Estruturas de Dados em Árvores: Relatório com uma análise comparativa com os resultados de dois algoritmos simples implementados com as árvores ABB e AVL.

Dayan Ramos Gomes

Resumo:

Este relatório apresenta informações sobre as árvores de estruturas de dados utilizadas para resolução dos algoritmos definidos, sendo elas: Árvore Binária de Busca (ABB) e Árvore Binária AVL, a partir delas, dois algoritmos foram desenvolvidos e resolvidos com a utilização das mesmas. Com base no resultado obtido, foi feita uma análise comparativa de tempo nas operações de inserção e busca dos algoritmos para esclarecer e definir qual é a melhor implementação de árvore para determinado algoritmo.

1. Introdução

Árvore é uma estrutura de dados que herda as características das topologias em árvore, conceitualmente diferente das listas encadeadas, onde, nas árvores os dados estão dispostos de forma hierárquica e o seu conceito e implementação está diretamente ligado a recursividade.

Este trabalho tem como objetivos demonstrar dois tipos de estruturas de dados em árvores mais básicos e seu funcionamento, características, desempenho e assim podermos discernir quando cada tipo de árvore é mais indicado para implementação. Para definir qual tipo de árvore utilizar em determinado problema é preciso primeiramente realizar uma análise do problema, quantidade informações que vão ser armazenada inicialmente, se os dados precisam estar bem distribuídos ou ordenados, entre outros fatores que podem ser vistos com uma análise prévia. Sabendo disso é possível podemos prover uma melhor otimização tanto do hardware e software da nossa máquina e prover uma melhor resolução maximizando as possibilidades de problemas.

Assim, os testes realizados dos algoritmos foram feitos em um notebook da Acer com as seguintes especificações técnicas:

- Modelo: Acer a515-51g-58vh
- Processador: Intel Core I5-7200U de 7ª G – 2 núcleos e 4 threads - (até 3.1 GHz) 3MB Cache
- Memória RAM: 7.9 GB DDR4 - 2133 MHz (2 x 4 GB)
- Placa de vídeo dedicada GeForce 940MX com 2GB GDDR5
- SSD SATA M.2 WD Green 240 GB – 545 MB/s leitura e escrita
- Sistema Operacional: Windows 10 de 64 bits

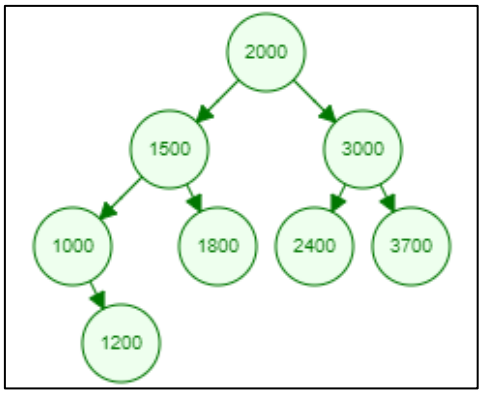
Desta forma, este documento está organizado da seguinte forma: A Seção 2 contém uma breve explicação sobre os conceitos dos tipos de árvores utilizados, os problemas propostos e uma apresentação sobre os aspectos funcionais presentes nos programas desenvolvidos para revolver os problemas; a Seção 3 apresenta um estudo comparativo feito a partir da execução dos programas e uma explanação sobre o que são e para que servem; a Seção 4 apresenta a conclusão do trabalho.

2. Seções Específicas

Primeiramente, para melhor entendimento do relatório é importante conhecer as estruturas de dados em árvores utilizadas. Abaixo está uma breve explicação das principais características de cada árvore e logo depois aos problemas propostos com suas resoluções com as árvores citadas.

2.1 Árvore Binária de Busca

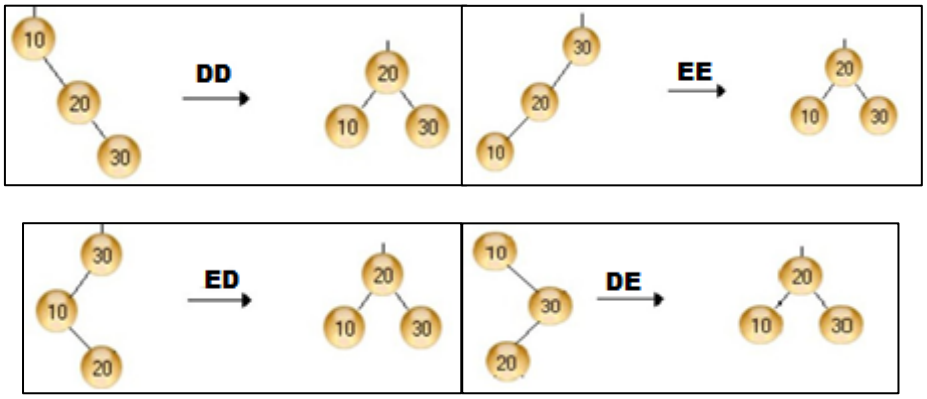
Uma árvore binária de busca é uma estrutura de dados baseada em nós. Cada nó armazena um conjunto de informações e ponteiros para subárvores da esquerda e direita, chamados filhos, onde todos os nós da esquerda contêm valores menores que o nó raiz que é o primeiro valor informado, e o todos os nós da direita contém valores maiores do que o nó raiz.



Sempre que um nó é inserido, ele se torna uma folha, ou seja, um nó que não tem subárvores. Ela é indicada para dados que estão bem distribuídos, pois assim, seu tempo de busca pode ser reduzido até pela metade, porém a forma com que os dados são inseridos influencia na estrutura da arvore, pois, essa árvore não faz a organização dos dados.

2.2 Árvore Binária AVL

A árvore AVL tem as mesmas características da árvore binária de busca, porém, em relação a organização dos nós na árvore, ela é mais organizada. Assim que um novo nó é inserido, a árvore AVL faz uma reorganização dos nós a partir do seu fator de balanceamento, que é um cálculo feito a partir da altura dos nós filhos e onde a altura de cada nó é guardada como uma informação dentro do próprio nó. Essa reorganização é feita a partir de rotações simples ou duplas, sendo elas as rotações: DD, EE, ED e DE, mantendo os nós nos padrões de uma árvore balanceada.



As rotações simples são as DD e EE, feitas quando o maior ramo é externo para direita ou para esquerda, respectivamente. A rotação dupla ED é feita quando o maior ramo é o interno e para esquerda, sendo nada menos que duas rotações simples, a rotação dupla DE é feita quando o maior ramo é o interno da direita, sendo duas rotações simples também. Assim, esta árvore busca otimizar as buscas, já que a árvore sempre estará balanceada, porém é um algoritmo custoso quando há muitos balanceamentos a serem feitos.

2.3 Problema 1

Nessa questão, é pedido para fazer a inserção de 1000 números aleatórios em uma árvore, buscar um número, mostrar o nó folha mais profundo e o menos profundo, e quantas vezes a diferença entre a profundidade maior e menor foram de 0, 1, 2, 3, 4 e assim por diante. Também é pedido para contabilizar o tempo para inserir todos os números e para buscar um elemento, tudo isso repetido 30 vezes para fazer a analiso dos resultados.

A solução deste problema foi feita com as árvores já apresentadas (ABB e AVL), e foi feita uma inserção de 5000 números, para uma melhor análise dos dados mostrados na próxima seção.

2.3.1 ABB

Nesta implementação as funções da *main* estão dentro de um loop de repetição até 30 vezes, assim foi feita uma inserção de 5000 números aleatórios na arvore binária de busca, onde primeiro foi inserido 2500 e depois mais 2500 para maior aleatoriedade, depois foi feita a coleta do tempo de inserção com um função da biblioteca *time.h* e mostrado o tempo de inserção em milissegundos.

Após isso, foi a chamada de uma função para coletar a profundidade do nó folha mais fundo e uma função para coletar a profundidade do nó folha menos profundo, e depois foi exibido o retorno dessas funções. Após isso é feita a busca de um número, coletado o tempo para a função de busca encontrar ou não esse número e mostrado esse tempo de busca em milissegundos.

No fim é array de structs alocado dinamicamente com a diferença da maior e menor profundidade, contabilizando quantas vezes uma diferença de 15 por exemplo se repetiu, depois exibe o resultado fora do primeiro loop de repetição. As outras demais função são as funções padrões para inserção e busca

- ehfolha(): recebe um nó da raiz e retorna um valor int caso o nó seja folha ou não.
- folhaFunda(): recebe a raiz, percorre toda a árvore a partir da recursividade onde o filho da direita e esquerda é passado para uma chamada da própria função, calcula a profundidade do nó e guarda o valor do nó mais profundo, retornando-o na função.
- menorDir(): recebe um nó da raiz, calcula a profundidade do nó a direita e o retorna.
- menorEsq(): recebe um nó da raiz, calcula a profundidade do nó à esquerda e o retorna.
- folhaRaza(): recebe a raiz, percorre toda a árvore e retorna a profundidade do menor nó folha.

2.3.2 AVL

A implementação deste problema com a utilização da árvore AVL é, em geral, semelhante ao da árvore ABB, diferenciando-se apenas na inserção que, como explicado e exemplificado na seção 2.2, na árvore AVL, quando um elemento é i inserido a árvore verifica se o fator de balanceamento são maiores que 2 ou menores que -2, caso sejam, após a inserção de um nó é feita a chamada das rotações que tem como objetivo balancear a árvore, na qual cada rotação é chamada de acordo como o nó está desbalanceado.

No fim, o escopo geral da implementação ficou igual, com a única diferença que na AVL a árvore estava sempre balanceada, mas na questão de coleta de tempo de inserção e busca, cálculo de maior e menor profundidade e exibição da diferença da maior e menor profundidade foram reutilizadas as mesmas funções da árvore binária de busca.

2.4 Problema 2

Nessa questão, é pedido para fazer um índice analítico de um livro que consiste na inserção termos principais em uma árvore binária ordenada alfabeticamente, onde cada termos tem um conjunto de páginas onde esse termo é mostrado e também tem um conjunto de variações desse termo, ou seja, seus subtermos, e cada subtermo tem suas páginas também onde ele aparece, tudo isso utilizando árvores binárias e listas para as páginas dos termos e subtermos.

Após isso, é necessário mostrar um termo numa linha, seguidos pelas páginas em que ele aparece, em ordem ascendente, e os subtermos da mesma forma, só que endentados em relação ao seu termo principal.

2.4.1 ABB

Nesta implementação, foi feita uma árvore binária ABB para os termos principais, que tem uma string, ponteiros para seus filhos, um ponteiro para uma árvore binária ABB de páginas, um ponteiro para outra árvore binária ABB de subtermos, que também possui uma string, um ponteiro para seus filhos e um ponteiro para uma árvore binária ABB de páginas.

Nesta implementação, foram feitas funções para inserir termos, subtermos e paginas nas suas respectivas árvores binárias ABB. As funções que insere os termos e subtermos em ordem alfabética, utilizam o comando *strcmp* para comparar as strings na hora da recursividade.

O tempo de inserção de todos os elementos também foram calculados, somados e mostrados, para sabermos o tempo total de todas as inserções, assim como o tempo de busca e exibição da função que mostra todos os elementos ao final do programa, de acordo como foi pedido no problema, mostrando os termos seguidos por suas páginas e em seguida mostra seus subtermos, se houver, seguidos por suas páginas.

- inserirTermo(): recebe como parâmetro a raiz da árvore e uma string com o termo principal, assim, insere o termo em ordem alfabética.
- inserirPagInTermo(): recebe como parâmetro a raiz de termos, uma string com o termo principal e o numero de uma página, onde a função faz as devidas comparações e insere a página no termo passado por parâmetro.
- inserirSubTermoInTermo(): recebe como parâmetro a raiz de termos, uma string com o termo principal e uma string com o subtermo, a função faz as devidas comparações e adiciona o subtermo ao termo passado por parâmetro.
- inserirPagInSub(): recebe como parâmetro a raiz de termos, uma string com um termo, uma string com um subtermo e uma página, onde a função faz as devidas comparações e se o termo informado existir, chama a função procuraSub() passando os subtermos desse termo encontrado.
- procuraSub(): recebe uma raiz de subtermos, uma string com um subtermo, e uma página, faz as devidas comparações e relaciona/adiciona a página informada com o subtermo do termo passado, caso ele exista.

2.4.2 AVL

A implementação deste problema com a utilização da árvore AVL é também, como o problema 1, semelhante ao da árvore ABB, diferenciando-se apenas na inserção que, como explicado e exemplificado na seção 2.2, na árvore AVL, quando os termos, os subtermos, e as paginas são inseridas, caso alguma dessas inserções deixe algum nó desbalanceado, as rotações (Seção 2.2) serão chamadas para fazerem os devidos ajustes na árvore, mantendo-a sempre balanceada.

No fim, o escopo geral desta implementação em árvore AVL também muito parecido, com apenas as diferenças já citadas, mas na questão de buscar e mostrar os elementos, da coleta de tempo de inserção e busca, foram reutilizadas as mesmas funções da árvore binária de busca.

3. Resultados da Execução do Programa

Nesta seção será apresentado uma análise comparativa da inserção e busca dos dados resultantes da execução dos algoritmos implementados e utilizados na resolução dos problemas citados na seção anterior.

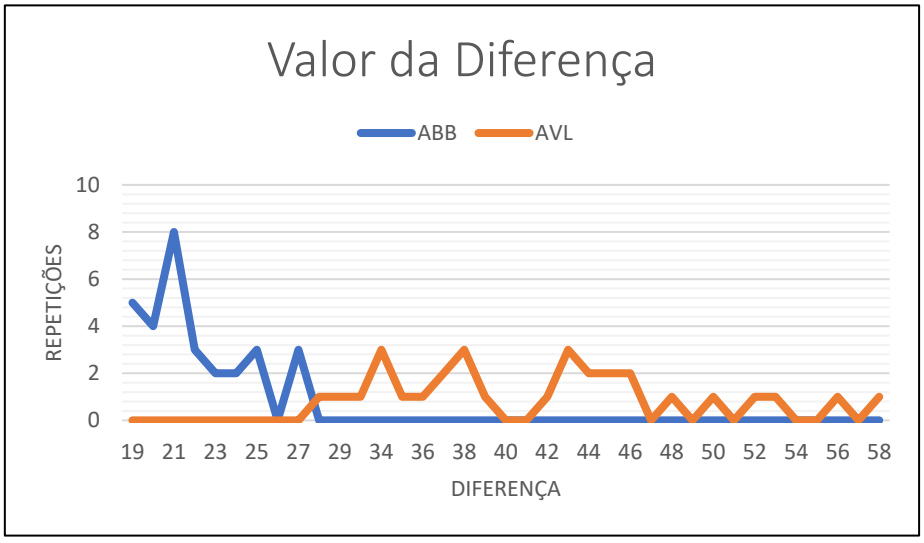
3.1 Árvore Binária de Busca x Árvore binária AVL (Problema 1)

Os seguintes resultados mostrados, são resultados da inserção de 5000 (cinco mil) números aleatórios inseridos nas duas árvores com um teste de 30 repetições para cada árvore mostrada (Seção 2.1 e 2.2).

ÁRVORE	Tempo médio para a Inserção (tempo em milissegundos)	Tempo médio para a Busca (tempo em milissegundos)
Árvore Binária de Busca	12.5	0.5
Árvore Binária AVL	651.6	0.08

Primeiramente, em relação a inserção de dados nas duas árvores, podemos claramente perceber que a árvore binária AVL tem tempo médio muito maior para inserir todos os 5000 números, isso se deve, pois como são muitos dados inseridos, repetidamente a árvore fica desbalanceada, e no caso da árvore AVL, todo vez que um nó está pendente pra algum lado, ela faz o balanceamento da árvore, na qual demanda um maior custo de tempo para isso, principalmente levando em consideração que são cinco mil números. Já a árvore binária de busca tem um tempo de inserção visivelmente menor, pois nela os números são apenas inseridos e não são balanceados, levando assim, a uma inserção mais rápida.

Em relação a busca de dados, apesar das árvores usadas não serem a estrutura de dados mais rápida a nível computacional, o processamento para busca é muito rápido, levando as duas árvores a terem um tempo de busca muito baixo, mas, se mesmo assim formos comparar, podemos perceber que a busca na árvore binária AVL é mais rápida, pois apesar dela ter um custo alto na inserção, na sua busca tem um desempenho melhor, pois os dados da árvores estão todos balanceados.



O gráfico demostra a diferença do nó mais profundo e do nó menos profundo das duas árvores nas duas execuções do programa. O resultado obtido não foi como esperado, pois esperávamos que o valor da diferença da árvore AVL fosse menor do que o da árvore binária de busca e com mais repetições, porém, pode ter tido algum erro na implementação do mesmo na qual resultou nesse resultado para o valor das diferenças nesse problema.

3.2 Árvore Binária de Busca x Árvore binária AVL (Problema 2)

Os seguintes resultados mostrados, são resultados da inserção de alguns termos, algumas páginas para os termos, da inserção de alguns subtermos e algumas páginas para eles inseridos nas duas árvores (Seção 2.1 e 2.2).

ÁRVORE	Tempo médio para a Inserção (tempo em milissegundos)	Tempo médio para a Busca e Exibição (tempo em milissegundos)
Árvore Binária de Busca	0	48
Árvore Binária AVL	0	15

Em relação a inserção de dados, apesar das árvores usadas não serem a estrutura de dados mais rápida a nível computacional, o processamento para inserção com apenas alguns elementos é muito rápido, levando as duas árvores a terem um tempo de inserção zerado.

Em relação a busca e exibição de todos os dados, as duas árvores tem um tempo de busca muito baixo, mas a árvore AVL por estar balanceada tendo a ter um tempo menor do que a árvore binária de busca que não está balanceada, podendo ser muito mais rápida para buscas em casos em que há muitos elementos adicionados, lembrando que quanto mais elementos a inserir, mais a inserção demora, mas quanto mais elementos inseridos, mais rápida a busca por determinado dado.

4. Conclusão

A partir deste estudo, foi possível concluir, a partir do propósito de descobrir qual estrutura de dados em árvores das duas apresentadas, sendo elas, a árvore binária de busca e a árvore binária AVL é mais rápida para determinado problema. Com base na análise dos testes executados observamos a proporção entre o tempo de inserção de dados e o tempo de busca nas duas árvores presentes nesse relatório.

Em relação a árvore binária de busca, concluímos que o seu tempo de inserção é o menor tempo entre as duas árvores, pois como ela não realiza o balanceamento dos seus nós, consequentemente a sua inserção vai ser mais rápida, porém, em relação ao seu tempo de busca, concluímos que é o maior entre as duas árvores, pois como seus dados estão mal distribuídos na estrutura, para buscar um determinado elemento ou para concluir que ele não está inserido é demando um custo de processamento maior e consequentemente, um tempo maior.

Em relação a árvore binária AVL, concluímos que o seu tempo de inserção é muito maior do que a outra árvore, devido que dado inserido que afeta o balanceamento da árvore, rotações são feitas para que a árvore continue balanceada a cada inserção, desta forma, aumento muito o seu custo de processamento e consequentemente o seu tempo, e em relação ao tempo de busca, concluímos que é o menor dentre as duas árvores, pois como muito processamento foi gasto para deixar ela balanceada, a busca de um elemento em uma árvore balanceada vai demandar muito menos processamento e assim, menos tempo.

Desta forma, podemos ver a diferença de cada estruturas de dados em árvores em cada inserção e busca dos problemas. Com essa análise parcial foi aprendido a como implementar as árvores em determinados problemas e a relacionar os dados de mais de uma árvore, como feito no problema 2, foi descoberto também as vantagens de cada árvore e um possível erro na implementação da árvore AVL no primeiro problema que influenciou no resultado das diferenças das profundidades dos nós, onde o mesmo será corrigido mais a frente, após analisar a implementação.

5. Referências

BOERES, Cristina. **ESTRUTURAS DE DADOS Árvores, árvores binárias e percursos.** 73 slides. Disponível em < http://www2.ic.uff.br/~boeres/slides_ed/ed_ArvoresPercursos.pdf>. acesso em 10 de março de 2022.

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