RELATÓRIO DE RESOLUÇÃO DO TRABALHO 1 CORRESPONDENTE A PRIMEIRA AVALIAÇÃO DA DISCIPLINA DE ESTRUTURAS DE DADOS II

Eduardo de Sousa Gomes Vieira

eduardosousa@ufpi.edu.br

Estruturas de Dados II - Juliana Oliveira de Carvalho

Resumo

Este relatório tem como propósito abordar a resolução das questões da primeira atividade avaliativa da disciplina de Estrutura de Dados II, com o intuito de apresentar os elementos necessários para compreender o problema e sua solução. As questões foram solucionadas utilizando a linguagem de programação C em conjunto com a estrutura de dados das árvores binárias. Tanto a abordagem sem balanceamento quanto a abordagem com balanceamento conhecida como AVL foram aplicadas, permitindo a criação de programas capazes de solucionar as questões propostas.

Palavra-chave:

Árvores Binárias, AVL, inserção, busca.

Introdução

Árvores binárias são estruturas de dados que consistem em nós interconectados, onde cada nó possui no máximo dois filhos: um filho à esquerda e um filho à direita. Cada nó pode conter um valor ou uma chave associada a ele. Uma árvore AVL é uma árvore binária de busca balanceada. A principal característica de uma árvore AVL é que ela mantém o equilíbrio entre suas subárvores para garantir uma altura mínima e, consequentemente, um desempenho eficiente em suas operações.

Os tópicos subsequentes deste trabalho estão organizados de forma a apresentar as questões propostas, assim como a lógica essencial para a sua resolução, exemplos de execução dos programas, testes de eficiência comparando a árvore binária sem balanceamento e a árvore balanceada (AVL), conclusão dos resultados obtidos e um apêndice contendo todos os algoritmos utilizados. As questões da atividade são divididas de modo a demonstrar a solução de um mesmo problema utilizando os dois tipos de árvores mencionados anteriormente, permitindo comparar a velocidade de inserção e busca de dados.

Hardware utilizado

Todos os testes a seguir foram feitos utilizando o mesmo hardware, com as seguintes características:

Marca	Lenovo
Sistema Operacional	Windows 11
Processador	Intel Core i5-1035G1

Memória RAM	8GB
Tipo de memória	DDR4 2666MHz

Seções Específicas

Os tópicos a seguir, serão apresentados os enunciados das questões, juntamente com suas resoluções e testes de desempenho. Isso possibilitará aos leitores compreender a lógica por trás da solução e avaliar a eficiência do método utilizado.

Estruturas de Dados Utilizadas

Foram utilizados dois tipos de estruturas não lineares Árvore binária e AVL.

- **Árvore Binária**
  - Estrutura Curso: contém os campos código do curso, nome do curso, quantidade de blocos, número de semanas para cada disciplina, endereço para a árvore de disciplinas daquele curso, endereços para o nó esquerdo e direito.
  - Estrutura Disciplina: armazena os campos código da disciplina, nome da disciplina, bloco da disciplina, carga horária da disciplina, endereços para o nó esquerdo e direito.
- **AVL**
  - A AVL utiliza a mesma Estrutura de Curso e Disciplina da Árvore Binária com acréscimo de um campo chamado altura que indica a altura de cada nó da AVL.

Questão 1 – a

Nessa parte da questão foi proposto uma implementação em C que cadastre em uma árvore binária, os dados de um curso, pra isso foi implementado uma struct Curso com os seguintes campos:

- Código: Essa informação vai servir como um identificador único para cada curso, pois além dos números não poderem se repetir, a árvore vai estar organizada por esse código.
- Nome do curso: Essa variável vai armazenar o nome do curso em um vetor do tipo char com 50 posições
- Quantidade de blocos: Uma variável do tipo inteiro que irá armazenar a quantidade de blocos que o curso possuí.
- Número de semanas: Essa informação irá armazenar os números de semanas para cada disciplina, essa informação é do tipo inteiro.
- Endereço para a árvore de disciplinas: Nessa variável, do tipo ponteiro, será armazenado o endereço de memória de uma nova árvore (Questão 2). A Arvore de Disciplinas ficará dentro de cada curso, com isso, cada nó da árvore de curso possuirá uma árvore de Disciplinas.
- Endereço de um nó da esquerda e da direita da raiz: Essas informações não foram mencionadas na atividade, mas por se tratar de árvores de busca é necessário a implementação dos mesmos. Esses ponteiros são inicializados nulos.

Para solucionar esse problema da letra a da primeira questão, primeiro foi criada uma função para criar o nó do curso em questão, que tem como parâmetro código, nome

do curso, quantidades de blocos e as semanas do curso, ao final a função retorna o nó criado. Logo após a criação do novo nó, ele será passado para outra função de inserir nó na árvore, que tem como parâmetro a raiz da árvore o qual é passada por referência e o novo nó, por fim é inserido o novo nó na árvore, essa função não possui retorno.

Questão 1 – b

Nessa parte da questão foi proposto uma implementação em C que cadastre em uma árvore binária, os dados de uma disciplina, pra isso foi implementado uma struct Disciplina com os seguintes campos:

- Código da disciplina: Essa informação vai servir como um identificador único para cada disciplina, pois além dos números não poderem se repetir, a árvore vai estar organizada por esse código.
- Nome da disciplina: Essa variável vai armazenar, em um vetor do tipo char com 50 posições, o nome da disciplina.
- Bloco da disciplina: Variável inteira que representa o bloco que a disciplina pertence.
- A carga horária da disciplina: Variável inteira que irá armazenar a carga horária da disciplina.
- Endereço de um nó da esquerda e da direita da raiz: Essas informações não foram mencionadas na atividade, mas por se tratar de árvores de busca é necessário a implementação dos mesmos. Esses ponteiros são inicializados nulos.

Para solucionar esse problema da letra b da primeira questão, foi criado uma função que vai criar o nó da disciplina, que tem como parâmetro o código, nome, o bloco e a carga horária da disciplina, e ao final a função retorna o nó criado. Logo após a criação do novo nó, ele é passado para outra função que irá inserir nó na árvore de disciplinas, que tem como parâmetro a raiz da árvore de curso o qual é passada por referência e o novo nó de disciplina. Essa função de inserir o curso pode ser combinada com outra função que verifica se o nó do curso existe, caso exista ele retorna o no do curso, se não, é retornado nulo, essa função tem como parâmetro a raiz do curso e o código do curso ao qual é passada por referência e o novo nó. A função inserir disciplina não tem retorno.

Questão 1 – c

Essa questão é dividida em partes, que são implementações de funções, e essas funções podem ou não ter outras funções auxiliares, a baixo as funções pedidas:

1. Imprimir a árvore de cursos em ordem crescente pelo código do curso;
2. Imprimir os dados de um curso dado o código do mesmo;
3. Imprimir todos os cursos com a mesma quantidade de blocos, onde o usuário informe a quantidade de blocos;
4. Imprimir a árvore de disciplinas em ordem crescente pelo código das disciplinas dado o código do curso;
5. Imprimir os dados de uma disciplina dado o código dela e do curso ao qual ela pertence;
6. Imprimir as disciplinas de um determinado bloco de um curso, dado o bloco e o código do curso;
7. Imprimir todas as disciplinas de um determinado curso com a mesma carga horária, onde o código do curso e a carga horária devem ser informadas pelo usuário;

8. Excluir uma disciplina dado o código da disciplina e o código do curso;
9. Excluir um curso dado o código do mesmo, desde que não tenha nenhuma disciplina cadastrada para aquele curso.

Questão 1 – c – 1

Para imprimir a árvore de cursos em ordem crescente é utilizada a função chamada “imprimirCursos”, que tem como parâmetro a raiz da árvore de curso, essa função percorre a árvore de cursos em ordem. Dessa maneira a função irá imprimir todos os cursos inseridos na árvore com os seus respectivos dados e é importante destacar que a ordem de impressão vai depender do código do curso, pois ele funciona como um identificador único para cada curso.

Questão 1 – c – 2

Nessa questão temos a função “imprimirCursoPeloCodigo”, que tem como parâmetros a raiz da árvore de curso e o código do curso que é um número inteiro. Essa função vai verificar primeiramente se a raiz não é nula, não sendo, ocorrerá uma comparação do código da raiz com o código do curso que foi passado por parâmetro, se for igual, os dados do curso em questão serão imprimidos, se isso não acontecer ocorrerá outra verificação para ver se o código do curso é menor que o código da raiz, se for, a função é chamada recursivamente passando a raiz da esquerda da árvore curso como parâmetro e o código do curso e se o código do curso não for menor que o código da raiz, a função é chamada recursivamente passando a raiz da direita da árvore curso como parâmetro e o código do curso.

Questão 1 – c – 3

Na terceira parte da primeira questão da letra c é pedido uma função para imprimir todos os cursos com a mesma quantidade de blocos. Nessa função temos como parâmetro a raiz da árvore cursos e a quantidade de blocos que é um número inteiro, onde esse dado é informado pelo usuário. Essa função vai verificar primeiramente se a raiz não é nula, não sendo, ocorrerá uma comparação da quantidade de blocos da raiz com a quantidade de blocos do curso que foi passado por parâmetro, se for igual, os dados do curso em questão serão imprimidos. Se isso não acontecer a função é chamada recursivamente primeiramente passando a raiz da esquerda da árvore curso como parâmetro e a quantidade de blocos e logo em seguida a função é chamada recursivamente primeiramente passando a raiz da direita da árvore curso como parâmetro e a quantidade de blocos. Como isso toda a árvore vai sendo percorrida até o momento que a quantidade de blocos da raiz for a mesma que a quantidade de blocos que o usuário informou.

Questão 1 – c – 4

Na quarta parte é pedido uma função para imprimir a árvore de disciplinas em ordem crescente pelo código das disciplinas dado o código do curso. Nessa função temos como parâmetro a raiz da árvore cursos e o código do curso que é um número

inteiro, que será informado pelo usuário. A função vai percorrer a árvore até achar o código do curso que equivale ao valor passado ou até encontrar NULL. Dentro da função é declarado uma variável inteira chamada “encontrou” que é inicializada com zero. A função vai verificar primeiramente se a raiz não é nula, não sendo, ocorrerá uma verificação se o código do curso é menor que o código do curso da raiz, se for a variável encontrou recebe a chamada recursiva da função passando a raiz da esquerda como parâmetro e o código do curso informado pelo usuário. A próxima verificação serve para comparar se a raiz do código do curso é igual ao código do curso, se for igual a função “imprimirDisciplinas” onde a raiz da árvore disciplina é passado por parâmetro, os dados da disciplina serão imprimidos por ordem crescente em relação ao seu código e com isso a variável encontrou recebe 1. Depois disso ocorre a última verificação, para ver se o código do curso é maior que o código do curso da raiz, se for a variável encontrou recebe a chamada recursiva da função passando a raiz da direita como parâmetro e o código do curso informado pelo usuário. Ao final da função a variável encontrou é retornada.

Questão 1 – c – 5

Para solucionar o problema foi criado uma função para imprimir os dados de uma disciplina dado o código dela e do curso ao qual ela pertence. Nessa função temos como parâmetro a raiz da árvore cursos e o código do curso e o código da disciplina que são números inteiros informados pelo usuário. Se a raiz não estiver vazia é verificado se o código do curso da raiz é igual ao código do curso que o usuário informou se for, a função “procura Disciplina” é chamada, essa função vai auxiliar na impressão dos dados da disciplina, nessa função temos como parâmetro a raiz da árvore disciplina e o código da disciplina. Na função ocorre algo semelhante já apresentado aqui, se a raiz não for nula vai ser verificado se o código da disciplina raiz é igual ao código da disciplina, se for os dados dela serão impressos, caso contrário a função vai ser percorrida recursivamente para a esquerda e depois para direita. Voltando para a função principal, se os códigos não forem iguais essa função também será percorrida recursivamente primeiramente para a esquerda da raiz se o código do curso for menor que o código da raiz, se não a função será percorrida recursivamente passando a raiz da direita como parâmetro.

Questão 1 – c – 6

Para solucionar o problema foi criado uma função para imprimir as disciplinas de um determinado bloco de um curso, dado o bloco e o código do curso. Nessa função temos como parâmetro a raiz da árvore cursos e o código do curso e o bloco da disciplina que são números inteiros informados pelo usuário. Se a raiz não estiver vazia é verificado se o código do curso da raiz é igual ao código do curso que o usuário informou se for, a função “procuraDisciplina2” é chamada, essa função vai auxiliar na impressão dos dados das disciplinas, nessa função temos como parâmetro a raiz da árvore disciplina, o bloco da disciplina e o código da disciplina. Na função ocorre algo semelhante já apresentado aqui, se a raiz não for nula vai ser verificado se o bloco da disciplina raiz é igual ao bloco da disciplina, se for os dados dela serão impressos, caso contrário a função vai ser percorrida recursivamente para a esquerda e depois para direita. Voltando para a função principal, se os códigos não forem iguais essa função também será percorrida recursivamente primeiramente para a esquerda da raiz se o

código do curso for menor que o código da raiz, se não a função será percorrida recursivamente passando a raiz da direita como parâmetro.

Questão 1 – c – 7

Para solucionar o problema foi criado uma função para imprimir todas as disciplinas de um determinado curso com a mesma carga horária, onde o código do curso e a carga horária devem ser informadas pelo usuário. Nessa função temos como parâmetro a raiz da árvore cursos e o código do curso e a carga horária da disciplina que são números inteiros informados pelo usuário. Se a raiz não estiver vazia é verificado se o código do curso da raiz é igual ao código do curso que o usuário informou se for, a função “procuraDisciplina3” é chamada, essa função vai auxiliar na impressão dos dados das disciplinas, nessa função temos como parâmetro a raiz da árvore disciplina, carga horária da disciplina e o código da disciplina. Na função ocorre algo semelhante já apresentado aqui, se a raiz não for nula vai ser verificado se a carga horária da disciplina raiz é igual a carga horária da disciplina, se for os dados dela serão impressos, caso contrário a função vai ser percorrida recursivamente para a esquerda e depois para direita. Voltando para a função principal, se os códigos não forem iguais essa função também será percorrida recursivamente primeiramente para a esquerda da raiz se o código do curso for menor que o código da raiz, se não a função será percorrida recursivamente passando a raiz da direita como parâmetro.

Questão 1 – c – 8

Para solucionar o problema foi criado uma função para fazer a exclusão de uma disciplina, ela tem como parâmetros a raiz da árvore de cursos, código da disciplina e código do curso. Para remoção acontecer, ela busca o curso com o código correspondente na árvore e chama a função remover Disciplina para remover a disciplina com o código fornecido desse curso. A função remover disciplina tem como parâmetros a raiz da árvore disciplina que é passada como parâmetros e o código da disciplina, nessa função existem três casos principais:

Caso 1: O nó a ser removido não possui filhos. Nesse caso, o nó é simplesmente liberado da memória e o ponteiro para ele é definido como nulo.

Caso 2: O nó possui apenas um filho. O ponteiro para o nó atual é substituído pelo ponteiro para o filho existente e o nó original é liberado da memória.

Caso 3: O nó possui dois filhos. É necessário encontrar o nó folha mais à direita na subárvore esquerda. A função buscar folha é chamada para realizar essa busca. O nó folha encontrado substitui o nó original e o nó original é liberado da memória.

A função buscar folha tem como parâmetros a raiz da árvore disciplina que passada por referência e a raiz da árvore disciplina, essa função é usada para encontrar a folha mais à direita em uma subárvore. Essa função é chamada recursivamente até que o último nó à direita seja encontrado. O ponteiro passado por referência é atualizado para apontar para o novo nó folha, ou seja, o ponteiro que foi passado como parâmetro.

Questão 1 – c – 9

Para solucionar o problema foi criado uma função para excluir um curso, ela tem como parâmetros a raiz da árvore cursos passados por referência e o código do curso que o usuário deseja remover. Essa função possuí duas funções auxiliares a enderecoFilho e a buscarfolhaCurso. A função verifica se a raiz não é nula, se o código do curso na raiz corresponde ao código fornecido e se o curso não possui disciplinas. Em seguida, realiza a remoção do curso, considerando três casos: sem filhos, um filho ou dois filhos. Se o código fornecido for menor ou maior do que o código do curso na raiz, a função é chamada recursivamente para o filho esquerdo ou direito, respectivamente. A função enderecoFilho é usada no segundo caso da função principal quando o curso tem um filho, ela tem como parâmetro a raiz do curso, essa função retorna o endereço do filho da direita se existir, caso contrário, retorna o endereço do filho da esquerda de um nó raiz de um curso. A função buscarfolhaCurso é usada no terceiro caso da função principal quando o curso a ser removido possuí dois filhos. Ela tem como parâmetros a raiz da árvore curso passada por referência e a raiz da árvore da direita, essa função encontra a folha mais à direita em uma subárvore de cursos. A função atualiza o ponteiro filho1 que é passado por referência para apontar para a folha mais à direita encontrada (ou seja, filho2) que foi passado por parâmetro, através de uma busca recursiva.

Diferença na implementação da árvore binária para a árvore AVL

Essas soluções acima quase todas são usadas igualmente na árvore binária quanto na árvore com balanceamento (AVL). Variando apenas as estruturas das árvores, o que implica que no momento de inserção de um novo nó, na Árvore AVL, acontece o cálculo do fator de balanceamento, e a depender dele, a árvore poderá ser rotacionada para manter balanceada. Para isso ela possuí algumas funções auxiliares para isso. Na remoção também temos diferenças, pois quando a remoção do nó termina é verificado se ela está balanceada ou não, se ela estiver desbalanceada, vai ser feito o balanceamento da árvore. As funções a seguir estão relacionadas à inserção e balanceamento de nós em uma árvore AVL de disciplinas.

A função calclrAltrDis recebe como parâmetro um ponteiro para um nó de disciplina e retorna a altura desse nó. Se o nó for nulo, é retornado -1.

A função atlzrAltrDis serve para atualizar a altura do nó. Ela recebe como parâmetro um ponteiro para um nó de disciplina, calcula as alturas dos nós filhos. Em seguida, atribui a altura do nó como a maior altura entre os filhos, mais 1.

A função calclrFatrBalncmntDis calcula o fator de balanceamento de um nó de disciplina. Ela recebe um ponteiro para um nó e calcula a diferença entre as alturas do filho esquerdo e do filho direito. Se o nó for nulo, é retornado 0.

As funções rotDirtDis e rotEsqrdDis realizam rotações simples em uma árvore AVL de disciplinas. Elas recebem um ponteiro para um ponteiro de nó de disciplina, executam as rotações necessárias e atualizam as alturas dos nós envolvidos.

A função inserirDis é responsável por inserir um nó de disciplina na árvore AVL. Ela recebe como parâmetro a raiz da disciplina passada por referência e um ponteiro para o novo nó de disciplina. A função insere o nó na posição correta da árvore, de acordo com o código da disciplina. Em seguida, verifica o fator de balanceamento do nó

atual e realiza as rotações necessárias para manter a balanceamento da árvore. As rotações podem ser simples ou duplas, dependendo da situação. Após as rotações, as alturas dos nós são atualizadas.

As principais diferenças entre as inserções de cursos e disciplinas estão nas estruturas dos nós (usando as respectivas structs Curso e Disciplina) e nos critérios de ordenação (código do curso ou código da disciplina). No entanto, o processo de balanceamento e as rotações utilizadas são os mesmos para manter a propriedade AVL da árvore.

A diferença principal entre as funções de remover uma disciplina de uma árvore AVL e de uma árvore binária de busca está na manutenção do balanceamento da árvore. Pois na árvore AVL, além de realizar as mesmas operações de ajuste de ponteiros para remover o nó, é necessário garantir que a propriedade de balanceamento da árvore seja mantida.

A função remover Disciplina é responsável por remover um nó da árvore AVL. Após a remoção, ela verifica se a árvore está desbalanceada e realiza as rotações necessárias (rotação simples ou rotação dupla) para manter o balanceamento. Além disso, a função atlzrAltrDis é chamada para atualizar as alturas dos nós após as modificações.

A diferença na função que é usada para excluir um curso dado o código do mesmo na AVL é que ao final da remoção se o nó estiver desbalanceado, vai ser feito o balanceamento da árvore. A função de remoção (removerCurso) precisa levar em consideração o fator de balanceamento dos nós durante a remoção e se necessário realizar as rotações para manter a árvore balanceada. Isso é feito por meio da verificação do fator de balanceamento (valorBalan) e das rotações (rotDirt e rotEsqrd) dentro da função removerCurso da árvore AVL.

Os códigos de árvore binária e AVL possuem funções auxiliares que ajudam no processo, uma função para embaralhar vetor quando necessário que tem como parâmetro o vetor e o tamanho dele, uma função para sortear numero de um vetor que não possui parâmetros, essa função retorna um número sorteado e é útil na hora de fazer as inserções e uma função que lê o arquivo txt. Tem ainda um arquivo separado que cria arquivos txt que são úteis e necessários na hora de inserir milhares de nós nas árvores.

Resultados da Execução do Programa:

O projeto acima foi desenvolvido utilizando árvore binária e árvore AVL. A diferença principal entre árvores binárias e árvores AVL está no balanceamento, pois uma árvore binária comum não possui restrições de balanceamento, já uma árvore AVL é uma árvore binária balanceada. Em termos gerais, uma árvore binária comum tem uma velocidade de inserção mais rápida do que uma árvore AVL. A busca em uma árvore binária comum e em uma árvore AVL possui uma complexidade semelhante, sendo a busca eficiente em ambas as estruturas.

Com isso chegamos a um ponto importante do nosso trabalho, os testes para verificar os resultados do tempo de inserção e busca em ambas as árvores para que possamos comparar as diferenças de tempos entre elas. Para isso os testes foram desenvolvidos da seguinte maneira, no primeiro teste foram inseridos nas árvores 50000

mil cursos para o segundo teste a inserção de cursos aumentou para 75000 mil e outro teste com 100000 mil em ordem aleatória. As inserções e buscas foram repetidas 30 vezes e ao final foi calculado a média de tempo em ambos os casos.

Os testes de desempenho são importantes para validar e comparar as estruturas de dados, otimizar algoritmos, garantir a escalabilidade e a confiabilidade das implementações, além de auxiliar na tomada de decisões informadas sobre o uso das estruturas de dados em diferentes contextos. Os dados coletados estão disponíveis nas tabelas abaixo e servem para mostrar o desempenho de ambas as árvores ao serem expostas em diferentes cenários de inserções e busca. Para melhor entendimento das diferenças nas medias de tempos entre as inserções e buscas nas árvores, dois gráficos também estão disponíveis para análise.

Árvore	**Média de tempo da inserção**	**Média de tempo da busca**
Binária	84.88943	0.00021
AVL	99.18731	0.00011

Tabela 01: Experimento inserindo 50000 nós de cursos em ordem aleatória.

Com base nos resultados apresentados na tabela 01, pode-se concluir que a inserção na árvore binária tem um desempenho melhor do que na árvore AVL comparando a média de tempo das duas inserções. Enquanto a inserção da árvore binária tem uma média de tempo de aproximadamente 85 milissegundos o tempo de inserção da árvore AVL é de aproximadamente 100 milissegundos. Já na busca a média de tempo da AVL é superior ao da binária, uma diferença de 0,0001 milissegundos.

Árvore	**Média de tempo da inserção**	**Média de tempo da busca**
Binária	117.68570	0.00024
AVL	143.34492	0.00013

Tabela 02: Experimento inserindo 75000 nós de cursos em ordem aleatória.

Nesse segundo experimento o número de inserções na árvore de cursos foi aumentado em 25 mil, com base nos resultados apresentados na tabela 02, pode-se concluir também que a inserção na árvore binária tem um desempenho melhor do que na árvore AVL comparando a média de tempo das duas inserções, é importante se atentar que a média de tempo de inserção está crescendo. Agora ambas as inserções

ultrapassam os 100 milissegundos. Já na busca a média de tempo da AVL continua superior ao da binária.

Árvore	Média de tempo da inserção	Média de tempo da busca
Binária	154.74308	0.00029
AVL	182.91042	0.00016

Tabela 03: Experimento inserindo 100000 nós de cursos em ordem aleatória.

Para esse terceiro teste o número de inserções chegou a 100 mil, os tempos de inserção continuam a subir, a média de tempo da inserção na árvore binária continua com um desempenho melhor, os valores de busca estão maiores em relação aos experimentos passados, mesmo assim a busca na árvore AVL continua com melhor desempenho.

Árvore	Média de tempo da inserção	Média de tempo da busca
Binária	5.34088	0.00650
AVL	1.27594	0.00012

Tabela 04: Experimento inserindo 1000 nós de cursos ordenado.

Nesse experimento o número de inserções foi menor em relação aos outros, entretanto podemos perceber diferenças desse experimento para os outros, pois o txt de números que é usado para a criação dos códigos dos cursos estava ordenado. Nesse tipo de inserção o desempenho da árvore AVL se mostrou mais eficiente em ambas os casos, por isso a importância dos testes.

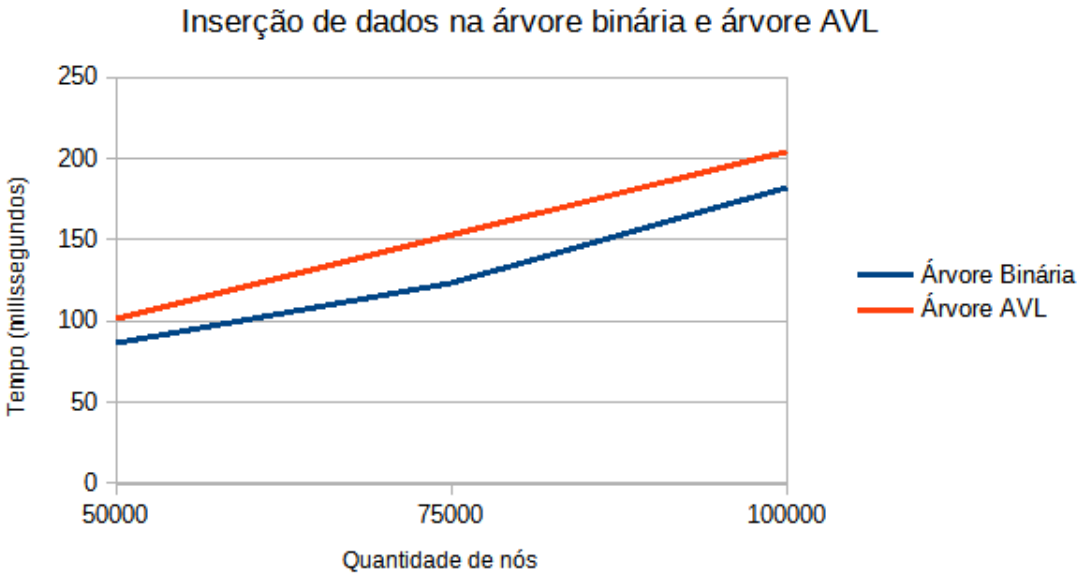

Figura 05 - Comparação do tempo de inserção das árvores binárias.

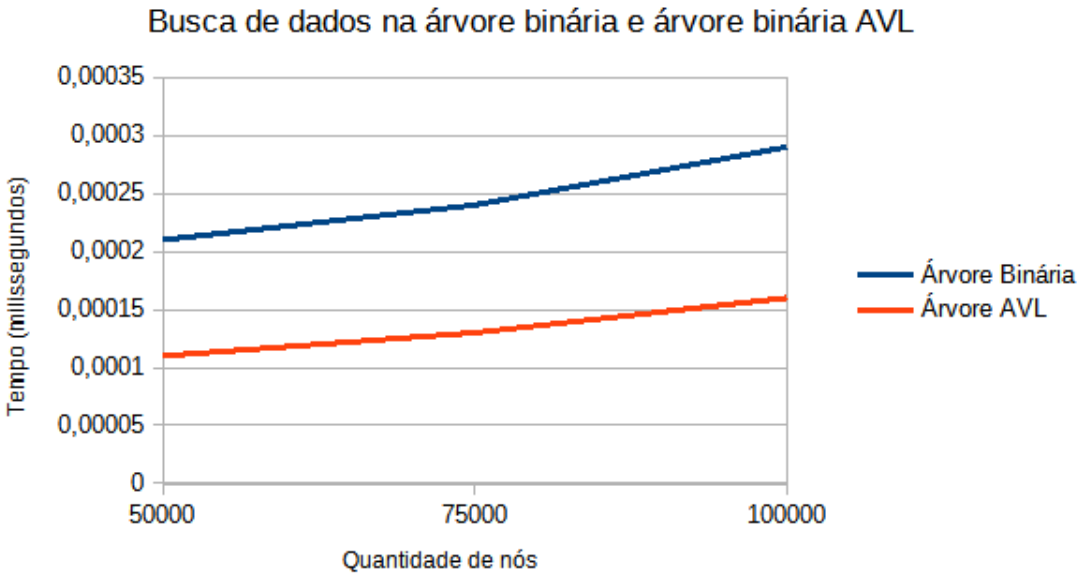

Figura 06 - Comparação do tempo de busca das árvores binárias.

A partir do gráfico da figura 05, é possível notar que o tempo de inserção na árvore binária AVL é maior do que na árvore sem balanceamento. Isso se dá pela necessidade de efetuar o balanceamento da árvore ao ponto que se insere um novo nó. Entretanto, na Figura 06, o tempo de busca da informação na árvore AVL é mais rápido do que na binária sem balanceamento. Isso ocorre devido ao balanceamento, que faz com que a árvore tenha a menor profundidade possível, fazendo com que leve menos tempo para que a busca chegue na parte mais profunda da árvore (nós folhas de maior profundidade), tornando a árvore AVL mais rápida para buscas.

Conclusão

Com base nos dados apresentados e nos conceitos discutidos, foi possível analisar as características e o desempenho das diferentes estruturas de árvores binárias, permitindo avaliar suas vantagens e desvantagens. O propósito do experimento foi comparar o desempenho das árvores binárias, tanto com quanto sem balanceamento, quando os valores estão ordenados ou não para inserção, e também comparar o tempo de busca das árvores. O proposito dos experimentos foi alcançado, podemos perceber que os resultados do experimento mostraram que a árvore binária teve um desempenho superior em termos de velocidade de inserção em comparação com a árvore AVL, enquanto que na busca a árvore AVL se mostrou superior. Com e execução do programa fazendo testes inserindo dados nas árvores com valores ordenados, foi descoberto que a árvore AVL foi mais eficiente nos processos de inserção e busca. Com a realização dos experimentos, aprendemos que a escolha da estrutura de árvore adequada depende do contexto específico.