BST Trees and Time Complexity

April 11, 2024

Contents

1 Binary Search Trees

Binary Search Trees are a specific type of a binary tree where the right child of a node is greater than the parent node and the left child of the node is lesser than the parent. (It could also work with the places switched). The time complexity of searching for a value in a **binary search tree** is $O(h)$ where the variable h is the height of the tree itself which is the largest number of pointers followed from the top to the leaf node. In the case of a **balanced binary search tree**, the time complexity will be $O(h) = O(log(n))$.

1.1 Insert operation

When thinking about the insert operation, consider which nodes are smaller or larger than the value x?

Remember that in a binary search tree, the children of each node has a specific pattern where the left child is smaller than the parent and the right side would be larger than the parent.

1.2 Predecessor and Successor

The predecessor is the node with the largest value less than the input value. The successor is the node with the smallest value larger than the input

value. If you think of the nodes as being "ordered", these are just the nodes "before" and "after" the input value x. However, there are no specific rules to a binary search tree where the successor or predecessor nodes have to be at a specific place, they technically be anywhere in the tree.

1.3 Remove operation

The main concern of the remove operation is to preserve the BST property of the tree.

2 Big-O, Big-Ω, Big-θ

Big-O:
The Big-O notation is a representation of the time complexity of the worst case scenario. Remember, if $f(x) \in O(g(x))$, we can multiply a function $g(x)$ by some constant where it will be greater than or equal to $f(x)$.

Big-Ω:
The Big-Ω notation is a representation of the best case scenario. It would be the opposite of the Big-O notation so $f(x) \geq c \cdot g(x)$

Big-θ:
The Big-θ notation is a representation of the average case, so it would be a representation of two constants multiplied by $g(x)$ where $c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)$

In summary, consider Big-O to be the upper bound for $f(x)$ and the Big-Ω to be the lower bound where Big-θ is where the function $f(x)$ is bounded by both an upper and lower bound.

3 Space vs Time

In the same way we classify algorithms according to their time complexity, we can also classify them by their space complexity. **Spatial complexity** represents the memory requirements of a program. Often, computational efficiency comes at a cost of spatial complexity and the same vise versa. We're often more concerned about computational complexity than the spatial complexity.