**UNIVERSIDADE FEDERAL DO MATO GROSSO** *CAMPUS* ARAGUAIA

ANTHONY MUNIZ PRADO DE OLIVEIRA, VINICIUS SPANHOL FERRARI

**TRABALHO 2 – Árvores Binárias Balanceadas**

BARRA DO GARÇAS - MT

2022

**1 INTRODUÇÃO**

O objetivo deste trabalho é exercitar a implementação e manipulação de algoritmos de árvores de busca, visando verificar sua funcionalidade, e sua optimização em diferentes cenários onde se deve a grande quantidade de inserções de dados e busca de diferentes nós(“um nó é um dos membros da estrutura de uma árvore de busca”) nas mesmas com a linguagem de programação C. Também foi trabalhado a manipulação de arquivos.

O desenvolvimento consiste em 5 etapas em seu total, onde a **1ª etapa** consiste em realizar a implementação da Árvore Binária de Busca, Árvore AVL, Rubro Negra e B (ordem d=5), com adaptações no código para armazenar dados de escolas, de modo que a busca, inserção e remoção seja feita com base no id da escola. No caso da busca, se a escola existir, os dados dela deverão ser retornados por referência.

Implementar também as funções para análise da árvore, especialmente a função que identifica a altura da árvore e que contabilize o número de rotações executadas (Caso existam) em um procedimento de inserção (no caso de árvores balanceadas). As implementações deveriam ser independentes (um programa/algoritmo para cada tipo de árvore).

A **2ª etapa**, nós tivemos que fazer para cada algoritmo de árvore, um algoritmo que escreva um programa para ler um arquivo contendo os dados de 30228 escolas no ENEM. Cada linha do arquivo representa uma escola e é composta por nove informações, separadas por ponto e vírgulas.

A **3ª etapa** deveria ser avaliado o desempenho das Árvore Binária de Busca, Árvore AVL, Rubro-Negra e B a medida que os dados são inseridos:

* Número de níveis;
* Altura da árvore;
* Número de rotações/redistribuição (caso da B);
* Tempo total de construção da árvore.

E a **4ª etapa** deu-se a realização de uma estatística do tempo que as árvores implementadas consomem a medida que se aumenta o número de operações de busca. Para isso, deveriamos gerar 8 conjuntos de dados com as seguintes quantidade de índices para cada conjunto: 100, 1000, 10.000, 50.000, 100.000, 300.000, 500.000, 1.000.000. Os valores nos conjuntos devem ser selecionados aleatoriamente. Deveria ser aplicado a cada árvore e dividido o tempo de cada conjunto de buscas.

A **5ª etapa** é a escrita deste relatório aqui que deve descrever e discutir os resultados obtidos ao final da implementação das etapas anteriores.

Foi utilizado o editor de textos Visual Studio Code e a linguagem e bibliotecas implementadas em c para a programação, dos algoritmos responsáveis pelos dados que serão apresentados neste trabalho.

**2 DESENVOLVIMENTO**

Etapa 1:

Foi realizada a implementação das Árvore Binária de Busca, Árvore AVL, Rubro Negra, a árvore B não foi implementada, pois não conseguimos adaptar ela para aceitar os dados do aluno. Mas as outras conseguimos realizar esta adaptação e fizemos as mesmas armazenarem os dados do arquivo “DadosEnem.txt”, desta fora a estrutura final das árvores ficou desta forma:

**ESTRUTURA ALUNO (Utilizado na implementação da estrutura das árvores)**

typedef struct Aluno{

//chave da futura arvores

int id;

//dados do estudante

char estado[3];

char municipio[40];

char rede[20];

//notas do estudante

double media\_ciencias\_natureza;

double media\_ciencias\_humanas;

double media\_linguagem;

double media\_matematica;

double media\_redacao;

} Alunos;

**ARVORE BINÁRIA:**

typedef struct No{

int chave;

Alunos \*aluno;

struct No \*esq;

struct No \*dir;

} ArvBin;

**ARVORE AVL:**

typedef struct sNoA{

int chave;

int fb; //fator balanceamento

int h; //altura no

Alunos\* aluno;

struct sNoA \*esq;

struct sNoA \*dir;

}ArvAVL;

**ARVORE RUBRO-NEGRA:**

typedef struct NoRB{

int chave;

int cor;

Alunos \*aluno;

struct NoRB \*esq;

struct NoRB \*dir;

struct NoRB \*pai;

} ArvRB;

Utilizamos nosso algoritmo “Main” para contabilizar o total de rotações executadas e avaliar a altura das árvores resultantes após a inserção dos valores informados pelo arquivo ‘txt’ de entrada. Para isso criamos funções próprias no algoritmo de cada árvore para contabilizar o nível e o número de rotações das mesmas que realizam rotações, e por fim retornar esses valores a Main por meio do ‘printf’.

Etapa 2:

Fizemos uma função no Main que realiza a leitura do arquivo (“DadosEnem.txt” que contém os dados de 30228 escolas no ENEM) por meio da função ‘fscanf’ e armazenar os mesmos em um ponteiro de vetor do tipo Alunos (Alunos \*v[30228]) e depois passa esses dados por parâmetro para uma função que irá inserir esses dados na árvore. Para essa inserção foi criado um arquivo chamado “insercao\_time.c”, que contém as funções que chamaram a inserção do vetor nas estruturas de busca implementadas, como um exemplo:

ArvBin \*tempo\_e\_Insercao\_ARVBinaria(ArvBin \*Arvore\_Binaria, Alunos \*v[], int tam);

Esta função vai inserir o vetor na árvore binária, e é passado para ela a raiz da estrutura, o vetor dos dados, e o tamanho do vetor. A implementação foi feita para deixar o arquivo Main mais limpo, então fizemos apenas a implementação de preenchimento de vetor dentro dele e por fim apenas chamamos as funções de inserção e de busca que serão melhor explicadas nas seguintes etapas.

Etapa 3:

Nesta parte é pedido que seja realizado a avaliação do desempenho de inserção das Árvores Binárias, porém esta avaliação já é feita durante a inserção que foi explicado acima, então ela já é pré-montada durante ela, e sendo necessário apenas chamar na Main a função que retorne os dados dos Níveis e Rotações de cada árvore se a mesma fizer rotações, para que os mesmos pudessem ser comparados para a conclusão do objetivo deste trabalho.

Etapa 4:

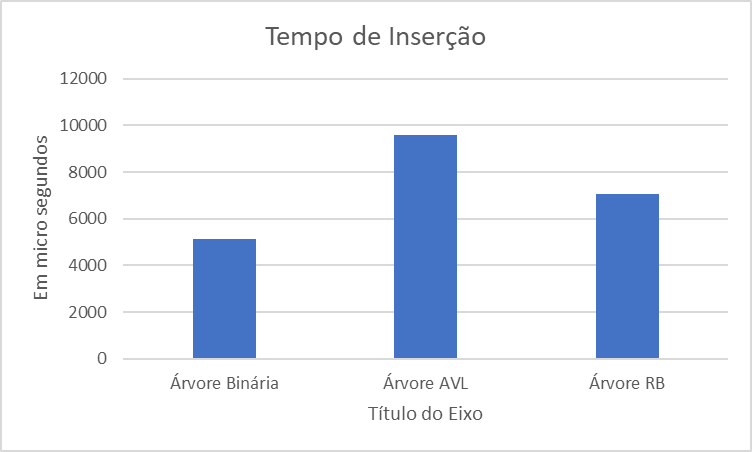
Com o auxílio do algoritmo desenvolvido nesta etapa 4 “Etapa\_4/Etapa4.c”, foram gerados 9 conjuntos de dados de chaves aleatórias e armazenados esses mesmos em arquivos de entrada nomeados com base em seu conteúdo, utilizados nas funções implementadas no arquivo ‘insercao\_time.c’ que chama a função de busca de cada árvore e informar a chave(Que fora retirado desses arquivos gerados) que será utilizada para o rastreamento do respectivo nó com base na chave informada. Ao fim da busca de todos as chaves de todos os conjuntos é feito um cálculo do tempo decorrido na busca de cada conjunto eo tempo total para a busca em todos os conjuntos de chaves e retornar esse tempo na Main para a comparação com os tempos das outras árvores que também passaram pelos mesmos processos para um cálculo e comparação de ambas, com os mesmos dados de buscas para que obtivéssemos os dados de uma maneiras mais organizada e realizar uma análise estatística mais eficiente para cada caso.

**3 RESULTADOS**

**3.1 Analisando o tempo de inserções**

**tabela 1**

| **Tempo de Inserção em microsegundos [µs]** | | |
| --- | --- | --- |
| **Árvore Binária** | **Árvore AVL** | **Árvore RB** |
| **5144µs** | **9579µs** | **7073µs** |

****

Tempo de inserção da avl maior como esperado.

**3.1 Analisando o número de rotações**

**tabela 2**

| **Número de Rotações** | | |
| --- | --- | --- |
| **Árvore Binária** | **Árvore AVL** | **Árvore RB** |
| **não realiza** | **31766** | **32427** |

obs: na árvore RB as rotações duplas contam como 2 diferente da avl, logo o numero de rotações é menor.

Avl fez mais rotações como o esperado.

**3.2 Analisando os níveis das árvores**

**tabela 3**

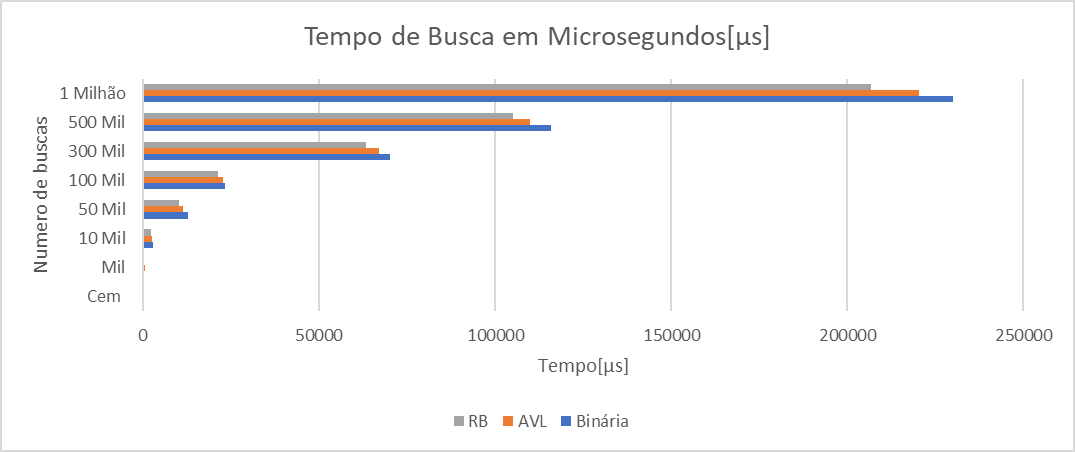
| **Número de Níveis das Árvores** | | |
| --- | --- | --- |
| **Árvore Binária** | **Árvore AVL** | **Árvore RB** |
| **44** | **20** | **17** |

Aqui houve uma quebra de padrão onde a Árvore Avl ficou com mais níveis que a RB, o que não é o esperado normalmente.

**3.3 Analisando o tempo de busca das árvores**

**tabela 4**

| **Tempo de Busca em Microsegundos[µs] em cada arquivo** | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Arquivos** | **Cem** | **Mil** | **10 Mil** | **50 Mil** | **100 Mil** | **300 Mil** | **500 Mil** | **1 Milhão** |
| **Árvores** |  |  |  |  |  |  |  |  |
| **Binária** | **108** | **506** | **2737** | **12899** | **23432** | **70053** | **115794** | **230006** |
| **AVL** | **105** | **474** | **2553** | **11302** | **22647** | **67076** | **109884** | **220417** |
| **RB** | **98** | **468** | **2421** | **10363** | **21329** | **63238** | **105129** | **206898** |



Aqui houve uma quebra de padrão onde a Árvore Avl demorou mais para realizar as inserções que a RB, o que não é o esperado normalmente.

**4 CONSIDERAÇÕES FINAIS**

Ao fim das execuções e comparações dos valores finais fornecidos pelo algoritmo, nós acabamos por chegar nas seguintes conclusões:

Mesmo o algoritmo da árvore AVL realizando um grande número de rotações, como pode ser visto ao analisar a tabela 2, acabou gerando uma árvore com mais níveis que a Rubro-Negra, o que explicaria o porquê desse algoritmo resultar em tempos melhores para a Árvore RB, tanto nas inserções, quanto nas buscas que são o ponto fraco dessa árvore.

Já a árvore binária clássica por não realizar rotações se torna a mais simples de ser executada, porém se torna muito grande e nada otimizada em tamanho, sendo demorado sua inserção que em todas as execuções percorrerá um caminho muito grande dependendo da situação diferente da AVL e Rubro-Negro gerando assim uma árvore capaz de ter uma altura igual ao número de valores inseridos, tornando ela a pior na prática quando for necessário fazer uma buscas ou uma exclusão.

Ou seja, podemos concluir que neste caso a melhor e mais rápida é a Rubro-Negra quando se quer agilidade na busca e inserção de muitos valores.