Relatório Árvore Binária de Busca

Adilson Paulo Da Silva Aquino
Cicero Paulino de Oliveira Filho
Felipe Marley de Oliveira Gomes

May 27, 2023

Breve introdução sobre o programa em Java e seu propósito.

Função Insert

- Descrição da função insert: Responsável por criar e inserir um novo valor em uma árvore binária de busca, entretanto essa ação só ocorre após verificar se o nó já existe na árvore.
- complexidade assintótica: A complexidade assintótica dessa função depende da altura da árvore e pode variar de O(log n) a O(n), onde n é o número de elementos na árvore.

Função Search

- Descrição da função search: Responsável por procurar um valor específico em uma árvore binária de busca, caso o encontre retornará um valor booleano indicando que o nó foi encontrado.
- complexidade assintótica: A complexidade assintótica dessa função também depende da altura da árvore e pode variar de O(log n) a O(n), onde n é o número de elementos na árvore.

Função Print

- Descrição da função print: A função print e suas funções auxiliares auxiliarPrint, serialize e barDiagram são responsáveis por imprimir a árvore binária de busca em diferentes formatos: como uma string serializada ou como um diagrama de barras.
- complexidade assintótica:
  - print: A complexidade dessa função é O(1), pois ela apenas chama a função auxiliarPrint.

- auxiliarPrint: A complexidade dessa função é O(1), pois a execução das operações condicionais if e else if é constante.
- serialize: A complexidade dessa função é proporcional ao número de nós na árvore, pois ela percorre recursivamente todos os nós. Portanto, a complexidade é O(n), onde n é o número de nós na árvore.
- barDiagram: A complexidade dessa função é proporcional ao número de nós na árvore, pois ela percorre recursivamente todos os nós para imprimir o diagrama. Portanto, a complexidade também é O(n), onde n é o número de nós na árvore. Portanto, a complexidade total do código é O(n), pois as funções serialize e barDiagram têm complexidade O(n), e as outras funções têm complexidade constante.

Função Remove

- Descrição da função remove: A função remove e seu método auxiliar backRemove são responsáveis por remover um nó específico de uma árvore binária de busca.

- complexidade assintótica:
  - remove: A complexidade dessa função depende da complexidade da função search, que é O(log n) no melhor caso e O(n) no pior caso.
  - backRemove: A complexidade dessa função depende do número de operações realizadas durante a remoção do nó. No pior caso, quando o nó a ser removido tem dois filhos, a função precisa encontrar o sucessor ou predecessor do nó. Assim, a complexidade de encontrar o sucessor ou predecessor é O(log n) no caso de uma árvore balanceada, e O(n) no caso de uma árvore desbalanceada. Portanto, a complexidade total do código de remoção é O(log n) no melhor caso (quando a árvore está balanceada) e O(n) no pior caso (quando a árvore está desbalanceada).

Função nElement

- Descrição da função nElemente: A função nElement e sua função auxiliar searchSymmetricOrder são responsáveis por encontrar o elemento que ocupa uma posição enquanto faz o percuso em ordem simétrica.

- complexidade assintótica: A complexidade dessa função depende da complexidade da função searchSymmetricOrder No pior caso, a função searchSymmetricOrder percorre todos os nós da árvore até encontrar o elemento desejado, o que pode levar à visita de todos os nós na árvore. Portanto, a complexidade total do código é O(n) no pior caso.

Função Position

- Descrição da função position: A função position e sua função auxiliar backPosition são responsáveis por encontrar a posição de um determinado valor em uma árvore binária de busca.
- complexidade assintótica:
  - position: A complexidade dessa função depende da complexidade da função backPosition
  - backPosition: A complexidade dessa função depende do número de operações realizadas durante a busca pelo valor. No pior caso, a função percorrerá todos os nós da árvore. Portanto, a complexidade é O(n).

Função Media

- Descrição da função media: A função media e sua função auxiliar countElements são responsáveis por calcular a média de todos os elementos de cada nó de uma árvore binária de busca.
- complexidade assintótica:
  - media: A complexidade dessa função depende da complexidade da função search e da função countElements.
  - countElements: A complexidade dessa função depende do número de operações realizadas durante a contagem dos elementos. A função percorre todos os nós da árvore e realiza a contagem de acordo com o parâmetro x fornecido. Portanto, a complexidade total do código é O(n) no pior caso.

Função Median

- Descrição da função median: A função median e sua função auxiliar backMedian são responsáveis por encontrar a mediana dos elementos em uma árvore binária de busca.
- complexidade assintótica:
  - median: A complexidade dessa função depende da complexidade da função countElements e da função backMedian.
  - backMedian: A complexidade dessa função depende do número de operações realizadas durante a busca pela mediana. A função position é chamada para encontrar a posição do nó atual, e sua complexidade é O(n). A função backMedian percorre os nós da árvore, comparando a posição do nó atual com a posição da mediana. A função é chamada recursivamente até encontrar a mediana. Portanto, a complexidade total do código é O(n^2) no pior caso.

Função PreOrder

- Descrição da função preOrder: A função preOrder e sua função auxiliar backPreOrder são responsáveis por percorrer a árvore em pré-ordem e retornar uma string com os elementos visitados.

- complexidade assintótica:

  - preOrder: A complexidade dessa função depende da complexidade da função backPreOrder.
  - backPreOrder: A complexidade dessa função depende do número de operações realizadas durante o percurso em pré-ordem da árvore. A função percorre todos os nós da árvore uma vez, adicionando cada elemento visitado ao StringBuilder. Portanto, a complexidade é O(n).

Função IsFull

- Descrição da função isFull: A função isFull e sua função auxiliar backIsFull são responsáveis por verificar se a árvore é uma árvore binária cheia, ou seja, se todos os nós possuem zero ou dois filhos.

- complexidade assintótica:

  - isFull: A complexidade dessa função depende da complexidade da função backIsFull.
  - backIsFull: A complexidade dessa função depende do número de operações realizadas durante a verificação da propriedade de árvore binária cheia. A função percorre todos os nós da árvore, verificando se cada nó possui zero ou dois filhos. No pior caso, em uma árvore totalmente cheia, a função visitará todos os nós. Portanto, a complexidade é O(n).

Função IsComplete

- Descrição da função isComplete: A função isComplete e suas funções auxiliares countNodes e backIsComplete são responsáveis por verificar se a árvore é uma árvore binária completa, ou seja, Se v é um nó com uma subárvore vazia, então v está no último ou no penúltimo nível de T.

- complexidade assintótica:

  - isComplete: A complexidade dessa função depende da complexidade das funções countNodes e backIsComplete.
  - countNodes: A complexidade dessa função depende do número de operações realizadas durante a contagem dos nós da árvore. A função percorre todos os nós da árvore uma vez, incrementando o contador. Portanto, a complexidade é O(n)

– backIsComplete: A complexidade dessa função depende do número de operações realizadas durante a verificação da completude da árvore. A função percorre todos os nós da árvore, comparando o índice do nó atual com o número total de nós. A função é chamada recursivamente para os filhos esquerdo e direito. No pior caso, a função visitará todos os nós da árvore. Portanto, a complexidade é O(n).