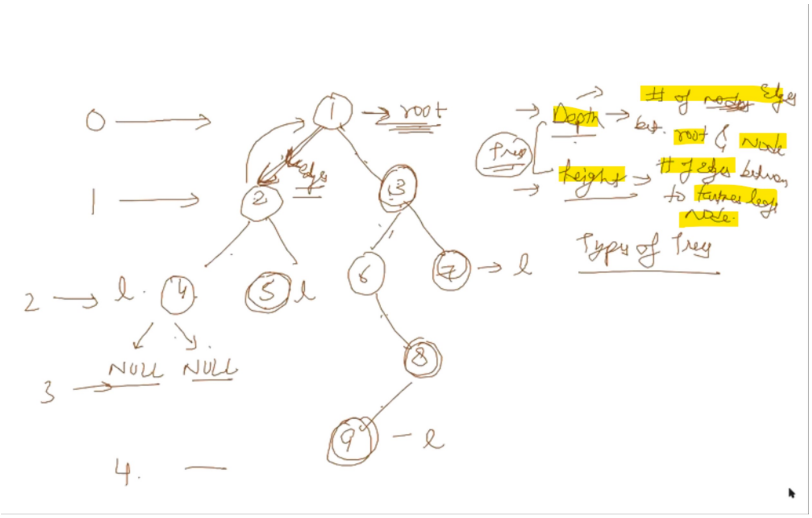
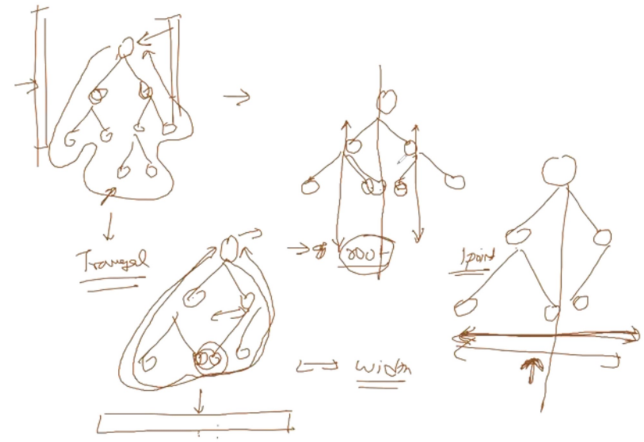
Trees Demux

18 October 2021 22:03

Topic	Question	Difficulty	Link	STATUS	Similar pblms
Trees	96.Unique BST	med	https://leetcode.com/problems/unique-binary-search-trees/		
	94. Inorder traversal binary tree	easy			https://leetcode.com/problems/binary-tree-level-order-traversal-ii/ || https://leetcode.com/problems/binary-tree-zigzag-level-order-traversal/ || https://leetcode.com/problems/n-ary-tree-level-order-traversal/
	102.level order traversal				
	105. Construct binary tree from preorder and inorder 106. Construct binary tree from postorder and inorder		https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/	Must To do	https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/
	101. Symmetric tree	easy	https://leetcode.com/problems/symmetric-tree/	To do	
	226. Invert binary tree			To do	
	104. Find max depth				https://leetcode.com/problems/minimum-depth-of-binary-tree/ https://leetcode.com/problems/time-needed-to-inform-all-employees/ https://leetcode.com/problems/balanced-binary-tree/
	1038. Binary search tree to greater sum tree				



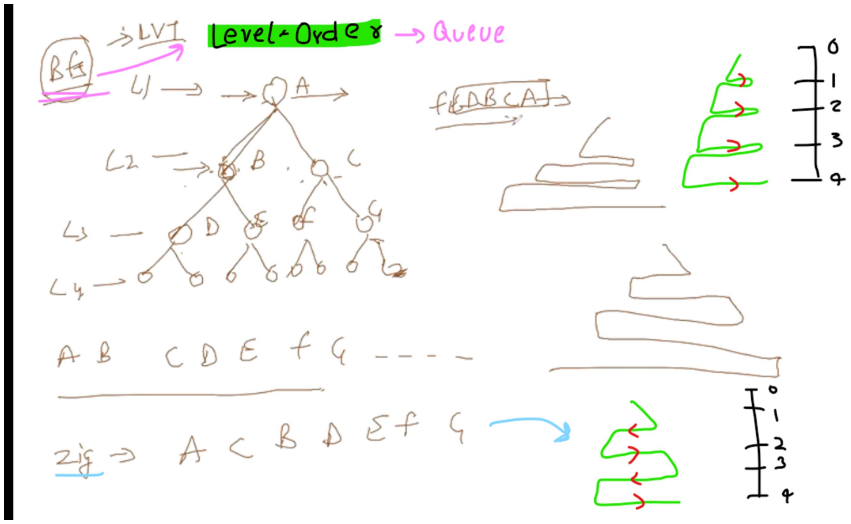
Depth of tree: # of edges between node and root of tree
Height of node(/tree) : # of edges between node(/root) and farthest leaf node
Examples : Height (3) = 3 , Height(1) =4 , Depth(3) = 1 , Depth(1)=0

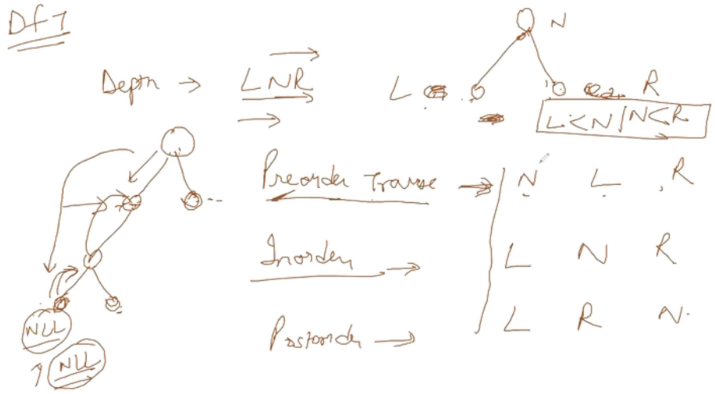


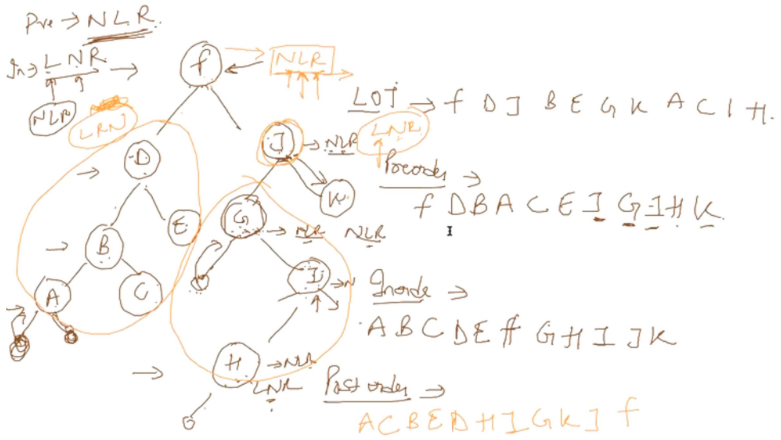
Diameter / Width of tree

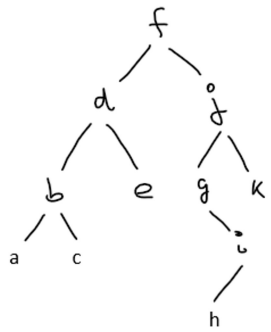
- BFS (in graphs) = Level Order Traversal (in trees)
- DFS (in graphs) = Preorder/Inorder/Postorder Traversal (in trees)

•

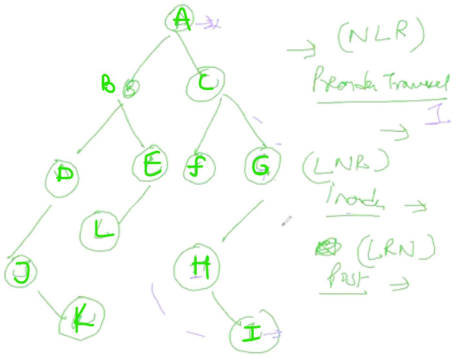

BFT
LVT
Level-Order
Queue
L1
A
L2
B
C
L3
D
E
F
G
L4
A B C D E F G ----
zig → A C B D E F G
fBDBCA
0
1
2
3
4


DFT
Depth → LNR
L
N
R
L<N/N<R
Preorder Traverse → N L R
Inorder → L N R
Postorder → L R N
NULL
NULL

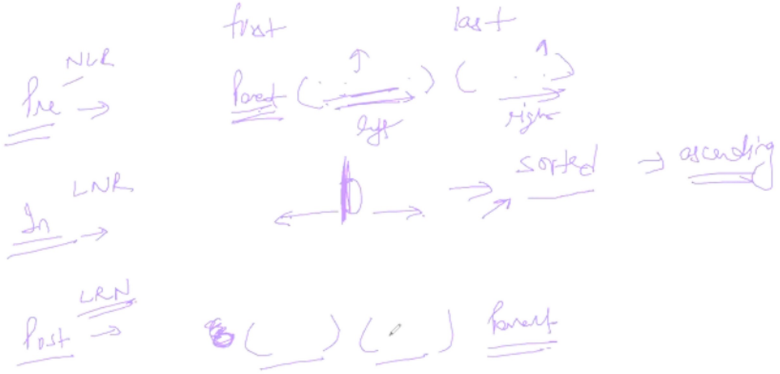

Pre → NLR
In → LNR
Post → LRN
NLR
LRN
LNR
LOT → f D J B E G K A C I H
Preorder → f D B A C E J G I H K
Inorder → A B C D E f f G H I J K
Post order → A C B E D H I G K J f



LOT(Level Order Traversal) : f,d,j,b,e,g,k,a,c,i,h
Inorder Traversal : a,b,c,d,e,f,g,h,i,j,k
Postorder Traversal : a,c,b,e,d,h,i,g,k,j,f
Preoder Traversal: f,d,b,a,c,e,j,g,i,h,k



Last element in Pre-order : **i**
Last element in In-order : **g** (Another way to look at the solution: Since g is the greatest element in the BST)
Last element in Post-order : **a**



---

96. Unique Binary Search Trees

Medium 3405 123 Add to List Share

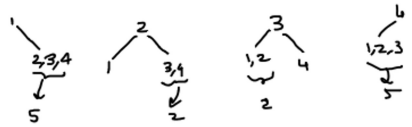
Given *n*, how many structurally unique **BST's** (binary search trees) that store values 1 ... *n*?

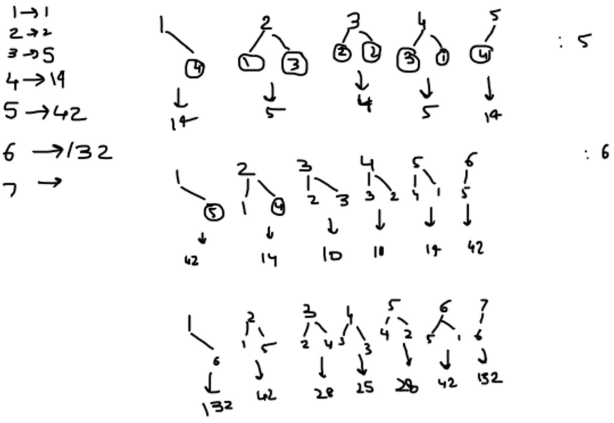
Example:

```
Input: 3
Output: 5
Explanation:
Given n = 3, there are a total of 5
unique BST's:

   1         3     3      2      1
    \       /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

Accepted 303.5K Submissions 578K

```
class Solution {
public:
    int numTrees(int n) {

    }
};
```





Success Details >

Runtime: 0 ms, faster than 100.00% of C++ online su
Unique Binary Search Trees.

Memory Usage: 6.1 MB, less than 49.57% of C++ or
Unique Binary Search Trees.

Next challenges:

Unique Binary Search Trees II

Show off your acceptance:

```
class Solution {
public:
    int numTrees(int n) {
        int dp[n+1];
        dp[0]=1;
        dp[1]=1;
        // dp[n] -> answer for n number of nodes
        for(int nodes=1;nodes<=n;nodes++){
            dp[nodes]=0;
            for(int root=1;root<=nodes;root++){
                dp[nodes]+=(dp[nodes-root]*dp[root-1]);
            }
        }
        return dp[n];
    }
};
```

Testcase Run Code Result Debugger

Time Submitted	Status	Runtime	Men
a few seconds ago	Accepted	0 ms	6.1 I
19 days ago	Accepted	0 ms	6.2 I

Success Details ›

Runtime: 4 ms, faster than 15.21% of C++ online submissions for Unique Binary Search Trees.

Memory Usage: 6.1 MB, less than 36.93% of C++ online submissions for Unique Binary Search Trees.

Next challenges:

Unique Binary Search Trees II

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
10/19/2021 01:57	Accepted	4 ms	6.1 MB	cpp
10/19/2021 01:56	Runtime Error	N/A	N/A	cpp
10/19/2021 01:31	Wrong Answer	N/A	N/A	cpp

```cpp
class Solution {
public:
    int numTrees(int n) {
        vector<int> dp(n+5,-1);int x,mid;
        dp[0]=0,dp[1]=1,dp[2]=2;
        for(int i=3;i<=n;i++){
            x=0;
             for(int j=1;j<=i;j++){ // root as j : 1....i/2-1
                x=x+(max(1,dp[j-1]) * max(1,dp[i-j]));
            }
            // for(int j=1;j<ceil((floor)(i+1)/2);j++){ // root as j : 1....i/2-1
            //     x=x+(max(1,dp[j-1]) * max(1,dp[i-j]));
            // }x*=2;
            // //for middle element
            // mid=dp[i/2];
            // if(i%2==0) mid*=2;
            // x+=mid;
            dp[i]=x;
        }
        // for(auto i:dp) cout<<i<<" ";
        return dp[n];
    }
};
```

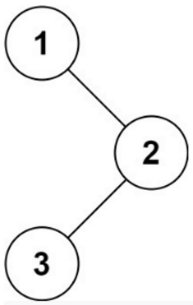
---

94. Binary Tree Inorder Traversal

Easy 5949 252 Add to List Share

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values.*

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,3,2]
```

Example 2:

LeetCode Explore Problems Interview Contest Discuss Store — October LeetCoding Challenge 2021

Description Solution Discuss (999+) Submissions — C++ — Autocomplete

Runtime: 4 ms, faster than 43.49% of C++ online submissions for Binary Tree Inorder Traversal.

Memory Usage: 8.4 MB, less than 25.44% of C++ online submissions for Binary Tree Inorder Traversal.

Next challenges:

Validate Binary Search Tree
Binary Tree Preorder Traversal
Binary Tree Postorder Traversal
Binary Search Tree Iterator
Kth Smallest Element in a BST
Closest Binary Search Tree Value II
Inorder Successor in BST
Convert Binary Search Tree to Sorted Doubly Linked List
Minimum Distance Between BST Nodes

Show off your acceptance:

Time Submitted	Status	Runtime	Memory	Language
10/19/2021 13:31	Accepted	4 ms	8.4 MB	cpp

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void inorder(TreeNode *node,vector<int> &ans){
         //base case
        if(node==NULL) return;

        //recursive traversal
        // left Root right
        inorder(node->left,ans);
        ans.push_back(node->val);
        inorder(node->right,ans);
    }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> ans;
        inorder(root,ans);
        return ans;
    }
};
```

//try iterative approach of the 3 traversals: inorder,preorder,postorder

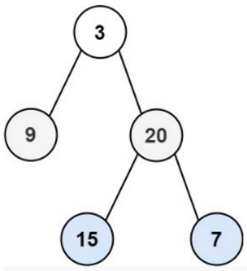
---

102. Binary Tree Level Order Traversal

Medium 6113 127 Add to List Share

Given the `root` of a binary tree, return *the level order traversal of its nodes' values.* (i.e., from left to right, level by level).

Example 1:



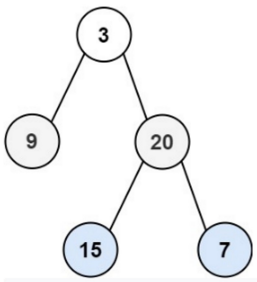

```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```


102. Binary Tree Level Order Traversal

Medium 6113 127 Add to List Share

Given the `root` of a binary tree, return *the level order traversal of its nodes' values.* (i.e., from left to right, level by level).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void levelTraversal(TreeNode* node,vector<vector<int>> &ans,int level){
        if(node==NULL) return;
        // TRICK: to overcome the need for size knowing of 2D ans vector in advance for initialization
        if(level==ans.size()) ans.push_back({});
        //above is step is important/makes the job easy, otherwise ,we had 2 options:
        //1. we first needed to find the height of tree (an extra time to find it)
        //2. or,we had to initialize: vector<vector<int>> ans(2005,vector<int> (0) ); and then trim off the empty lists at the end
        // [[3],[9,20],[15,7],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],...

        //recursive call
        ans[level].push_back(node->val);
        levelTraversal(node->left,ans,level+1);
        levelTraversal(node->right,ans,level+1);
    }
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> ans;
        // vector<vector<int>> ans(2005,vector<int> (0) );
        levelTraversal(root,ans,0);
        return ans;
    }
};
```

Q) To construct a binary tree from 2 of the following : preorder,inorder,postorder , Which of the combinations will work and why?

Ans)

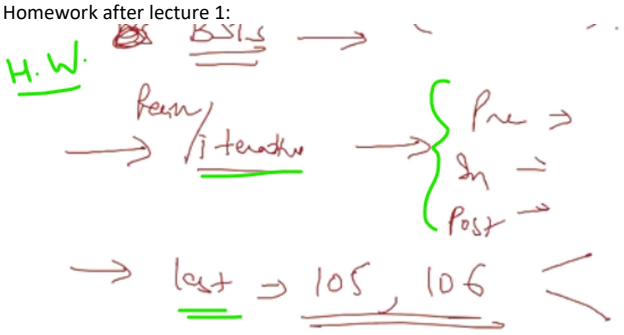
- Works: Inorder + Preorder
- Works: Inorder + Postorder
- NOT Works : Postorder + Preorder

Reason :

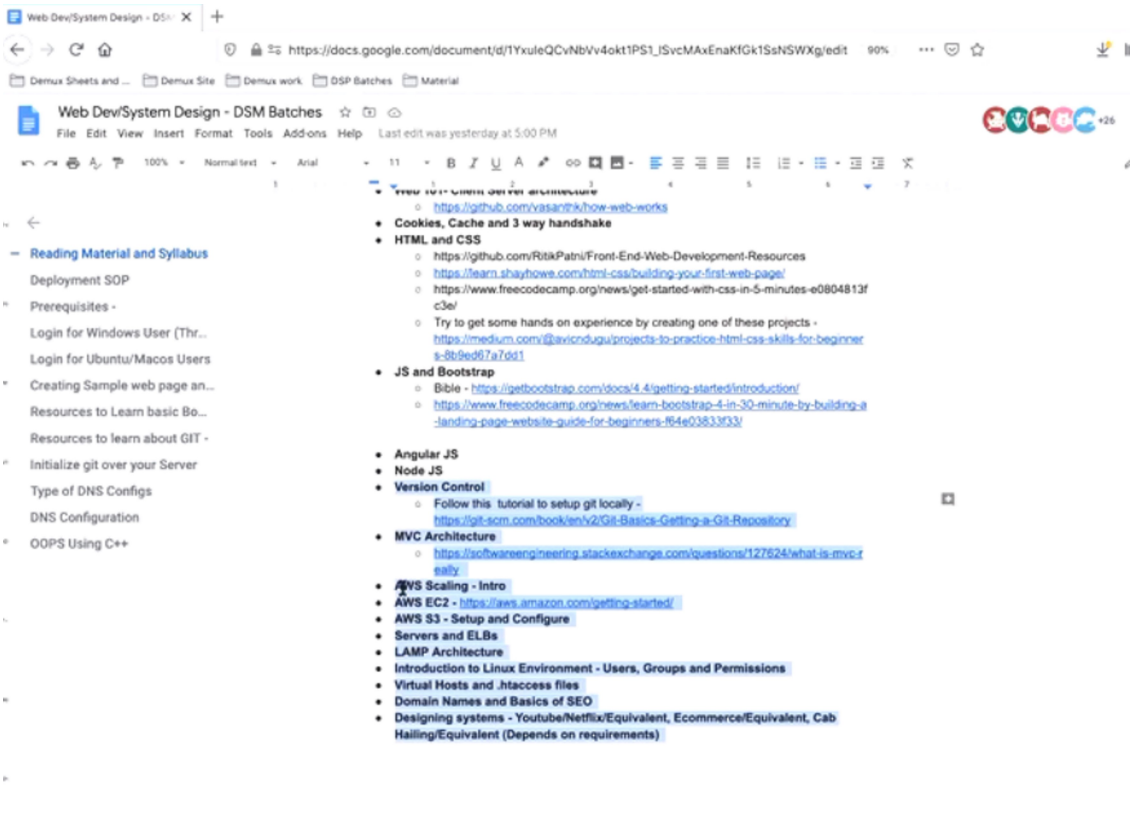
- we can get the **current root node** from either the preorder or postorder
- Now we need to differentiate between the left and right subtree elements for the current node ,which can be done only with help of inorder (left **node** right) ,where the **current_node** acts as a divider , We can then recurse the above 2 steps to build the binary tree

---

Homework after lecture 1:



1. Iterative approach for : pre,post,in -order traversals
2. Pblms : 105,106



Referrals -> same clg seniors
Learnings in **startups** very high: Meesho , khatabook, bounce , unacademy ; fintech : open,payu, bankbaja ; edtech : => email
Funding in last 6 months dekhlo , they will be on a hiring spree !

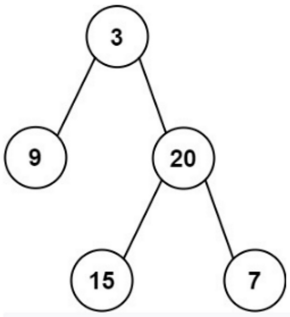
---

105. Construct Binary Tree from Preorder and Inorder Traversal

Medium 6704 165 Add to List Share

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree.*

Example 1:



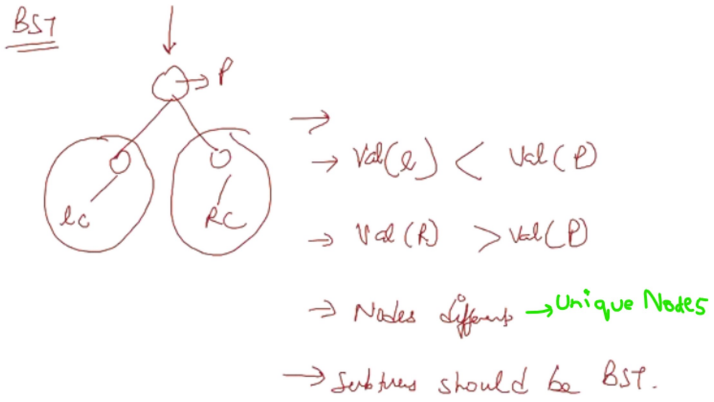

```
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
```

```
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        int rootIdx=0;//for preorder taversal
        TreeNode* ans = build(preorder,inorder,0,rootIdx,preorder.size()-1);
        return ans;
    }
    TreeNode* build(vector<int>& preorder, vector<int>& inorder,int left,int &rootIdx,int right){
        //base case
        if(left>right){
            return NULL;
        }

        int pivot=left;
        while(inorder[pivot]!=preorder[rootIdx]) pivot++;

        TreeNode* newNode=new TreeNode(preorder[rootIdx]);
        rootIdx++;//Also note : rootIdx must be passed by reference
        //reason: after building the leftSubtree of the root,
        // the rootIdx will become = no of elements in the left subtree

        //left subtree  => left ... pivot-1
        newNode->left=build(preorder,inorder,left,rootIdx,pivot-1);
        //right subtree => pivot+1 ... right
        newNode->right=build(preorder,inorder,pivot+1,rootIdx,right);
        return newNode;

    }
};
```

==========

Lecture 2

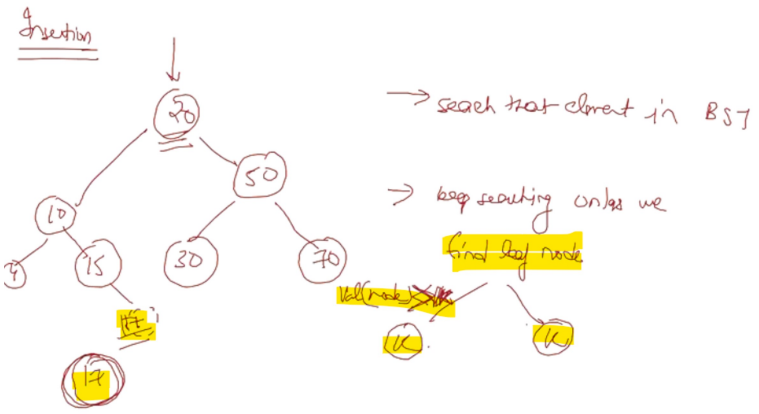
Binary Search Trees

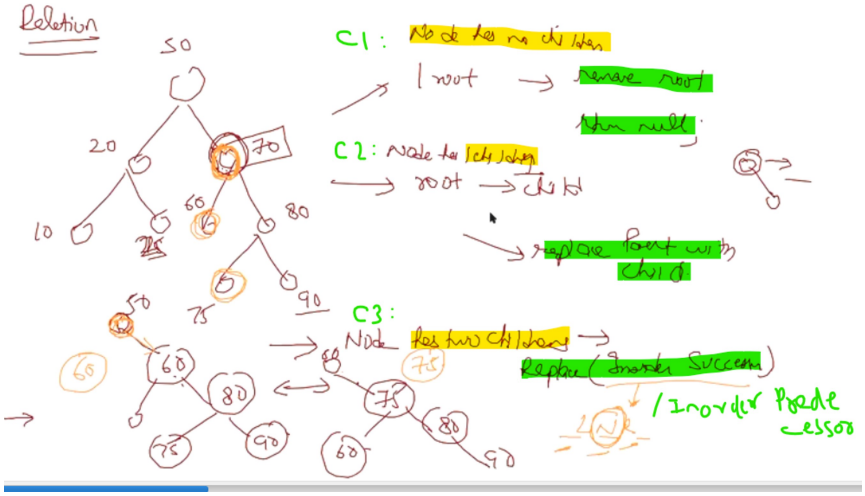


Data Structure	Pros	Cons
arrays	Search : O(log n) ,access : O(1)	Insertion/deletion : O(n)
Linked list	Insertion/deletion : O(log n)	Search : O(n) , access : O(n)
bst	Insertion/deletion : O(log n) ; Search : O(log n)	

NOTE: BST , overcomes cons of both arrays and linked lists as it provides both Searching and insertion/deletion in log n

- Searching
- Insertion , deletion in BST





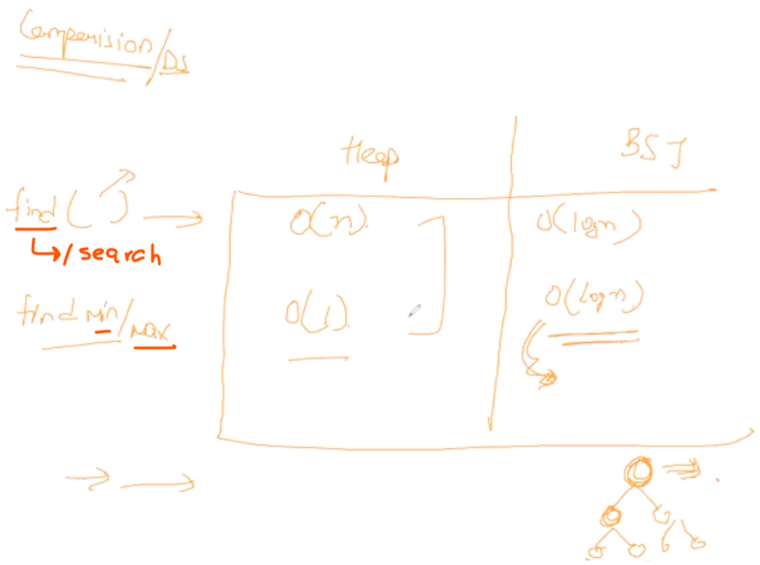
Deletion in BST : There are **3 cases:**

- Case 1 : Only 1 node => that is root => remove root; return NULL;
- Case 2 : Node/root has only 1 side children (i.e either only on the left side or only right side) => replace parent with child
- Case 3 : Node/root has children on both sides ,then we have 2 options:

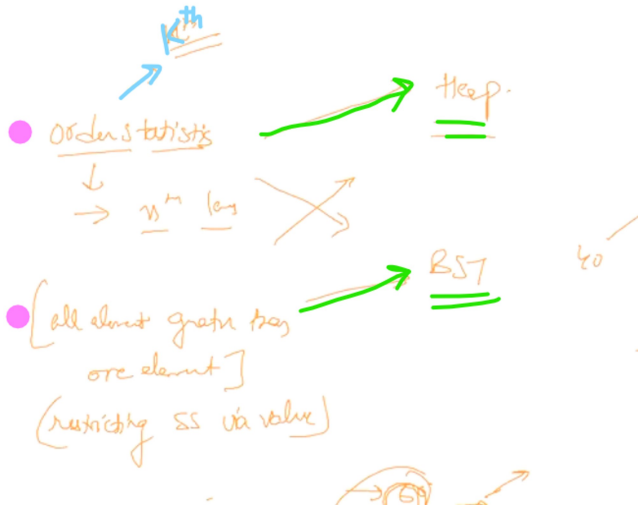
- Replace Root with Inorder Predecessor of the root (/largest element in the left subtree)
- Replace Root with Inorder Successor of the root (/smallest element in the right subtree)

Use Cases of BST :

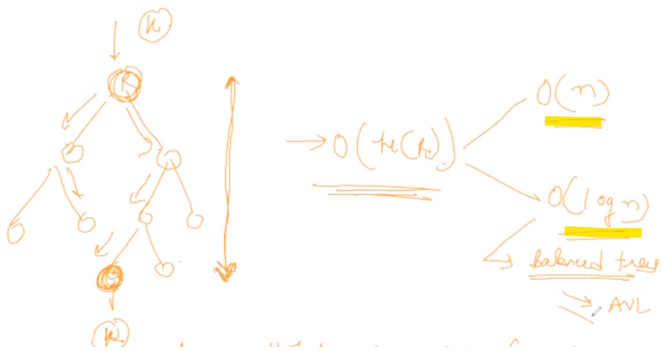
- Used in Compression algorithms ,for example: huffman coding
- In Router DNS configuration the data structure used is BST
- Custom functions using BST ,for example: sets,maps

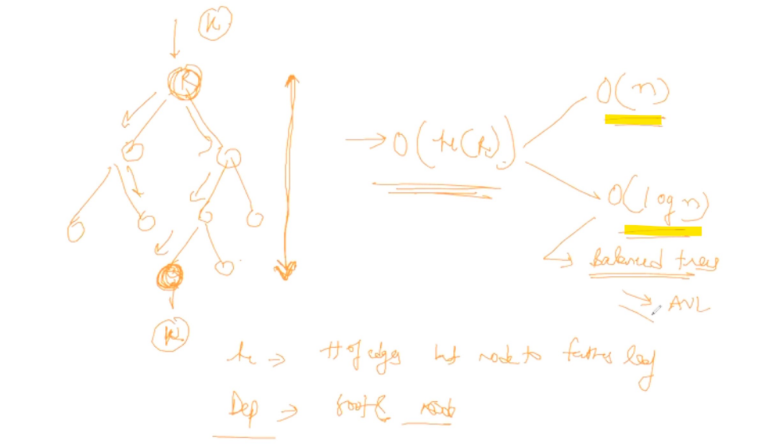


- When to use Heap?
  - K th order statistics => heap
- When to use BST?
  - Subproblem is : all elements greater(/smaller) than a certain element => BST

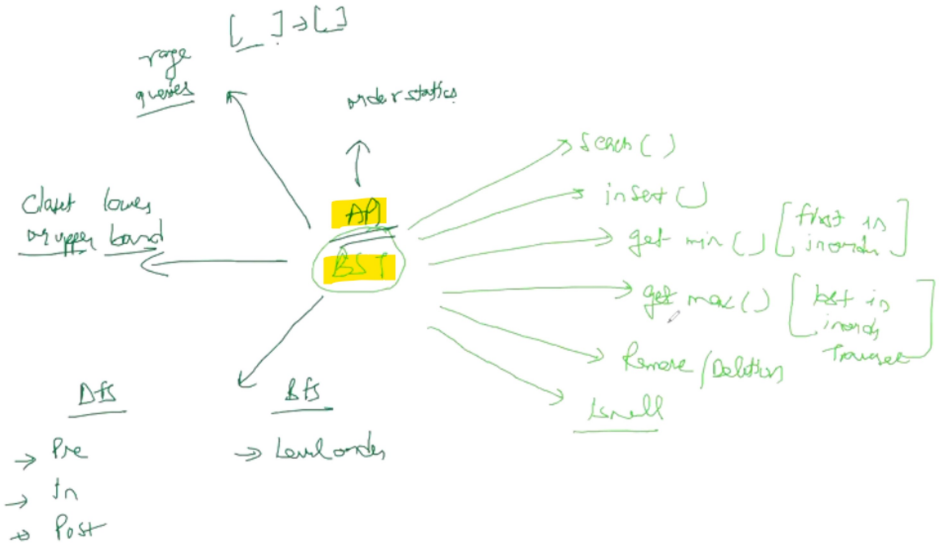


Note :
BST :: Avg Case to find an element in BST = O(log n) ; **Worst case** to find an element in **BST = O(n)** ; as BST maybe skewed/not balanced
Balanced BST :: AVL trees ; Red Black trees :: Avg Case to find an element in ***balanced BST*** = O(log n) ; ***Worst case*** to find an element in ***balanced BST*** = **O(log n)**
[Balanced BST : The difference between height of any 2 sister nodes (/nodes at same level) is at most 1]

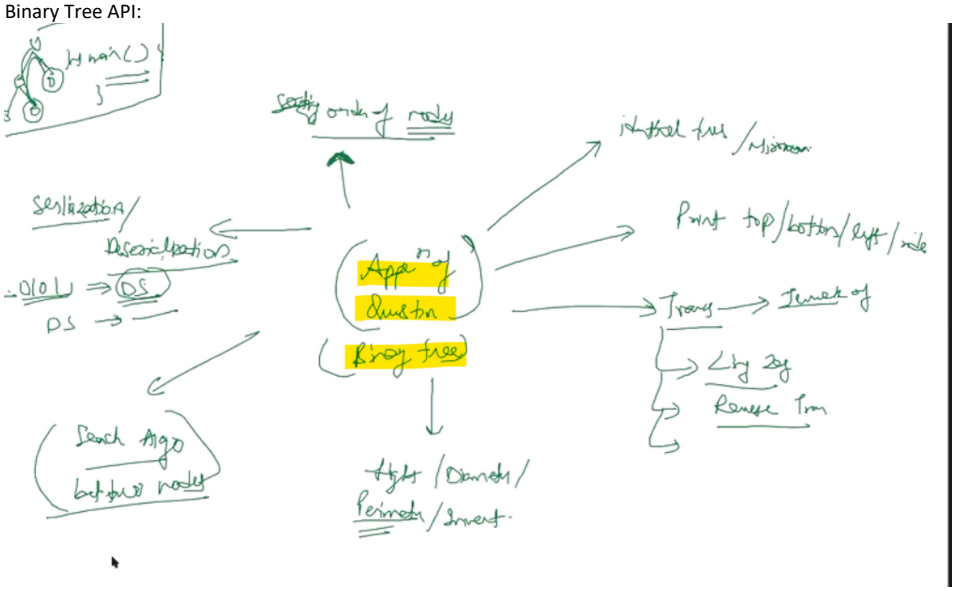




BST API:



Binary Tree API:



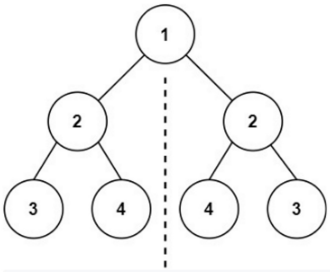
Leetcode 101: symmetric tree

101. Symmetric Tree

Easy 7598 191 Add to List Share

Given the `root` of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

Example 1:




```
Input: root = [1,2,2,3,4,4,3]
Output: true
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    bool isSymmetric(TreeNode* root) {

    }
};
```

Success Details >

Runtime: 8 ms, faster than 65.19% of C++ Symmetric Tree.

Memory Usage: 16.4 MB, less than 62.8 for Symmetric Tree.

Next challenges:

Remove Boxes

Convert Binary Search Tree to Sorted Doubl

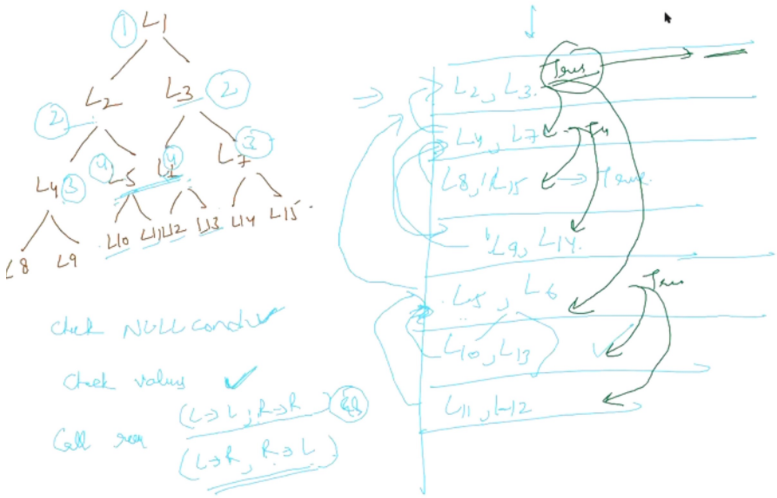
Distribute Coins in Binary Tree

Show off your acceptance:

Time Submitted Status R

```
class Solution {
public:

    bool isSym(TreeNode* l,TreeNode* r){
        // Terminating nodes
        if(l==NULL && r==NULL) return true;
        if(l==NULL || r==NULL) return false;
        // Compare the values;
        if(l->val!=r->val) return false;
        // l->left,r->right
        // l->right,r->left
        return isSym(l->left,r->right) && isSym(l->right,r->left);
    }


    bool isSymmetric(TreeNode* root) {
        if(root==NULL) return true;
        return isSym(root->left,root->right);
    }
};
```

Recursion stack for above pblm:



2:00:00 resume watching a bit b4
Leetcode 226. Invert binary tree

226. Invert Binary Tree

Easy 6506 92 Add to List Share

Given the `root` of a binary tree, invert the tree, and return *its root*.

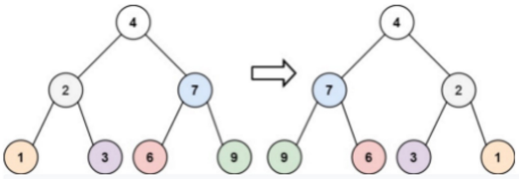
Example 1:

226. Invert Binary Tree

Easy 6506 92 Add to List Share

Given the `root` of a binary tree, invert the tree, and return *its root*.

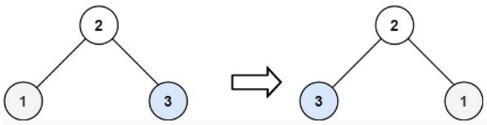
Example 1:



```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```


```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```


Example 2:



```
Input: root = [2,1,3]
Output: [2,3,1]
```

Example 3:

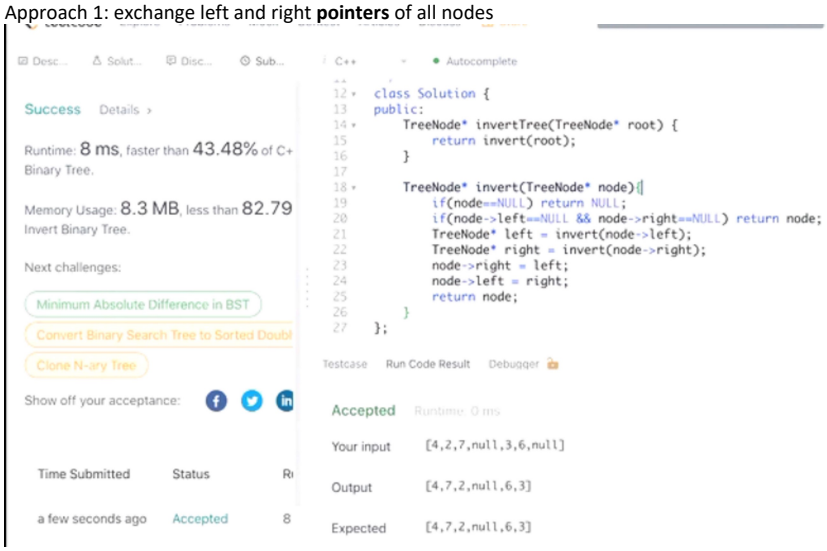
```
Input: root = []
Output: []
```

Constraints:

- The number of nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    TreeNode* invert(TreeNode* root,TreeNode* inverse){
        if(root==NULL)
    }
    TreeNode* invertTree(TreeNode* root) {
        if(root==NULL) return root;
        TreeNode* inverse;
        inverse->val=root->val;
        invert(root,inverse);
        return inverse;
    }
};
```

Approach 1: exchange left and right **pointers** of all nodes



Desc... Solut... Disc... Sub... C++ Autocomplete

Success Details >

Runtime: 8 ms, faster than 43.48% of C++ Binary Tree.

Memory Usage: 8.3 MB, less than 82.79 Invert Binary Tree.

Next challenges:

Minimum Absolute Difference in BST

Convert Binary Search Tree to Sorted Doubl

Clone N-ary Tree

Show off your acceptance:

Time Submitted	Status	R
a few seconds ago	Accepted	8

```
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        return invert(root);
    }

    TreeNode* invert(TreeNode* node){
        if(node==NULL) return NULL;
        if(node->left==NULL && node->right==NULL) return node;
        TreeNode* left = invert(node->left);
        TreeNode* right = invert(node->right);
        node->right = left;
        node->left = right;
        return node;
    }
};
```

Testcase Run Code Result Debugger

Accepted Runtime: 0 ms

Your input [4,2,7,null,3,6,null]

Output [4,7,2,null,6,3]

Expected [4,7,2,null,6,3]

Approach 2:
By swapping values and not the pointers (HOW??)

---

104. Maximum Depth of Binary Tree

Easy 2532 74 Add to List

Given a binary tree, find its maximum depth.

The maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.

Note: A leaf is a node with no children.

Example:

Given binary tree `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

return its depth = 3.

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {

    }
};
```

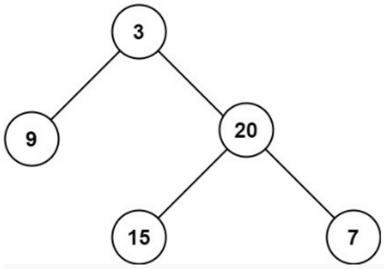
104. Maximum Depth of Binary Tree

Easy 5012 106 Add to List Share

Given the `root` of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: 3
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==NULL) return 0;
        return 1+max(maxDepth(root->left),maxDepth(root->right));
    }
};
```

Testcase | Run Code Result | Debugger

Accepted Runtime: 0 ms

Your input [1,2,3,4,5]

Output 3

Expected 3

1038. Binary Search Tree to Greater Sum Tree

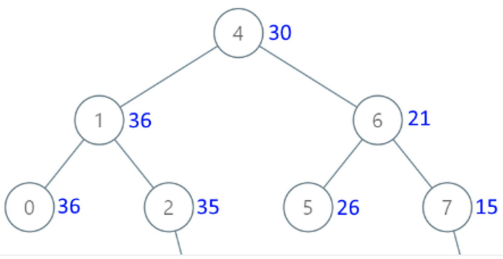
Medium 1982 120 Add to List Share

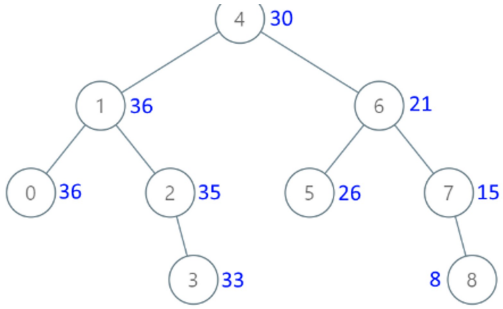
Given the `root` of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:






```
Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

Example 2:

```
Input: root = [0,null,1]
Output: [1,null,1]
```

Example 3:

```
Input: root = [1,0,2]
Output: [3,3,2]
```

Solution:

1038. Binary Search Tree to Greater Sum Tree

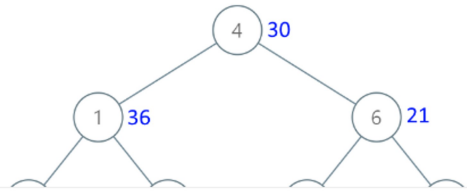
Medium 1982 120 Add to List Share

Given the `root` of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.

As a reminder, a *binary search tree* is a tree that satisfies these constraints:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void convert(TreeNode* root,int &x){
        //base case
        if (root==NULL) return;
        convert(root->right,x);
        root->val=root->val+x;
        x=root->val;
        convert(root->left,x);
        // root->val=root->val+x;
        // x=root->val;

    }
    TreeNode* bstToGst(TreeNode* root) {
        if(root==NULL) return NULL;
        int x=0;
        convert(root,x);
        return root;

    }
};
```

Note:
Leetcode pe tree k questions m directly node diya hua h , try tree questions where input (tree input) is in array form so do 2-3 pblms based on that input format also so that you are comfortable in interviews! Because in interviews : aise input bhi de skte h!

```
Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

Sample Question: https://leetcode.com/contest/weekly-contest-264/problems/count-nodes-with-the-highest-score/
