UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

ESTRUTURAS DE DADOS BÁSICAS II

ÁRVORE BINÁRIA DE BUSCA ESTENDIDA

Carlos Antonio de Oliveira Neto
Jonathan Rocha de Almeida
Pedro Henrique Bezerra Cavalcante

NATAL 2016

INTRODUÇÃO

Este documento visa compartilhar uma explicação sobre a forma como desenvolvemos o algoritmo de busca binária estendida proposto pela professora Silvia Maria Diniz Monteiro Maia para disciplina Estrutura de Dados Básicas II do curso de Bacharelado em Tecnologia da Informação da Universidade Federal do Rio Grande do Norte.

Além da explicação da nossa abordagem na resolução do problema, serão também apresentadas as análises de complexidade assintótica dos métodos implementados.

ABORDAGEM DE SOLUÇÃO

Na criação de nossa árvore binária de busca estendida, optamos por utilizar a linguagem de programação Java. Criamos duas classes, uma classe chamada Driver para tratar das leituras dos arquivos de entrada e criar a árvore, e outra chamada No para realizar propriamente as operações na árvore criada.

Na classe Driver, criamos um JFileChooser, para simplificar a escolha dos arquivos de manipulação da árvore. Ao iniciar o programa, o JFileChooser irá requisitar a escolha de um arquivo texto de uma linha que contém os números que serão inseridos na árvore e depois irá requisitar um arquivo contendo as operações que serão feitas na árvore. Após os processamento dos arquivos será impresso no terminal a saída para todos os comandos, os quais podem ser:

ENESIMO N - Encontra o enésimo nó (contanto a partir do 1) no percurso em ordem simétrica da árvore e retorna seu valor (int).

POSICAO X - Retorna a posição do nó X no percurso em ordem simétrica da árvore, contando a partir do 1.

MEDIANA - Retorna o valor (int) do nó que está na posição média no percurso em ordem simétrica da árvore. Caso a árvore tenha um número par de elemento retorna o menor elemento.

CHEIA - Retorna true se a árvore é cheia, false se não é.

COMPLETA - Retorna true se a árvore é completa, false se não é.

IMPRIMA - Imprime os valores inseridos na árvore num percurso por nível.

REMOVA N - Remove o nó N da árvore.

Já a classe No é responsável pela estrutura da árvore. Para garantir a melhor complexidade temporal possível construímos os nós com os seguintes atributos:

valor - Inteiro armazenado no nó. (Valor único para cada nó).

pai - Aponta para o nó pai do atual, ou null se não houver um pai.

esq - Aponta para o nó filho esquerdo do atual, ou null, caso não o tenha.

dir - Aponta para o nó filho direito do atual, ou null, caso não o tenha.

nosEsq - Número de nós na sub-árvore esquerda do atual.

nosDir - Número de nós na sub-árvore direita do atual.

altura - Altura do nó atual.

ehCheia - Retorna se a árvore enraizada no nó atual é cheia.

ehCompleta - Retorna se a árvore enraizada no nó atual é completa.

É importante salientar que para a execução correta do algoritmo, os arquivos de entrada tem que seguir o seguinte padrão:

Arquivo com os números:

- Conter apenas uma linha formada por números separados por espaços simples;
- **Após o último número também deve haver um espaço simples.**

Arquivo com as instruções:

- É necessário ter o java instalado na máquina.
- Fornecer apenas uma instrução por linha.
- **Após as instruções que não possuem parâmetro deve haver um espaço simples.**
- Nas instruções com parâmetro só há o espaço entre a instrução e o número passado. Não há espaço após o número no parâmetro como no arquivo com os números.

Juntamente com os arquivos de código-fonte existem 3 arquivos de exemplificação de entrada: entrada.txt e instrucoes.txt para melhor entendimento.

A organização da hierarquia dos arquivos segue a seguinte estrutura:

Extended-Binary-Search-Tree
- README.md
- bin
  - Driver.class
  - No.class
- data
  - entrada.txt
  - instrucoes.txt
- src
  - Driver.java
  - No.java

Para executar via linha de comando:

Na pasta src:

```
javac Driver.java No.java -d ../bin
```

Na pasta bin:

```
java Driver
```

ANÁLISE DE COMPLEXIDADE ASSINTÓTICA

- atualizaAltura()
- atualizaCheia()
- atualizaCompleta()

Estes são métodos privados chamados durante a inserção de um nó para atualizar os atributos com os quais são nomeados de todos os nós no caminho de inserção de um novo nó. Deste modo suas complexidades no pior caso é O(h), onde h é a altura da árvore. Numa árvore ideal esta complexidade seria de log n.

- max()
- temSubarvoreVazia()
- temSubarvoreNaoCompleta()

Estes são métodos auxiliares chamados na execução dos métodos mostrados acima. Estes métodos fazem checagens simples em apenas um nó e por isso possuem complexidade temporal constante.

- corrigeEsqDir()
- corrigeEsqDirRemover()

Estes são métodos executados apenas quando uma inserção ou remoção falha com o intuito corrigir os atributos nosEsq e nosDir alterados no processo padrão de inserção ou remoção, que os modifica independente do sucesso dos mesmos. Para realizar esta correção eles percorrem todo o caminho de inserção ou remoção do nó e por esse

motivo possuem complexidade temporal O(h), onde h é a altura da árvore.

- adicionar()

Este é o método em que se concentra a maior parte da lógica de todo o sistema, uma vez que é a partir dele que todos os atributos de controle são alterados. Apesar de possuir uma complexidade considerável, sua execução percorre apenas o caminho de inserção do novo nó e, do mesmo modo, todos os métodos auxiliares chamados dentro dele também não fogem a essa regra. Deste modo sua complexidade também é O(h), muito embora o percurso h seja percorrido uma série de vezes para realizar a inserção.

- atualizaAlturaRemover()
- menorElemento()

Estes são métodos chamados exclusivamente dentro do procedimento de remoção de um nó. O atualizaAlturaRemover() é similar ao atualizaAltura() mostrado anteriormente, porém leva em conta nós que não possuem filhos (cenário impossível na inserção de um elemento). Já o método menorElemento() retorna o menor elemento de uma árvore e, para tal, deve percorrer, no máximo, h posições, onde h é a altura da árvore. Deste modo ambos estes métodos possuem complexidade O(h).

- remover()

Este é o segundo método mais complexo de toda a estrutura da árvore binária e nele são tratadas as exclusões de nós. Assim como no

adicionar(), a árvore é percorrida em apenas um de seus caminhos e sua complexidade, assim como a complexidade de seus métodos auxiliares, é O(h).

- procurar()

Este método verifica a existência de um elemento na árvore e, para tal, percorre apenas um caminho dentro da árvore, como já é comum nas árvore binárias de busca. Por esse motivo possui complexidade O(h).

- enesimoElemento(n)

Este método retorna o valor do nó na posição n numa leitura em ordem simétrica da árvore. Em uma árvore binária comum teria complexidade O(n), mas como esta fazer uso dos atributos nosEsq e nosDir descritos anteriormente este método teve sua complexidade otimizada para O(h), onde h é a altura da árvore.

- posicaoDosAncestrais()
- posicao(x)

Em conjunto estes dois métodos retornam a posição do nó x numa leitura em ordem simétrica da árvore. Ambos percorrem apenas o caminho do nó x até a raiz da árvore por conta das variáveis nosEsq e nosDir, que informam quantos nós teriam sido visitados anteriormente, proporcionando um cálculo mais rápido da posição do nó atual. Por este motivo possuem complexidade O(h), onde h é a altura da árvore.

- mediana()

Este método é apenas um caso especial do enesimoElemento(), para uma posição n que se encontra no centro de uma visitação em ordem simétrica da árvore. Por apenas chamar o enesimoElemento() para um número específico, eles possuem a mesma complexidade: O(h).

- ehCheia()
- ehCompleta()

Estes métodos foram substancialmente otimizados pela presença dos atributos que retornam, pois não calculam absolutamente nada, mas apenas retornam tais atributos, os quais são atualizados no corpo do inserir() e remover(). Deste modo a complexidade deles é constante.

- toString()

Este método percorre toda a árvore e então retorna uma string contendo todos os nós ordenados por nível. Por acessar todos os nós da árvore possui complexidade O(n) onde n é o número de nós.