UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS – PICOS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

VANDIRLEYA BARBOSA DA COSTA

RELATÓRIO DE ESTRUTURAS DE DADOS

Picos – PI, 2023.

RELATÓRIO DE ESTRUTURAS DE DADOS - ÁRVORE BINÁRIA E ÁRVORE AVL

Relatório com requisito para aprovação e apresentado à disciplina de Estruturas de Dados II, no curso Bacharelado em Sistemas de Informação pela Universidade Federal do Piauí, ministrada pela Prof. Juliana Carvalho no semestre letivo 2023.1.

Picos – PI, 2023.

Resumo

Este trabalho aborda as árvores de busca binária (ABB) e as árvores AVL como estruturas utilizadas para resolver problemas de inserção, busca e remoção de cursos e disciplinas. Serão apresentados os resultados do tempo de inserção e busca obtidos por meio de análises comparativas, visando identificar a melhor performance entre as duas abordagens.

1. Introdução

No campo da computação, as árvores desempenham um papel crucial como uma estrutura de dados essencial, oferecendo eficiência no armazenamento e organização de informações. Essas estruturas são compostas por nós que contêm valores e ponteiros para outros nós, conhecidos como filhos. As árvores são classificadas de acordo com suas características e propriedades, e cada tipo apresenta suas próprias vantagens e desvantagens, sendo aplicadas em diferentes contextos e algoritmos.

O objetivo deste trabalho é apresentar o funcionamento de dois tipos fundamentais de árvores, discutindo suas características e desempenho, para auxiliar na seleção adequada de cada um deles. Ao enfrentar um problema, é crucial analisar suas particularidades, como a quantidade de informações envolvidas e se a ordenação ou distribuição dos dados é necessária. Através dessa análise criteriosa, torna-se possível otimizar tanto o hardware quanto o software, alcançando soluções mais eficientes para os problemas em questão. A execução dos algoritmos foi realizada em um notebook Samsung com as seguintes especificações:

Modelo	Lenovo 82MF
Processador	AMD Ryzen 5 5500U with Radeon Graphics (12 GPUs), ~2.1GHz
Memória RAM	8.192MB RAM
Placa de vídeo	Vega 7
Sistema Operacional	Pop!_OS

2. Problema Apresentado

O problema proposto consiste em desenvolver um programa em linguagem C que realiza o cadastro e manipulação de dados utilizando uma árvore binária e

posteriormente uma árvore AVL. O programa deve ser capaz de cadastrar informações de cursos e disciplinas, organizando-os de acordo com suas especificações. Para os cursos, devem ser cadastrados o código, o nome do curso, a quantidade de blocos e o número de semanas para cada disciplina. Além disso, cada curso terá uma árvore de disciplinas associada a ele. Para as disciplinas, devem ser cadastrados o código da disciplina, o nome, o bloco (que deve ser menor que a quantidade de blocos do curso) e a carga horária (que deve ser um número múltiplo do número de semanas para cada disciplina no curso).

O programa oferecerá funcionalidades como impressão em ordem crescente pelo código do curso, impressão dos dados de um curso específico, impressão de cursos com a mesma quantidade de blocos informada pelo usuário, impressão em ordem crescente pelo código das disciplinas em um determinado curso, impressão dos dados de uma disciplina específica informando seu código e o código do curso ao qual pertence, impressão das disciplinas de um determinado bloco de um curso fornecendo o bloco e o código do curso, impressão de todas as disciplinas de um curso com a mesma carga horária informada pelo usuário. Além disso, o programa realizará medições de tempo de execução para duas operações principais: o tempo de inserção de cada elemento na árvore de cursos e o tempo de busca de um curso específico na árvore de cursos. Essas medições serão realizadas utilizando a função "time" da linguagem C, obtendo o tempo inicial antes de iniciar a operação e o tempo final ao finalizá-la.

3. Seções Específicas

Para o melhor entendimento das seções a seguir, é importante conhecer as estruturas de dados utilizadas na resolução do problema. A seguir, uma breve apresentação sobre os dois tipos de árvores:

3.1 Árvore Binária de Busca (ABB)

Uma árvore de busca binária é uma estrutura de dados em que cada nó possui no máximo dois filhos: um filho à esquerda e um filho à direita. Essa estrutura segue uma regra específica, na qual todos os valores menores que o valor do nó atual são armazenados no filho esquerdo, e todos os valores maiores são armazenados no filho direito. Essa organização facilita a busca de um valor específico na árvore de forma eficiente, reduzindo o número de comparações necessárias.

Além disso, a árvore de busca binária possui uma propriedade adicional: para qualquer nó, todos os valores na subárvore esquerda são menores que o valor do nó, e todos os valores na subárvore direita são maiores. Essa propriedade permite realizar operações como inserção, remoção e busca em tempo médio de complexidade O(log n), tornando-a adequada para armazenar e recuperar informações ordenadas de forma rápida.

3.2 Árvore Binária AVL

Uma árvore AVL é uma estrutura de dados em forma de árvore binária de busca balanceada. Ela é chamada de AVL em homenagem aos seus inventores, Adelson-Velskii e Landis. A principal característica de uma árvore AVL é que ela mantém o equilíbrio entre suas subárvores. Isso significa que a diferença de altura entre as subárvores esquerda e direita de cada nó é no máximo 1.

Esse equilíbrio é alcançado por meio de rotações e reorganizações automáticas dos nós da árvore, sempre que uma operação de inserção ou remoção pode comprometer o balanceamento. O objetivo do balanceamento é evitar que a árvore se torne desequilibrada e perca sua eficiência. Com uma árvore AVL, as operações de busca, inserção e remoção podem ser realizadas em tempo médio de complexidade O(log n), onde "n" é o número de nós na árvore.

4. Estrutura

Esta seção apresenta a estrutura e organização das funções utilizadas nos códigos-fonte da Árvore Binária de Busca e da Árvore AVL para resolver o problema proposto. A fim de facilitar o entendimento, são especificados os parâmetros e as funcionalidades das funções presentes em cada tipo de árvore.

4.1 Árvore Binária de Busca

A primeira funcionalidade a ser criada na árvore binaria para a resolução do problema foram duas estruturas, uma para o curso e outra para a disciplina.

1. *Estrutura de disciplina:* esta estrutura contem os campos para o código da disciplina, nome da disciplina, bloco da disciplina(o qual deve ser menor que a quantidade de blocos do curso), a carga horária da disciplina(deve ser um número múltiplo do número de semanas para cada disciplina no curso). A árvore deve estar organizada pelo código da disciplina, o mesmo não deve se repetir
2. *Estrutura de curso:* esta estrutura contem os campos para o código do curso, nome do curso, quantidade de blocos, número de semanas para cada disciplina e o endereço para a árvore de disciplinas daquele curso.

Após isso as funções utilizadas estão organizadas da seguinte forma:

//Disciplinas

1. *inserirDadosDisc:* Esta função recebe um código de disciplina (codDisc), um nome de disciplina (nome[]), um número de bloco (blocoDisc) e uma carga horária (carga). Depois cria e preenche uma estrutura de disciplina (Disciplina) com os dados fornecidos. Por último retorna um ponteiro para a estrutura de disciplina criada.

2. *verificarCodDisc:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc) e um código de disciplina (codDisc). Verifica se o código de disciplina já existe na árvore. E retorna um valor inteiro (0 ou 1) indicando se o código de disciplina foi encontrado (1) ou não (0).
3. *inserirDisciplina:* Esta função recebe um ponteiro para um ponteiro da raiz da árvore de disciplinas (raizArvDisc) e um ponteiro para uma disciplina a ser inserida (novaDisc). Insere a disciplina na árvore de disciplinas.
4. *imprimirDadoDisc:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc) e imprime os dados de todas as disciplinas presentes na árvore de disciplinas.
5. *imprimirArvDiscCurso:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso) e um código de curso (codCurso). Imprime os dados de todas as disciplinas associadas a um determinado curso. E retorna um valor inteiro indicando se o curso foi encontrado (1) ou não (0).
6. *buscarDisc:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc) e um código de disciplina (codDisc). Busca uma disciplina na árvore de disciplinas pelo código e imprime seus dados.
7. *imprimirDadosDisc:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso), um código de curso (codCurso) e um código de disciplina (codDisc). Imprime os dados de uma disciplina específica associada a um curso específico.
8. *buscarDiscBlocos:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc), um número de bloco (blocoDisc) e um código de disciplina (codDisc). Busca e imprime as disciplinas que pertencem a um determinado bloco.
9. *imprimirDisciplinasBloco:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso), um código de curso (codCurso) e um número de bloco (blocoDisc). Imprime as disciplinas de um determinado bloco em um curso específico.
10. *buscarDiscCargaH:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc) e uma carga horária (carga). Busca e imprime as disciplinas que possuem uma determinada carga horária.
11. *imprimirDisCargaHoraria:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso), um código de curso (codCurso) e uma carga horária (carga). Imprime as disciplinas de uma determinada carga horária em um curso.
12. *buscarFolhaDisc:* Esta função recebe um ponteiro para a raiz da árvore de disciplinas (raizArvDisc). Busca a última folha à direita da árvore de disciplinas. Retorna o ponteiro para a última folha à direita da árvore de disciplinas.
13. *removerDisciplina:* Esta função recebe um ponteiro para um ponteiro da raiz da árvore de disciplinas (raizArvDisc) e um código de disciplina (codDisc). Remove uma disciplina da árvore de disciplinas pelo seu código.

14. *excluirDisc:* Esta função recebe um ponteiro para um ponteiro da raiz da árvore de cursos (raizArvCurso), um código de curso (codCurso) e um código de disciplina (codDisc). Exclui uma disciplina de um curso específico na árvore de cursos, dado o código do curso e o código da disciplina.

//Curso

15. *inserirDadosCurso:* Esta função recebe um código de curso (codCurso), um nome de curso (nome[]), um número de blocos (numBlocos) e um semestre de início (semestre). Cria e preenche uma estrutura de curso (Curso) com os dados fornecidos. Retorna um ponteiro para a estrutura de curso criada.
16. *inserirCurso:* ponteiro para um ponteiro da raiz da árvore de cursos (raizArvCurso) e um ponteiro para um curso a ser inserido (novoCurso). Insere o curso na árvore de cursos.
17. *buscarCurso:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso) e um código de curso (codCurso). Busca um curso na árvore de cursos pelo seu código e retorna o ponteiro para o curso encontrado. Retorna um ponteiro para a estrutura de curso encontrada ou NULL se o curso não for encontrado.
18. *imprimirDadosCurso:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso) e um código de curso (codCurso). Imprime os dados de um curso específico.
19. *imprimirCursosBlocosIguais:* Esta função recebe um ponteiro para a raiz da árvore de cursos (raizArvCurso) e a quantidade de blocos desejada (qtdBlocos). Imprime os dados dos cursos que possuem a quantidade de blocos igual a qtdBlocos.
20. *buscarFolhaCurso:* Esta função recebe um ponteiro para um ponteiro que aponta para o último curso (ultimo) e um ponteiro para o curso filho (filho). Busca a última folha à direita da árvore de cursos e atualiza o ponteiro "ultimo" para apontar para ela.
21. *excluirCurso:* Esta função recebe um ponteiro para um ponteiro da raiz da árvore de cursos (raizArvCurso) e um código de curso (codCurso). Remove um curso da árvore de cursos pelo seu código, levando em consideração diferentes casos (sem filhos, um filho ou dois filhos).

4.2 Árvore AVL

Para a resolução do problema com a AVL, foram reutilizadas algumas funções e estruturas do curso e disciplina. Esta última teve o campo "altura" adicionado para ambas as estruturas. Além disso, algumas funções foram adicionadas para auxiliar na inserção e exclusão dos cursos. As alterações serão explicadas a seguir.

//Disciplinas

1. *Rotacao_esq_disciplina:* Essa função recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc). Ela realiza uma rotação para a esquerda na árvore de disciplinas.

2. *Rotacao_dir_disciplina*: Recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc). Realiza uma rotação para a direita na árvore de disciplinas.*Altura_no_disciplina:* Recebe um ponteiro para uma Disciplina (Disciplina *disciplina). Retorna a altura do nó da disciplina.
3. *Fb_disc:* Recebe um ponteiro para uma Disciplina (Disciplina *disciplina). Calcula e retorna o fator de balanceamento (FB) da árvore de disciplinas.
4. *Atualiza_altura_disc:* Recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc). Atualiza a altura da árvore de disciplinas.
5. *Calcula_altura_disc:* Recebe um ponteiro para uma Disciplina (Disciplina *raizDisc). Calcula a altura da árvore de disciplinas e retorna o valor.
6. *Balanceia_disc:* Recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc). Realiza as rotações necessárias para balancear a árvore de disciplinas.
7. *Insere_disc*: Recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc) e um ponteiro para uma Disciplina (Disciplina *Novo). Insere a nova disciplina na árvore de disciplinas.
8. *Eh_folha_disc*: Recebe um ponteiro para uma Disciplina (Disciplina *raizDisc). Verifica se a disciplina é uma folha (não tem filhos). Retorna 1 se for folha, 0 caso contrário.
9. *Dois_filhos_disc:* Recebe um ponteiro para uma Disciplina (Disciplina *raizDisc). Verifica se a disciplina tem dois filhos. Retorna 1 se tiver dois filhos, 0 caso contrário.
10. *Maior_esq_disc*: Recebe um ponteiro para uma Disciplina (Disciplina *raizDisc). Retorna o nó com o maior valor à esquerda da árvore de disciplinas.
11. *Disc_filho:* Recebe um ponteiro para uma Disciplina (Disciplina *raizDisc). Retorna o filho da disciplina (esquerda ou direita).
12. *Excluir_disc_Aux:* Recebe um ponteiro para um ponteiro de Disciplina (Disciplina **raizDisc), um código de disciplina (int CodDisciplina) e um ponteiro para uma variável removeu (int *removeu). Realiza a remoção da disciplina com o código especificado na árvore de disciplinas.
13. *Excluir_disc:* Recebe um ponteiro para um ponteiro de Curso (Curso **raizCursos), um código de curso (int CodCurso) e um código de disciplina (int CodDisciplina). Realiza a remoção da disciplina com o código especificado na árvore de disciplinas de um determinado curso.

// **Cursos**

1. *Rotacao_esq_curso:* Esta função recebe um ponteiro para um ponteiro de Curso raizCursos. Ela realiza uma rotação para a esquerda na árvore de cursos. A rotação esquerda envolve a atualização dos ponteiros dos nós para manter a estrutura da árvore.
2. *Rotacao_dir_curso:* Essa função recebe um ponteiro para um ponteiro de Curso raizCursos. Ela realiza uma rotação para a direita na árvore de cursos.

A rotação direita envolve a atualização dos ponteiros dos nós para manter a estrutura da árvore.

3. *Altura_no_curso:* Essa função recebe um ponteiro para um curso. Ela retorna a altura do nó do curso. A altura é um valor inteiro que representa a distância entre o nó e a folha mais distante na subárvore abaixo dele.
4. *Fb_curso:* Essa função recebe um ponteiro para um curso. Ela calcula e retorna o fator de balanceamento (FB) do nó da árvore de cursos. O fator de balanceamento é a diferença entre as alturas das subárvores esquerda e direita de um nó. Um FB positivo indica que a subárvore esquerda é mais alta, enquanto um FB negativo indica que a subárvore direita é mais alta.
5. *Atualiza_altura_curso:* Esta função recebe um ponteiro para um ponteiro de Curso raizCursos. Ela atualiza a altura da árvore de cursos. A função chama a função Calcula_altura_curso para calcular a altura do nó da raiz e atualiza o valor da altura no campo correspondente da estrutura do Curso.
6. *Calcula_altura_curso*: Essa função recebe um ponteiro para um Curso raizCursos. Ela calcula a altura da árvore de cursos. A função utiliza uma abordagem recursiva para calcular a altura. Se a raiz for nula, a altura é definida como -1. Caso contrário, a função chama a si mesma para calcular as alturas das subárvores esquerda e direita e retorna a maior delas mais 1.
7. *Balanceia_cursos:* Esta função recebe um ponteiro para um ponteiro de Curso raizCursos. Ela verifica o fator de balanceamento do nó raiz da árvore de cursos e realiza as rotações necessárias para balancear a árvore, conforme necessário. Se o FB for igual a -2, é feita uma rotação para a esquerda ou uma rotação dupla esquerda-direita, dependendo do FB do filho direito. Se o FB for igual a 2, é feita uma rotação para a direita ou uma rotação dupla direita-esquerda, dependendo do FB do filho esquerdo. A função não retorna nenhum valor.
8. *InsereCurso:* Esta função recebe um ponteiro para um ponteiro de Curso raizCursos e um ponteiro para um Curso Novo. Ela insere um novo curso na árvore de cursos, mantendo a propriedade da árvore de busca binária. A função realiza a inserção recursivamente com base no valor do código do curso. Após a inserção, chama as funções Balanceia_cursos e Atualiza_altura_curso para garantir que a árvore esteja balanceada e atualizada em termos de altura.
9. *Eh_folha_curso:* Esta função recebe um ponteiro para um Curso raizCursos. Ela verifica se o nó da árvore de cursos é uma folha, ou seja, se não possui filhos. Retorna 1 se for uma folha e 0 caso contrário.
10. *Dois_filhos_curso*: Esta função recebe um ponteiro para um Curso raizCursos. Ela verifica se o nó da árvore de cursos possui dois filhos. Retorna 1 se tiver dois filhos e 0 caso contrário.
11. *Maior_esq_curso:* Essa função recebe um ponteiro para um Curso raizCursos. Ela encontra e retorna o maior elemento na subárvore esquerda do nó especificado.

12. *Curso_filho:* Essa função recebe um ponteiro para um Curso raizCursos. Ela retorna o filho do curso na árvore. Se o nó possuir um filho à esquerda, ele é retornado. Caso contrário, o filho à direita é retornado.
13. *Excluir_curso:* Esta função recebe um ponteiro para um ponteiro de Curso raizCursos e um inteiro CodCurso. Realiza a remoção do curso com o código especificado na árvore de cursos. A função primeiro verifica se o curso existe na árvore. Se existir, verifica se o curso é uma folha, um nó com dois filhos ou um nó com apenas um filho. Com base na situação, realiza as operações apropriadas para excluir o curso da árvore. Após a remoção, chama as funções Balanceia_cursos e Atualiza_altura_curso para garantir que a árvore esteja balanceada e atualizada em termos de altura.

5. Resultados da execução dos programas

Nessa seção serão apresentados os resultados obtidos nas comparações de tempo para a operação de inserção e para a operação de busca nas árvores binária de busca e AVL.

5.1 Inserção - Árvore Binária de Busca X Árvore AVL.

Para medir o tempo de inserção em ambas as árvores, foram realizados testes variando a ordem de entrada. Cada teste de inserção foi medido em milissegundos, sendo realizados 30 testes com 50 mil elementos únicos em cada um. Dessa forma, foi possível obter um tempo médio de inserção com mais precisão.

A análise dos tempos médios de inserção demonstrou que a árvore ABB apresentou um desempenho superior à árvore AVL em todos os casos. Isso é plausível, uma vez que, para cada inserção na árvore AVL, é realizada uma verificação para determinar se está balanceada ou não, o que demanda mais tempo devido ao uso de chamadas recursivas. Além disso, é válido ressaltar que o tempo médio de inserção na ordem decrescente é maior do que na ordem crescente em ambas as árvores. Enquanto o tempo médio de inserção na ordem aleatória na árvore ABB é pequeno, na AVL é alto.

Tabela 1: Tempo médio de inserção.

Árvore	Ordem Crescente	Ordem Decrescente	Ordem Aleatória
ABB	0,014867 (ms)	0,022200 (ms)	0,000100 (ms)
AVL	0,385462 (ms)	0,424259 (ms)	0,911999 (ms)

5.2 Busca - Árvore Binária de Busca X Árvore AVL.

A realização dos testes de busca seguiu o mesmo padrão dos testes de inserção. No entanto, ao contrário da inserção, onde a árvore ABB apresenta os

melhores resultados, na busca a AVL apresenta um desempenho melhor. Isso ocorre porque, à medida que elementos são inseridos ou removidos, o balanceamento é realizado, reduzindo a quantidade de níveis na árvore e proporcionando uma distribuição mais equilibrada. Isso torna a busca por um elemento mais eficiente.

Para obter os tempos nesse cenário de busca, foram realizadas pesquisas no pior caso possível em termos de ordens crescente e decrescente, com os últimos elementos inseridos. No caso da ordem aleatória, a busca foi realizada procurando uma determinada quantidade de números aleatórios em ambas as árvores.

Tabela 1: Tempo médio de busca.

Árvore	Ordem Crescente	Ordem Decrescente	Ordem Aleatória
ABB	0,451067 (ms)	0,493667 (ms)	0,000333 (ms)
AVL	0 (ms)	0 (ms)	0 (ms)

6. Conclusão

Com base nos testes realizados para medir o tempo de inserção e busca nas árvores ABB e AVL, podemos concluir que cada estrutura apresenta um desempenho superior em diferentes cenários. No caso da inserção, a árvore ABB demonstrou um desempenho superior em relação à AVL. Isso pode ser atribuído ao fato de que a AVL requer verificações adicionais para garantir o balanceamento da árvore a cada inserção, o que demanda mais tempo. No entanto, na busca, a árvore AVL apresentou um desempenho superior. Isso ocorre porque, devido ao balanceamento realizado após cada inserção ou remoção, a árvore AVL mantém uma distribuição mais equilibrada e reduz a quantidade de níveis, tornando a busca por um elemento mais eficiente.

Portanto, é possível concluir que a escolha da estrutura de árvore (ABB ou AVL) deve levar em consideração o tipo de operação predominante no contexto do sistema. Se as inserções forem mais frequentes e a busca for menos intensiva, a ABB pode ser a melhor opção devido ao seu desempenho superior na inserção. Por outro lado, se a busca por elementos for a operação crítica, a AVL se destaca pela sua eficiência na distribuição e busca balanceada dos elementos.