RELATÓRIO EBD II

Componentes:

Yago Beserra Marques

Análise assintótica e descrição da resolução:

- **Inserção**: O algoritmo de inserção procura o local onde o nó que desejamos inserir deveria estar. Aproveitando o fato de ser uma ABB, vamos percorrendo recursivamente para a esquerda ou direita (caso o valor do nó seja menor ou maior, respectivamente) até encontrarmos um local vazio. Se encontramos alguém de mesmo valor lá, apenas ignoramos, pois não pode ter mais de um valor na ABB. Note que esse algoritmo parece ser O(log n), o que pode ser! Caso a árvore seja completa, visto que na recursão nó sempre estamos dividindo a árvore em 2. Mas note também que se a árvore não for completa, teremos O(n).
- **Busca**: O algoritmo funciona de maneira análoga a inserção! Compara o nó com a raiz, se igual retorna-o. Se menor, chama recursivamente o algoritmo de busca para o filho à esquerda, se maior, chama recursivamente o algoritmo de busca para a direita. Assim até encontrar um nó nulo. Portanto, a complexidade é igual a da inserção.
- **Remoção**: Análogo ao algoritmo de busca, o algoritmo de remoção vai percorrendo a ABB (mesmo esquema recursivo de ir para a esquerda e direita). Assim, quando encontramos, temos três situações: o nó é folha, tem um filho, tem dois filhos. Podemos agrupar o caso 1 e 2, retornando o filho esquerdo se o direito for nulo, e o direito se o esquerdo for nulo (note que se os dois forem nulos, retornará nulo também, resolvendo o 1 caso). Para o caso 3, é substituído pelo elemento à direita da sub-árvore à esquerda (que chamei de elemento bilbo) e chama a remoção para remover esse elemento bilbo a partir do nó atual. Note que passamos por um nó somente uma vez e note também que da remoção passamos do nó encontrado para baixo para removê-lo, assim, não há desperdício, e a complexidade assintótica fica igual a da busca e inserção, O(n).
- **ToString**: Nesse algoritmo percorremos a árvore por nível. Para isso, utilizamos uma fila, pois assim colocamos na ordem e no mesmo laços os dois filhos do nó em questão na fila. Assim, basta irmos retirando o nó da fila e colocando em uma string de saída, retornando-a no final. A complexidade deste algoritmo é O(n), pois passamos uma vez por cada nó (n é a quantidade de nós).
- **N-ésimo Elemento**: A lógica desse é fazer um percurso Inorder iterativo, com um contador incrementando a cada laço. Assim, só precisamos comparar se o contador é igual a N, e retornar o nó que estava no lao no momento. Para fazer o percorrimento in-order (peça fundamental do algoritmo) utilizei uma

Pilha, adiconando sempre o filho à esquerda até que o filho da esquerda seja null, assim retiramos um elemento da pilha, verificamos se ele é o nó que queremos encontrar (retornamos-o se sim) e, caso negativo, pegamos o seu filho à direita e retomamos o processo de ir à esquerda até o final da ABB. Note que só passamos por um nó apenas uma vez, portanto a complexidade assintótica é de O(n).

- **Mediana**: Não consegui fazer da maneira que a senhora pediu. Mas preferi fazer da maneira “errada” e entregar do que enviar em branco! Nesse algoritmo, utilizei uma função recursiva para contar a quantidade de nós da ABB. Depois disso, chamei a função N-ésimo elemento com a metade da quantidade de nós, portanto ela é O(n-ésimo elemento) = O(n).
- **Posição**: É análogo à função N-ésimo elemento, a única diferença é que comparamos o valor da raiz atual com o valor passado por parâmetro (que seria o nó, mas para fins de simplificação só passei o valor) e se for igual retorna o contador. Se chegar ao final e não tivermos encontrado retorne -1 para saber que não obtivemos sucesso. Pelos mesmos motivo da função N-ésimo elemento, a complexidade assintótica desse algoritmo é O(n).
- **ehCheia**: Esse é um algoritmo recursivo que retorna verdadeiro se chegamos em alguma folha e falso se uma dos seus filhos não for nulo, quando ele tem dois filhos é chamada a recursão. A recursão é feita chamando ehCheia para o filho à esquerda e à direita da raiz. Depois é feito um & lógico para compararmos se as duas sub-árvores são cheias também. Note que só passamos uma vez por cada nó, assim, a complexidade é O(n).