

Estrutura de Dados I

Exercício Programa III - Verificador Ortográfico

Rulian Dos Reis

01 de outubro de 2021

# Sumário

<b>I</b>	<b>Introdução</b>	<b>3</b>
<b>II</b>	<b>Metodologia adotada para os experimentos</b>	<b>3</b>
<b>1</b>	<b>Linguagem</b>	<b>3</b>
<b>2</b>	<b>Estruturas utilizadas</b>	<b>3</b>
2.1	Fila . . . . .	3
2.2	Árvores binárias de busca . . . . .	4
<b>3</b>	<b>Banco de Palavras</b>	<b>5</b>
<b>4</b>	<b>Comparação de Palavras</b>	<b>5</b>
<b>5</b>	<b>Exibição de sugestões</b>	<b>5</b>
<b>6</b>	<b>Implementação</b>	<b>5</b>
<b>III</b>	<b>Resultados</b>	<b>6</b>
<b>IV</b>	<b>Conclusões</b>	<b>11</b>

## Parte I

# Introdução

---

Uma estrutura de dados é uma forma inteligente de armazenar e organizar informações de maneira que possamos realizar operações de forma mais eficiente. Diferentes tipos de estruturas de dados são mais adequadas para diferentes tipos de aplicações, dentre elas, estão presentes listas encadeadas, pilhas, listas, árvores binárias de busca, *heaps*, entre outras. Cada uma delas possui uma implementação diferente, com diferentes atributos e finalidades.

No presente relatório, serão explicadas duas dessas estruturas: filas e árvores binárias de busca, além de explicitar a diferença de uso entre árvores binárias de busca balanceadas e não balanceadas. Tais estruturas foram utilizadas na implementação de um verificador ortográfico, ferramenta cotidianamente utilizada para revisar textos. Nas sessões abaixo, destrincharemos o programa a fim de entendê-lo minuciosamente.

## Parte II

# Metodologia adotada para os experimentos

---

## 1 Linguagem

A linguagem utilizada para a implementação do verificador ortográfico foi a linguagem *C*, mais especificamente, a versão de 2011.

## 2 Estruturas utilizadas

### 2.1 Fila

Uma fila é uma estrutura de dado linear que segue uma ordem particular na forma em que as operações são executadas. Tal ordem pode ser explicada pela frase em inglês "*First in*

*first out*", que, traduzida para o português significa algo como "primeiro a chegar, primeiro a sair".

Na implementação, deve-se basicamente definir dois atributos principais, são eles: início e fim. Além destes dois atributos principais, também foi acrescentado na *struct Fila* um atributo que armazenava o tamanho da fila, ou seja, a quantidade de itens(ou nós) presentes na mesma.

As principais funções que realizam o processo lógico na estrutura são as funções *enqueue*, que acrescenta um item no fim da fila, e a função *dequeue*, que remove o primeiro item da fila.

## 2.2 Árvores binárias de busca

Uma árvore binária de busca é uma estrutura um pouco mais complexa de se entender que a fila. Ela possui um nó principal, chamado comumente de raiz e, para cada nó da árvore, pode existir um nó a esquerda e/ou um a direita. Eles são organizados da seguinte forma: à esquerda, ficam os dados de menor grandeza em relação ao nó analisado, já a direita, os de maior grandeza.

Essa estrutura pode ser dividida em duas: árvores balanceadas e árvores não balanceadas. O que as difere é o seguinte: uma árvore só pode ser considerada balanceada caso, para qualquer nó, a altura das subárvores, ou seja, o número de nós no caminho mais longo presente na árvore, se distanciem em no máximo uma unidade.

Para uma árvore não balanceada, as funções de inserção de um nó e remoção de um nó são simples. Para inserir, basta percorrer os nós, comparando o dado inserido com os dados dos nós. Caso o dado seja de maior grandeza, deve-se caminhar para a direita, caso contrário, para esquerda, até que chegue num último nó, determinado de nó folha, e acrescentá-lo à direita ou à esquerda deste nó, seguindo a mesma lógica. A função de remoção também é bem simples: percorre-se a árvore da mesma maneira, até identificar o nó que contenha a informação a ser removida. Assim que encontrá-lo, apenas é necessário removê-lo e organizar as ligações dos nós restantes da árvore.

Já para uma árvore balanceada, é necessário no final de cada um dos passos ditos acima, garantirmos que a árvore continue balanceada. Esse processo é realizado com o auxílio de quatro funções: rotação simples à direita, rotação simples à esquerda, rotação dupla à direita e rotação dupla à esquerda. Cada uma delas é realizada baseado no fator de balanceamento da árvore, que consiste na altura do nó raiz da subárvore à direita subtraído pela altura do nó raiz da subárvore à esquerda.

### 3 Banco de Palavras

Como parâmetros de comparação, foram utilizados dicionários em português e em inglês, disponibilizados pelo docente, para a realização dos testes. Cada um deles era passado como argumento na compilação e geração do executável pelo arquivo *Makefile*. Eles possuíam, em cada linha, palavras escritas corretamente, que foram utilizadas para a identificação de possíveis erros e na sugestão de novas palavras para os substituírem.

### 4 Comparação de Palavras

Para a comparação e verificação das palavras, utilizou-se da função *strcmp*, da biblioteca *<string.h>*. Essa função, por sua vez, é uma função do tipo *int* que recebe duas *strings*. Caso o conteúdo das mesmas sejam iguais, ela retorna zero. Caso contrário, se o primeiro caractere que as diferencia for tiver um maior valor na primeira *string*, ela retorna  $>0$ , se não,  $<0$ .

### 5 Exibição de sugestões

O método utilizado para a exibição de sugestões foi o seguinte: com o auxílio da função *distanciaEdicao*, é verificado a quantidade de caracteres que diferenciam a palavra analisada e cada uma das palavras do dicionário inserido. Quanto maior o valor retornado, mais caracteres se diferenciam. Dessa forma, deve-se definir um limite de caracteres diferentes no momento que a análise é feita, de forma que: quanto maior esse limite, mais palavras serão validadas como sugestão para a palavra incorreta.

### 6 Implementação

O programa possui uma lógica relativamente simples: cada uma das palavras do dicionário é inserida em uma árvore binária de busca, definida se é balanceada ou não balanceada no arquivo *Makefile*. Assim, para cada uma das palavras do texto verificado, uma comparação (explicada na sessão 4) é realizada e, nos casos em que a palavra não é encontrada na árvore, ela é acrescentada à uma fila com a função *enqueue*, utilizada apenas para armazenar as palavras incorretas. Este processo só ocorre caso a palavra não esteja presente na fila, a fim de apresentar sugestões de correção apenas uma vez para cada palavra.

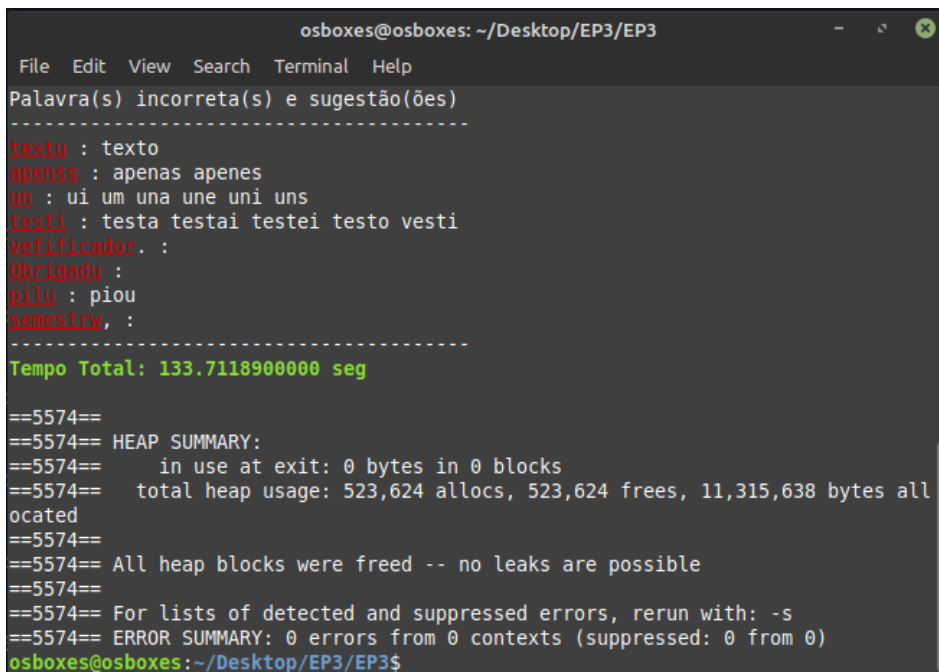
Assim, ao finalizar o texto, é iniciado o percorrido da fila de palavras incorretas. Para cada primeiro nó da fila, analisamos a palavra incorreta e sugerimos outras utilizando o

procedimento explicado na sessão 5. Após isso, com a função *dequeue*, excluimos o primeiro nó, e analisamos o novo primeiro nó, até que a fila esteja concluída. A estrutura da fila foi utilizada pois garante que as primeiras palavras inseridas são listadas como erro primeiro, mantendo a ordem lógica das aparições.

## Parte III

# Resultados

Inicialmente, verificamos que não houve nenhum vazamento de memória durante o processo executando o comando *make memcheck* no terminal.



```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
Palavra(s) incorreta(s) e sugestão(ões)
-----
textu : texto
apenss : apenas apenes
un : ui um una une uni uns
testi : testa testai testei testo vesti
verificador. :
Obrigado :
piu : piou
semestru, :
-----
Tempo Total: 133.7118900000 seg

==5574==
==5574== HEAP SUMMARY:
==5574==   in use at exit: 0 bytes in 0 blocks
==5574== total heap usage: 523,624 allocs, 523,624 frees, 11,315,638 bytes allocated
==5574==
==5574== All heap blocks were freed -- no leaks are possible
==5574==
==5574== For lists of detected and suppressed errors, rerun with: -s
==5574== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
osboxes@osboxes:~/Desktop/EP3/EP3$
```

Figura 1: Teste de vazamento de memória

Utilizando o dicionário Português.txt disponibilizado, executamos o programa em uma máquina virtual com de 4096 MB de memória RAM. Foram utilizados para os testes à seguir ambos os formatos de árvore: balanceada (AVL) e não balanceada. Como parâmetro para sugestão, foi utilizado apenas 1 (um) caractere diferente da palavra analisada, portanto, as sugestões foram bem limitadas.

```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
osboxes@osboxes:~/Desktop/EP3/EP3$ make run
./EP3 -d Portugues.txt -t TextoEP3.txt -avl
Este texto é apenas um testi do verificador. Obrigadu piau semestrw, professor.

-----
-      Número de palavras lidas: 12
-      Número de palavras incorretas: 8
Palavra(s) incorreta(s) e sugestão(ões)
-----
texto : texto
apenas : apenas apenas
un : ui um una une uni uns
testi : testa testai testeí testo vesti
verificador. :
obrigadu :
piaú : piou
semestrw, :
-----
Tempo Total: 2.3127520000 seg
osboxes@osboxes:~/Desktop/EP3/EP3$
```

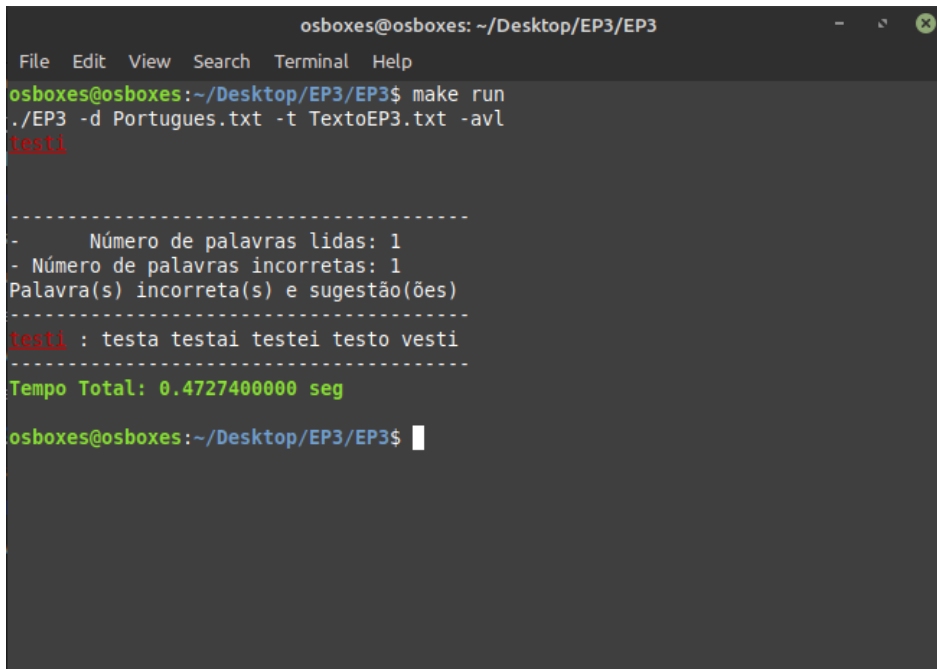
Figura 2: Funcionamento do programa com uma árvore balanceada

```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
osboxes@osboxes:~/Desktop/EP3/EP3$ make run
./EP3 -d Portugues.txt -t TextoEP3.txt
Este texto é apenas um testi do verificador. Obrigadu piau semestrw, professor.

-----
-      Número de palavras lidas: 12
-      Número de palavras incorretas: 8
Palavra(s) incorreta(s) e sugestão(ões)
-----
texto : texto
apenas : apenas
un : uh
testi : testa
verificador. :
obrigadu :
piaú : piou
semestrw, :
-----
Tempo Total: 923.1731590000 seg
osboxes@osboxes:~/Desktop/EP3/EP3$
```

Figura 3: Funcionamento do programa com uma árvore não balanceada

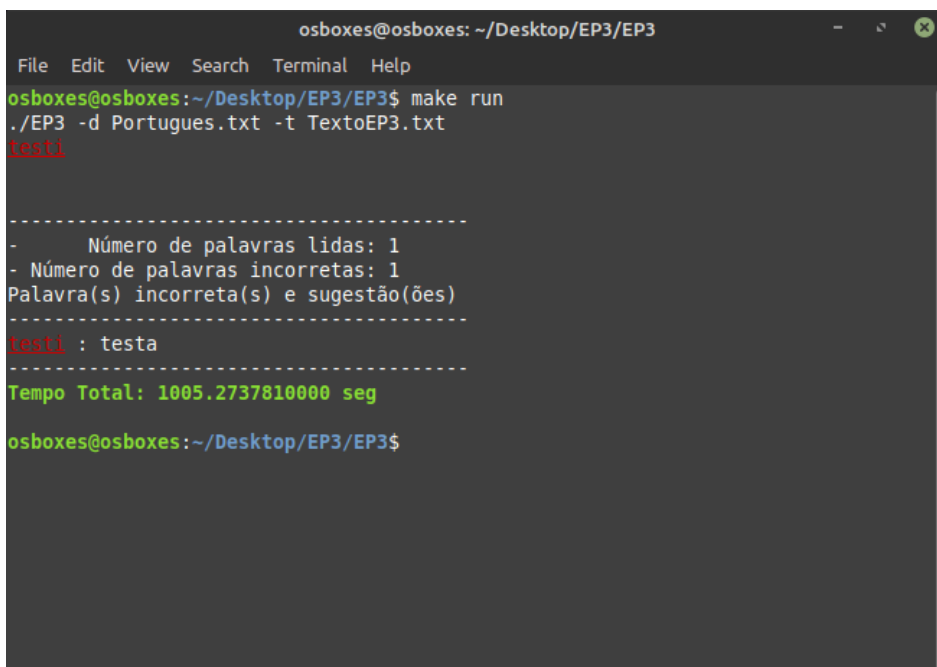
Segue abaixo outra execução, no entanto, com uma única palavra no texto analisado :



```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
osboxes@osboxes:~/Desktop/EP3/EP3$ make run
./EP3 -d Portugues.txt -t TextoEP3.txt -avl
testi

-----
-      Número de palavras lidas: 1
-      Número de palavras incorretas: 1
Palavra(s) incorreta(s) e sugestão(ões)
-----
testi : testa testai testei testo vesti
-----
Tempo Total: 0.472740000 seg
osboxes@osboxes:~/Desktop/EP3/EP3$
```

Figura 4: Única palavra analisada com uma árvore balanceada



```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
osboxes@osboxes:~/Desktop/EP3/EP3$ make run
./EP3 -d Portugues.txt -t TextoEP3.txt
testi

-----
-      Número de palavras lidas: 1
-      Número de palavras incorretas: 1
Palavra(s) incorreta(s) e sugestão(ões)
-----
testi : testa
-----
Tempo Total: 1005.2737810000 seg
osboxes@osboxes:~/Desktop/EP3/EP3$
```

Figura 5: Única palavra analisada com uma árvore não balanceada



Utilizando o livro texto "**Dom Casmurro**" de Machado de Assis, disponibilizado no site indicado pelo docente **Projeto Gutenberg**, e o dicionário "Portugues.txt" o teste com a árvore balanceada ficou da seguinte forma:

```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
edicao.
Most people start at our Web site which has the main PG search
facility: www.gutenberg.org

This Web site includes information about Project Gutenberg-tm,
including how to make donations to the Project Gutenberg Literary
archive Foundation, how to help produce our new ebooks, and how to
subscribe to our email newsletter to hear about new ebooks.

-----
- Número de palavras lidas: 67988
- Número de palavras incorretas: 12735
Palavra(s) incorreta(s) e sugestão(ões)
-----
the : lhe
Project :
Gutenberg :
ebook :
of : o oi os ou
by :
this :
```

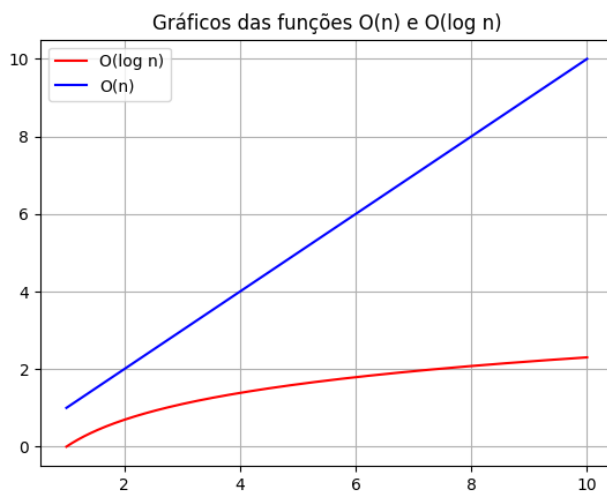
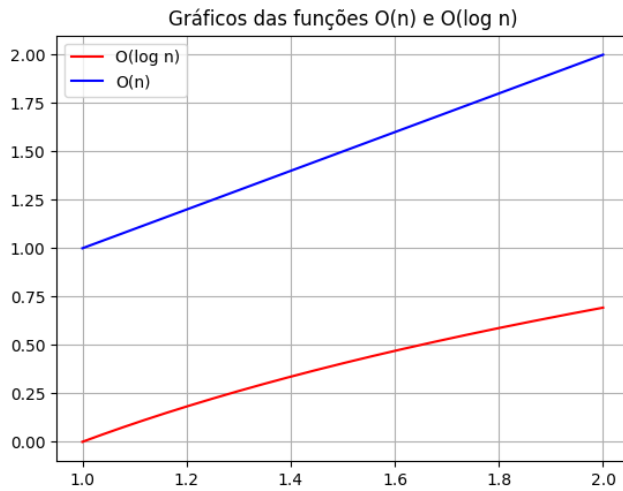
Figura 6: Teste do livro Dom Casmurro com o dicionário Português e árvore AVL

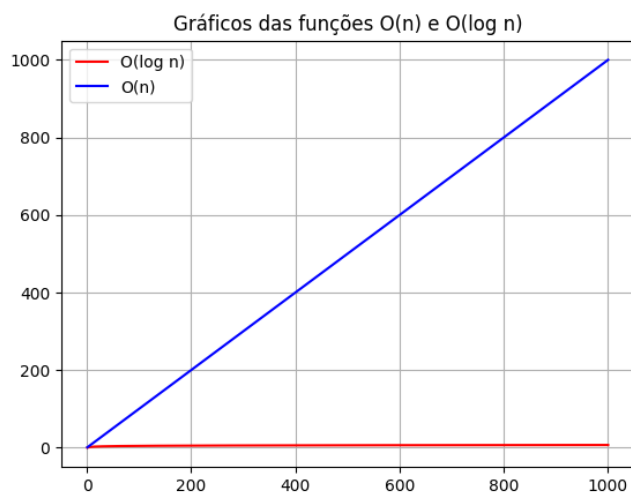
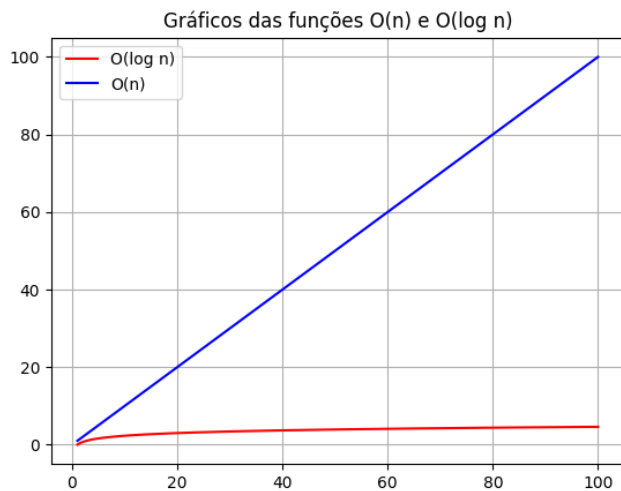
```
osboxes@osboxes: ~/Desktop/EP3/EP3
File Edit View Search Terminal Help
confirmed : confirme
included. :
thus, :
necessarily :
paper : papar pape papem papes
edition :
best :
start :
main : maia
PG :
search :
facility: :
includes :
gutenberg-tm, :
produce : produze
subscribe :
email :
newsletter :
hear : cear gear tear
ebooks. :
-----
Tempo Total: 1925.2864710000 seg
osboxes@osboxes:~/Desktop/EP3/EP3$
```

Para uma árvore não balanceada o tempo de execução não seria prático, portanto, não foi realizado o teste.

Como pudemos perceber, o tempo de execução com uma árvore balanceada é muito menor do que uma não balanceada. Isso ocorre por que os dados na árvore balanceadas estão organizados de forma que a complexidade de suas operações de busca, inserção e remoção dos nós sejam  $O(\log n)$ , enquanto uma árvore não balanceada possui complexidade  $O(n)$ .

Nos gráficos abaixo, que foram gerados utilizando a biblioteca *Matplotlib* do Python, vemos a diferença entre as complexidades:





Assim como esperado, é mostrado no gráfico que quanto maiores os valores de  $n$ , mais a complexidade  $O(\log n)$  se torna mais otimizada que a complexidade  $O(n)$ . Aproximando os dados para a realidade do trabalho, quanto maior o acervo de palavras presente na nossa árvore que armazena o dicionário, mais a árvore binária balanceada se torna mais útil para a comparação.

## Parte IV

# Conclusões

---

Devido este exercício programa, pudemos perceber as principais diferenças entre a utilização de árvores binárias de buscas balanceadas e não balanceadas, e o porque da escolha de cada uma delas para diferentes situações, devido suas complexidades, além de exercitarmos a criação, manipulação e organização de diferentes arquivos *.h* e *.c* e abordarmos, principalmente, o uso de diversos tipos abstratos de dados(ou TAD's).

**GitHub do Autor:** [github.com/ruliandosreis](https://github.com/ruliandosreis)