DECEMBER 22, 2019 / #DATA STRUCTURES

Binary Search Tree Data Structure Explained with Examples

A tree is a data structure composed of nodes that has the following characteristics:

1. Each tree has a root node (at the top) having some value.
2. The root node has zero or more child nodes.
3. Each child node has zero or more child nodes, and so on. This create a subtree in the tree. Every node has it's own subtree made up of his children and their children, etc. This means that every node on its own can be a tree.

A binary search tree (BST) adds these two characteristics:

1. Each node has a maximum of up to two children.
2. For each node, the values of its left descendent nodes are

less than that of the current node, which in turn is less than the right descendent nodes (if any).

The BST is built up on the idea of the binary search algorithm, which allows for fast lookup, insertion and removal of nodes. The way that they are set up means that, on average, each comparison allows the operations to skip about half of the tree, so that each lookup, insertion or deletion takes time proportional to the logarithm of the number of items stored in the tree, `O(log n)`.

However, some times the worst case can happen, when the tree isn't balanced and the time complexity is `O(n)` for all three of these functions. That is why self-balancing trees (AVL, red-black, etc.) are a lot more effective than the basic BST.

Worst case scenario example: This can happen when you keep adding nodes that are *always* larger than the node before (it's parent), the same can happen when you always add nodes with values lower than their parents.

Basic operations on a BST

- Create: creates an empty tree.
- Insert: insert a node in the tree.

- Search: Searches for a node in the tree.
- Delete: deletes a node from the tree.

Create

Initially an empty tree without any nodes is created. The variable/identifier which must point to the root node is initialized with a `NULL` value.

Search

You always start searching the tree at the root node and go down from there. You compare the data in each node with the one you are looking for. If the compared node doesn't match then you either proceed to the right child or the left child, which depends on the outcome of the following comparison: If the node that you are searching for is lower than the one you were comparing it with, you proceed to to the left child, otherwise (if it's larger) you go to the right child. Why? Because the BST is structured (as per its definition), that the right child is always larger than the parent and the left child is always lesser.

Insert

It is very similar to the search function. You again start at the root of the tree and go down recursively, searching for the right place to insert our new node, in the same way as explained in the

search function. If a node with the same value is already in the tree, you can choose to either insert the duplicate or not. Some trees allow duplicates, some don't. It depends on the certain implementation.

Delete

There are 3 cases that can happen when you are trying to delete a node. If it has,

1. No subtree (no children): This one is the easiest one. You can simply just delete the node, without any additional actions required.
2. One subtree (one child): You have to make sure that after the node is deleted, its child is then connected to the deleted node's parent.
3. Two subtrees (two children): You have to find and replace the node you want to delete with its successor (the letfmost node in the right subtree).

The time complexity for creating a tree is `O(1)`. The time complexity for searching, inserting or deleting a node depends on the height of the tree `h`, so the worst case is `O(h)`.

Runtime

- Worst-case performance: `O(log n)`
- Best-case performance: `O(1)`
- Average performance: `O(log n)`

Where `n` is the number of nodes in the BST.

If this article was helpful, tweet it.

Learn to code for free. freeCodeCamp's open source curriculum has helped more than 40,000 people get jobs as developers.

Get started

Donations to freeCodeCamp go toward our education initiatives, and help pay for servers, services, and staff.

You can make a tax-deductible donation here.

What is Docker?

TCP/IP Model

RTF File

CSS Transition

How to Use Instagram?

MBR VS GPT

FAT32 Format

Error 503 Code

Windows Hosts File

Mobi to PDF