UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DEPARTAMENTO DE INFORMÁTICA E MATEMÁTICA APLICADA
ESTRUTURA DE DADOS BÁSICA II

IANCO SOARES OLIVEIRA
LUCAS VINÍCIUS GÓIS NOGUEIRA

ANÁLISE DE COMPLEXIDADE DA ÁRVORE BINÁRIA DE BUSCA

NATAL/RN
2022

IANCO SOARES OLIVEIRA
LUCAS VINÍCIUS GÓIS NOGUEIRA

ANÁLISE DE COMPLEXIDADE DA ÁRVORE BINÁRIA DE BUSCA

Relatório técnico apresentado à disciplina Estrutura de Dados Básica II, como requisito parcial para obtenção da nota referente à Unidade II.

NATAL/RN
2022

SUMÁRIO

1. INTRODUÇÃO

Um problema computacional pode ser definido como um problema que possui entradas e retorna uma solução a partir de uma instância deste problema. É possível solucionar este problema a partir de algoritmos, que nada mais são do que uma sequência de passos bem definidos para resolvê-lo. Existem diversas maneiras de escrever um algoritmo que resolva o mesmo problema. Por isso, existe uma forma de avaliar diferentes algoritmos de modo que determine qual o mais eficiente, visto que a execução desses algoritmos demandam recursos do computador (processador, memória RAM, etc). Este método de análise de eficiência é a análise de complexidade.

Este relatório objetiva realizar análises de complexidade temporal das principais operações da Árvore Binária de Busca, sendo elas a busca, inserção, remoção e outras mais. Uma árvore binária é quando temos um nó que pode se conectar a até no máximo dois nós-filhos. Já a árvore binária de busca é uma árvore binária que se encaixa nas seguintes propriedades:

1) A raiz possui uma chave;
2) As chaves dos nós da subárvore esquerda da raiz são menores que a chave da raiz;
3) As chaves dos nós da subárvore direita da raiz são maiores que a chave da raiz:
4) As subárvoras esquerda e direita são árvores binárias de busca.

2. ANÁLISE DE COMPLEXIDADE DA ÁRVORE BINÁRIA DE BUSCA

Foram realizadas as implementações dos métodos a seguir utilizando, principalmente, a recursão como principal ferramenta.

Procedimento	Complexidade O	Discussão
buscar(int valor)	log(n) ou n	A complexidade de buscar() de uma árvore binária de busca com n elementos é relativa a sua altura h. No pior caso (em uma árvore zig-zag sua altura será n e no melhor caso, uma árvore completa, sua altura será log(n).
inserir(int valor)	log(n) ou n	O inserir é uma modificação do método buscar. Entretanto o inserir sempre irá até as folhas da árvore enquanto o buscar pode parar na raiz.
remover(int valor)	log(n) ou n	Assim como o buscar o remover tem sua complexidade relativa a altura da árvore, No caso do elemento a ser removido ter os dois filhos o método buscar também usa o método antecessor() que é um método que também depende da altura da árvore.
enesimoElemento(int x)	n	Esse método percorre a árvore de forma simétrica até achar o elemento com índice x, portanto sua complexidade em pior caso é linear.
posicao(int x)	n	Assim como o método enesimoElemento() o método posicao() faz o percurso simétrico da árvore para achar o elemento em uma determinada posição.
mediana()	n	Assim como os métodos anteriores, o mediana() percorre a árvore de forma simétrica, entretanto ele sempre irá

		percorrer metade dos elementos da árvore, tanto no pior quanto no melhor caso.
media(int x)	n	No pior caso, quando o valor passado for a raiz da árvore, o método irá percorrer todos os elementos da árvore para achar sua média, no melhor caso, quando o valor passado for de um nó folha, o método terá apenas que buscar o elemento tendo, assim, a complexidade correspondente a altura da árvore.
ehCheia()	n	Usa o método getAltura(), para o cálculo da altura da árvore esse método, por sua vez, é uma aplicação do percurso em pós ordem. Como qualquer algoritmo que faz um percurso na árvore é um método linear então o getAltura() é linear e, portanto, o ehCheia() é um método linear.
ehCompleta()	n	Assim como o método ehCheia() o método linear usa o getAltura(), então ehCompleta() também é linear.
preOrdem()	n	Como preOrdem() percorre todos os elementos da árvore então ele é um método linear (em qualquer caso)
imprimeArvore(int s)	n	Assim como o método preOrdem() o método imprimeArvore() percorre todos os elementos da lista, portanto é linear.