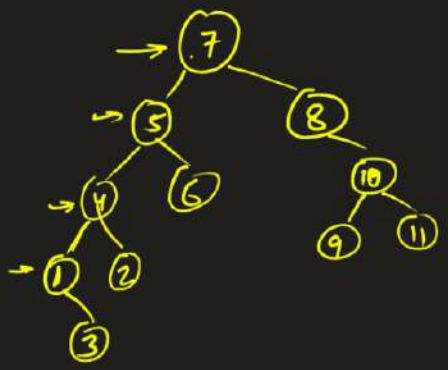
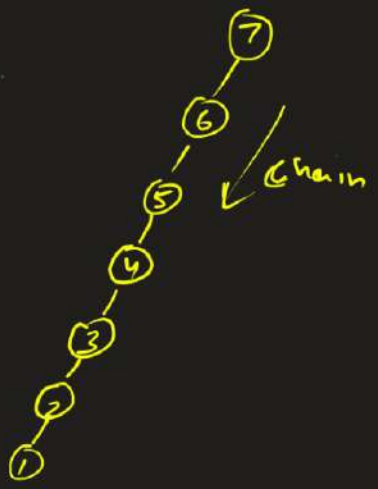
Trees & Graphs Lecture 2

Saturday, 17 August 2024 1:10 PM

Binary Search Trees



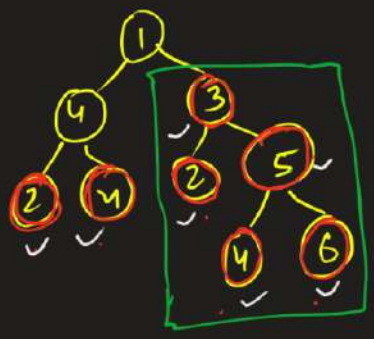


Worst Case :- O(n)

https://leetcode.com/problems/search-in-a-binary-search-tree/

```
class Solution {
public:
    TreeNode* searchBST(TreeNode* root, int val) {
        if(!root) return nullptr;
        if(val == root->val) return root;
        if(val < root->val) return searchBST(root->left, val);
        return searchBST(root->right, val);
    }
};
```

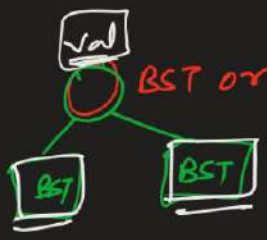
https://leetcode.com/problems/maximum-sum-bst-in-binary-tree/



7 BSTs Rooted at 2, 4, 2, 3, 5, 4, 6

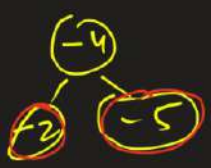
Highest Sum :- Rooted at 3

Sum = 20



val > left → val

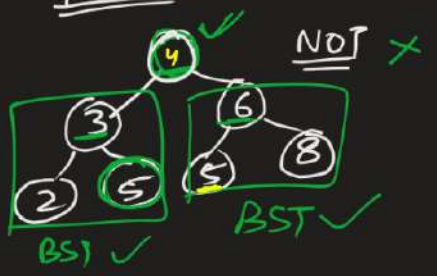
val < right → val



Brute :- For every node, check that the tree rooted at that node is BST or not. return the max sum BST

$O(n^2)$

→ Optimise



val > max (left subtree)

&& val < min (right subtree)

&& left & right should be BST

val > max left
val < min right
isBST left ✓
isBST right ✓

min = min(min left, min Right, val)
max = max(max left, max Right, val)
Sum = sum left + Sum Right + val
isBST = AND(_, _, _, _).

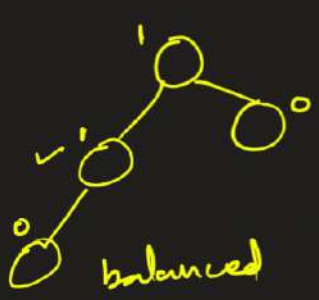
```python
class Solution:
    maxBstSum = 0

    def dfs(self, root):
        if not root:
            return {'max': -100000, 'min': 100000, 'sum': 0, 'isBST': True}
        l = self.dfs(root.left)
        r = self.dfs(root.right)
        ans = {}
        ans['max'] = max(l['max'], r['max'], root.val)
        ans['min'] = min(l['min'], r['min'], root.val)
        ans['sum'] = l['sum'] + r['sum'] + root.val
        ans['isBST'] = False
        if l['isBST'] and r['isBST'] and root.val > l['max'] and root.val < r['min']:
            ans['isBST'] = True
            self.maxBstSum = max(self.maxBstSum, ans['sum'])
        return ans

    def maxSumBST(self, root: Optional[TreeNode]) -> int:
        self.dfs(root)
        return self.maxBstSum
```

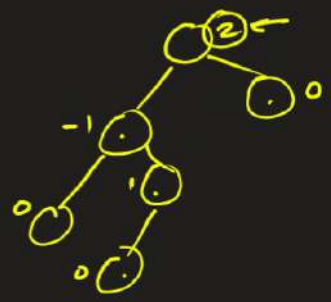
T = O(n)
S = O(n)
← map

Balanced BST

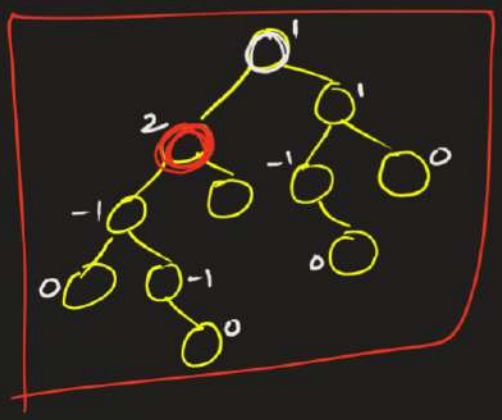
> At every node, height balancing factor $\boxed{-1 \leq bf \leq 1}$

Depth of left subtree − Depth of Right subtree





NOT balanced

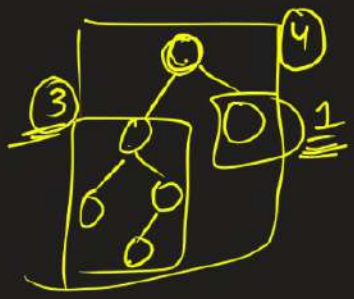


NOT balanced

https://leetcode.com/problems/balanced-binary-tree/

$T = O(n), \quad S = O(n)$

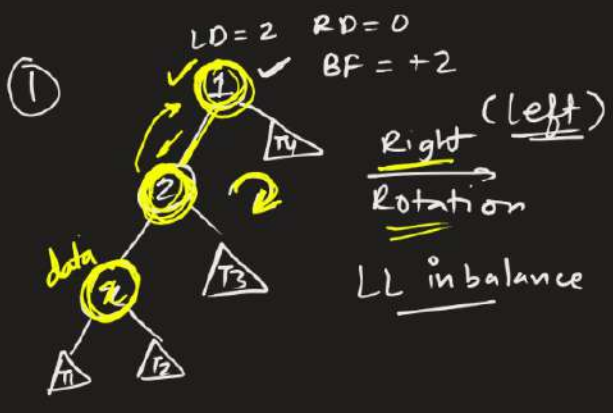
```python
class Solution:
    maxBalancingFactor = 0

    def dfs(self, root):
        if not root:
            return 0
        ld = self.dfs(root.left)
        rd = self.dfs(root.right)
        self.maxBalancingFactor = max(self.maxBalancingFactor, abs(ld-rd))
        return max(ld, rd)+1

    def isBalanced(self, root: Optional[TreeNode]) -> bool:
        self.dfs(root)
        return self.maxBalancingFactor <= 1
```

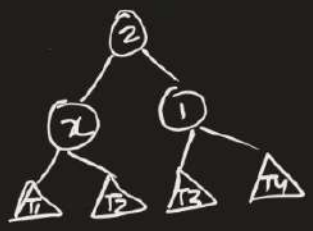


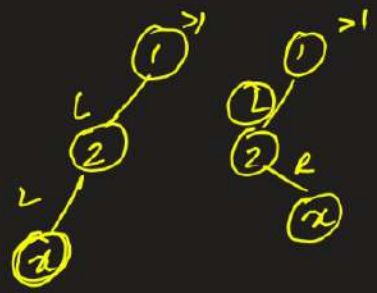
AVL Tree

↳ self balancing Binary Search Tree. $h = O(\log n)$

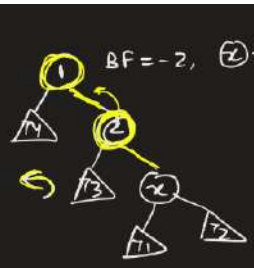
↳ Insert in BST



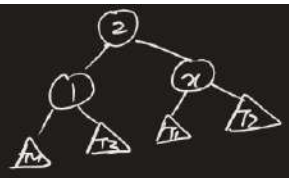




(2) BF = -2, (x) → Right of Right child

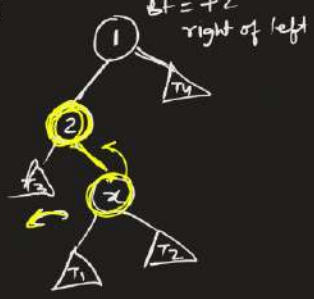


(Right) Left Rotation

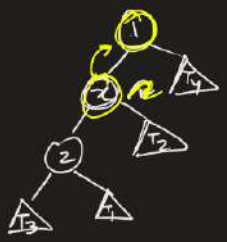


RR imbalance

(3) BF = +2, right of left

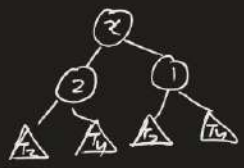


(Left) Left Rotate



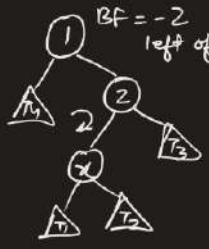
LR imbalance

node (~~Left~~) Right Rotate

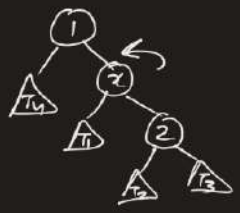


LL imbalance

(4) BF = -2, left of right



(Right) Right Rotate



Rt imbalance

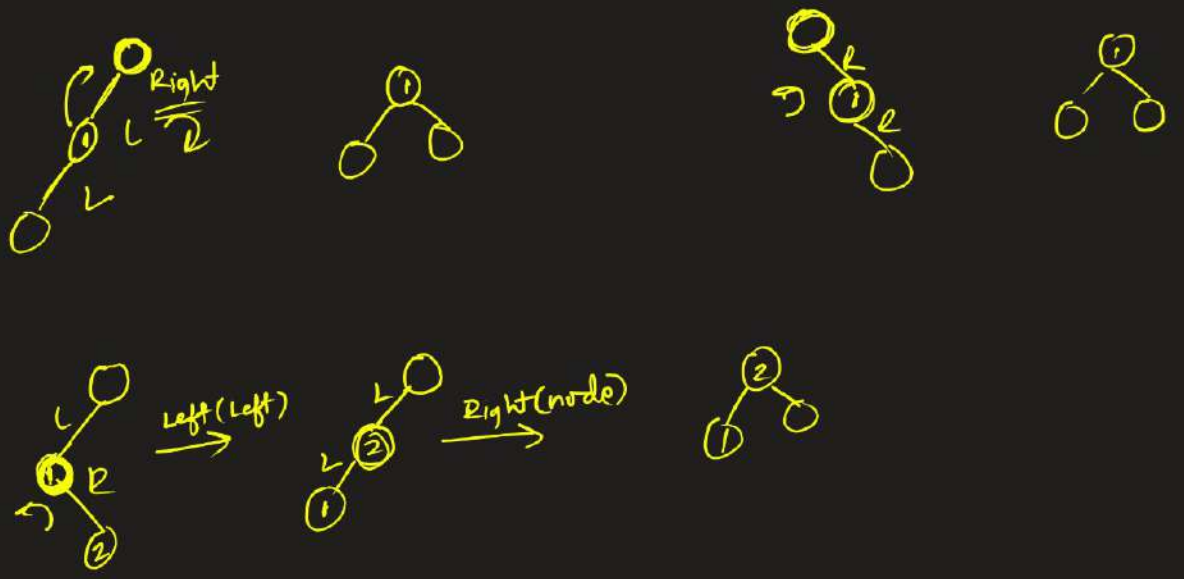
(Right) Left Rotate

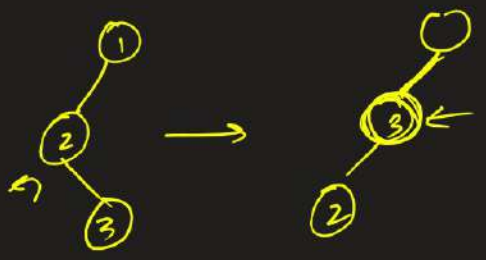
RR imbalance

Left Rotate ✓ Right Rotate ✓

Depth of subtree ✓

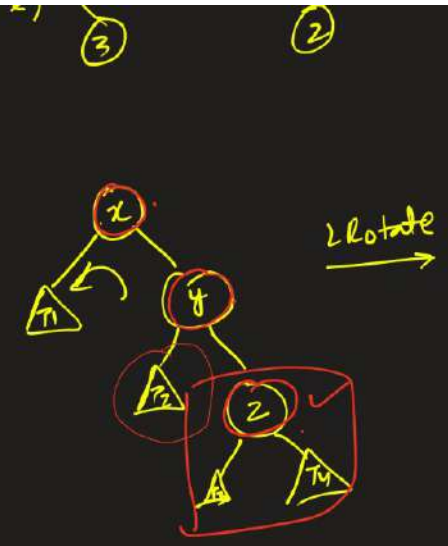
https://www.geeksforgeeks.org/problems/avl-tree-insertion/1



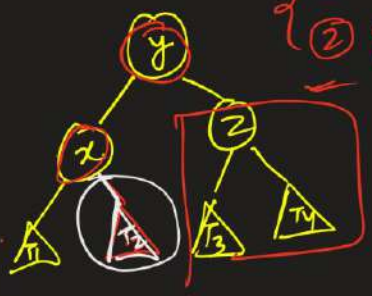


node → left = leftRotate (node → left)

(1) x → R = y → L

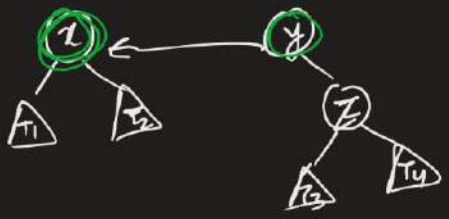


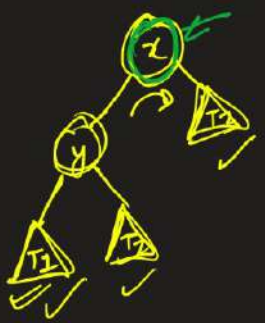
LRotate



① x → R = y → L
② y → L = x

(y) height = 1 + max(x height, z. height)
(x) height = 1 + max(T1. height, T2. height





② y → right = x
① x → left = y. right

```
class Solution{
    public:

    int height(Node* node) {
        if(!node) return 0;
        return node->height;
    }

    Node* leftRotate(Node* x) {
        Node* y = x->right;
        x->right = y->left;
        y->left = x;
        x->height = 1+max(height(x->left), height(x->right));
        y->height = 1+max(height(y->left), height(y->right));
        return y;
    }

    Node* rightRotate(Node* x) {
        Node* y = x->left;
        x->left = y->right;
        y->right = x;
        x->height = 1+max(height(x->left), height(x->right));
        y->height = 1+max(height(y->left), height(y->right));
        return y;
    }
```

```
    Node* insertToAVL(Node* node, int data) {
        if(!node)   return new Node(data);
        if(data < node->data)   node->left = insertToAVL(node->left, data);
        else if(data > node->data)   node->right = insertToAVL(node->right, data);
        else    return node;
        node->height = 1+max(height(node->left), height(node->right));
        int bf = height(node->left) - height(node->right);

        // L-L imbalance
        if(bf>1 && data < node->left->data) {
            return rightRotate(node);
        }
        // R-R imbalance
        if(bf<-1 && data > node->right->data) {
            return leftRotate(node);
        }
        // L-R imbalance
        if(bf>1 && data > node->left->data) {
            node->left = leftRotate(node->left);
            return rightRotate(node);
        }
        // R-L imbalance
        if(bf<-1 && data < node->right->data) {
            node->right = rightRotate(node->right);
            return leftRotate(node);
        }
        return node;
    }
};
```