UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

CARLOS NEGRI (00333174)
LEONARDO KAUER LEFFA (00333399)

Comparação de Eficiência entre as Árvores ABP e AVL: Estrutura de Dados

Trabalho de Final de Semestre para a disciplina de Estrutura de Dados – INF 01203 no semestre de 2021/2

Prof. Viviane Moreira
Turma A

Porto Alegre, maio de 2022

Índice:

Apresentação

O objetivo do trabalho é utilizar as estruturas de dados aprendidas durante todo o semestre da disciplina em uma situação simulada semelhante a uma situação real.

Para esse trabalho, foi apresentado o seguinte problema: criar um programa que atue como um gerador de paráfrases, utilizando um dicionário, para alterar determinadas palavras em um arquivo por outras com o significado equivalente.

Além disso, foi solicitada a utilização de dois tipos de árvores, a Árvore de Binária de Pesquisa (ABP) e outra de nossa escolha, nesse caso a Árvore de Velsky e Landis (AVL). Essas árvores serão utilizadas no armazenamento do dicionário de sinônimos.

O programa tem como entrada dois arquivos, arquivo de entrada com o texto à ser analisado e o arquivo contendo o dicionário de sinônimos, a sua saída é composta do mesmo arquivo de entrada porém com as palavras modificadas. Na chamada do programa, é pedido o nome do arquivo de dicionário, o nome do arquivo de entrada e o nome do arquivo de saída à ser criado.

Desenvolvimento

O programa foi desenvolvido, aplicado, testado e depurado por ambos os integrantes do grupo, sendo as funções divididas de maneira igual entre os mesmos. Visando a facilitação do acesso foi criado uma pasta compartilhada no Google Drive, onde é armazenada o programa em todas as suas fases de desenvolvimento.

O desenvolvimento do trabalho foi dividido em algumas etapas, as quais estão listadas abaixo:

1. Os dados de entrada foram divididos em grupos para facilitar seu manuseio pelo programa, sua depuração e seu desenvolvimento.
2. Esses grupos foram estruturados em uma árvore (ABP e AVL) utilizando-se de TADs desenvolvidos durante o semestre, essas árvores irão auxiliar na busca rápida e eficiente das palavras.
3. Os TADs, propriamente modificados e funcionais, foram separados em arquivos levando em conta sua função, também foram criadas funções auxiliares com o objetivo de auxiliar os TADs na execução de sua função.
4. Implementação do código.
5. Depuração do código.

Visão Geral do Programa

O programa é dividido em 4 partes principais, todas em seus próprios arquivos: *main.c, avl.c, abp.c, funcoes.c*. Em cada um desses arquivos estão as funções responsáveis pelo funcionamento do programa.

O programa passa por 4 momentos principais durante sua execução:

1. Abertura de todos os arquivos que serão utilizados junto com a iniciação do cronômetro.
2. Leitura do dicionário de sinônimos junto com sua colocação adequada na respectiva árvore (ABP ou AVL).
3. Leitura do arquivo de texto, checagem de palavras em tempo real com a árvore de sinônimos e escrita no arquivo de saída.
4. Fechamento de todos os arquivos abertos e impressão das informações relevantes na tela.

Estruturas de Dados e TAD's

Foram criadas 2 Estruturas de dados para o programa, uma utilizada para a árvore ABP e outra para a árvore AVL.

Estrutura NO (ABP)

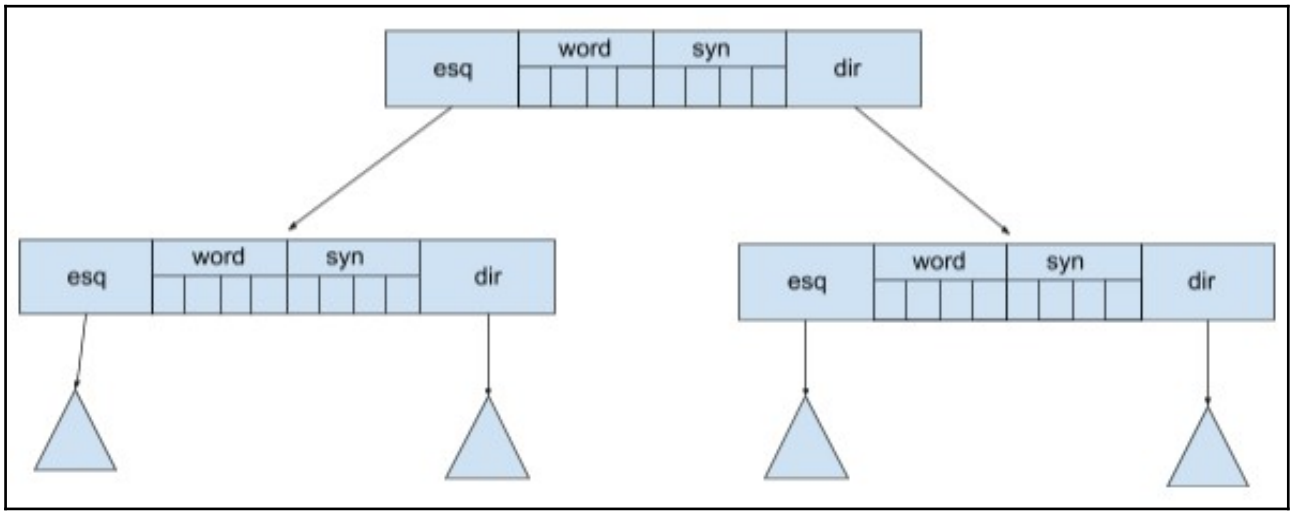


Diagrama 1 – Esquema de dados da ABP

Na estrutura NO da árvore ABP encontram-se 4 elementos:

1. char word[MAX]; Responsável pela string da palavra que está armazenada nesse nó.
2. char syn[MAX]; Responsável pela string do sinônimo que está armazenado nesse nó.
3. struct NO *esq; Aponta para o nó abaixo do nó atual no lado esquerdo.
4. struct NO *dir; Aponta para o nó abaixo do nó atual no lado direito.

Com essa estrutura é possível criar uma árvore com os nodos encadeados entre si, permitindo a realização de operações diversas com eles.

Os nodos são organizados de acordo com a ordem alfabética das palavras e são adicionados na ordem em que aparecem no documento de dicionário.

Estrutura NO (AVL)

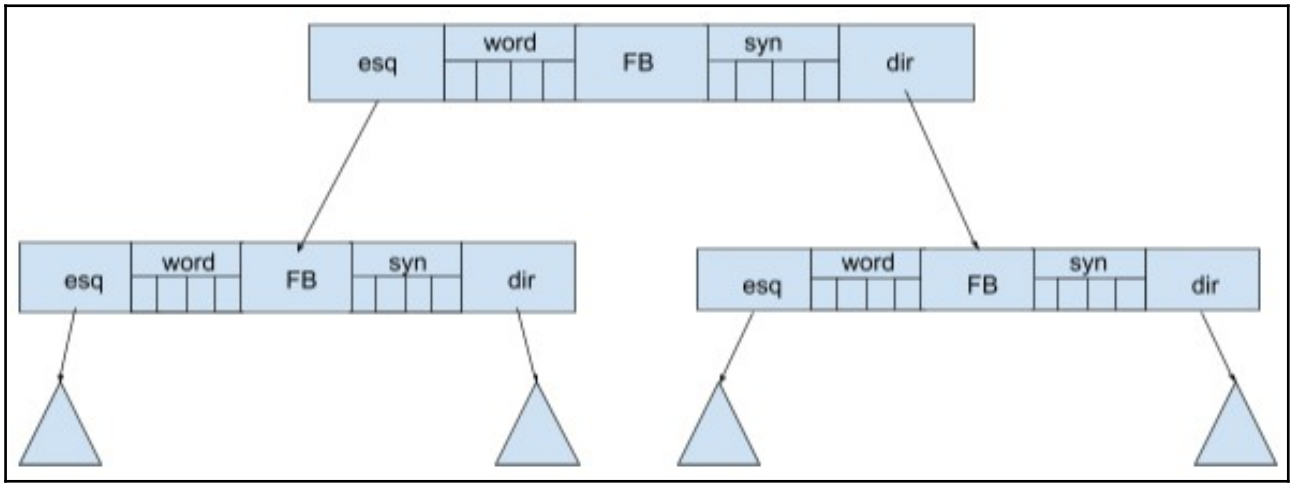


Diagrama 2 – Esquema de dados da AVL

Na estrutura NO da árvore AVL encontram-se 5 elementos:

1. char word[MAX]; Responsável pela string da palavra que está armazenada nesse nó.
2. char syn[MAX]; Responsável pela string do sinônimo que está armazenado nesse nó.
3. int *FB; Armazena fator de balanceamento do nó.
4. struct NO *esq; Aponta para o nó abaixo do nó atual no lado esquerdo.
5. struct NO *dir; Aponta pata o nó abaixo do nó atual no lado direito.

Com essa estrutura é possível criar uma árvore com os nodos encadeados entre si, permitindo a realização de operações diversas com eles.

Os nodos são organizados de acordo com a ordem alfabética das palavras e são adicionados na ordem em que aparecem no documento de dicionário.

Funcionamento Geral do Programa e Suas Funções

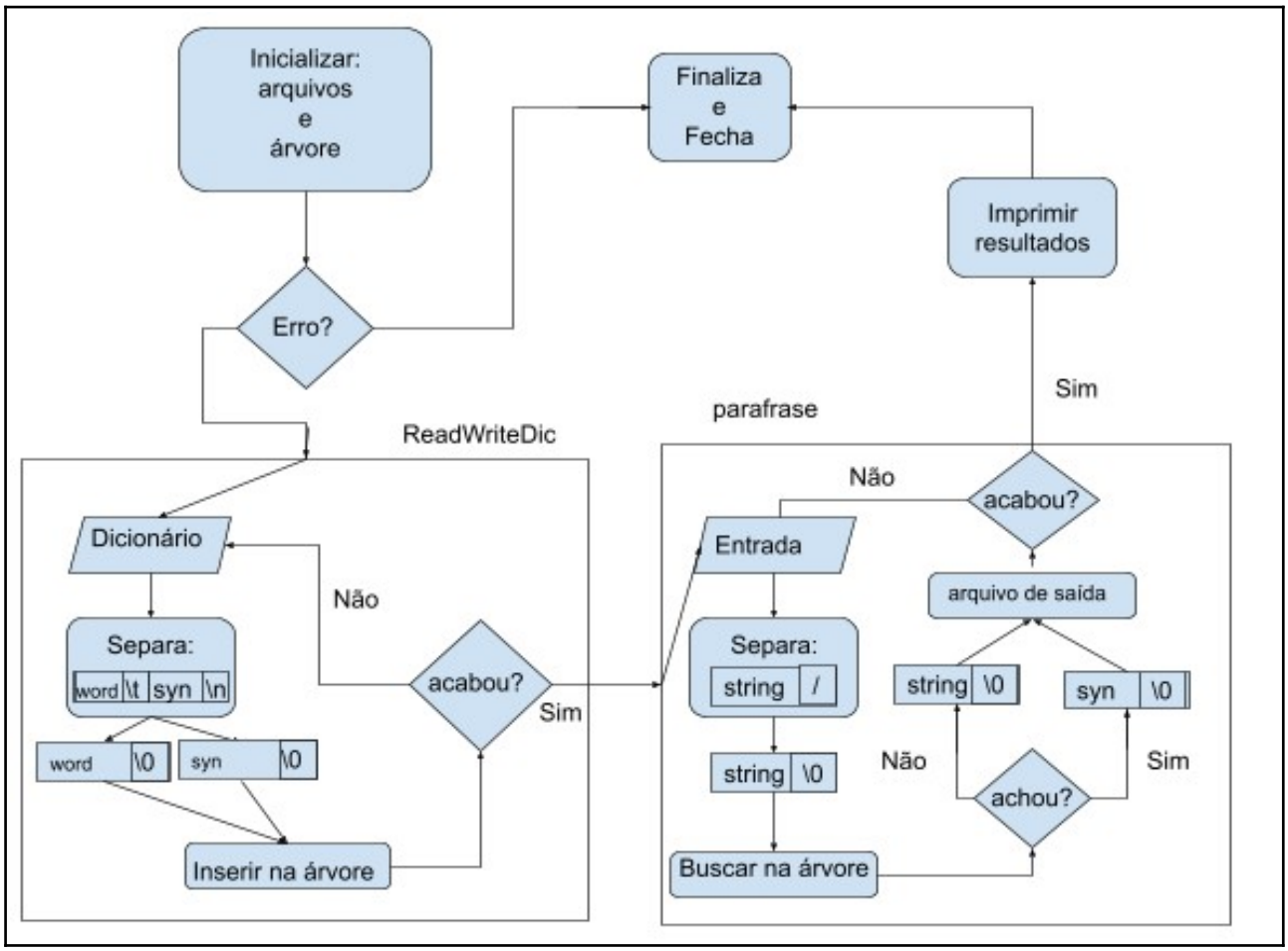


Diagrama 3 – Funcionamento geral do programa

Cada uma das estruturas apresentadas possui suas próprias funções, que podem ser divididas entre funções básicas de manipulação de cada estrutura e funções específicas para a proposta da aplicação. Elas foram separadas em arquivos C distintos e são chamadas pela função main, também em um arquivo distinto, quando necessário.

Além das funções relacionadas as estruturas, há as funções que manipulam arquivos, visando a leitura e escrita de palavras.

funcoes.c

- *ReadWriteDic*: Lê o arquivo de dicionário e adiciona seu conteúdo à árvore.
- *parafrase*: Lê a linha do texto de entrada, pega cada palavra separada e procura se existe na árvore, se existir, troca pelo sinônimo, senão mantém igual.

abp.c

- *inicializa*: Inicia uma árvore binária de pesquisa com todos os nodos nulos.
- *InsereArvore*: Insere um novo nodo na árvore binária de pesquisa.
- *busca*: Busca um nodo na árvore utilizando sua palavra (word[]) como parâmetro de pesquisa e retorna o nodo em que se encontra a word se achado, senão retorna NULL.
- *destruir*: Destrói a árvore, liberando o seu espaço da memória.
- *contaNodos:* Conta a quantidade de nodos da árvore
- *Altura:* Calcula a altura da árvore

avl.c

- *inicializa*: Inicia uma AVL com todos os nodos nulos.
- *rotacao_direita*: Realiza o movimento de rotação direita na árvore.
- *rotacao_esquerda*: Realiza o movimento de rotação esquerda na árvore.
- *rotacao_dupla_direita*: Realiza o movimento de rotação dupla direita na árvore.
- *rotacao_dupla_esquerda*: Realiza o movimento de rotação dupla esquerda na árvore.
- *InsereArvore*: Insere um novo nodo na árvore.
- *Caso1*: Analisa se é preciso fazer uma rotação direita após a inserção de um novo nodo.
- *Caso2*: Analisa se é preciso fazer uma rotação esquerda após a inserção de um novo nodo.
- *Altura*: Calcula a altura de um nodo.
- *fator*: Calcula o fator de balanceamento de um nodo.
- *destruir*: Destrói a árvore, liberando o seu espaço da memória.
- *busca:* Busca um nodo na árvore utilizando sua palavra (word[]) como parâmetro de pesquisa e retorna o nodo em que se encontra a word se achado, senão retorna NULL.
- *contaNodos:* Conta a quantidade de nodos na árvore

Arquivos de entrada e saída

São recebidos como parâmetros da função *main* dois arquivos de entrada e um arquivo de saída.

O primeiro arquivo de entrada é um dicionário de sinônimos em formado texto. Ele contém palavras separadas entre si por linhas e seus respectivos sinônimos separados por <TAB> da palavra à qual eles se referem.

O segundo arquivo de entrada é o texto ao qual será feita a análise em busca de sinônimos. A busca é feita ignorando-se a diferença entre letras maiúsculas e minúsculas.

O arquivo de saída, por sua vez, contém o texto do arquivo de entrada, porém com todas as palavras encontradas no dicionário e trocadas pro seus sinônimos.

No final da execução do programa, são exibidos na tela as informações pertinentes sobre a eficiência do programa.

Testes executados

Foram executados testes com cada forma de implementação (ABP e AVL) utilizando um dicionário normal (não ordenado).

Tempo

texto de entrada	tempo (ms)
Cap. 1 - Alienista (ABP-N)	31
Cap. 1 - Alienista (AVL-N)	31

texto de entrada	tempo (ms)
Dicionário (ABP-N)	93
Dicionário (AVL-N)	140

texto de entrada	tempo (ms)
Dicionário Ord. (ABP-N)	88
Dicionário Ord. (AVL-N)	143

Comparações

texto de entrada	Comparações
Cap. 1 - Alienista (ABP-N)	361429
Cap. 1 - Alienista (AVL-N)	292573

texto de entrada	Comparações
Dicionário (ABP-N)	984728
Dicionário (AVL-N)	789388

texto de entrada	Comparações
Dicionário Ord. (ABP-N)	984728
Dicionário Ord. (AVL-N)	789388

Comparações Inserção

texto de entrada	Comparações Inserção
Cap. 1 - Alienista (ABP-N)	316652
Cap. 1 - Alienista (AVL-N)	256898

texto de entrada	Comparações Inserção
Dicionário (ABP-N)	316652
Dicionário (AVL-N)	256898

texto de entrada	Comparações Inserção
Dicionário Ord. (ABP-N)	316652
Dicionário Ord. (AVL-N)	256898

Comparações Busca

texto de entrada	Comparações Busca
Cap. 1 - Alienista (ABP-N)	44777
Cap. 1 - Alienista (AVL-N)	35675

texto de entrada	Comparações Busca
Dicionário (ABP-N)	668076
Dicionário (AVL-N)	532490

texto de entrada	Comparações Busca
Dicionário Ord. (ABP-N)	668076
Dicionário Ord. (AVL-N)	532490

Número de Nodos

texto de entrada	Número de nodos
Cap. 1 - Alienista (ABP-N)	10000
Cap. 1 - Alienista (AVL-N)	10000

texto de entrada	Número de nodos
Dicionário (ABP-N)	10000
Dicionário (AVL-N)	10000

texto de entrada	Número de nodos
Dicionário Ord. (ABP-N)	10000
Dicionário Ord. (AVL-N)	10000

Altura

texto de entrada	Altura
Cap. 1 - Alienista (ABP-N)	30
Cap. 1 - Alienista (AVL-N)	16

texto de entrada	Altura
Dicionário (ABP-N)	30
Dicionário (AVL-N)	16

texto de entrada	Altura
Dicionário Ord. (ABP-N)	30
Dicionário Ord. (AVL-N)	16

Rotações

texto de entrada	Rotações
Cap. 1 - Alienista (ABP-N)	0
Cap. 1 - Alienista (AVL-N)	6979

texto de entrada	Rotações
Dicionário (ABP-N)	0
Dicionário (AVL-N)	6979

texto de entrada	Rotações
Dicionário Ord. (ABP-N)	0
Dicionário Ord. (AVL-N)	6979

Palavras Trocadas

texto de entrada	Palavras trocadas
Cap. 1 - Alienista (ABP-N)	120
Cap. 1 - Alienista (AVL-N)	120

texto de entrada	Palavras trocadas
Dicionário (ABP-N)	10000
Dicionário (AVL-N)	10000

texto de entrada	Palavras trocadas
Dicionário Ord. (ABP-N)	10000
Dicionário Ord. (AVL-N)	10000

Análise gráfica de dados

Foram criados alguns arquivos de texto para a realização de alguns testes para a análise deles em formato gráfico, os resultados estão mostrados abaixo:

Árvore ABP

ABP		
palavras trocadas	**comparações busca**	**tempo (ms)**
84	45932	17
152	81506	20
212	110548	23
272	139656	27
562	264579	37
706	322940	44
786	360240	48
832	378781	51
898	402922	50
1010	450116	57
1038	461968	59
1180	521977	63
1329	578299	71

Tabela 1 – Teste com diferentes entradas na árvore ABP

Utilizando a tabela acima foram criados os dois gráficos mostrados abaixo:

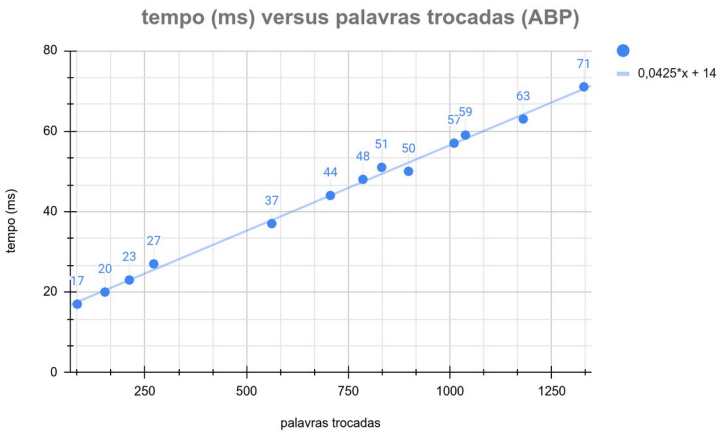


Gráfico 1 – Tempo X Palavras Trocadas (ABP)

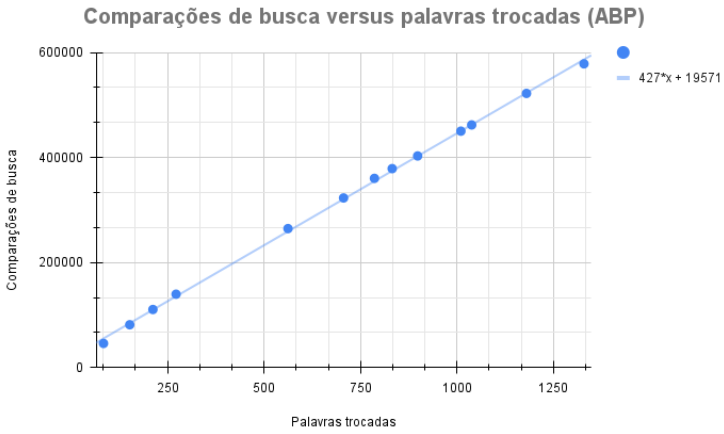


Gráfico 2 – Comparações de Busca X Palavras Trocadas (ABP)

No primeiro gráfico, o “Tempo” representa o tempo necessário para a leitura do arquivo de dicionário, sua inserção na árvore, sua busca de sinônimos e posterior escrita no arquivo de saída.

Já no segundo, as “Comparações” representam o número de comparações realizadas para a operação de busca na árvore.

Em ambos os gráficos, é possível observar que a taxa de crescimento é linear tanto para o tempo de execução quanto para o número de comparações.

Árvore AVL

AVL		
palavras trocadas	**comparações busca**	**tempo (ms)**
84	36314	29
152	64506	35
212	87448	39
272	110448	41
562	208353	46
706	254504	49
786	284244	55
832	299047	59
898	318262	61
1010	355356	65
1038	364724	66
1180	412445	69
1329	457155	72

Tabela 2 – Teste com diferentes entradas na árvore AVL

Utilizando a tabela acima foram construídos os seguintes gráficos:

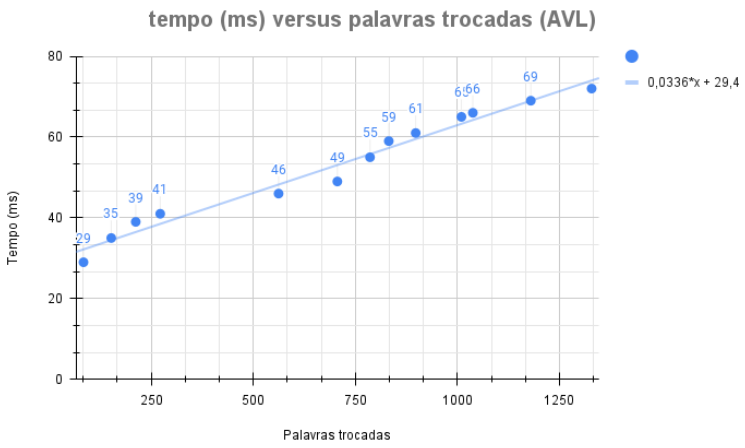


Gráfico 3 - Tempo X Palavras Trocadas (AVL)

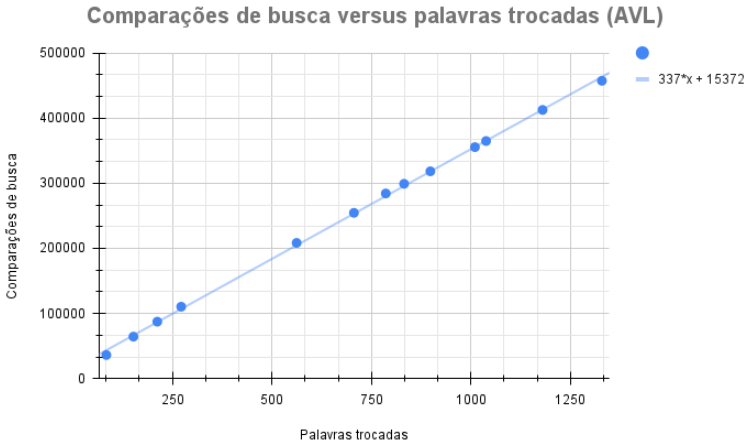


Gráfico 4 – Comparações de Busca X Palavras Trocadas (AVL)

No primeiro gráfico, o “Tempo” representa o tempo necessário para a leitura do arquivo de dicionário, sua inserção na árvore, sua busca de sinônimos e posterior escrita no arquivo de saída.

Já no segundo, as “Comparações” representam o numero de comparações realizadas para a operação de busca na árvore.

Em ambos os gráficos, é possível observar que a taxa de crescimento é linear tanto para o tempo de execução quanto para o número de comparações.

Análise dos Algoritmos

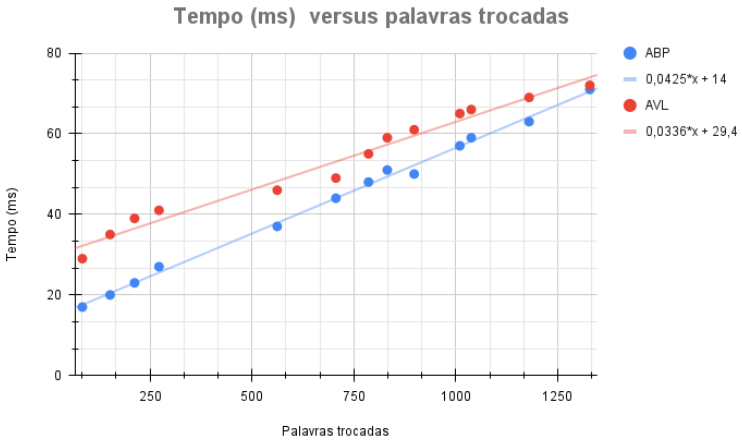


Gráfico 5 – Comparação entre ABP e AVL

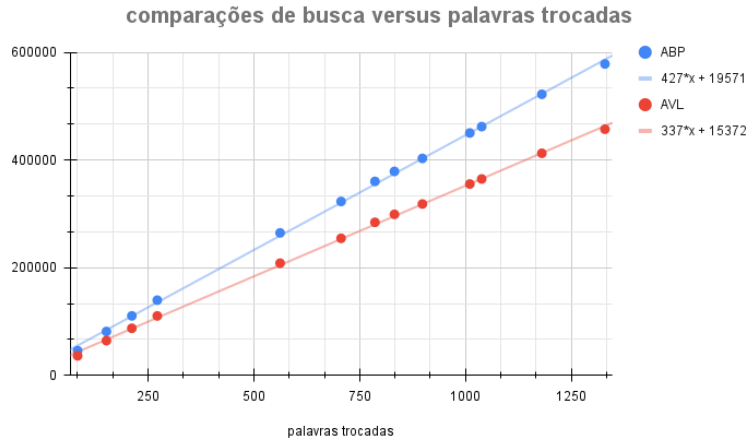


Gráfico 6 – Comparação entre ABP e AVL

Após diversos testes utilizando os dois algoritmos (ABP e AVL) foi possível chegar a algumas conclusões, as quais serão desenvolvidas abaixo.

Analisando e comparando os gráficos acima, a árvore AVL é mais rápida para buscas e mais lenta para inserção do que a árvore ABP, já que as árvores AVL são mais rigidamente balanceadas do que as árvores ABP, o que permite uma operação de busca mais rápida, mas também compromete o desempenho das operações de modificação. Portanto, se a aplicação da árvore realiza, de forma mais intensa, operações de busca, é mais apropriado o uso de uma árvore AVL.

Além disso, é evitável o uso de árvores ABP quando é possível e/ou comum que a árvore esteja desbalanceada, pois demoraria quase o mesmo tempo do que uma busca em um vetor, o que não é preferível, já que é usado uma árvore para obter buscas mais rápidas.

Embora não demonstrado nos testes acima, foi realizado um teste utilizando um dicionário ordenado como dicionário de entrada, porém, após mais de uma hora de teste, o programa não exibiu sinais de finalização. Isso acontece porque a árvore não tem como separar as palavras em

ordem alfabética e colocá-las em seus lados corretos na árvore, resultando em uma árvore com o valor de altura igual o valor de nodos.

Após analisar as tabelas da seção 7 – Testes Executados, foi constatado que a AVL tem uma altura relativamente menor em comparação com a ABP, o que acaba resultando em menos comparações entre nodos, isso se dá graças aos algorítimos de rotação presentes nas árvores AVLs e ausentes nas árvores ABP.

Analisando o gráfico 5, é notável que, a partir de algum número de nodos, a árvore AVL se tornará mais vantajosa, nesse caso, mesmo fazendo o uso de rotações, pois a taxa de crescimento do número de comparações da busca em função da quantidade de palavras trocadas é maior na árvore ABP do que a árvore AVL.

Usando-se as equações das linhas de tendência do gráfico 5, temos:

$$0{,}0425x+14=0{,}0336x+29{,}4$$

$$0{,}0089x=15{,}4$$

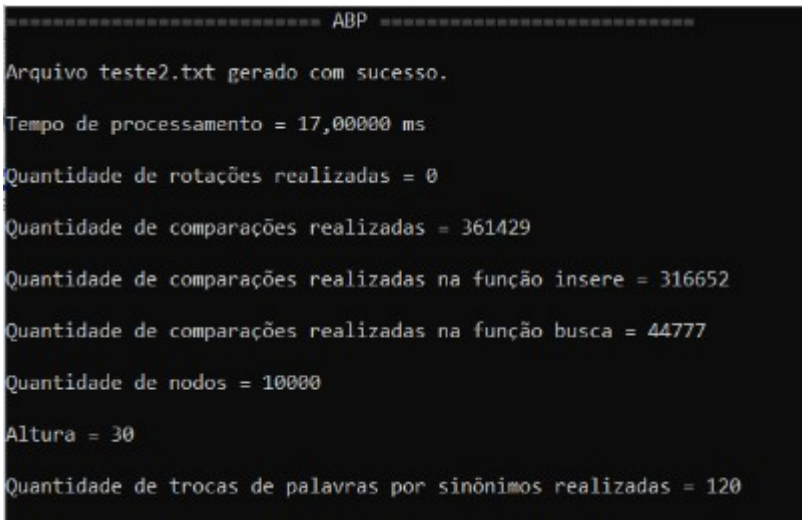
$$x=1730{,}33$$

Nesse caso, a partir de aproximadamente 1730 trocas de palavras a avl se torna mais vantajosa na busca para um mesmo numero de inserções.

Analisando o gráfico 6 nesse contexto, o número de comparações da ABP é sempre maior ou igual do que o número de comparações da AVL.

Porém, para árvores mais degeneradas do que essa, a AVL se torna a preferível, apesar do números de rotações ser maior, a busca será muito mais eficiente, já que o consumo de tempo de cada uma das funções busca e insere é proporcional à altura da árvore binária de busca, no pior caso, e as funções que descrevemos não garantem altura logarítmica, enquanto a árvore AVL garante.

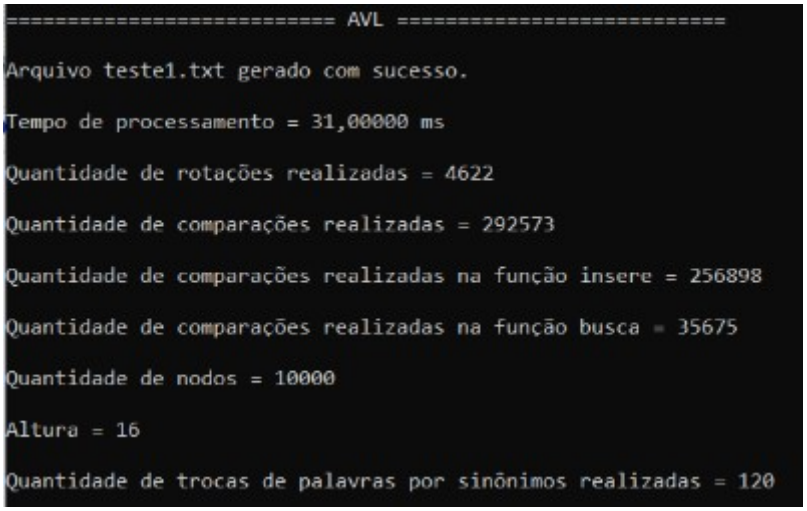
Imagens da Execução do Programa



Árvore: ABP

Entrada: Alienista Cap 1

Dicionário: Normal



Árvore: AVL

Entrada: Alienista Cap 1

Dicionário: Normal

Referências

1. Moodle da Disciplina. Disponível em <https://moodle.inf.ufrgs.br/course/view.php?id=572>. Acesso em Maio de 2022.
2. FRANCESQUINI, Emilio. MC202 - Estrutura de Dados. Disponível em <https://www.ic.unicamp.br/~francesquini/mc202/files/aula12.pdf> Acesso em Maio de 2022.