Program for Interview Preparation

for doubts please contact *Kevin Paul* *or* *Utkrisht Sikka*

Week-5 - Binary Trees (BST, Traversal, AVL)

BST

A binary search tree (BST), also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left sub- tree and less than those in its right sub-tree. In case of duplicate elements in the tree, user can decide to include equal elements either in the left sub-tree or the right but should be consistent in the entire tree.

Problems

• Rearrange BST (Easy)

• Lowest Common Ancestor (Easy)

• BST Tilt (Easy)

• Distribute Coins (Medium)

• Greater Sum Tree (Medium)

Traversal

Traversal of a binary tree refers to **the process of visiting (e.g. retrieving, updating, or deleting) each node in that tree, exactly once.**

Depth First Traversals:

(a) Inorder (Left, Root, Right)
(b) Preorder (Root, Left, Right)
(c) Postorder (Left, Right, Root)

```
Algorithm for Inorder(tree) is as follows
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)


Similarly algorithm for preorder and postorder can be written.
```

Uses of Inorder

In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order.

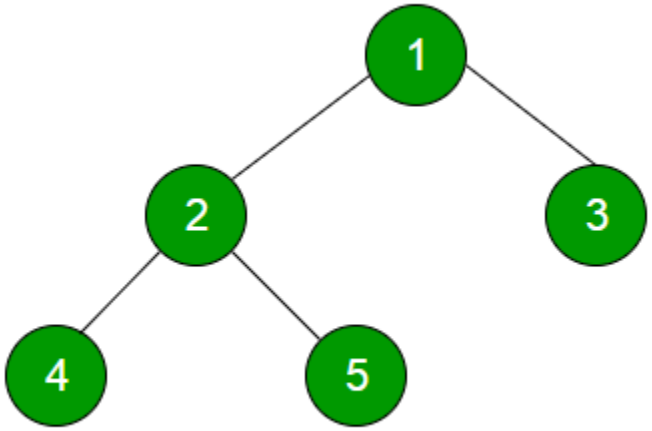
Uses of Preorder

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expression on an expression tree

Uses of Postorder

Postorder traversal is used to delete the tree. We should use the postorder transversal because before deleting the parent node, we should delete its child nodes first. Postorder traversal is also useful to get the postfix expression of an expression tree.

Breadth-First or Level Order Traversal:



Level order traversal of the above tree is 1 2 3 4 5
Basically we print nodes of current level before printing nodes of next level.

We can recover a tree completely if we are given its

1. inorder and preorder traversal or,
2. inorder and postorder traversal

Refer to this blog to know more about it
https://www.geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal/

Practice Questions:

1. https://leetcode.com/problems/binary-tree-inorder-traversal (easy)
2. https://leetcode.com/problems/binary-tree-level-order-traversal/ (medium)
3. https://leetcode.com/problems/binary-tree-maximum-path-sum (hard)
4. https://leetcode.com/problems/sum-of-distances-in-tree (hard)
5. https://leetcode.com/problems/delete-node-in-a-bst (medium)
6. https://leetcode.com/problems/recover-a-tree-from-preorder-traversal (hard)

AVL

Refer to [the MIT 6.006 video on AVL trees.]

An AVL tree allows the following operations in O(logN) time.

• find(x): Check if the value x exists in the AVL tree. • insert(x): Insert the value x in the AVL tree.

• delete(x): Erase the value x from the AVL tree.

The C++ set and Java TreeSet are implemented as balanced binary search trees, however, the balancing technique used is different from that used in AVL trees(specifically Red and Black Trees). They also support the 3 operations mentioned above in O(logN) time, as you see in section 1.

Consider the problem of finding the kth largest number number in an AVL tree. Can this be done in logarithmic time? How?

Implementation

Even though you almost never need to implement your own AVL Tree (or any other variant of Balanced Binary Search Tree), it is good to at least think about it's implementation. How will you implement an AVL tree? Give it a sincere thought.

(Taught in DSA course)