Advanced Trees

Types of Advanced Trees

- Red-Black Trees
- 2-3-4 Trees
- B Trees

Reflection

1. **Compare the heights of the resultant trees – how do they compare with a Binary Search Tree (BST) for the same input values?**

The heights of the resultant trees are smaller than the height of the binary search trees. Due to the fact a binary search tree is NOT self-balancing - there may be certain instances where the tree is more left/right dominant. This leads to increase hight of the tree. Red-Black, 2-3-4 and B trees are all self-balancing, meaning nodes may split, expand and replace each other. This means the trees will always balance to a height height of (log n). Note that binary search trees can be (log n) height, but falter in worst-case scenarios.

2. **Compare the complexity of the algorithms, how much work would be required for the main operations: insert | find | delete? Compare this to BST.**
    - The insert, find and delete time complexity for a Binary Search tree has a worst case scenario of `O(h)` (h is the height of the tree), because of the binary splitting of levels generally a BST can run in `O(log n)` time. One potential disadvantage of this is that a BST can degrade over time into a degenerate tree, reducing search time to `O(n)`
    - A red-black tree shares the same time complexity of `O(log n)` for insert, find and delete
3. **Compare the understandability of the algorithms, which would be easier to implement?**
    - 2-3-4 Trees and Red-Black trees are equivalent data structures, how ever 2-3-4 trees can involve more complex node splits and expansions compared to Red-Blacks Trees, making them harder to implement
    - B trees are not as complex as 2-3-4 trees but not as simple as red-black trees. They are much simpler in terms of traversal as keys are stored in a sequential sorted order. This is especially useful for storing database and hard-drive data.
4. **Describe how an in-order traversal would work on each type of tree.**

Red-Black, 2-3-4 and B Trees allow for efficient in-order traversal just like any ordinary binary search tree. A recursive method can be used that searches starting from the left most leaf node, and traversing all sub trees, and will finish

once the right side sub tree is explored.

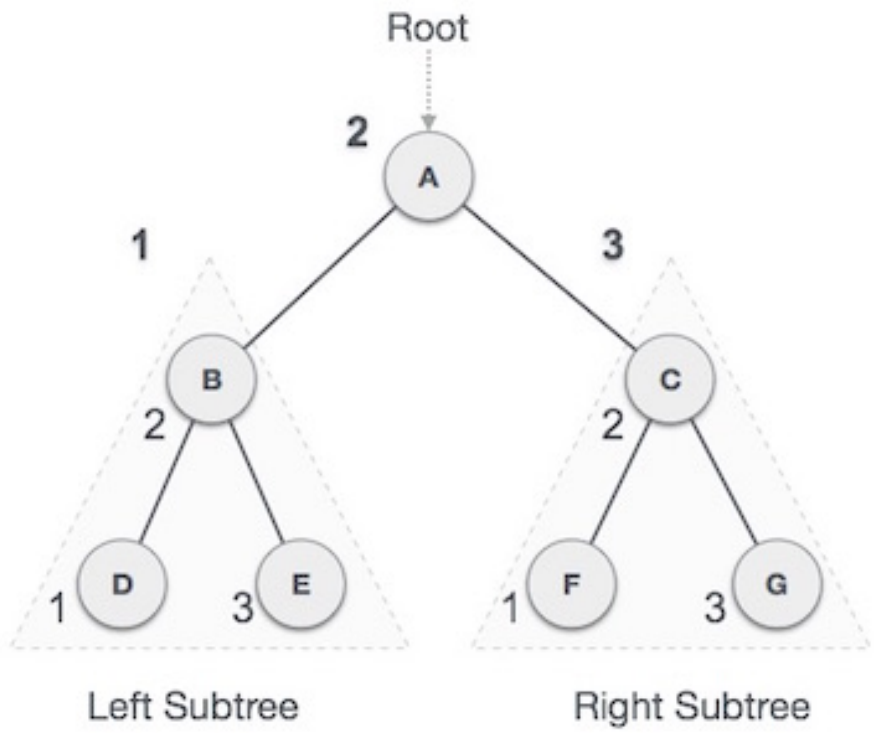

Figure 1: in-order traversal

Recap

Binary Search Trees run in O(log n) time. (fast) They can degrade to O(n) "linear time", depending on the data. (slow down) Eg: input of [12,13,14,17] will degrade a BST as they stack on the right, as elements are increasing in value. - To traverse to '14' it will take n(3) aka the height of '14' in the tree. - This is called a degenerate tree. - But we want a balanced/complete/semi-complete tree.

Red-Black Trees

2-3-4 Trees

- Each node can store 1,2 or 3 entries
- Number of children is equal to number of entries + 1, else its zero

Reference

https://www.tutorialspoint.com/data_structures_algorithms/tree_traversal.htm