Relatório do RA2

Gabriel Vidal Andreoli

Primeiramente uma breve descrição do código criado para essa atividade.

O código é divido em 2 pacotes, o normal que possui o código para criar uma arvore binária normal, o Pacote AVL possui o código para criar uma arvore AVL

No pacote normal há as seguintes classes: Node, ArvoreBinariaNormal e Main

A classe node possui os atributos privados esquerda, direita e info que armazena o valor do nó. Nessa mesma classe há também o construtor do nó e os getters e setters dos atributos.

A classe ArvoreBinariaNormal possui como atributo um nó chamado de raiz, um construtor, getters da raiz e alguns métodos. Esses métodos consistem em dois para a inserção de elementos na arvore, onde um chamado apenas de recursao faz toda a parte lógica da inserção utilizando recursão para alocar os novos nós na arvore enquanto outro método chamado de inserir apenas chama esse método recursão, isso é feito para simplificar na hora de inserir elementos na arvore instanciada.

A logica para inserir os elementos na arvore é a seguinte, caso a raiz seja nula, ela vai ser instanciada e recebe como valor o valor passado como parâmetro, caso a raiz já exista é feita uma comparação que verifica se o valor a ser adicionado é maior ou menor que a raiz, caso seja menor é criado um nó na esquerda e caso seja maior é criado na direita, é utilizada a recursão para que o referencial da raiz se mova pela arvore até encontrar um nó adequado.

Há os métodos de emOrdem, preOrdem e posOrdem para printar a arvore e mostrar se os elementos adicionados estão dentro da arvore.

Há o método de busca na arvore, em que se insere um valor no parâmetro e é utilizado recursão para facilitar a busca, se um nó com info igual ao valor passado como parâmetro o método retorna esse nó, caso não seja encontrado nenhum nó com esse valor o método retorna null.

Há o método menor que é utilizado na remoção de elementos da arvore, sua função é retornar o menor número inteiro armazenado por um dos nós a direita do nó em que ele foi chamado.

Por fim há os métodos de remoção, assim como na inserção, a função de remoção é dividida em dois métodos, um que possui toda a parte lógica e outro que apenas chama esse método com a parte logica a fim de tornar a remoção mais simples no futuro.

Para remover um nó da arvore primeiro o método remover procura o nó que deve ser removido com a ajuda de recursão, ao encontrar o nó ele vai verificar se esse nó possui filhos, caso não possua ele apenas é removido, caso possua só um filho ele é substituído por esse filho e caso tenha 2 filhos é encontrado o menor nó a sua direita e é substituído por esse nó.

A classe Main só serve para instanciar a arvore e realizar os testes para o trabalho.

No pacote AVL há as classes Node, ArvoreAVL e Main.

A classe Node é muito parecida com a mencionada anteriormente, com a diferença de que possui um atributo de altura que será utilizado para calcular o balanceamento da arvore.

A classe ArvoreAVL possui como atributo uma raiz, possui um construtor e alguns métodos diferentes da ArvoreBinariaNormal.

Há o método altura que retorna a altura do nó que for passado como parâmetro, eu utilizo ela ao invés de usar o getter da altura do node pois evita erros de NullPointerException que podem ocorrer ao tentar pegar a altura de um nó que não existe.

Há também um método maior que recebe dois nós como parâmetro e retorna a altura do maior deles, é utilizado para realizar o balanceamento da arvore.

Há um método balanço que calcula qual é o balanço atual da arvore, caso seja menor que -1 ou maior que 1 a arvore sofrerá um rebalanceamento.

Há os métodos de giraDireita e giraEsquerda que servem para fazer o balanceamento da arvore, eles mudam as referências de memória de alguns nós para fazer a arvore girar.

A inserção é bem parecida com a de uma arvore normal, com exceção de que a cada inserção é realizado o balanço da arvore e dependendo do resultado ocorre o rebalanceamento.

Os métodos de busca, emOrdem, preOrdem e posOrdem não tiveram nenhuma mudança significativa.

A remoção também permanece muito parecida ao mostrado na arvore normal, com a diferença de que no método de recursão da remoção é realizada a verificação e rebalanceamento da arvore caso necessário.

A classe Main é igual à do pacote Normal.

Com o código explicado podemos prosseguir para os resultados obtidos na inserção de diferentes quantidades de elementos aleatórios inseridos, para realizar esse teste eu criei medi o tempo de execução de todas as inserções em nanosegundos e os resultados foram os seguintes (Lembrando que esses testes vão ter resultados diferentes de acordo com o hardware que for utilizado para a execução dos testes):

Para a inserção de 100 elementos:

Arvore normal: variou entre 407.500 e 632.700 nanosegundos

Arvore AVL: variou entre 716.200 e 1.568.700 nano segundos.

Para a inserção de 500 elementos;

Arvore normal: variou entre 904.900 e 1.667.400 nanosegundos

Arvore AVL: variou entre 2.020.400 e 2.761.900 nanosegundos.

Para a inserção de 1000 elementos:

Arvore normal: variou entre 915.100 e 1.179.900 nanosegundos

Arvore AVL: variou entre 2.330.900 e 3.042.700 nanosegundos

Para a inserção de 10.000 elementos:

Arvore normal: variou entre 3.223.050 e 4.942.700 nanosegundos

Arvore AVL: variou entre 8.249.800 e 10.136.000 nanosegundos

Para a inserção de 20.000 elementos:

Arvore normal: variou entre 6.856.400 e 9.087.100 nanosegundos

Arvore AVL: variou entre 14.339.600e 18.776.600 nanosegundos

Com isso, é possível expor que a inserção em uma arvore sem rebalanceamento é consideravelmente mais rápido que uma ArvoreAVL para a inserção.

Agora os testes de Remoção:

(esses testes foram feitos apenas com 10.000 e 20.000 elementos em cada arvore)

Remoção com 10.000:

Arvore Normal: a variação ficou entre 7.100 e 10.500 nanosegundos

Arvore AVL: a variação ficou entre 9.400 e 13.300 nanosegundos

Remoção com 20.000:

Arvore Normal: a variação ficou entre 7.000 e 14.500 nanosegundos

Arvore AVL: a variação ficou entre 12.500 e 20.700 nanosegundos

Testes De busca:

(esses testes foram feitos apenas com 10.000 e 20.000 elementos em cada arvore)

Busca com 10.000:

Arvore Normal: a variação ficou entre 8.700 e 10.900 nanosegundos

Arvore AVL: a variação ficou entre 7.100 e 9.200 nanosegundos

Busca com 20.000:

Arvore Normal: a variação ficou entre 8000 e 20.900 nanosegundos

Arvore AVL: a variação ficou entre 9.400 e 14.300 nanosegundos

Conclusões.

Com os testes realizados, é possível concluir que a arvore binaria normal possui um desempenho superior a uma arvore AVL tanto na inserção de um elemento quanto na remoção, isso se deve ao fato de que por não possuir um rebalanceamento, esse algoritmo se torna mais rápido de ser executado enquanto uma arvore AVL tem que ficar conferindo o balanço a cada elemento inserido e a cada recursão da inserção e remoção também são recalibradas as alturas dos nós, o que causa uma certa perda de desempenho.

Enquanto isso a arvore AVL consegue um desempenho superior ao realizar uma busca, pois não precisa conferir a altura dos nós nem o balanço da arvore, que já foram feitas durante as inserções e remoções. Enquanto a arvore binaria normal pode ficar bem desproporcional ao longo das inserções e remoções, a arvore AVL permanece sempre simétrica, o que a favorece ao realizar uma busca, o que tende a ficar mais nítido conforme se insere cada vez mais elementos.