Árvore Binária e Árvore AVL

Hector José Rodrigues Salgueiros
email: hectorsalg@ufpi.edu.br

Junho 2023

1 Resumo do projeto

O projeto consiste em criar uma estrutura de dados baseada em árvores binárias, com a árvore de disciplinas dentro da árvore de cursos. Serão implementadas funções de inserção, impressão e remoção de nós, além de realizar testes de inserção e busca. Posteriormente, a estrutura será aprimorada com a implementação da Árvore AVL, garantindo melhor desempenho. O projeto tem como objetivo organizar e manipular cursos e disciplinas de forma eficiente, seguindo a lógica das árvores binárias.

2 Introdução

Relatório destinado a explicação da árvore binária e árvore AVL, como funcionam, a forma de implementar cada uma das árvores, problemática, funções na linguaguem C destinadas para solução do problema em questão, lógica por trás das funções, além de comparar a eficiência dos códigos e árvores em determinadas condições.

3 Árvores

3.1 Árvore Binária

Uma árvore binária para programação é definida em um nó, sendo raiz da árvore, ou não. Sua estrutura é definida por: Informações da estrutura, Nó para esquerda e Nó para a direita, tendo assim, no máximo dois filhos, direcionando valores onde o filho da esquerda tem informação númerica inferior a informação númerica do nó pai, além do filho da esquerda do nó que possui informação númerica maior que o nó pai. Em situações onde o nó não possui filhos é chamado de folha.

3.2 Árvore AVL

Uma árvore AVL contém as mesmas situações das explicadas acima, com o porém de possuir valores a mais na sua estrutura, como: Altura e Fator de balanceamento. A Altura é utilizada para calcular o Fator de balanceamento da Árvore. Com a seguinte fórmula: FB =(Altura

do filho da esquerda) − (Altura do filho da direita). A Altura é definida pelo acréscimo de +1 ao filho de maior altura, sendo definido por: Folha possui $altura = 0$ e um Filho inexistente ou Filho nulo possui $altura = -1$.

4 Problemática

Faça a implementação de uma árvore binária e AVL de cursos e disciplinas de uma universidade, o curso possuindo Código, Nome, Quantidade de Blocos, Número de semanas para cada disciplina e uma árvore de disciplinas, na árvore de disciplinas possuindo Código da disciplina, Nome, bloco da disciplina e carga horária.

5 Funções

5.1 Funções da Árvore Binária

5.1.1 criarCurso

Uma função do tipo *Struct Curso* que retorna NULL, serve para inicialização da árvore.

5.1.2 criarNoCurso

Uma função do tipo *Struct Curso* que retorna um ponteiro do tipo *Struct Curso*, inicializado pelos parâmetros da função sendo o *codC* do tipo inteiro, destinado ao código do curso, *nome* um vetor de caracteres do tipo char, *qtdBCurso* do tipo inteiro para definir a quantidade de blocos do curso, e *semana* do tipo inteiro para definir a quantidade de semanas do curso. Reservando memória para todo o novo nó e atribuindo os valores em seus devidos lugares.

5.1.3 inserirCurso

Uma função do tipo *void*(vazio), ou seja, não possui retorno. Seus parâmetros são a *raiz* do tipo *Struct Curso* passado por referência e *no* também do tipo *Struct Curso*. Essa função verifica o *codC* do *no* e inserer ou percorre a árvore com recursividade até que se encontre NULL.

5.1.4 existeCurso

Uma função do tipo *Struct Curso* que retorna um ponteiro do tipo *Struct Curso*. Tem parâmetros como *raiz* do tipo *Struct Curso* e *codC* do tipo inteiro para procurar o curso. Percorre a árvore de curso com revursividade até que encontre e retorne o valor pesquisado, caso não encontre é retornado NULL.

5.1.5 criarNoDisciplina

Uma função auxiliar do tipo *Struct Disciplina* que retorna um ponteiro do tipo *Struct Disciplina*, inicializado pelos parâmetros da função sendo o *codD* do tipo inteiro, destinado ao código da disciplina, *nome* um vetor de caracteres do tipo char, *bloco* do tipo inteiro para

definir qual o bloco, e *cargHor* do tipo inteiro para definir a quantidade de horas da disciplina. Reservando memória para todo o novo nó e atribuindo os valores em seus devidos lugares.

5.1.6 inserirDisciplina

Uma função do tipo *void* que tem como parâmetros *raiz* do tipo *Struct Disciplina* passada por referência e *no* do tipo *Struct Disciplina*. Essa função verifica o *codD* do *no* e inserer ou percorre a árvore com recursividade até que se encontre NULL.

5.1.7 auxiliarInserirDisc

Uma função auxiliar do tipo *void* que insere a disciplina em determinado curso, recebendo os parâmetros *Curso* do tipo *Struct Curso* passado por referência, codC do tipo inteiro para pesquisar o código do curso e *no* do tipo *Struct Disciplina*, utiliza a função auxiliar *existeCurso* para acessar o curso que desejo atribuir a disciplina em seguida é realizado a chamada de função recursiva *inserirDisciplina*.

5.1.8 imprimirCurso

Uma função do tipo *void* para imprimir um nó da arvore, recebe como parâmetro *no* do tipo *Struct Curso*.

5.1.9 imprimirCursos

Uma função do tipo *void* para imprimir todos os cursos da árvore em ordem de forma recursiva pelo parâmetro *raiz* do tipo *Struct Curso*.

5.1.10 imprimirDadosCurso

Uma função do tipo *void* que recebe os parâmetros *raiz* do tipo *Struct Curso* e *cod* do tipo inteiro para pesquisar o código específico do curso, utilizando a função auxiliar *existeCurso* para imprimir o nó especifico da árvore.

5.1.11 imprimirCursosQtdB

Uma função do tipo *void* que recebe os parâmetros *raiz* do tipo *Struct Curso* e *qtdB* do tipo inteiro para pesquisar os cursos com a mesma quantidade de blocos, imprime ordenado de forma recursiva todos os cursos que possuem a quantidade de bloco desejada.

5.1.12 imprimirDisciplina

Uma função do tipo *void* para imprimir um nó da arvore, recebe como parâmetro *no* do tipo *Struct Disciplina*.

5.1.13 imprimirDisciplinas

Uma função do tipo *void* para imprimir todos os cursos da árvore em ordem de forma recursiva pelo parâmetro *raiz* do tipo *Struct Disciplina*.

5.1.14 imprimirDisciplinaCod

Uma função do tipo *void* que recebe os parâmetros *raiz* do tipo *Struct Disciplina* e *codD* do tipo inteiro para pesquisar o código específico da disciplina, imprime a disciplina pelo código.

5.1.15 imprimirDisciplinasCurso

Uma função do tipo *void* que recebe os parâmetros *raiz* do tipo *Struct Curso* e *codC* do tipo inteiro para pesquisar o código do curso. Utiliza-se a função auxiliar *existeCurso* para retornar o nó desejado e imprime a árvore de disciplinas do curso específico.

5.1.16 imprimirDiscCurso

Uma função do tipo *void* que recebe os parâmetros *raiz* do tipo *Struct Curso*, *codC* do tipo inteiro para pesquisar o código do curso e *codD* do tipo inteiro para pesquisar o código da disciplina. Utiliza-se a função auxiliar *existeCurso* para retornar o nó desejado, em seguida utiliza a função *imprimirDisciplinaCod* para imprimir a disciplina específica pelo código.

5.1.17 discBloco

Uma função do tipo *void* para imprimir as disciplinas pelo bloco, recebendo os parâmetros *raiz* do tipo *Struct Disciplina* e *bloco* do tipo inteiro para verificar qual o bloco, imprime ordenado de forma recursiva todos as disciplinas com mesma quantidade de blocos.

5.1.18 imprimirDiscBloco

Uma função do tipo *void* para imprimir as disciplinas de um dado curso de um dado bloco, tem os parâmetros *raiz* do tipo *Struct Curso*, *codC* do tipo inteiro e *bloco* do tipo inteiro. Utiliza a função *existeCurso* para acessar o curso desejado em seguida utiliza a função *discBloco* para imprimir as disciplinas pelo bloco.

5.1.19 cargaHorDisc

Uma função do tipo *void* para imprimir as disciplinas pelo bloco, recebendo os parâmetros *dis* do tipo *Struct Disciplina* e *CargaHor* do tipo inteiro para verificar qual o horário, imprime ordenado de forma recursiva todos as disciplinas com mesma carga horária.

5.1.20 imprimirDiscCursoHorario

Uma função do tipo *void* para imprimir as disciplinas de um dado curso de um dada carga horária, tem os parâmetros *raiz* do tipo *Struct Curso*, *codC* do tipo inteiro e *cargaHor* do tipo inteiro. Utiliza a função *existeCurso* para acessar o curso desejado em seguida utiliza a função *cargaHorDisc* para imprimir as disciplinas pela carga horária.

5.1.21 folha

Uma função auxiliar do tipo inteiro para verificar se o nó é folha, recebendo o parâmetro *raiz* do tipo *Struct Curso* que retorna 1 caso seja folha ou 0 caso não seja.

5.1.22 enderecoFilho

Uma função auxiliar do tipo *Struct Curso* que retorna um ponteiro do tipo *Struct Curso*, recebe o parâmetro *raiz* do tipo *Struct Curso* para retornar um dos filhos da função, com adendo de que se tenha a certeza de que o nó só possua um filho.

5.1.23 maiorFilhoEsq

Uma função auxiliar do tipo *void*, seus parâmetros são *filhoRecebe* do tipo *Struct Curso* passado por referência e *outrofilho* do tipo *Struct Curso*, retorna no *filhoRecebe* o nó mais a direita dele.

5.1.24 removerCurso

Uma função do tipo *void*(vazio), seus parâmetros são a *raiz* do tipo *Struct Curso* passado por referência e *codC* também do tipo inteiro, percorre a árvore de cursos pelo código, e remove o curso e libera da memória caso encontre o código e não possua disciplinas, utilizando as funções auxiliares *maiorFilhoEsq*, *folha* e *enderecoFilho* para fazer as verificações para a remoção.

5.1.25 liberarArvoreCurso

Uma função do tipo *void* para remover toda a árvore e liberar a memória utilizada por ela, recebe os parâmetros *raiz* do tipo *Struct Curso* e em pôs ordem de forma recursiva libera a árvore.

5.1.26 folhaDis

Uma função auxiliar do tipo inteiro para verificar se o nó é folha, recebendo o parâmetro *raiz* do tipo *Struct Curso* que retorna 1 caso seja folha ou 0 caso não seja, porém agora para disciplina.

5.1.27 enderecoFilhoDis

Uma função auxiliar do tipo *Struct Curso* que retorna um ponteiro do tipo *Struct Curso*, recebe o parâmetro *raiz* do tipo *Struct Curso* para retornar um dos filhos da função, com adendo de que se tenha a certeza de que o nó só possua um filho, porém agora para disciplina.

5.1.28 maiorFilhoEsqDis

Uma função auxiliar do tipo *void*, seus parâmetros são *filhoRecebe* do tipo *Struct Curso* passado por referência e *outrofilho* do tipo *Struct Curso*, retorna no *filhoRecebe* o nó mais a direita dele, porém agora para disciplina.

5.1.29 removerDisc

Uma função do tipo *void*(vazio), seus parâmetros são a *raiz* do tipo *Struct Curso* passado por referência e *codC* também do tipo inteiro, percorre a árvore de cursos pelo código, e remove o curso e libera da memória caso encontre o código e não possua disciplinas, utilizando as funções auxiliares *maiorFilhoEsq*, *folha* e *enderecoFilho* para fazer as verificações para a remoção, porém agora para disciplina.

5.1.30 auxRemoverDisc

Uma função do tipo *void* que remove a disciplina de um dado curso e dado código de disciplina, recebe os parâmetros *raiz* do tipo *Struct Curso*, *codC* do tipo inteiro e *codD* do tipo inteiro, utiliza a função *existeCurso* para acessar o curso deseja em seguida utiliza a função *removerDisc*.

5.2 Funções da Árvore AVL

5.2.1 Funções repetidas

Possui as mesmas funções da Árvore binária, sendo os tópicos: 4.1.1, 4.1.4, 4.1.6, 4.1.8 ao 4.1.23, 4.1.25, 4.1.26, 4.1.27, 4.1.28, 4.1.29.

5.2.2 Mudanças de estrutura

Como na árvore AVL precisa de altura para calcular o fator de balanceamento, sua estrutura deve ser acrescentada a informação *altura* do tipo inteiro, com isso as funções 4.1.2 e 4.1.5 recebem uma atualização inicializando a informação altura da árvore.

5.2.3 pegarAltura

Uma função auxiliar do tipo inteiro para retornar a altura do nó, recebe como parâmetro *raiz* do tipo *Struct Curso*, para retornar sua altura, caso não haja nó existente, é retornado -1.

5.2.4 atualizarAltura

Uma função do tipo *void* que recebe como paramêtro *raiz* do tipo *Struct Curso* e com auxílio da função *pegarAltura*, atualiza a altura do nó.

5.2.5 fb

Uma função do tipo inteiro que retorna a altura do nó, recebe como parâmetro a função auxiliar *pegarAltura* para realizar a formula citada na introdução da árvore AVL.

5.2.6 rotacaoEsquerda

Uma função do tipo *void* que rotaciona a raiz para esquerda e atualiza a altura através da função *atualizarAltura*, recebendo o parâmetro *raiz* do tipo *Struct Curso* passado por referência.

5.2.7 rotacaoDireita

Realiza os mesmos passos da função 4.2.6, no sentido da direita.

5.2.8 balancear

Uma função do tipo *void* que realiza rotações nos nós da árvore para que ela permaneça balanceada, recebe o parâmetro *raiz* do tipo *Struct Curso* passado por referência e faz rotações simples ou duplas, a depender do caso, utilizando as funções *fb*, *rotacaoEsquerda* e *rotacaoDireita*.

5.2.9 Funções para balancear

As funções dos tópicos: 4.1.3, 4.1.7, 4.1.24 e 4.1.30. Realizam os mesmo passos porém acrescentando as novas funções citadas acima.

5.3 Funções de Teste da Árvore Binária

5.3.1 embaralharVetor

Uma função do tipo *void* para adicionar valores aleatórios para o teste, sendo passado como parâmetro o *vet*, um vetor do tipo inteiro.

5.3.2 criarArquivo

Uma função do tipo *void* para criar o arquivo de testes, recebendo um parâmetro *vet*, um vetor do tipo inteiro.

5.3.3 lerArquivo

Uma função do tipo *void* para abrir o arquivo e ler o que tem dentro dele, recebendo o parâmetro *vetor*, um vetor do tipo inteiro.

5.3.4 inserirValoresTestes

Uma função do tipo *void* para inserir na árvore os cursos dentro do vetor e calcular o tempo de inserção, sendo passados os parâmetros *raiz* do tipo *Struct Curso* passado por referência, *vet*, um vetor do tipo inteiro, *temps* um vetor em double.

5.3.5 tempoBusca

Uma função do tipo *void* para buscar um nó da árvore e calcular o tempo de busca, recebe os parâmetros *raiz* do tipo *Struct Curso* passado por referência, *vet*, um vetor do tipo inteiro, *temps* um vetor em double.

5.4 Funções de Teste da Árvore AVL

5.4.1 Funções Repetidas

Reutilizei as funções 4.3.1, 4.3.3, 4.3.4, 4.3.5.

6 Testes

Foi utilizado a inserção de 100.000 valores e realizado uma média de 30 testes na inserção e busca.

6.1 Ordenado

Árvores	Média Inserir	Média Buscar
Árvore Binária	115188.214 milissegundos	4.06 milissegundos
Árvore AVL	298.77223 milissegundos	0.00580 milissegundos

6.2 Desordenado

Árvores	Média Inserir	Média Buscar
Árvore Binária	242.136 milissegundos	0.002 milissegundos
Árvore AVL	309.30330 milissegundos	0.0018 milissegundos

7 Conclusão

Conforme os testes e lógica por trás dos acrescimos da árvore binária para se tornar árvore AVL, e perceptível a melhora em busca, independente dos casos, onde o primeiro caso o método ordenado torna a árvore binária em uma lista agravando o seu resultado em inserção e busca, onde, pelos próprios dados e notável que ao utilizar árvore AVL, o balanceamento a torna mais eficiente, pois a construção da sua árvore cresce mais lentamente de forma vertical, o que torna a busca mais direta, porém como visto na inserção desordenada ou aleatória, a inserção da árvore binária é mais eficiente pois executa menos passos de inserção, dependendo de como está dos dados aleatoriamente, mas mesmo com essa pouca vantagem na inserção se perde na busca, pois a árvore AVL estará sempre balanceada.