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

Willians Silva Santos
willianssilva@ufpi.edu.br
Estruturas de Dados II - Juliana Oliveira de Carvalho

Resumo

Este relatório tem como principal fundamento a resolução das questões da primeira atividade avaliativa da disciplina de Estrutura de dados II, visando apresentar todos os aspectos necessários para o entendimento do problema, bem como sua solução. Todas as questões foram resolvidas com uso da linguagem de programação C, que alinhado à estrutura de dados das árvores binárias, sem balanceamento ou com o balanceamento, conhecida como AVL, pode proporcionar a criação de programas capazes de solucionar as questões.

Palavra-chave: Árvores Binárias, AVL, Estruturas de Dados.

Introdução

As árvores binárias são estruturas de dados que armazenam informações em forma hierárquica. Cada elemento da árvore é chamado de nó e cada nó pode ter, no máximo, dois filhos. A árvore é composta por um nó raiz, que é o nó principal, e pelos nós filhos, que podem ter outros nós filhos e assim por diante. Uma árvore AVL é uma árvore binária de pesquisa balanceada. Isso significa que ela mantém seus nós balanceados de maneira a garantir que a altura da árvore seja sempre a menor possível, o que resulta em operações mais rápidas.

A atividade propunha o desenvolvimento de um programa em C que gerencia o cadastro de cursos e disciplinas em uma árvore binária. O programa permite a inserção e manipulação desses dados, além de realizar consultas específicas, como imprimir os cursos em ordem crescente de código, exibir informações de um curso dado o código, listar cursos com a mesma quantidade de blocos, entre outras. A atividade também solicita a verificação dos tempos de inserção e busca na árvore de cursos, utilizando a função time do C para medir o tempo de execução. Por fim, é proposto repetir todo o processo utilizando uma Árvore AVL, uma variação da árvore binária que possui um mecanismo automático de balanceamento. Os tópicos seguintes deste trabalho estão divididos de modo a apresentar as estruturas das árvores, implementação das funções, os testes de eficiência entre a árvore binária sem balanceamento e com balanceamento (AVL) e conclusão dos resultados alcançados.

Hardware utilizado

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

Marca	Lenovo
Sistema Operacional	BigLinux

Processador	Intel i5 10° geração
Memória RAM	8GB
Tipo de memória	DDR4 2666MHz

Seções Específicas

Os tópicos seguintes apresentam as estruturas utilizada, escopos das funções e os testes de eficiência.

Estruturas de Dados Utilizadas

Foram utilizadas 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.

Implementação das funções da Árvore Binária

Adiante será apresentado a descrição do que cada função do programa realiza, quais seus parâmetros e o seu retorno.

- **criarNoCurso**: Essa função cria um nó do tipo curso. Ela possui os seguintes parâmetros, código do curso, nome do curso, quantidade de blocos que o curso possui e as semanas do curso. Por fim a função retorna o nó de curso criado.
- **inserirCurso**: Essa função percorre toda a árvore de cursos até encontrar onde o nó se encaixa e o insere. Ela possui como parâmetro a raiz e nó. A função não possui retorno.
  - **raiz**: esse é o ponteiro da árvore de curso o qual é passada por referência
  - **nó**: é o novo nó que será inserido na árvore.
- **existeCurso**: Essa função verifica se um curso existe. Ela possui como parâmetro a raiz e código. Por fim ela retorna o nó do curso se o encontra, ou nulo caso não encontre.
  - **raiz**: esse é o ponteiro da árvore de curso.
  - **código:** código do curso.
- **criarNoDisciplina**: Essa função cria o nó de disciplinas, ela possui os seguintes parâmetros, o código, nome, o bloco e a carga horária. Por fim a função retorna o nó de disciplina criado.
  - **código:** código da disciplina.
  - **nome:** nome da disciplina.
  - **bloco:** bloco da disciplina.
  - **carga horária:** carga horária da disciplina.

- **auxiliarInserirDisc**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista o endereço da árvore de disciplina é passado por referência para a função **inserirDisciplina**. Ela possui como parâmetro, raiz, código, nó. A função não possui retorno.
    - **raiz**: ponteiro do endereço da árvore de cursos o qual é passado por referência.
    - **código:** código do curso.
    - **nó**: é o novo nó que será inserido na árvore.
- **inserirDisciplina**: Essa função percorre toda a árvore de cursos até encontrar onde o nó se encaixa e o insere. Ela possui como parâmetro a raiz e nó. A função não possui retorno.
    - **raiz**: ponteiro do endereço da árvore de disciplina o qual é passado por referência.
    - **nó**: é o novo nó que será inserido na árvore.
- **imprimirCurso**: Essa função imprimi o conteúdo do nó da árvore de cursos. Ela possui como parâmetro nó. A função não possui retorno.
    - **nó**: este é nó atual da árvore.
- **imprimirCursos**: Essa função percorre toda a árvore de cursos em INORDER, e utiliza a função **imprimirCurso** para imprimir os dados do curso. Ela possui como parâmetro raiz. A função não possui retorno.
    - **raiz**: raiz é o ponteiro de curso;
- **imprimirDadosCurso**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista o nó atual do curso é enviado para a função **imprimirCurso** que no final irá imprimir os dados de um curso específico. Ela possui como parâmetro raiz e código. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **código:** código do curso.
- **imprimirCursosQtdB**: Essa função percorre toda a árvore de cursos e quando encontra os nós que possuem a mesma quantidade de bloco requerido, ela utiliza a função **imprimirCurso.** Ela possui como parâmetro raiz, bloco. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **bloco:** a quantidade de bloco requerido
- **ImprimirDisciplina:** Essa função imprimi o conteúdo do nó da árvore de disciplinas. Ela possui como parâmetro nó. A função não possui retorno.
    - **nó**: este é nó atual da árvore.
- **imprimirDisciplinas:** Essa função percorre toda a árvore de disciplinas em INORDER, e utiliza a função **imprimirDisciplinas** para imprimir os dados da disciplina. Ela possui como parâmetro raiz. A função não possui retorno.
    - **raiz**: raiz é o ponteiro de disciplina;
- **imprimirDisciplinaCod**: Essa função percorre a árvore de disciplinas até encontrar o nó que possuí o código informando, ao encontrar o nó é enviado para a função **imprimirDisciplina** que no final irá imprimir os dados da disciplina. Ela possui como parâmetro raiz e código. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de disciplinas.
    - **código:** código do disciplinas.
- **imprimirDisciplinasCurso**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista as disciplinas desse curso serão passada para a

função **imprimirDisciplinas**, que no final irá imprimir toda a árvore de disciplinas. Ela possui como parâmetro raiz e código. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **código:** código do curso.
- **imprimirDiscCurso**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista a árvore de disciplinas é passada para a função **imprimirDisciplinaCod** que no final irá imprimir os dados de uma disciplina específica. Ela possui como parâmetro raiz, códigoC e códigoD. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **códigoC:** código do curso.
    - **códigoD:** código da disciplina.
- **discBloco**: Essa função percorre toda a árvore de disciplinas verificado se a disciplina estar no bloco especificado, caso esteja, ela utiliza a função **imprimirDisciplina** para imprimir os dados das disciplinas. Ela possui como parâmetro raiz e bloco.
    - **raiz**: esse é o ponteiro da árvore de disciplinas.
    - **bloco:** o bloco especificado.
- **imprimirDiscBloco**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista a árvore de disciplinas é passada para a função **DiscBloco** que no final irá imprimir os dados de todas as disciplinas que estão no mesmo bloco. Ela possui como parâmetro raiz, código, bloco. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **código**: código do curso.
    - **bloco:** o bloco especificado.
- **cargaHorDis**: Essa função percorre toda a árvore de disciplinas verificado se a disciplina a carga horária especificado, caso esteja, ela utiliza a função **imprimirDisciplina** para imprimir os dados das disciplinas. Ela possui como parâmetro raiz e carga horária.
    - **raiz**: esse é o ponteiro da árvore de disciplinas.
    - **carga horária:** a carga horária especificada.
- **imprimirDiscCursoHorario**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista a árvore de disciplinas é passada para a função **cargaHorDis** que no final irá imprimir os dados de todas as disciplinas possuem a mesma carga horária. Ela possui como parâmetro raiz, código, carga horária. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **código**: código do curso.
    - **carga horária:** a carga horária especificada.
- **auxRemoverDisc**: Essa função utiliza a função **existeCurso** para verificar se um curso existe, caso exista a árvore de disciplinas é passada por referência para a função **removerDisc** que no final irá remover o nó de disciplina. Ela possui como parâmetro raiz, códigoC e códigoD. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de curso.
    - **códigoC:** código do curso.
    - **códigoD:** código da disciplina.

- **removerDisc**: Essa função tem como objetivo remover um nó da árvore de disciplina dado seu código. Ela possui como parâmetro raiz e código. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de disciplinas.
    - **códigoD:** código da disciplina.
- **folhaDis**: Essa função verifica se um nó da árvore de disciplina é folha. Ela possui como parâmetro somente o nó. Ela retorna 1 se o nó é folha e 0 se não for.
    - **nó**: nó atual da árvore de disciplinas.
- **enderecoFilhoDis**: Essa função verifica se o nó atual tem somente um filho. Ela possui como parâmetro somente o nó. A função retorna o filho caso tenha somente um ou nulo se tiver dois.
    - **nó**: nó atual da árvore de disciplinas.
- **maiorFilhoEsqDis**: Essa função tem como objetivo encontrar o maior filho à esquerda e fazer o seu ponteiro da direita apontar para o outro nó que foi enviado. Ela possui como parâmetro o filho da esquerda e filho da direita. Essa função não possui retorno.
    - **filho da esquerda**: filho da esquerda do nó que vai ser removido.
    - **filho da direta**: filho da direita do nó que vai ser removido.
- **removerCurso**: Essa função tem como objetivo remover um nó da árvore de cursos dado seu código. Ela possui como parâmetro raiz e código. A função não possui retorno.
    - **raiz**: esse é o ponteiro da árvore de cursos.
    - **código:** código do curso.
- **folha**: Essa função verifica se um nó da árvore de cursos é folha. Ela possui como parâmetro somente o nó. Ela retorna 1 se o nó é folha e 0 se não for.
    - **nó**: nó atual da árvore de cursos.
- **enderecoFilho**: Essa função verifica se o nó atual tem somente um filho. Ela possui como parâmetro somente o nó. A função retorna o filho caso tenha somente um ou nulo se tiver dois.
    - **nó**: nó atual da árvore de cursos.
- **maiorFilhoEsq**: Essa função tem como objetivo encontrar o maior filho à esquerda e fazer o seu ponteiro da direita apontar para o outro nó que foi enviado. Ela possui como parâmetro o filho da esquerda e filho da direita. Essa função não possui retorno.
    - **filho da esquerda**: filho da esquerda do nó que vai ser removido.
    - **filho da direta**: filho da direita do nó que vai ser removido.
- **alturaArvore:** Essa função tem como objetivo calcular a altura da árvore binária, ou seja, o número máximo de nós percorridos da raiz até uma de suas folhas. Ela possui como parâmetro somente o nó. A função retorna a altura da árvore.
    - **nó**: nó atual da árvore de cursos.

Implementação das funções da AVL

AVL é uma estrutura de dados do tipo árvore binária de busca que possui uma propriedade adicional de balanceamento. Por esse motivo não serão apresentadas as funções que já foram descritas anteriormente. A única diferença é que nas funções de inserir e remover um nó, serão implementadas a função de balanceamento e a de atualizar a altura do nó. Além disso, só serão apresentadas as funções da estrutura de curso, pois

essas funções possuem a mesma lógica, o que diferencia uma dá outra é o tipo de parâmetro, pois uma é do tipo curso e outra do tipo disciplina.

- **pegarAltura**: Essa função tem o objetivo de obter a altura do nó atual. Ela possui como parâmetro somente o nó. A função retorna a altura do nó caso ele seja diferente de nulo, se não retorna -1.
    - **nó**: nó atual da árvore de cursos.
- **atualizarAltura**: Essa função tem o objetivo de atualizar a altura do nó atual. Ela possui como parâmetro somente o nó. A função não possui retorno.
    - **nó**: nó atual da árvore de cursos.
- **fb**: Essa função indica o fator de balanceamento do nó, que é a diferença entre a altura do subárvore esquerda e a altura do subárvore direita. Ela possui como parâmetro somente o nó. A função retorna o fator de balanceamento do nó;
    - **nó**: nó atual da árvore de cursos.
- **rotacaoEsquerda**: Essa função realiza uma rotação simples à esquerda no nó especificado, ela é utilizada para reequilibrar uma árvore AVL quando ocorrem desequilíbrios. Ela possui como parâmetro somente o nó. A função não possui retorno.
    - **nó**: nó atual da árvore de cursos.
- **rotacaoDireita**: Essa função realiza uma rotação simples à direita no nó especificado, ela é utilizada para reequilibrar uma árvore AVL quando ocorrem desequilíbrios. Ela possui como parâmetro somente o nó. A função não possui retorno.
    - **nó**: nó atual da árvore de cursos.
- **balancear**: Essa função tem o objetivo de verificar se o nó atual da árvore precisa fazer rotação. Ela possui como parâmetro somente o nó. A função não possui retorno.
    - **nó**: nó atual da árvore de cursos.

Resultados da Execução do Programa

No tópico a seguir serão exibidos os resultados das comparações de tempo entre as operações de inserção e busca nas estruturas de dados: árvores binárias e AVL. O tempo de execução foi medido em milissegundos como unidade de medida.

Testes

Os testes foram feitos com 100.000 números indo de 1 a 100.000, cada teste foi repetido 30 vezes, para obter a média de tempo para inserção e busca. Os testes foram feitos com números aleatórios e ordenados na forma crescente. Os mesmo números aleatórios usados na árvore binária foram utilizados na AVL.

A tabela a seguir obtida a partir dos testes com números ordenados, no qual vai de 1 a 100.000

Teste 1: Números ordenados.

Árvore	Média de tempo para a inserção	Média de tempo para a busca
Binária	15557.197200000	0.000042010
AVL	0.001591014	0.000000020

Tabela 1. Números em ordem crescente.

Com base na tabela 1 é possível observar que com números ordenas a árvore AVL se sobressai a árvores binárias em questão de tempo de inserir e busca. Pois no momento que um nó é inserido é verificado se a árvore ficou desbalanceada, e com isso ela vai se balanceando o que faz que ela tenha um melhor desempenho do que uma árvore binária. No caso da árvore binária ela se torna uma lista encadeada, com isso ela gasta mais tempo para inserir e buscar um elemento na árvore.

Teste 2: Números desordenados.

Árvore	Média de tempo para a inserção	Média de tempo para a busca
Binária	18.158080000	0.000000011
AVL	0.002882437	0.000000008

Tabela 2. Números desordenados.

A tabela 2 demonstra que a árvore AVL teve o melhor tempo de inserção e de busca, enquanto a árvore binária teve o pior caso na inserção e busca.

Conclusão

Através dos experimentos, foi possível analisar as características e desempenho das diferentes estruturas de árvores binárias, permitindo avaliar suas vantagens e desvantagens, e assim, escolher a estrutura mais adequada para aplicações específicas. A análise dos resultados obtidos permitiu concluir que as árvores binárias são uma ferramenta valiosa para a organização e busca de dados. Os dados nos permitem concluir que a árvore AVL é mais eficiente do que a árvore binária sem balanceamento, devido ao seu alto grau de balanceamento, além de ser menos propensa a se tornar desequilibrada.