Intermediate Software Engineering

Intermediate-Software-Engineering-Summer-2022 (@ Intermediate)
Personal Member ID#: 52751

Binary Trees UMPIRE Cheat Sheet

Definition

Binary Search Trees (BST) are special cases of binary trees. Binary trees are trees where each element or node has no more than 2 children.

The tree below **is** a binary tree

```
        1
      /   \
    2       3
   / \
  4   5
```

Binary trees are composed of **nodes**, where each node in the tree has **at most 2 children**. Nodes typically hold the following properties:

- data element (usually an integer, but can be anything)
- left pointer
- right pointer

```
class TreeNode {
    int data;
    TreeNode left;
    TreeNode right;
}
```

`1` `2` `3` `4` `5` are the data value of tree nodes in the example above.

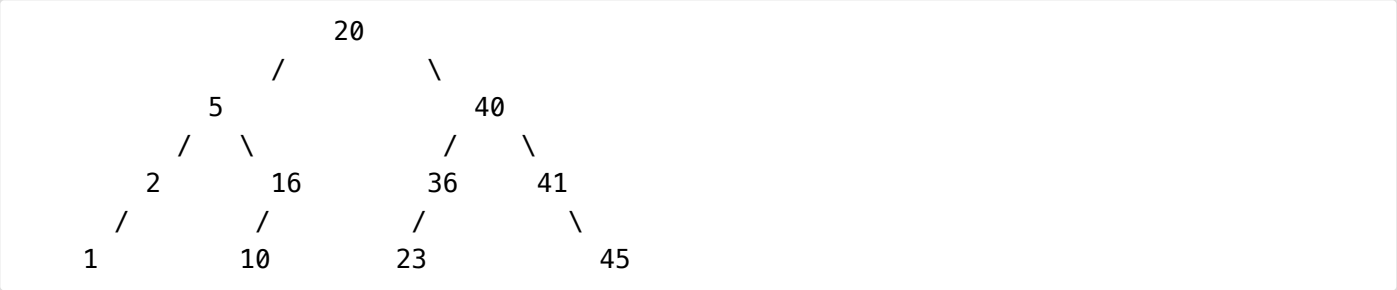
For example, the `root` is the base of the tree and the first node at the top. In our example, the node holding the data value `1` is the root. We can call the value of a Node following typical class syntax.

This tree, however, **is not** a binary tree because `1` has 4 children `2 3 4 5`

```
            1
      /   /   \    \
    2    3     4    5
```

BSTs are special in that for each node, all nodes to the left are less than or equal to it, and all nodes to the right are greater than it. Due to this property, **lookups in a BST take O(logN) time on average** if the tree is well-balanced.

If we were to put our winning lottery numbers into BST form, it could look something like this:
`1, 2, 5, 10, 16, 20, 23, 36, 40, 41, 45`



Pros and Cons

BSTs are the best options when trying to optimize for **efficient searching and flexible updates**. Unsorted linked lists have `O(1)` insertion and deletion operations, but require `O(N)` for search operations. If an array is sorted, searching can be `O(logN)`, but updating the array by adding or deleting elements is can be `O(N)` in the worst cases.

On the other hand, BSTs have more overhead and complexity to initialize and maintain. Arrays and linked lists are more straight forward due to their one dimensional structure. Trees require more thought due to its multi-dimension structure.

Check out CodePath guide on Binary Trees

U-nderstand

What are some common questions we should ask our interviewer?

- Are there memory constraints?
- What's the required time complexity?
- What kind of data will the inputs be?
- Can I assume all the inputs will be valid?
- What if the input is empty?
- What should we return if there is no solution to the problem?
- What should we return if there are multiple solutions to the problem?

M-atch

Are there any special techniques that we can use to help make this easier?

Prerequisites that you should be familiar with before: Recursion, stack, queue

- Some type of Tree Traversal: Pre-Order, In-Order, Post-Order, Level-Order
- Store nodes within a HashMap to refer to later
- Using Binary Search to find an element
- Applying a level-order traversal with a queue

- A basic instinct for solving DFS based questions is to do a recursive call and for all BFS(level order traversal) is to make queue and iterate, but also think upon how to iterate in DFS(Hint think on stack) and recurse in BFS based.

Common Questions

- Path problems
  - You are given root, you have to perform operations on a path, (path is root to leaf). Think upon the type of traversal you will apply when going from root to leaf.
- Lowest Common Ancestor
  - You are given two nodes and you have to return their ancestor at as least depth possible.
- Depth First Search
  - When the problem asks the traversal of a tree, you should think about Depth First Search (DFS) pattern and using it in combination with a recursive approach. Most tree DFS problems can be solved by in-order, pre-order, or post-order traversal. Hence, the first step to solve this problem is to decide which kind of traversal we should apply.
  - It is quite obvious that the in-order traversal is a more suitable way as a path contains nodes starting from root to leaf, which is a top-down traversal. Therefore, the algorithm process will be like:

```
See if the current node is a leaf and whether the accumulated sum, i.e. path sum, is
Recursively calculating the path sum of left subtrees and right subtrees.
```

P-lan/Pseudocode

- Can you create any **magic** helper methods that would simplify the solution?
- Talk through different approaches you can take, and their tradeoffs
- Be able to verbally describe your approach and explain how an example input would produce the desired output

Tips:

- Try to avoid nested loops
  - This is usually a brute force solution and is $O(n^2)$ time complexity

E-valuate

Binary Search Trees Time Complexity

	Average Case	**Worst Case**
Access	O(logn)	O(n)
Search	O(logn)	O(n)

	Average Case	**Worst Case**
Insertion	O(logn)	O(n)
Deletion	O(logn)	O(n)