Instituto de Engenharia - Várzea Grande

AVL TREE VS BINARY SEARCH TREE

GUSTAVO DA COSTA GONÇALVES
LUCAS STEFFENS DE OLIVEIRA

Cuiabá
Abril de 2017

UNIVERSIDADE FEDERAL DE MATO GROSSO
Instituto de Engenharia - Várzea Grande

Avl Tree vs Binary Search Tree

GUSTAVO DA COSTA GONÇALVES
LUCAS STEFFENS DE OLIVEIRA

Trabalho apresentado ao Instituto de Engenharias de Várzea Grande, em comprimento das exigências legais, sob orientação do Prof. Frederico S. Oliveira, para obtenção de nota da disciplina de Estrutura de Dados.

Abril de 2017

Sumário

1 INTRODUÇÃO

As árvores binárias, são ótimas estruturas de dados, essas que podem ser utilizadas de diversas formas para inúmeros fins. A usabilidade é tamanha que se fossemos citar alguns exemplos falariamos este trabalho todo sobre isso.

Mas o objetivo principal desta documentação é discutir a diferença entre a Árvore binária de Busca (ABB) e a Árvore binária balanceada(AVL), estas que devem receber um texto e alocar em cada nó uma palavra realizando a impressão em ordem alfabetica e ao mesmo tempo contabilizar a frequência em que as palavras se repetem, além dos indicadores de performace de cada algoritmo testado. Esses indicadores são número de comparações e tempo de processamento.

1.1 Árvore Binária de Busca

Uma árvore binária é uma estrutura de dados com o intuito de facilitar as buscas dos dados. Isto por ser muito flexivel, pois como uma lista ela é dinâmica e pode possuir o tamanho que for necessário (ou que a memória deixar) para alocar dados. Mas não é esta sua principal vantagem e sim como ela se organiza ("ordena"), uma árvore binária idependente de qual seja se organiza da seguinte forma:

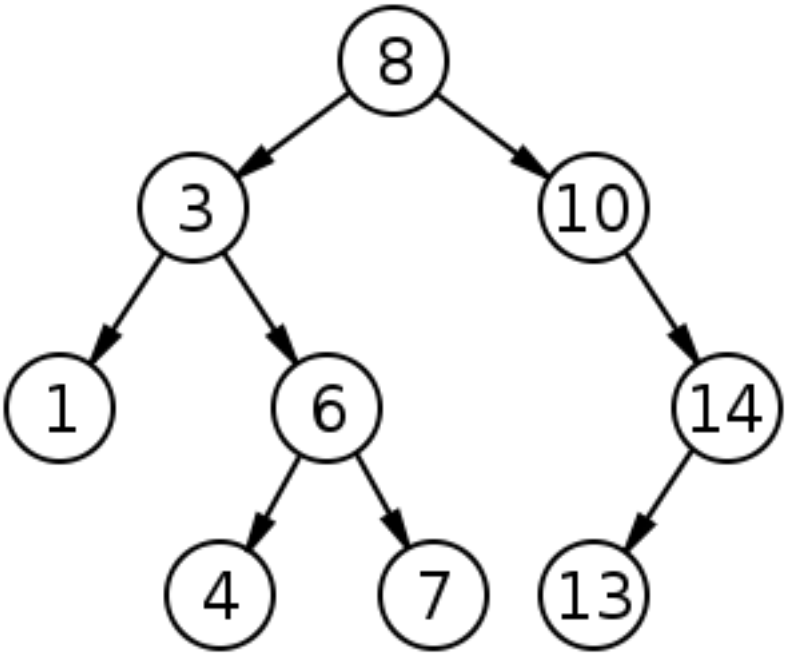

Figura 1: Exemplo de uma árvore binária

As regras que regem uma Árvore binária de busca são:

`Inserção`

- No ínicio de uma ABB é inserido um nó raiz.
- Após a inserção do nó raiz os próximos nós serão inseridos a esquerda caso ele for menor que o nó raiz e a direita caso ele for maior que o elemento do nó raiz.
- Para os próximos N elementos adicionados o segundo procedimento é realizado.
- Como não há nenhuma regra de balaceamento na ABB, a árvore pode acabar parecendo uma lista dinâmica encadeada.

`Busca`

- Para a busca por um elemento específico é examinando a raiz. Se o valor for igual a raiz o elemento buscado é a raiz.
- Se o valor for maior do que a raiz, então deve buscar na sub-árvore da direita, e este mesmo teste é feito em todos os nós da sub-árvore. Da mesma forma caso o elemento buscado seja menor do que a raiz, a busca ocorre na sub-árvore da esquerda.
- O procedimento é realizado até encontrar o elemento ou então chegar nos nós folhas que indicam o fim da árvore e que o elemento não existe.

`Remoção`

Remoção na folha: É o caso mais simples, onde o elemento a ser removido está em uma folha, logo basta apenas exclui-lo.

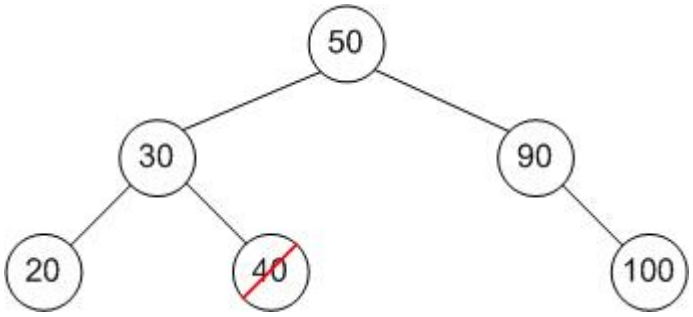

Figura 2: Exemplo de remoção na folha de uma ABB

Remoção de um nó filho:

Caso o elemento a ser removido possua apenas um nó filho ele ocupa o lugar de seu pai.

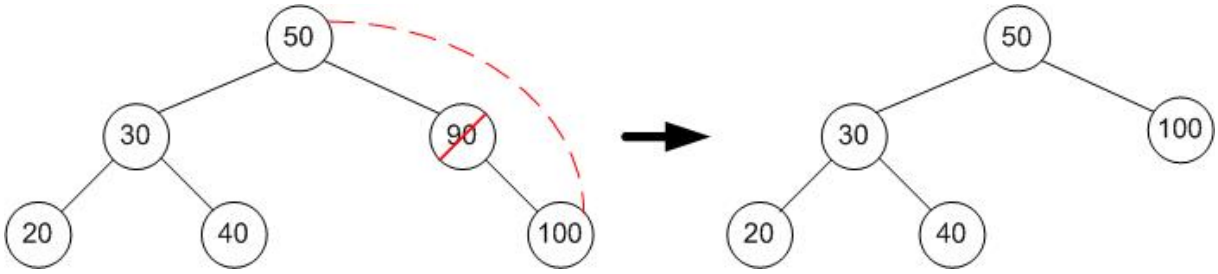

Figura 3: Exemplo de remoção nó filho de uma ABB

Caso o elemento a ser removido possua dois filhos o elemento sucessor (o elemento da direita mais a esquerda) que toma o lugar do pai.

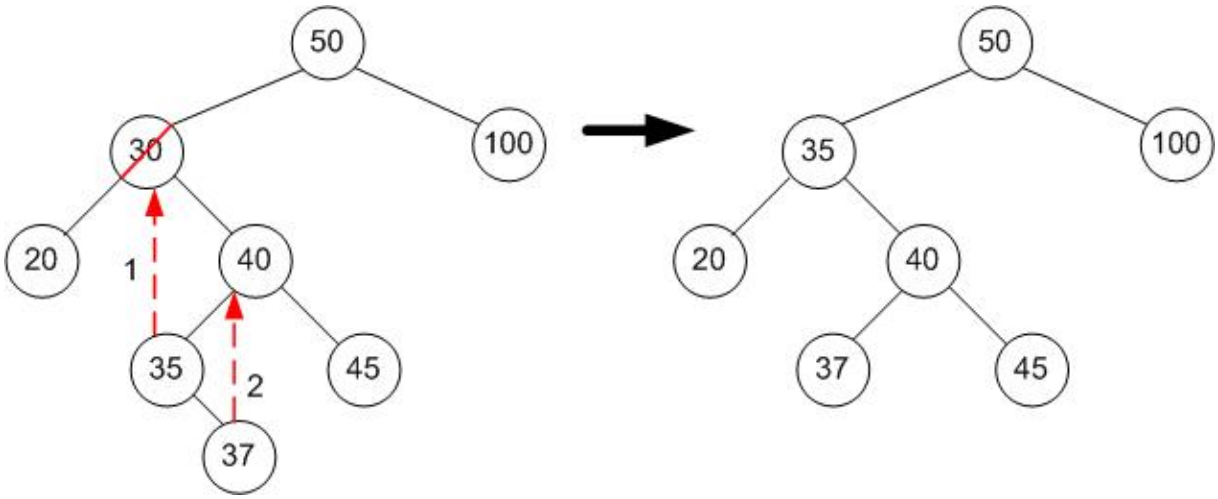

Figura 4: Exemplo de remoção nó filho de uma ABB

Podemos notar que este último caso serve caso o nó a ser removido for a raiz.

`Trájetos de Percursão`

Existem três formas de percorrer uma ABB.

Pré-Ordem

- Inicia na raiz.
- Percorre a sub-árvore esquerda em pré-ordem(Recusivamente realiza estes mesmos passos).
- Percorre a sub-árvore direita em pré-ordem.

Em Ordem

- Percorre a sub-árvore da esquerda (recusivamente).
- Visita na raiz.
- Percorre a sub-árvore da direita (recusivamente)

Pós Ordem

- Percorre a sub-árvore da esquerda (recursivamente) em pós ordem.
- Percorre a sub-árvore da direita (recursivamente) em pós ordem.
- Visita a raiz.

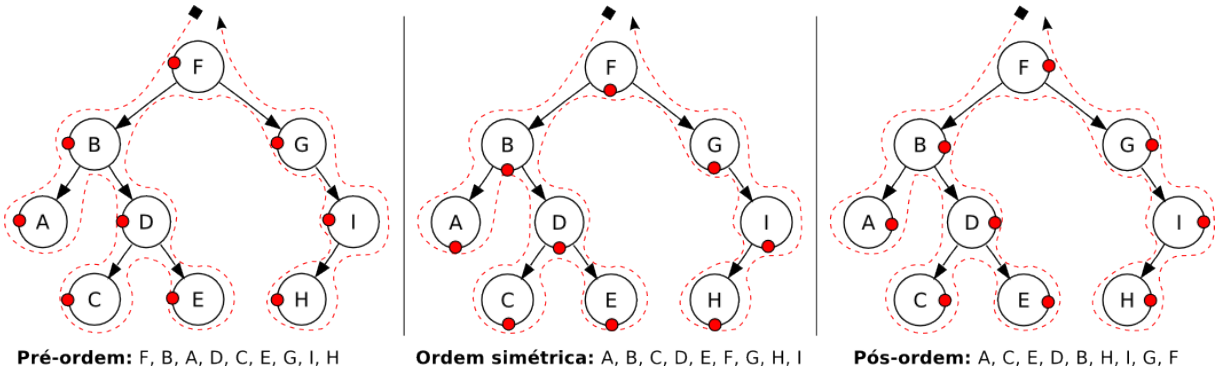

Figura 5: Exemplo das percusões de uma ABB

1.2 Árvore AVL

Um outro tipo de estrutura de dados de árvore binária baseada no balanceamento do nós, **a sua principal vantagem é a otimização na operação de busca**. A AVL TREE tolera no máximo 1 nivel de diferença de altura entre os nós filho, propriedade essa que a torna balanceada, logo após as operações de inserção e remoção dos dados é possivel que fique desbalanceada, para isso é realizado o cálculo do Fb, o cálculo do fator balanceamento(Fb) de cada nó é feito de acordo com a sua altura, ao constatar que ultrapassou o limite de 1 nivel, logo está desbalanceada, assim será realizada as operações de rotações para novamente deixá-lá balaceada.

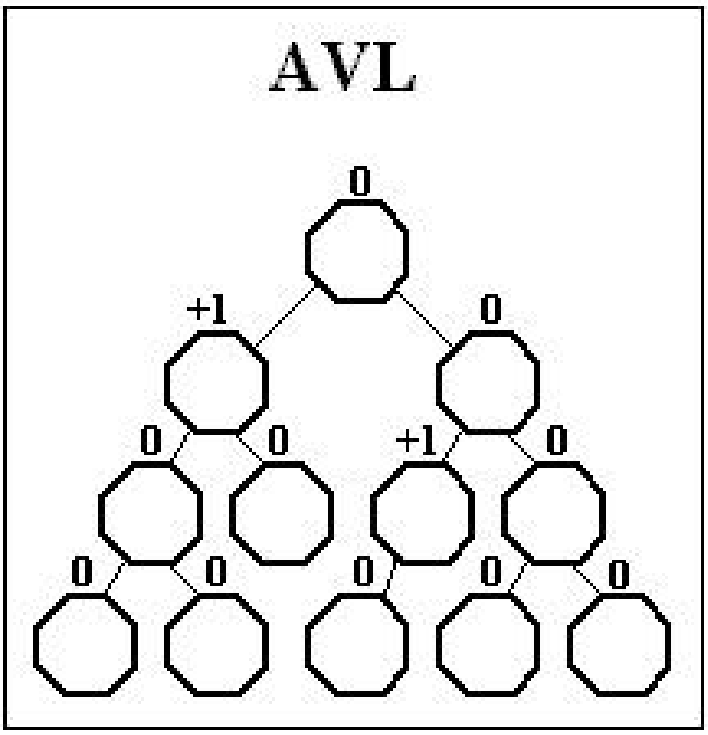

Figura 6: Exemplo de uma Árvore AVL

1.2.1 Propriedades

- A altura é de um nó é sempre a maior altura de seus filhos até um nó folha
- O fator de balanceamente é calculado como a diferença da altura do filho da esquerda, com a altura do filho da direita: $Fb = he - hd$
- Caso o Fb difere de -1,1 ou 0 a AVL está desbalanceada sendo assim necessário as rotações, operação que balanceia novamenta a árvore após uma inserção ou remoção.

1.2.2 Rotações

As rotações ocorrem quando as operações de inserção e remoção acontecem desbalanceando a árvore, basicamente a rotação depende do resultado do Fb, após as rotações a AVL TREE volta a estar balanceada

- Caso o Fb do nó avaliado seja igual a -2 e o Fb do nó filho seja igual a -1 é realizada a operação rotação simples a direita.

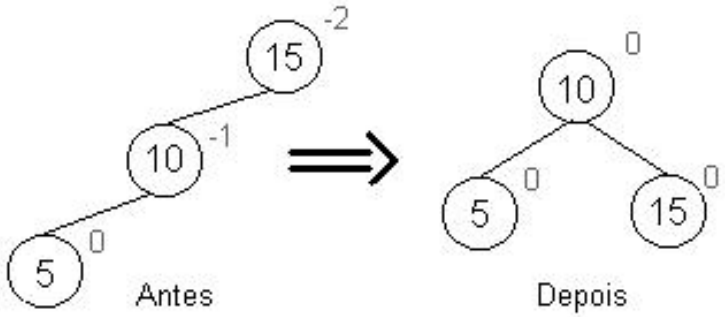

Figura 7: Rotação simples à direita

- Caso o Fb do nó avaliado seja igual a 2 e o Fb do nó filho seja igual a 1 é realizada a operação rotação simples a esquerda.

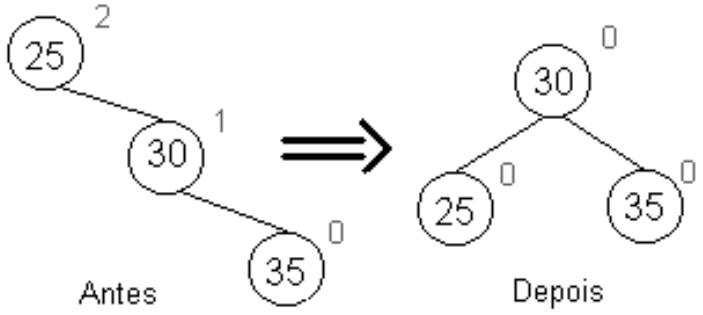

Figura 8: Rotação simples à esquerda

- Caso o Fb do nó avaliado seja igual a -2 e o Fb do nó filho seja igual a 1 é realizada a operação rotação dupla a direita.

  Rotação dupla para direita é composta de uma rotação simples à direita, seguida de uma rotação simples à esquerda.

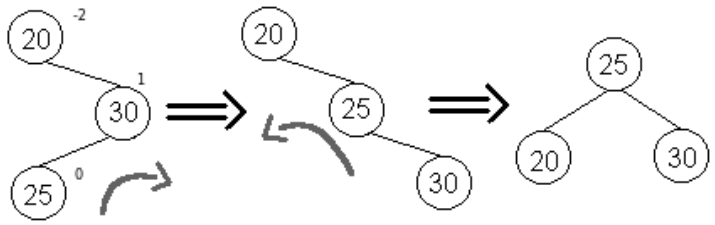

Figura 9: Rotação dupla para à direita

- Caso o Fb do nó avaliado seja igual a 2 e o Fb do nó filho seja igual a -1 é realizada a operação rotação dupla a esquerda.

  Rotação dupla para esquerda é composta de uma rotação simples à esquerda, seguida de uma rotação simples à direita.

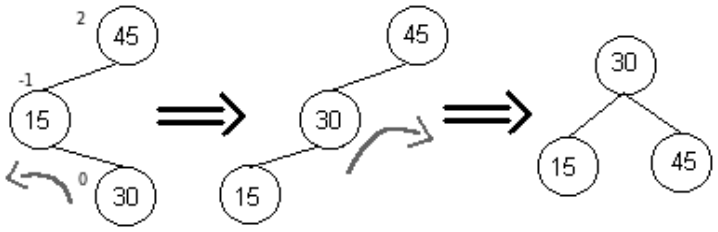

Figura 10: Rotação dupla para à esquerda

2 DESENVOLVIMENTO

O desafio deste trabalho prático consistiu em manipular os códigos de Árvores Binárias, pra a alocação de string advindas de entradas com a extensão .txt. Para isso foram retirado da internet códigos com as estruturas das árvores com sua devidas funções para manipulações. Estas que comumente são para tratar números e não strings.

A manipulação foi feita da seguinte maneira, primeiramente o arquivo main.c faz leitura do arquivo.txt, caracter por caracter, quando é encontrado um ponto de parada como um caracter especial por exemplo o espaço do teclado , o algoritmo detecta que é o final de uma palavra, assim adiciona-se todos os caracteres dessa palavra em uma variável, formando uma string(que é uma cadeia de caracters), desse modo é chamada as funções das respectivas árvores binária de busca e árvore avl. Dentro de cada função de cada árvore é adicionado as palavras de acordo com o tamanho e as regras de inserção de cada árvore, caso as palavras forem de tamanho igual sua frequência é incrementada. Após a manipulação, foi feita a análise do comportamento dos dois tipos de árvores implementadas, comparando-as com os resultados obtidos nos testes.

2.1 Utilização de bibliotecas

Tais bibliotecas foram utilizadas com a inteção de simplificar a codificação, pois, essas bibliotecas tem como próposito auxiliar a programação, as importações realizadas foram destinadas a manipulação de string e caracteres.

2.1.1 ctype.h

A biblioteca ctype.h contém funções e macros para manipulação de caracteres, as funções utilizadas foram:

- tolower - Converte o caracter em minúsculo
- isalpha - Verificar se o caracter é uma letra do alfabeto
- isspace - Verificar se o caracter é um espaço em branco
- isupper - Verifica se o caracter é uma letra maiúscula

2.1.2 string.h

A biblioteca string.h da linguagem C, contém uma série de funções para manipular strings, a função utilizada foi:

- strcmp - variável do tipo inteiro, x = strcmp(string1, string2), compara o conteúdo de duas strings, os possíveis valores de retorno são:

  se x == 0: o conteúdo das strings são iguais

  se x < 0: o conteúdo da string1 é menor do que string2

  se x > 0: conteúdo da string1 é maior do que string2

2.2 Análise ABB vs AVL

Foram realizados testes com sete diferentes textos, variando tamanho e "complexidade". Os arquivos .txt denominados complexos nesta análise se referem a textos com inumeros carateres especiais e pontuações, estas que devem ser evitadas na contagem estatistica. Isto foi realizado pois assim conseguimos testar a frágilidade e o desempenho tanto quanto ao tamanho quanto para possiveis defeitos na implementação.

Os textos utilizados se encontram na pasta do trabalho no diretório casos de testes, e foram escolhidos aleatóriamente sem a verificação.

Abaixo segue a tabela dos resultados obtidos:

Tabela 1: Resultados Obtidos

TEXTO	AVL COMPARAÇÔES	ABB COMPARAÇÔES	TEMPO
Caso de Teste	132	330	0.000015
Teste da Orientação do Trabalho	219	448	0.000022
Férias	152	830	0.000039
Biblia Cristã	7118971	17444652	0.287769
Alcorão	1290491	3007244	0.054931
Machado de Assis A mão e a Luva	312047	834496	0.016476
Drácula 1990	1491613	3499432	0.0625940

Com estes testes foi possivel notar explicitamente a diferença entre uma árvore bínaria de busca não balanceada para uma árvore balanceado no quesito comparações, em todos os casos de testes foram realizados pela ABB práticamente o dobro de comparações do que a AVL.

Oque já era esperado uma vez que o balanceamento tem como objeitivo diminuir as comparações quando se busca e/ou insere um elemento.

Até então não foi perceptivel nada anormal ao esperado contando com os conceitos aprendidos em sala de aula, pois os dois métodos se mostraram extremamente rápidos idependente da quantia de strings que foram alocadas na árvore deixando a diferencial apenas nos números de comparações realizadas.

Contudo foi notado também que o tempo de processamento tanto da ABB quanto da AVL, em todos os casos de testes foram idênticos, isso pode ser justificavél pois como são métodos extremamente eficazes para alocação de dados, e a máquina onde foi realizado os testes possue um hardware intermediario. Isto acabou invalidando a métrica de tempo de processamento uma vez que que foi uniforme.

Nos testes realizados, ocorreu de muitas palavras estarem com pontuações que conforme a orientação do trabalho, foi solicitado que fosse ignorado, fazendo assim com que algumas palavras na contagem viessem a perder o sentido. Mas este problema foi ignorado uma vez que o objetivo dos textos aleatóriamente escolhidos eram testar a capacidade das árvores em lidar com muitos dados.

Vale resaltar que na análise realizada não foram contabilizados as comparações realizadas para o balanceamento da árvore AVL, apesar de apresentar custo computacional, foi desconsiderada pois o objetivo foi as comparações no momento da buscas/inserção.

3 COMPILAÇÃO E EXECUÇÃO

Para compilar em qualquer distribuição Linux o leitor deve realizar os seguintes procedimentos:

1. Descompactar o aquivo TRABALHOPRATICO320152190100521521901011.tar.gz
2. Abrir a pasta TRABALHOPRATICO32015219010052152190101l.
3. No diretório Código Fonte abra o terminal e digite: *gcc main.c -o arvore.out*
4. E para executar digite: *./arvore.out entrada.txt*
5. Os resultados serão impressos na tela.

4 CONCLUSÃO

Ao término desta análise foi possível concluir que as árvores binárias são ferramentas altamente funcionais e faceis para implementação. Foi perceptivel também que com apenas algumas melhoras no método (balanceamento) é possivel diminuir muito o custo computacional quando se realiza uma busca/inserção em uma árvore. E apesar do custo para realizar o balanceamento ainda é muito viavél uma vez que sem balancear no pior caso poderiamos ter uma lista dinâmica encadeada que não seria nada eficaz no momento da busca e apresentaria um custo sempre linear para realizar a inserção.

Todas as teorias estudadas em sala de aula se comprovaram, principalmente as análisadas neste trabalho. Com tudo foi obtido dificuldades em tempo de processamento, pois os dois algortimos apresentaram tempo uniforme para todos os testes realizados. Ademais foi muito interessante trabalhar com árvores binárias manipulando strings, pois nos deu a oportunidade de conhecer a biblioteca *Ctype* que foi de grande ajuda e evitou que usássemos como de costume a tabela ASCII.

Referências

[1] UNICAMP. *Árvore Binária*. 2017. Disponível em: <http://www.ft.unicamp.br/liag/siteEd/definicao/arvore-binaria.php>.

[2] BACKES, A. *PROGRAMAÇÃO EM C DESCOMPLICADA*. 2017. Disponível em: <https://programacaodescomplicada.wordpress.com/sobre/about/>.

[3] USP, I. *Árvores Binárias*. 2017. Disponível em: <https://www.ime.usp.br/ pf/algoritmos/aulas/bint.html>.

[4] WIKIBOOKS. *Árvore Binária*. 2014. Disponível em: <https://pt.wikibooks.org/wiki/Programar$_e m_C$/$C381rvores_b inC3A1rias$>.