Universidade Federal do Piauí
Campus Senador Helvídio Nunes de Barros
Sistemas de Informação
Estruturas de Dados II
2021.1

Trabalho Prático I

Nome: Matheus Victor Ferreira da Silva

Resumo de Projeto

Este trabalho consiste em desenvolver quatro programas escritos em linguagem C que atendam às exigências de cada questão. A partir das estruturas de dados, árvores binárias e árvores AVL, esses programas devem fornecer funcionalidades que solucionem os problemas propostos e apresentem dados como o tempo de execução para que seja possível realizar alguns experimentos. A primeira e a segunda questão, solicita um programa que organize 1000 números gerados aleatoriamente em árvores binárias e árvores AVL. Já as questões 3 e 4, se referem a programas que armazenam unidades de vocabulários de palavras portuguesas junto com suas equivalentes inglesas.

Introdução

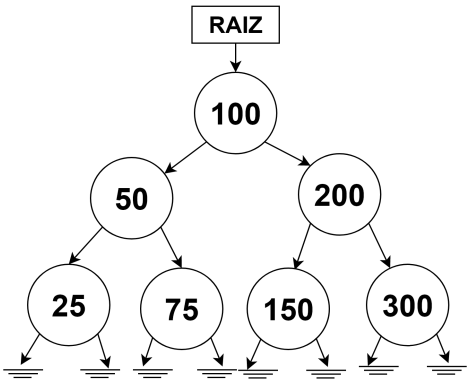
Árvores Binárias são estruturas de dados que organizam os Nós hierarquicamente. Sua principal característica é a limitação da quantidade de filhos que seus Nós podem possuir, que é no máximo dois. Embora as árvores binárias sejam bastante complexas em sua implementação, elas garantem uma poderosa busca binária de seus dados.

Os Nós são unidades básicas individuais em estruturas de dados. Em uma lista simples encadeada, a estrutura deve armazenar o Nó inicial da lista e esse Nó deve armazenar o endereço do próximo item da lista, que armazena o endereço do próximo, que armazena o do próximo, até o último item que armazena o valor *NULL* para indicar que não há mais próximos. Essa mesma idéia se aplica nas árvores binárias, que possuem um Nó inicial que chamamos de raiz, esse Nó pode possuir nenhum filho o que o torna um Nó folha, ou pode possuir um ou dois Nós filhos que podem ter filhos ou não.

A árvore binária possibilita uma rápida busca por seus dados, já que toda a busca se baseia na pesquisa binária que se apresenta muito mais eficiente que uma simples busca sequencial. Algumas outras informações sobre as árvores e seus Nós são de extrema importância, como a altura da árvore e de um Nó, a profundidade, a quantidade de Nós e ramos, e por fim o tamanho de um Nó.

A altura de um Nó significa o comprimento do caminho do Nó até a folha de maior profundidade. A altura da árvore é o total da altura da raiz. A altura de uma folha é 0, pois ele não possui filhos. A altura de um Nó é o total da soma da altura do seu filho de maior altura com + 1. Já a profundidade de um Nó se refere à distância do Nó até a raiz da árvore, a profundidade da raiz é 0 e profundidade de um Nó é 1 + a profundidade do seu pai.

A quantidade de Nós de árvore se refere ao número total de Nós inseridos em toda a estrutura da árvore, já o total de ramos significa o total de Nós que se ramificaram a partir da raiz, logo o total de ramos é a quantidade de todos os Nós menos 1. E por último, o tamanho de um Nó é o total de descendentes que esse Nó possui.



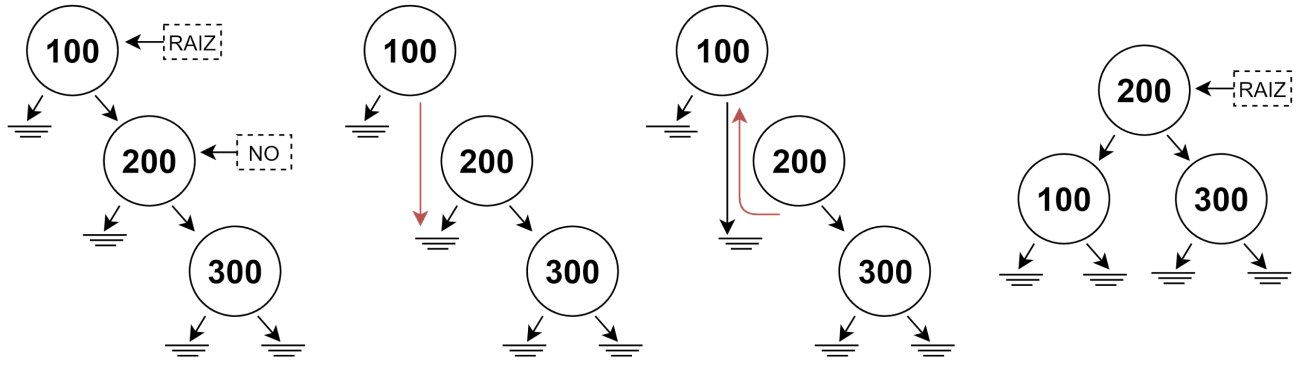
Embora as árvores binárias apresentam grande vantagem em relação às listas encadeadas, elas ainda possuem as suas desvantagens. Em alguns casos, essas árvores podem conter galhos muito profundos ao mesmo tempo que possuem galhos não tão profundos. Essa grande profundidade de um galho pode causar uma demora no momento das buscas, em alguns casos a busca nessas estruturas se tornam demoradas como uma busca sequencial.

Uma solução para este problema é o uso das Árvores AVL, que são basicamente Árvores Binárias que possuem algumas funções a mais que efetuam o balanceamento dos galhos dessas árvores, tornando todos os galhos do mesmo nível ou o mais próximo possível. As Árvores AVL apresentam uma grande vantagem em suas buscas pois elas já eliminam a possibilidade de haver esses galhos muito profundos.

As árvores AVL foram inventadas por (e posteriormente nomeadas em homenagem a) Georgy Adelson-Velsky e Evgenii Landis, por dois inventores soviéticos. Essas estruturas são criações bastante recentes; Adelson-Velsky e Landis apresentaram pela primeira vez a ideia por trás deles em 1962, em um artigo que a dupla co-autora e publicou intitulado “Um algoritmo para a organização da informação”.

Balanceamento é a operação de nivelar os galhos de uma árvore. Há um cálculo bem simples que é feito para saber se uma raiz está ou não desbalanceada, é subtraído a altura do Nó à esquerda do Nó raiz pela a altura do Nó à direita do Nó raiz, caso a diferença resulte (-1), 0 ou 1, significa que árvore que possui tal raiz está balanceada, caso contrário árvore estará desbalanceada e precisará passar pelo o processo de balanceamento.

Na imagem a seguir é demonstrado um exemplo de balanceamento em uma árvore binária de números inteiros. A sequência dos números inseridos nesta árvore foi: 100, 200 e 300. A folha desta árvore é o Nó que armazena o valor 300 e que possui a altura 0. O segundo Nó possui um único filho e armazena o valor 200 e possui a altura 1, o fator de balanceamento do Nó 200 é (-1). O primeiro Nó que corresponde pelo o 100 é a raiz e possui a altura 2, já o fator de balanceamento é total da altura do filho da esquerda que é (-1) pois é um valor *NULL* e a altura do filho da direita é 1, ou seja fator de balanceamento do Nó 100 é = (-1) - 1 = (-2), logo está esta simples árvore está desbalanceada, o galho mais profundo é o da direita e por isso que o giro será no sentido anti-horário.



Neste trabalho é apresentado como essas estruturas foram implantadas nos programas em C e também quais foram os resultados obtidos a partir do uso delas.

Seções Específicas

Questão 1: Árvore Binária para números aleatórios

Na primeira questão, é solicitado que seja desenvolvido um programa que gere e armazene 1000 números aleatórios, cada número corresponde a um Nó na árvore binária, é necessário gerar esses números, e realizar algumas operações: A primeira é imprimir o nível da folha de maior profundidade e o nível da folha de menor profundidade, e por fim removê-los para que sejam gerados outros 1000 números aleatoriamente, que em seguida se repetirá a remoção. Todo este processo deve ser repetido 30 vezes e deve ser registrado o tempo que levará para que esses processos sejam feitos, a fim de realizar comparações.

Após a geração dos números, o programa deve imprimir a quantidade de vezes que foi gerada uma árvore com os número aleatórios onde o nível diferença entre a profundidade máxima e mínima foram de 0, 1, 2 , 3 e assim por diante. Para obter informações sobre o tempo de execução foram utilizadas algumas funções da biblioteca padrão da linguagem C, "*time.h"*.

Na função main é chamada a função *start*(...) que é responsável por chamar as funções necessárias. Esta função possui um laço de repetição programado para realizar 30 iterações, que é o número total de árvores que serão criadas. Em cada iteração segue-se a ordem: primeiro uma árvore é alocada, em seguida é chamada a função que insere os valores na árvore recém alocada, depois é chamada a função que calcula a diferença de profundidade dos galhos, prossegue para a função que busca o valores na árvore e para finalizar, o espaço de memória reservada para a árvore é liberada. Todo esse processo é repetido.

A função que insere os valores na árvore foi nomeada de *etapa1*(...), nela contém um laço de repetição programado para ter 1000 iterações, total de números inteiros que serão inseridos na árvore. Dentro deste laço de repetição contém uma condição, que possibilita que os números aleatórios gerados sejam distantes. A primeira condição confere se a iteração é de número par, caso seja verdadeiro o número gerado estará entre o intervalo de 0 até 20000, caso contrário, o número gerado estará entre o intervalo do número 1000 a 60000. Assim que o número aleatório é gerado, ele é passado para a função *gestaoINSERIR*(...) que cuidará de inserir o número em seu devido lugar na árvore.

Já a função que ficou encarregada de calcular a diferença de profundidade dos galhos foi nomeada de *etapa2*(...), para auxiliar essa função foram criadas duas funções, *maiorPROFUNDIDADE*(...) e *menorPROFUNDIDADE*(...), essas funções retornam um valor inteiro que serão usadas no cálculo da diferença. A diferença é salva na variável onde ***d* aponta que mais adiante será contabilizada.

A última função criada tem como objetivo buscar os valores na árvore, nomeamos de *etapa3*(...), ela já foi programada para buscar automaticamente os valores salvo em um vetor de número inteiros, os valores são: 100, 500, 1000, 5000, 10000. Há um laço de repetição que tem 5 iterações, cada iteração se refere a busca de cada valor do vetor. Em cada iteração também é marcado o tempo de busca e assim que as iterações se encerram é marcado o tempo que todas as buscas levaram.

As funções principais estão presentes no arquivo "*arvoreBinaria.h*", e as funções auxiliares estão presentes no arquivo "*funcoes.h*", entre essas funções auxiliares estão a de gerar os números aleatórios conforme o intervalo definido a que calcula o menor valor e o que calcula o maior valor presente na árvore, e funções que contabilizam alguns eventos.

Questão 2: Árvore AVL para números aleatórios

A segunda questão pede exatamente o mesmo programa da questão 1, só que agora com o uso da estrutura da árvore AVL. Deve ser comparados os tempos que cada árvore teve para inserir e

buscar os dados. Para as funções de balanceamento foi criado o arquivo "*balanceamento.h"* onde essas funções são chamadas assim que algum novo Nó é inserido na árvore.

Questão 3: Vocabulário Inglês com Árvore Binária

Na terceira questão do trabalho é solicitado um programa que leia um arquivo *txt* que contenha unidades com vocabulários de palavras inglesas com suas palavras portuguesas equivalentes e salve todas as unidades deste arquivo em uma árvore binária. A estrutura deve possuir funções que realizam a inserção das unidades, que façam buscas, apresentem os dados e por fim, que façam a remoção das unidades da árvore.

A primeira operação deste programa é a leitura do arquivo, que foi nomeado como "arquivo.txt" e se encontra na mesma pasta que o programa "main.c". O programa deve ler cada linha do arquivo como uma entrada de dados, para isso foram utilizadas a função *fgetc* que lê um um caractere do arquivo por vez, essa função possibilita um controle mais avançado dos caracteres que vão ser salvos e também das regras de parada, evitando algum problema com caracteres especiais ou números.

Todas as funções que lidam diretamente com o arquivo *txt* estão presentes na biblioteca local que nomeamos como *arquivo.h*, nela possui a função *gestaoLER* que segue a seguinte sequência: [1] chama a função que ler a linha do arquivo, [2] depois ela envia para as funções de inserções nas árvores [3] depois verifica se deve repetir o processo.

A estrutura do arquivo segue as seguintes regras: Os nomes das unidades são precedidos por um símbolo de porcentagem, há somente uma entrada por linha e por fim, uma palavra em português é separada por dois pontos de sua(s) equivalente(s) inglesa(s); se há mais de uma equivalente, elas são separadas por vírgula. O programa deve ler o arquivo até encontrar o final do arquivo (EOF).

Assim que uma linha do arquivo é lida, deve ser feita uma pequena verificação da string. O primeiro caractere da string *linha* pode conter o caractere "%" que significa que a linha possui o nome de uma nova unidade que deve ser aberta e salva na árvore de unidades. Caso o primeiro caractere da *linha* seja uma letra significa que uma nova palavra deve ser alocada e salva na unidade que está aberta no momento. Caso o primeiro caractere da *linha* seja "\0" significa que todo o arquivo já foi lido e assim deve ser retornado para a função main.

Para esses programas foram criados seis tipos de dados (*structs*): O primeiro foi a (*arvUNIDADES*) que nada mais é que a árvore de unidades, possui um ponteiro para a unidade raíz e uma variável que armazena a quantidade Nós inseridas nela.

A unidade deve possuir um ponteiro para uma árvore de palavras referentes a unidade, logo a árvore de unidades (*arvUNIDADES*) é uma árvore onde seus Nós também são árvores. A unidade também deve possuir um nome, que é o nome obtido a partir do arquivo *txt*.

O registro de uma palavra possui uma string chamada de (*palavra_pot*) que deve armazenar a palavra portuguesa encontrada no arquivo. A palavra também possui um ponteiro para uma lista simplesmente encadeada chamada de (*palavras_eng*) onde serão armazenadas e organizadas por ordem alfabética as palavras inglesas equivalentes à palavra portuguesa já armazenada em (*palavra_pot*).

Em resumo, a árvore de unidades (*arvUNIDADES*) possui um ponteiro para uma unidade (*UNIDADE*) raíz. Essa unidade possui um ponteiro para uma árvore de palavras (*arvPALAVRAS*). Essa árvore de palavras possui um ponteiro para uma palavra (*PALAVRA*) chamada de raiz. A palavra possui uma string para a palavra portuguesa e uma lista (*LISTA*) para as palavras inglesas. A lista possui um ponteiro para a palavra inglesa (*palavraENG*) inicial e outro para a final. Cada palavra inglesa possui um ponteiro que aponta para a próxima palavra na lista ou o valor (*NULL*).

No programa *main.c* é apresentado um menu onde o usuário pode selecionar a opção de inserção dos dados do arquivo *txt* nas árvores, as funções de apresentar os dados nas ordens: pré-ordem, in-ordem e pós-ordem. Também imprimir todas as palavras portuguesas de uma determinada unidade. A função de imprimir todas as palavras inglesas de uma palavra buscada e por fim, a função de remover uma palavra de uma unidade.

Questão 4: Vocabulário Inglês com Árvore AVL

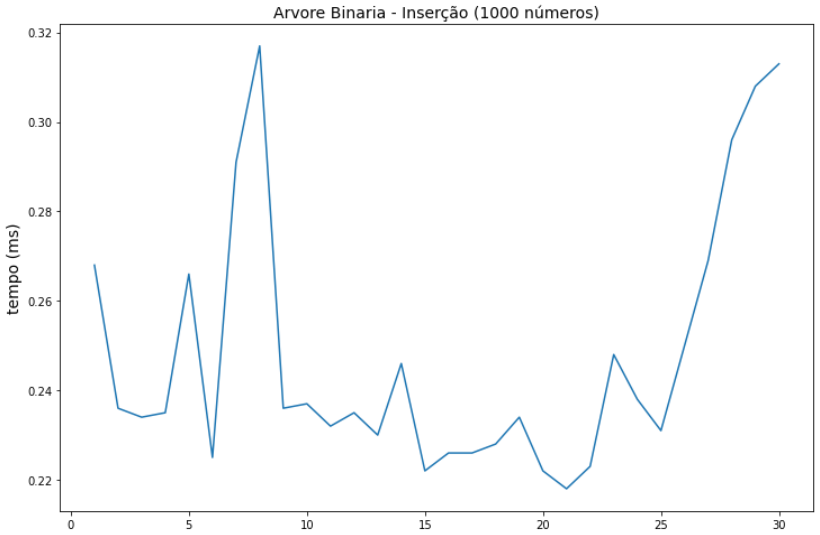
Na quarta questão é solicitado o mesmo programa que o da questão 3, agora com a gestão de uma árvore AVL. Como já citado, as árvores AVL acrescentam somente algumas etapas nas inserções e remoções dos Nós na árvore. As funções de balanceamento foram colocadas no arquivo "*balanceamento.h*" e as chamadas dessas funções, uma se encontra no final da função de inserção e a outra no final da função de remoção.

Resultados da execução do Programa

A seguir estão os resultados obtidos nos testes com os programas.

Questão 1 e Questão 2: INSERÇÃO na Árvore Binária e Árvore AVL

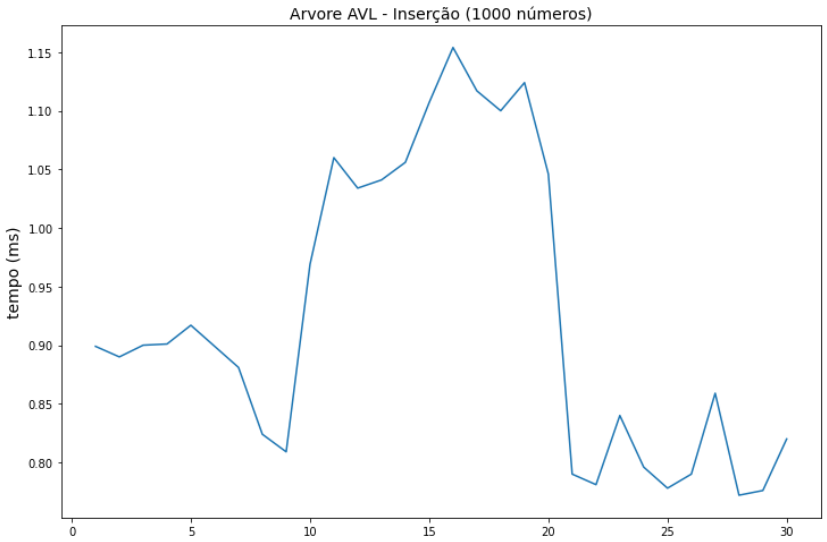
O gráfico a seguir é referente ao resultado da contagem de tempo (em milisegundos) que o programa levou para inserir 1000 valores na árvore binária.



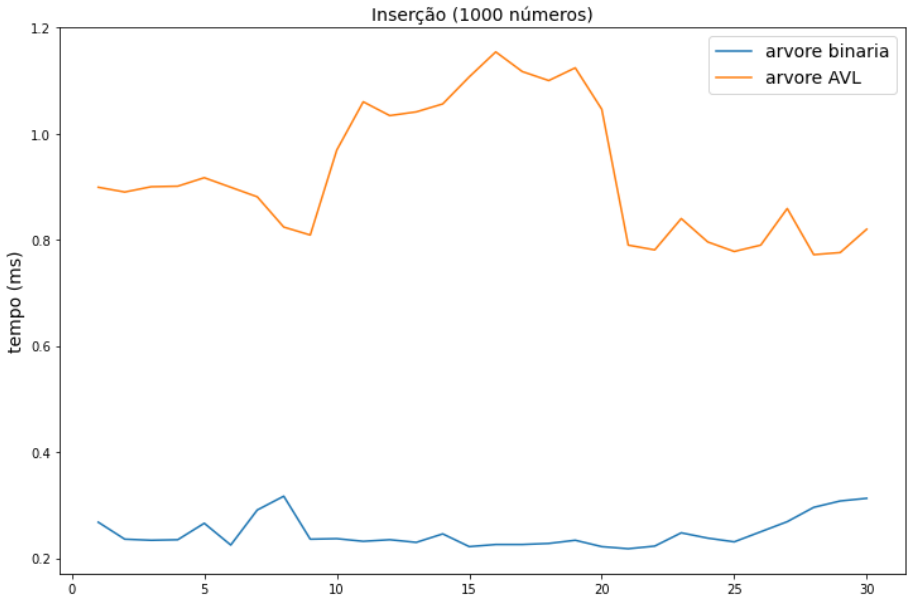
Maior tempo de 0,317ms e o menor tempo levado foi de 0,218ms.

Já o no gráfico é referente ao tempo levado que o programa levou para pesquisar

O gráfico a seguir é referente ao resultado da contagem de tempo (em milisegundos) que o programa levou para inserir 1000 valores na árvore AVL.



O maior tempo levado para inserção 1.154ms e menor tempo foi de 0.772ms. Como pode perceber, o tempo de inserção na árvore AVL é mais elevado que o tempo necessário para inserir os valores na árvore binário. No seguinte gráfico é demonstrado o comparativo entre os resultados entre as duas árvores.



Questão 1 e Questão 2: BUSCA na Árvore Binária e Árvore AVL

Para os testes de busca, estabelecemos que o programa iria realizar as buscas pelos seguintes valores: 100, 500, 1000, 5000 e por fim, 10000. Esses cinco valores foram buscados nas 30 árvore geradas, o que significa que tivemos o total de 5 * 30 = 150 buscas feitas em cada árvore. Focamos mais no tempo em que a estrutura levou para buscar do que o próprio resultado da busca, na tabela a seguir temos os resultados:

Árvore	Menor tempo	Maior Tempo	Média de Tempo	Tempo Total (Soma de todos)
Binária	0.001ms	0.003ms	0.0017933333ms	0.269ms
AVL	0.001ms	0.004ms	0.001673333ms	0.251ms

Embora a árvore AVL tenha tido um pico de 0,004 ms para realizar uma determinada busca, a árvore Binária apresentou uma média de tempo de busca maior. O total de tempo feito para realizar todas as buscas na árvore binária também foi superior ao tempo levado pela a árvore AVL.

Questão 3: Vocabulário Inglês com Árvore Binária

A fim de analisar o tempo de execução do programa utilizamos as funções da biblioteca padrão da linguagem C *time.h*. Para os testes na terceira e quarta questão é necessário analisar o conteúdo presente no arquivo *txt*. Criamos 2 versões do "*arquivo.txt*" que serão detalhadas adiante.

A primeira versão continha 4 unidades, onde cada unidade continha o número fixo de palavras para cada unidade, que eram 3 palavras, e cada palavra continha o total fixo de 3 palavras inglesas. Ou seja, no primeiro caso tínhamos a criação de 4 árvores de 3 palavras cada, onde cada palavra continha uma lista de 3 itens.

Unidades	Qnt. de Palavras	Total em segundos
Profissões	3	0.044000
Animais	3	0.027000
Cores	3	0.038000
Diversos	3	0.042000

O total para toda a execução do processo de leitura de arquivo e inserção das unidades foi de 0.297000 segundos. Deve-se levar em conta que esse total refere-se ao tempo gasto para realizar todo o processo de leitura, com todas as inserções nas árvores e listas, as impressões, escrita no arquivo e finalização do laço de repetição presente na função *gestaoLER*(...).

Para os testes de busca, realizei a busca por uma palavra em cada unidade e por fim, fiz a busca pela a palavra "laranja" que não foi escrita no arquivo *txt* logo ela não será encontrada.

Palavra buscada	Unidade da Palavra	Foi encontrada?	Total em segundos
Professor	**Profissões**	Sim	0.106000
Cobra	**Animais**	Sim	0.128000
Azul	**Cores**	Sim	0.185000
Mal	**Diversos**	Sim	0.094000
Laranja	**---**	Não	0.488000

A segunda versão do arquivo continha 4 unidades, onde cada unidade continha um número diferente de palavras. Outro ponto importante deste teste é que todas as palavras contêm uma lista de mesmo tamanho, que são de 3 itens cada.

Unidades	Qnt. de Palavras	Total em segundos
Profissões	1	0.012000
Animais	3	0.028000
Cores	5	0.032000
Diversos	7	0.046000

O total para a inserir todas as 4 unidades foi de 0.247000 segundos.

O total de tempo gasto para realizar as seguintes buscas:

Palavra buscada	Unidade da Palavra	Foi encontrada?	Total em segundos
Médico	**Profissões**	Sim	0.084000
Ave	**Animais**	Sim	0.184000

Rosa	**Cores**	Sim	0.038000
Obrigado	**Diversos**	Sim	0.084000
Cinza	**---**	Não	0.441000

Questão 4: Vocabulário Inglês com Árvore AVL

Na questão 4 avaliamos o tempo necessário para realizar os mesmos testes da questão 3 agora com o acrescento das funções de balanceamento.

Primeiro teste: 4 unidades com 3 palavras cada, onde cada palavra possui uma lista simplesmente encadeada com 3 palavras inglesas.

Primeira tabela se refere aos resultados da inserção dos dados presentes no arquivo *txt* nas árvores:

Unidades	Qnt. de Palavras	Houve Balanceamento?	Total em segundos
Profissões	3	Sim	0.081000
Animais	3	Não	0.040000
Cores	3	Sim	0.086000
Diversos	3	Não	0.052000

O total para toda a execução do processo de leitura de arquivo e inserção das unidades foi de 0.401000 segundos.

Resultados das buscas:

Palavra buscada	Unidade da Palavra	Foi encontrada?	Total em segundos
Professor	**Profissões**	Sim	0.085000
Cobra	**Animais**	Sim	0.090000
Azul	**Cores**	Sim	0.115000
Mal	**Diversos**	Sim	0.052000
Laranja	**---**	Não	0.224000

Segundo teste: 4 unidades onde cada uma possui uma quantidade diferente de palavras. Resultados na inserção:

Unidades	Qnt. de Palavras	Houve Balanceamento?	Total em segundos
Profissões	1	Não	0.046000
Animais	3	Não	0.054000
Cores	5	Sim	0.096000

Diversos	7	Sim	0.108000

Resultados das buscas:

Palavra buscada	Unidade da Palavra	Foi encontrada?	Total em segundos
Médico	**Profissões**	Sim	0.103000
Ave	**Animais**	Sim	0.076000
Rosa	**Cores**	Sim	0.063000
Obrigado	**Diversos**	Sim	0.050000
Cinza	**---**	Não	0.311000

Conclusão

Como foi visto neste trabalho, as árvores binárias se apresentam como uma poderosa estrutura capaz de realizar buscas mais velozes do que outras que dependem da busca sequencial. Mas ainda assim, as árvores binárias possuem um grande problema, o caso de seus galhos muito profundos que pode gerar um atraso nas operações, para isso foram desenvolvidas técnicas de balanceamento de uma árvore, tornando seus galhos mais próximos possíveis, assim surgiu a árvore AVL.

Como foi visto na seção dos resultados de execução, a árvore binária apresentou vantagem em relação a árvore AVL no quesito de inserção de valores, isso se dá por conta do processo de balanceamento que pode ocorrer depois da inserção de um novo Nó. Já no quesito de buscas, a árvore AVL mostrou resultados em tempos menores, em decorrência ao processo de balanceamento.