Estrutura de Dados em Árvores: Uma análise comparativa entre as árvores ABB, AVL, ARN e AVL com base na coleta de resultados de 3 algoritmos simples.

Samuel de Oliveira Ribeiro

Resumo:

Este relatório apresenta informações sobre as definições das estruturas de árvores básica, senso elas: Árvore Binária de busca (ABB), AVL, Árvore Rubro Negra (ARN) e a Árvore 23 (A23), com base nessas informações foi desenvolvido algoritmos para a resolução de 3 algoritmos propostos em sala. Com base na análise destes algoritmos e nos resultados coletados por ele foi feito um estudo comparativo em operações de inserção e busca no contexto dos exercícios para discernir sobre qual é a melhor implementação para cada tipo de algoritmo.

1. Introdução:

Este documento apresenta algumas informações sobre as estruturas de dados em árvores vistas em sala sendo organizado da seguinte forma: a Seção 2 contém uma breve caracterização dos tipos de árvore com algumas imagens demonstrativas de sua estrutura visual; a Seção 3 contém a descrição dos problemas propostos e uma explanação sobre as implementações desenvolvidas para cada problema; a Seção 4 é feito uma análise comparativa entre os resultados obtidos em cada algoritmo; e a Seção 5 contém a conclusão deste trabalho.

Este trabalho visa demonstrar os tipos de estrutura de dados em árvores mais básicos para assim discernir sobre o funcionamento, características e quando cada tipo de árvore é mais indicado para implementação. Para definir qual tipo de árvore utilizar em determinado problema é preciso primeiro realizar uma análise do problema, quantidade de informações a serem armazenadas, se os dados na árvore precisam estar bem distribuídos, entre outros fatores. Sabendo disso é possível prover uma otimização de recursos (hardware e software) e assim maximizar as possibilidades de evoluir o problema sem afetar a estruturação previamente realizada.

Para fins informativos todos os algoritmos foram executados num notebook da Samsung com as seguintes especificações:

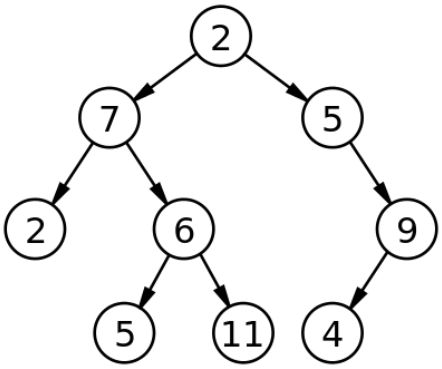
- Modelo: X51-2015
- Processador: Intel© Core™ i7-6500U CPU @ 2.50GHz × 1
- Sistema Operacional: Linux Mint 19.1 Cinnamon
- Memoria: 7.7 GiB

2. Seçoẽs Específicas:

Para a realização deste trabalho é importante conhecer algumas estruturas de árvores. Abaixo está uma breve descrição das principais características de cada estruturas de dados em árvore utilizada.

2.1. Árvore Binária de Busca (ABB)

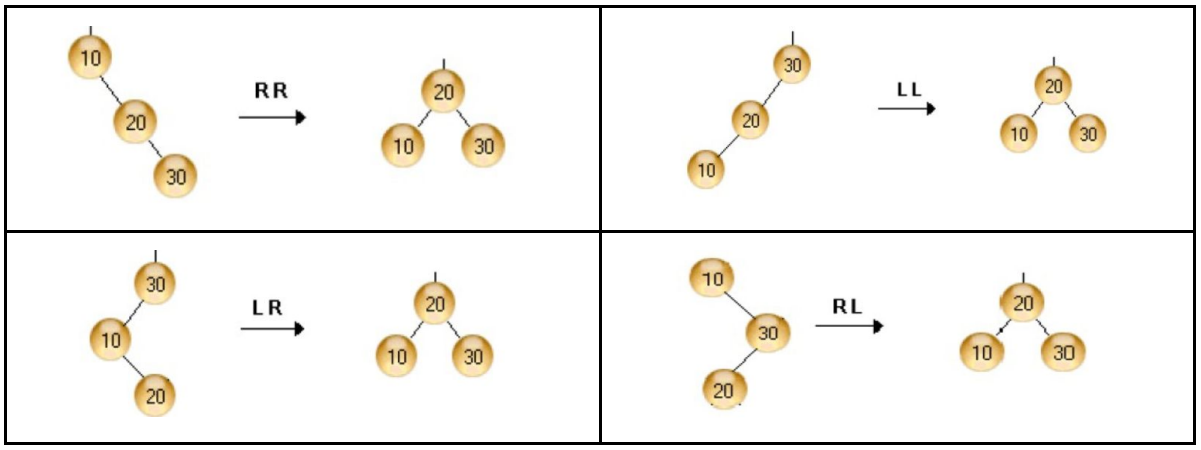
Uma árvore binária de busca é uma estrutura de dados baseada em nós. Cada nó armazena um conjunto de informações, um identificador da mesma e ponteiros para para associar a uma subárvore à esquerda e/ou à direita. Todos os nós da subárvore à esquerda contém valores identificadores menores que o identificador da raiz, e a subárvore da direita tem valores identificadores maiores que o da raiz (Por convenção).



Sempre o novo nó inserido irá se tornar uma folha. É indicado a utilização desta estrutura de dados quando é necessário realizar muitas buscas dos dados que estão bem distribuídos pois a mesma reduz em até 50% do esforço de busca, entretanto a organização (forma de inserção) dos dados influenciará diretamente da estruturação da árvore pois a mesa não dá suporte a reorganização dos dados durante/após a inserção.

2.2. Árvore AVL

Similar a árvore binária de busca (Seção 2.1) a árvore AVL tem as mesmas características citadas contudo ela é otimizada em relação a organização dos dados na mesma por meio da realização de balanceamentos (termo usado para definir a reestruturação da árvore em relação os níveis de cada nó). Ao final da inserção de uma informação na árvore (na volta da recursividade) é feito verificações para definir um número que representa a diferença entre a subárvore da esquerda pela subárvore da direita. Caso esta diferença seja igual ao valor absoluto 2 significa que naquele ponto a árvore está desbalanceada e que deve ser feita uma rotação específica (há 4 tipos de rotações: RR, LL, RL e LR) para mantê la nos padrões de uma árvore balanceada.

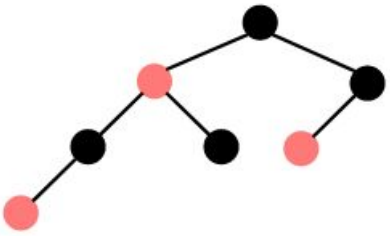


Para definir o tipo de rotação é verificado se o maior ramo é interno ou externo, caso seja interno é realizada uma rotação dupla, senão será uma rotação simple. Esta implementação visa otimizar as buscas em árvores mantendo a estrutura da mesma mais balanceada possível de acordo com as inserções. O algoritmo desta árvore se torna custoso quando há muitos balanceamentos a serem executados.

2.3. Árvore Rubro-Negra caída a esquerda (ARN_CE)

A árvore rubro negra consiste em um tipo especial de árvore binária de busca onde cada nó tem um atributo cor, podendo ter valores *RED* e *BLACK*. Essa estrutura é organizada de forma que os nós de cor vermelho fiquem sempre à esquerda de um nó pai

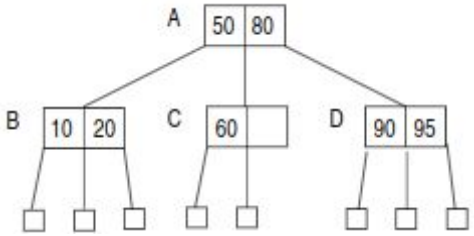
(Especifico da Árvore rubro negra caída a esquerda); o nó raiz sempre é preto; um novo nó é criado vermelho. Para que essas regras sejam cumpridas durante inserção é feito o balanceamento da árvore com base em 3 condições comparativos: Se o nó a direita é vermelho, se há 2 nó seguidos a esquerda da cor vermelha e se os dois filhos são da cor vermelha e para cada caso deste há uma modificação específica, sendo respectivamente o rotacionamento a esquerda, o rotacionamento a direita e a troca de cores.



A árvore rubro negra leva vantagem em relação a árvore AVL pois em sua função de balanceamento há apenas rotações simples diminuindo assim o tempo e custo de processamento na inserção.

2.4. Árvore 2-3 (A23)

A árvore 2-3 é uma estrutura de árvore que pode armazenar até 2 informções em um nó e que pode tem 2 ou 3 filhos. Sempre que houver uma informação no nó haverá 2 filhos, um à esquerda e outro no centro. Já se houver 2 informações no nó deverá ter 3 filhos, um à esquerda, um no centro e outro a direita. Após encontrar o local de inserção deverá ser feita o balanceamento do mesmo de acordo com as informações. A principal função a ser desenvolvida é a de quebrar o nó, que é utilizada quando é inserido uma informação em um nó ja cheio. Sempre promovendo um novo nó. A estrutura da árvore é apresentada abaixo.



3. Metodologia

Nesta seção será apresentado os problemas propostos e algumas as principais características de cada implementação realizada.

3.1. Inserção e busca de N números

Este algoritmo consiste em realizar a inserção de N números aleatórios numa determinada árvore e que seja realizada alguns testes de nível, busca, profundidade e a verificação do tempo de execução da inserção e da busca. A figura abaixo demonstra o fluxo de execução do problema.

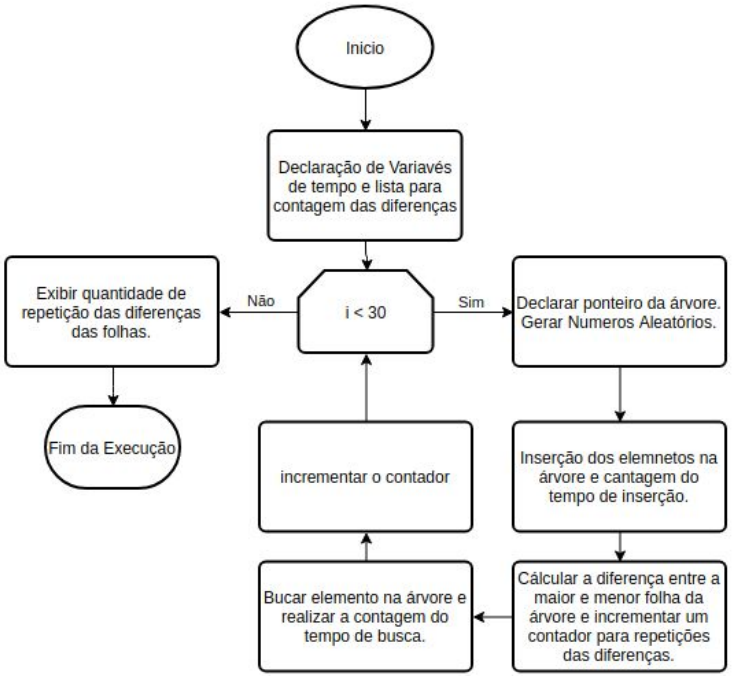


Figura 01: Fluxograma do algoritmo para inserção de números.

Para este problema foi desenvolvido a solução nas estruturas de árvore ABB, AVL e Rubro Negra. A seguir, para cada implementação, uma breve explicação do funcionamento geral do algoritmo e suas principais funções.

3.1.1. ABB

Esta implementação tem baixa complexidade pois o algoritmo pode ser facilmente implementado baseando-se nas funções de: inserção, busca, criar Folha e outras funções complementares. Neste caso a árvore binária de busca segue exatamente os passos do fluxograma acima, tendo apenas algumas funções não especificadas como a de calcular a maior e menor folha, e a função de maior ramo para assim facilitar no cálculo da altura de cada nó.

A função de inserção busca o local onde o novo valor será inserido (Esquerda ou Direita até encontrar um nó *NULL),* quando este local for encontrado o algoritmo volta às recursividades recalculando a altura de cada nó. Uma outra função importante é a de criar um nó, ela inicializa os valores dos ponteiros em *NULL*, a altura em 0 e o valor de acordo com a entrada dos dados.

É importante notar que a árvore binária de busca apenas adiciona os dados na árvore de acordo com a sua inserção, esta estrutura de árvore não reorganiza os dados, sendo assim é possível que ela se torne uma lista encadeada e não cumpra com seu propósito de facilitar a busca a dados.

3.1.2. AVL

O escopo geral desta implementação é similar a da árvore binária de busca (Seção 3.1.1), a única diferença de implementação é que, como exemplificado na seção 2.2, a árvore AVL realiza rotações logo após a inserção para que a árvore possa se manter o mais balanceada possível. É feita a chamada recursiva com a função de inserir dados, nesta função é feito as seguintes verificações: se a raiz é *NULL,* se a nova informação é menor ou maior que a atual; Caso seja encontrado o lugar correto de inserção é feito a verificação da diferença entre a altura dos ramos a

esquerda e á direita sendo que quando for igual a 2 será feito uma das rotações simples (LL e RR) ou rotações duplas (LR e RL) neste nó desbalanceado.

A árvore AVL tem os mesmos princípios que a ABB, contudo ela provê uma melhor organização dos dados realizando um balanceamento simples entre os nós durante a inserção dos mesmos. Essa implementação visa otimizar mais ainda uma posterior busca pois quanto mais balanceada a árvore menor será o maior nível da mesma e menor será o custo computacional para chegar na informação desejada.

3.1.3. Rubro Negra

Similar a Árvore AVL, a árvore rubro negra realiza a inserção de um nó e em seguida volta balanceando a árvore conforme citado na seção 2.3. A implementação deste algoritmo é semelhante aos demais citados acima, sua única diferença é que nas funções de rotação há um atributo a mais (Cor) e que há uma função que realiza a troca de cor do atual nó e seus filhos. A função de inserir serve apenas chamar a função *criaNo*, que de fato realiza a inserção, e para sempre manter uma propriedade da árvore onde a raiz sempre deve ter cor preta.

Da mesma forma que a AVL a árvore Rubro negra faz o balanceamento durante a inserção, mas o seu principal diferencial é que todas as rotações a serem realizadas serão apenas de movimentos simples (LL e RR) tornando assim a árvore ARN mais eficiente em questão custo computacional para inserção.

3.2. Dicionário Inglês-Português

Este algoritmo consiste em realizar a organização de um dicionário Português-Inglês, onde cada unidade será numa estrutura de árvore e cada nó desta árvore está relacionado a uma palavra em português e suas possíveis tradunçoẽs em inglês. O algoritmo deve fornecer possibilidade de criar N unidades e adicionar N palavras sempre tratando as possíveis repetições e mantendo a estrutura organizada em ordem alfabética.

3.2.1. ABB

Este algoritmo dá suporte para realizar a importação de dados, remoção de um palavra em português, buscar uma palavra em português e exibir dados de uma unidade. Todas as funções foram modularizadas a seguir apresento as principais delas:

- Importar Dados: Primeiro é realizado o tratamento da entrada onde a palavra em inglês fica numa variável a as palavras em português são armazenadas numa lista encadeada. Posteriormente é utilizado a função de inserção que adiciona cada palavra em inglês em um nó em português, desconsiderando as repetições. Este processo continua até que todas as linha da entrada sejam processadas e inseridas.
- Remover Palavra PTBR: A função de remover uma palavra busca um nó na árvore que exista a palavra desejada, caso este nó seja encontrado e feito o direcionamento dos ponteiros para que este nó seja substituído por um que mantenha a nesta ordem na árvore. Ao final o nó é dado um *free* no nó que removido. Se a palavra buscada não existir a função não fará nenhuma alteração na árvore.
- Buscar Palavra PTBR: É feita uma varredura em toda a árvore em busca de um nó que possua a palavra buscada. A função retorna 1 quando a palavra for encontrada.

- Exibir Unidade: É solicitado a unidade que deseja ser exibida e caso a unidade exista será exibido em ordem (Esquerda + Raiz + Direita)[Lista].

3.2.2. Árvore 2-3

Para este algoritmo foi implementado com a mesma lógica da Árvore Binária de Busca, contudo na árvore 2-3 há uma maior capacidade de armazenamento por nível (2 informações por nó), assim foi preciso desenvolver funções essenciais e específicas para o funcionamento. Abaixo algumas das principais funções:

- *lerArquivo*: Esta função é responsável por ler o arquivo de entrada, tratar as entradas linha a linha para que os dados possam ser utilizados para inserir em uma árvore. Internamente ela faz uso da função *inserirCapitulo(),* que cria um novo capítulo na unidade com nome de entrada, e a função *Inserir*.

- *Inserir*: A função inserir é responsável por buscar um nó para inserir a lista de palavras em inglês, caso este nó não exista ela irá criar um novo nó para esta palavra em português. Durante a inserção é preciso verificar o local exato na árvore para manter as suas definições. Dependendo do caso o local nó de inserção pode estar cheia, sendo assim este nó deve ser redefinido para que a estrutura da árvore continue dentro das regras. Para isso é utilizado a função *quebraNó.*

- *quebraNo*: Esta função tem por finalidade quebrar um nó passado por referência e um novo nó separando de forma coerente com as definições e retornar um novo nó que será promovido. É importante que todos os ponteiros no atual e novo nó sejam redefinidos para que nenhuma informação seja perdida.

- *Remover23: Função responsável por encontrar a informação a ser removida e fazer o redirecionamento dos dados. Após encontrar a informação para remover é preciso verificar se a estrutura da árvore está correta. Caso seja encontrado algum nó que esteja fora dos padroẽs o algoritmo irá fazer o redirecionamento dos dados para correção.*

3.3. Blocos de memória

Este problema visa a simulação da organização de um conjunto de blocos de memória em um computador, sendo possível alocar e liberar N blocos de memória sempre mantendo a estrutura organizada. Esta estrutura será organizarem em uma árvore cujo cada nó é referênte um um conjunto de blocos de memória sabendo do endereço inicial e final do mesmo.

3.3.1. AVL

A implementação deste algoritmo consiste em realizar operações similares às que ocorrem na memória de um computador simulando-a numa estrutura de árvore AVL. A seguir as principais funções:

- Inserir: Esta função sempre realiza a inserção no nó mais a direita possível, apenas realizando o balanceamento na volta das recursões.
- Alocar: É feito a busca pelo nó livre mais a esquerda cujo espaço seja capaz de alocar n blocos solicitados. Ao encontrar este nó é feito a junção com o nó mais à esquerda e mais a direita que são ocupados para assim manter a estrutura do algoritmo correta (Nós alternados entre Livre e Ocupado)

- Liberar: A função de liberar a memória é similar a de alocar, a única diferença é que para liberar a memória é feito a busca por um nó ocupado. Os restantes passos são como citados no tópico acima.

4. Resultados da Execução do programa

Nesta seção será apresentado alguns dados comparativos provenientes da execução dos algoritmos implementados e descritos na seção anterior. Para cada tipo de algoritmo foi realizado testes de inserção e busca.

4.1. ABB x AVL x RubroNegra (Inserção e busca de N números)

Todos os resultados a seguir foram obtidos após realizar um teste de 30 repetições de inserção sendo cada uma delas com 10000 (dez mil) elementos aleatórios.

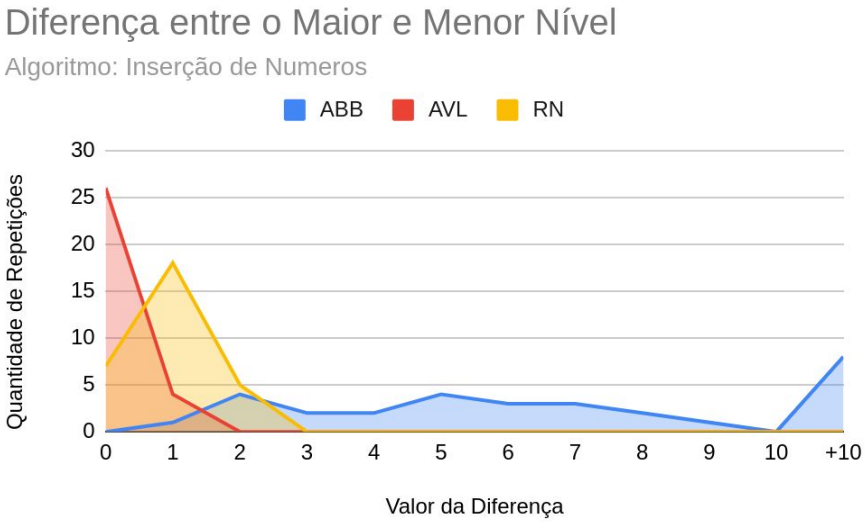
Tipo de Árvore	Tempo Médio de Inserção em milissegundos	Tempo Médio de Busca em milissegundos
ABB	3	0
AVL	981	0
Rubro Negra	4	0

Tabela 01: Tempo de inserção e busca de dados

Em relação a inserção de dados é notável que a árvore AVL demanda mais tempo, isso pode ser explicado pois ela, sempre ao inserir um dado, faz o processo de balanceamento e este balanceamento no melhor caso faŕa uma rotação simples e no pior caso fará uma rotação dupla. A árvore Rubro Negra também realiza este balanceamento, contudo ela não é totalmente balanceada e faz apenas rotações simples. Já a árvore ABB é muito rápida na inserção pois o custo de tempo é apenas para encontrar o local da inserção, não há balanceamento na árvore.

Em relação a busca de elementos a árvore ABB demanda um maior tempo pois os dados não estão organizados de forma balanceada, o tempo de busca na ABB no pior caso e melhor caso será o tempo para chegar ao maior nível e menor nível respectivamente. A árvore rubro negra demanda um tempo de busca mais rápido que a ABB contudo ela não tem o tempo de busca mais eficiente pois a árvore não é totalmente balanceada. Em relação a árvore AVL o tempo de busca é o mais rápido entre as 3 pois seus dados estão balanceados de uma melhor forma.

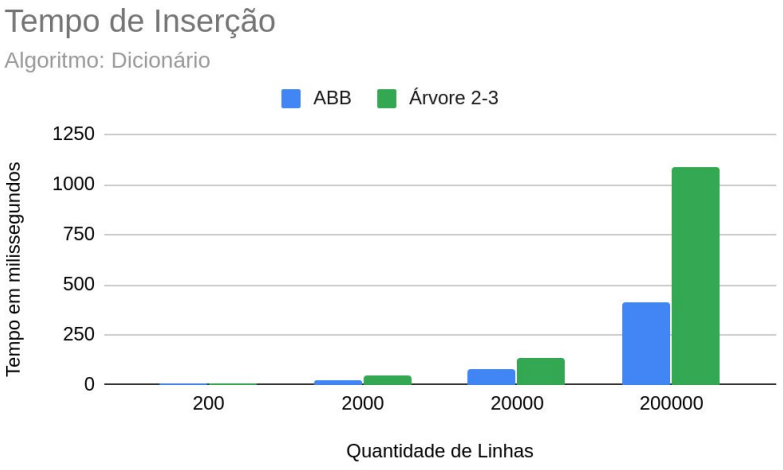
Podemos observar que em relação esses 3 tipos de árvore o tempo de busca é inversamente proporcional ao tempo de inserção, pois, neste caso, quando mais tempo para inserir melhor será a organização dos dados e menor será o tempo de busca de um elemento. Os tempos de busca na tabela acima ficou em 0 (zero) pois mesmo não sendo a estrutura de árvore mais rápida a nível computacional o processamento é muito rápido e somente seria capaz de coletar tempo de busca se houvesse um grande volume de dados.



O gráfico demonstra que a árvore AVL tem um elevado número de repetições com valores de níveis baixos, isso ocorre pois esta árvore, como falado anteriormente, é bem balanceada, dessa forma os níveis sempre ficam o mais próximo possível sem um grande nível de discrepância. Já a árvore ARN ainda assim como a AVL tem baixos valores de diferença pois a mesma também realiza balanceamentos porém não tão rigorosa quanto à AV. A árvore ABB tem diferença de níveis bastante variáveis pois ela não provê balanceamento dos dados.

4.2. ABB x Árvore 2-3 (Dicionário Inglês-Português)

Para os resultados a serem mostrados foi utilizado um arquivo de entrada base contendo 200 palavras em inglês e cerca de 500 diferentes palavras em português (as palavras em português utilizadas nos testes não condizem com a tradução literal do inglês). Estes dados foram utilizados para os arquivos testes maiores, onde os mesmos estão apenas repetindo 10x mais o arquivo base.



O gráfico acima demonstra o tempo de inserção na árvore ABB e A23 em relação a quantidade de linhas no arquivo de entrada. Observamos que a árvore ABB leva um menor tempo de inserção pois a mesma não realiza balanceamentos. Já a árvore A23 leva tem um elevado tempo de inserção pois sempre é realizado balanceamentos e redirecionamentos dos dados para manter as definições da árvore correta. A ABB se torna eficaz na inserção porém as buscas não tem um tempo de resposta tão rápido quanto a árvore A23 que leva um tempo

maior para manter a estruturação.Todos os testes de busca deram resultado 0, pois mesmo que grande o arquivo de teste não foi suficiente para gerar resultados.

Assim, podemos verificar que a árvore ABB é mais rápida na inserção porém possui o maior tempo de busca entre as três; a árvore ARN possui um pouco maior que a ABB e tem um tempo de busca próximo ao da árvore AVL; já a árvore AVL possui um elevado tempo de inserção e um tempo mínimo para realizar as buscas.

4.3. AVL (Blocos de memória)

Para todos os casos de teste realizados não foi possível coletar dados de tempo. O pior caso de teste utilizado teve cerca de 8000 nós na árvore com um intervalo de endereço de 10000000 (Um bilhão). A implementação para Alocar e Liberar blocos de memória está praticamente igual, alterando somente qual tipo de bloco é buscado (Livre ou ocupado) sendo assim o tempo será bem próximo. Em questão os tipos de caso de testes o algoritmo trata de forma igualitária.

5. Conclusão:

Este estudo discerniu sobre o funcionamento das árvores ABB, AVL, ARN e A23 com base em 3 algoritmos propostos: inserção de dados, dicionário português-inglês e blocos de memória. Com base nos testes executados podemos observar uma proporção entre tempo de inserção e tempo de busca nos dados. Quando maior o tempo para realizar a inserção menor será o tempo para finalizar a busca de dados (Em relação às estruturas de árvores presentes neste estudo).

Em questão a árvore binária de busca (ABB) o seu tempo de inserção é o menor e o tempo de busca é o maior pois a mesma não realiza balanceamentos. A árvore ARN têm tempos de inserção um pouco maiores que a ABB e tempo de busca um pouco menor que a AVl, isso ocorre pois a ARN contém balanceamentos não tão rigorosos quanto a AVL. Já a árvore AVL é mais otimizada para realizar buscas e bem custosas na inserção pois seus balanceamentos deixam os dados bem organizados mas detém um grande número de modificações. A árvore A23 é recomendada para grandes estudos com grande volume de dados pois seu tempo de busca e a sua organização é a mais otimizada entre todas deste estudo, contudo o seu tempo de inserção é elevado pois assim como na AVl realiza grandes volumes de modificações.

Este estudo contou com o objetivo de identificar características das estruturas de árvore (ABB, AVL, ARN e A23) com base na coleta de dados realizada, por meio da execução dos algoritmos propostos. Esta coleta de dados forneceu informações, como o tempo para inserção/busca e diferenças de níveis após a inserção na árvore, para fins comparativos. Com base nestes dados foi possível classificar os identificar a melhor estrutura de árvore para implementar cada algoritmo de acordo com os critérios de tempo de inserção, tempo de busca e/ou graus de diferenças de nível.

6. Referências:

VILLAS, Marcos V. et al. **Estruturas de Dados**. N.º 12. Local: Rio de Janeiro, Elsevier: Campus, 1993.

BOERES, Cristina. **ESTRUTURA DE DADOS Árvores, árvores binárias e percursos.** 73 slides. Disponivel em <http://www2.ic.uff.br/~boeres/slides_ed/ed_ArvoresPercursos.pdf>. acesso em 26 de set de 2019.

TENEMBAUM, Aaron M.; LANGSAM, Yedidyah; SOUZA, AUGENSTEIN, Moshe J. **Estruturas de Dados Usando C**. Local: São Paulo, MAKRON Books, 1995.