void imprimeDicionario (Dicionario *dict, FILE*
output);
int verificaLetraExisteNoDicionario(String palavra,
Dicionario *dict, PPalavraDict ref);
void imprimePalavraDict(DictSession *palavradict,
FILE *output);
void mostraTodasAsPalavras(Dicionario *);
```

Figura 10 - Métodos Dicionário



Já o arquivo “**main.c**” trata-se do código fonte do programa.(figura 3).

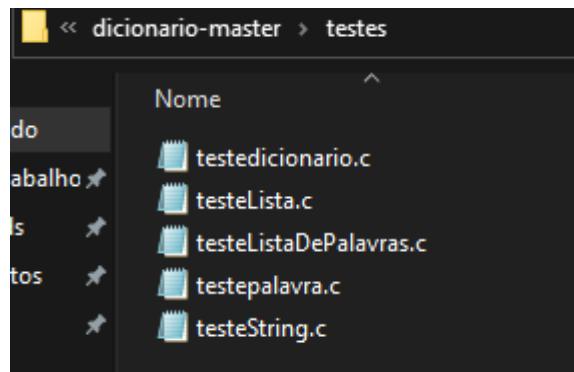


figura 11 - pasta de testes

Na pasta “/teste” encontram-se todos os arquivos .c necessários para testar os TADs criados (figura 12).

3. Desenvolvimento

A produção do programa teve início com um problema, que já a definição do grupo dos alunos se deu tardia, entretanto após reuniões conseguimos alinharmos as ideias e entendimentos centrais a respeito de como funcionaria o código e também como deveríamos fazer as implementações, levando em conta: um modo generalizado de declarar variáveis, comentar partes do códigos que fossem mais complexas ao entendimento deixando da maneira mais clara e explicativa possível.

O algoritmo foi trabalhado na estrutura de dados de listas encadeadas e vetores, listas encadeadas para organizar as seções e ocorrências das palavras, e vetores dentro das listas para serem usados como estrutura para algoritmos de ordenação.

3.2 identificação de palavras:

O arquivo é lido de linha a linha, ao chegar em um <espaço> retorna todo char até o espaço, considerando- o uma palavra.

3.3 ordenação:

Usamos algoritmos pré-estabelecidos para podermos organizar os vetores de palavras, sendo estes Selectsort, Insertsort, Bubblesort, Heapsort, Shellsort, Quicksort.

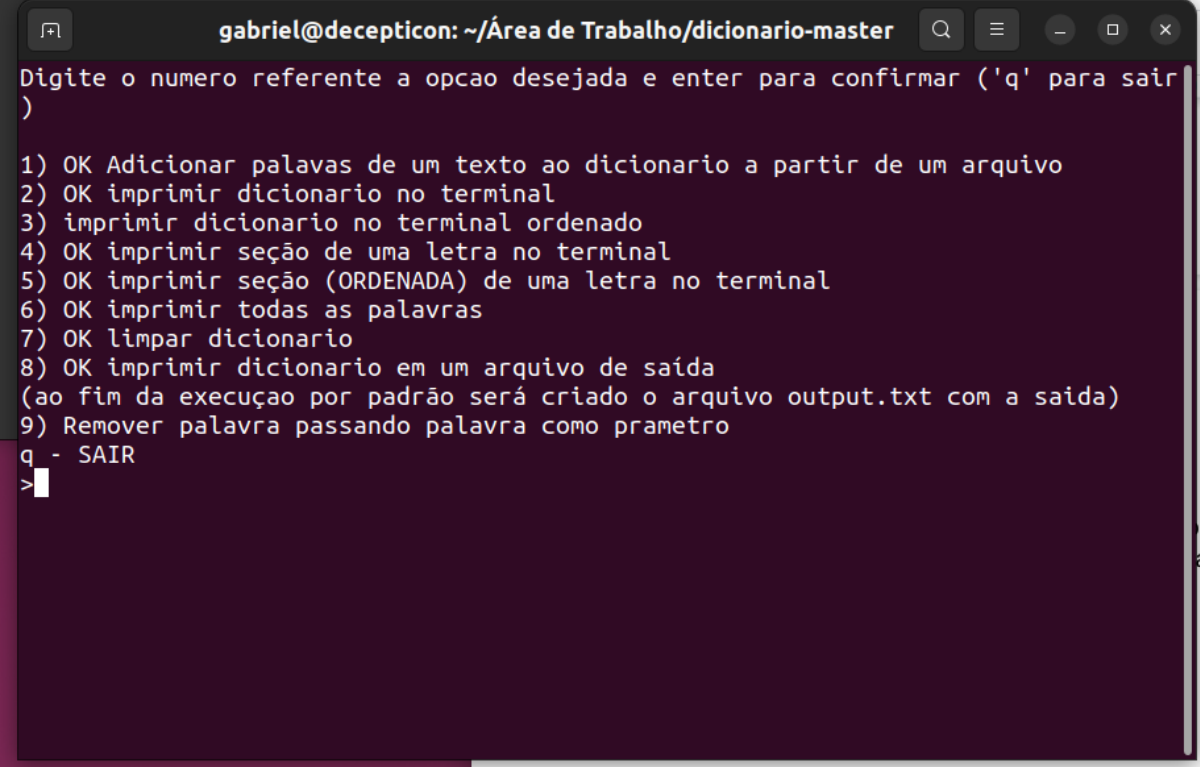
3.4 Continuação do código:

Esse algoritmo se mostrou muito promissor, pretendemos salvá-lo para aprimorar e usar em outras tarefas.



4. Resultados

Os resultados foram satisfatórios e esperados com o desejado, foi montado uma interface simples ,porém bem construída e intuitiva para servir de menu de opções ao usuário. Destacando a limpeza e clareza ao solicitar os comando no terminal e sua “interface” que nao contem lixo visual:



```
gabriel@decepticon: ~/Área de Trabalho/dicionario-master
Digite o numero referente a opcao desejada e enter para confirmar ('q' para sair)
)
1) OK Adicionar palavras de um texto ao dicionario a partir de um arquivo
2) OK imprimir dicionario no terminal
3) imprimir dicionario no terminal ordenado
4) OK imprimir seção de uma letra no terminal
5) OK imprimir seção (ORDENADA) de uma letra no terminal
6) OK imprimir todas as palavras
7) OK limpar dicionario
8) OK imprimir dicionario em um arquivo de saída
(ao fim da execucao por padrão será criado o arquivo output.txt com a saída)
9) Remover palavra passando palavra como prametro
q - SAIR
>
```

figura 12 - menu



Nosso programa possui uma maneira de mostrar a eficiência de cada algoritmo. Ao entrar com um arquivo de texto (opção 1), e selecionar opções para mostrar de maneira ordenada (escolhendo assim um algoritmo de ordenação específico), este retornará junto a tarefa de organizar as palavras a quantidade comparações, movimentações o tempo gasto - lembrando que o tempo pode variar de uma especificação de máquina para outra

```
gabriel@decepticon: ~/Área de Trabalho/dicionario-master
Digite o numero referente a opcao desejada e enter para confirmar ('q' para sair)
)
1) Adicionar palavras de um texto ao dicionario a partir de um arquivo
2) imprimir dicionario no terminal
3) imprimir dicionario no terminal ordenado
4) imprimir seção de uma letra no terminal
5) imprimir seção (ORDENADA) de uma letra no terminal
6) imprimir todas as palavras
7) limpar dicionario
8) imprimir dicionario em um arquivo de saída
(ao fim da execução por padrão será criado o arquivo output.txt com a saída)
9) Remover palavra passando palavra como parametro
q - SAIR
>1
digite o nome do arquivo :input.txt
```

figura 13 - opção 1

```
gabriel@decepticon: ~/Área de Trabalho/dicionario-master
Digite o numero referente a opcao desejada e enter para confirmar ('q' para sair)
)
1) Adicionar palavras de um texto ao dicionario a partir de um arquivo
2) imprimir dicionario no terminal
3) imprimir dicionario no terminal ordenado
4) imprimir seção de uma letra no terminal
5) imprimir seção (ORDENADA) de uma letra no terminal
6) imprimir todas as palavras
7) limpar dicionario
8) imprimir dicionario em um arquivo de saída
(ao fim da execução por padrão será criado o arquivo output.txt com a saída)
9) Remover palavra passando palavra como parametro
q - SAIR
>5
digite a letra que deseja buscar: a
```

figura 14 - opção 5 (letra “a”)



```
gabriel@decepticon: ~/Área de Trabalho/dicionario-master
Digite o numero referente a opcao desejada e enter para confirmar ('q' para sair)
)
1) Adicionar palavras de um texto ao dicionario a partir de um arquivo
2) imprimir dicionario no terminal
3) imprimir dicionario no terminal ordenado
4) imprimir seção de uma letra no terminal
5) imprimir seção (ORDENADA) de uma letra no terminal
6) imprimir todas as palavras
7) limpar dicionario
8) imprimir dicionario em um arquivo de saída
(ao fim da execução por padrão será criado o arquivo output.txt com a saída)
9) Remover palavra passando palavra como parametro
q - SAIR
>5
digite a letra que deseja buscar: a
Qual algoritmo de ordenação deseja usar?
1) bubbleSort, 2) insertionSort, 3) heapsort, 4) quickSort, 5) selectionSort, 6
) shellSort
3
```

figura 15 - opção 3 - heapsort

```
gabriel@decepticon: ~/Área de Trabalho/dicionario-master
-----
Palavra   : a
Linhas    : |1|2|3|4|5|
-----
Palavra   : aniversario
Linhas    : |5|
-----
Palavra   : apagou
Linhas    : |3|
-----
0.007ms 0s
Tempo gasto: 0.007 ms.
Tempo gasto: 0 s.
Numero de comparacoes: 16
Numero de movimentacoes: 4
*****

ENTER para continuar
```

figura 16 - comparações, tempo e movimentações mostrados



5. Conclusão

Os resultados foram alcançados, entendimento sobre a complexidade computacional e o uso de estruturas de dados para tratar informações dinâmicas quanto a seus tamanhos.

Através do uso de algoritmos de ordenação tivemos ainda mais contato com o comportamento computacional e o escalamento que determinadas ações podem ter de acordo com a estrutura e tratamento que os dados podem receber.

Pelo fato do alto potencial computacional atual não tivemos muita diferença entre o tempo de execução ao usar diferentes algoritmos de ordenação.

Embora seja certo que algoritmos simples funcionam de maneira excepcional para entrada de arquivos pequenos e de maneira ruim para arquivos grande; o que se prove ao contrário no caso de algoritmos complexos de ordenação, que são bons para casos grandes e desnecessários para entradas pequenas.

6. Referências

Pesquisamos meios, funções, códigos e algoritmos para adequarmos e compararmos nos seguintes veículos.

- github.com
- deviamidia.com.br
- vivalinux.com.br
- w3schools.blog
- asciitable.com
- clubedohardware.com.br
- linguagemc.com.br
- Slides da Thais
- livro ziviani - Projeto de Algoritmos

