05/05/2020

Threaded Binary Tree.

A binary tree is threaded by making all right child pointers that would normally be null point to the in-order successor of the node (if it exists), and all left child pointers that would normally be null point to the inorder predecessor of the node.

The idea of threaded binary trees is to make inorder traversal faster and do it without stack and without recursion.

Types of Threaded binary trees :

1. Single threaded : each node is threaded towards either the in-order predecessor or successor (left or right).

(a) Left in-threaded binary tree :

A left NULL pointer can be used as threads to store the address of inorder predecessor of the node then the binary tree is called a left in-threaded binary tree.

(b) Right in-threaded binary tree :

A right NULL pointer can be used as threads to store the address of inorder successor of the no then the binary tree is called a right in-threaded binary

2. <u>Double threaded</u> : each node is threaded towards both the in-order predecessor and successor (left and right)

i.e. If both left and right NULL pointers are used as threads to store the address of inorder predecessor and inorder successor then the binary tree is called a fully in-threaded tree or in-threaded or double threaded tree.

<u>Example.</u>

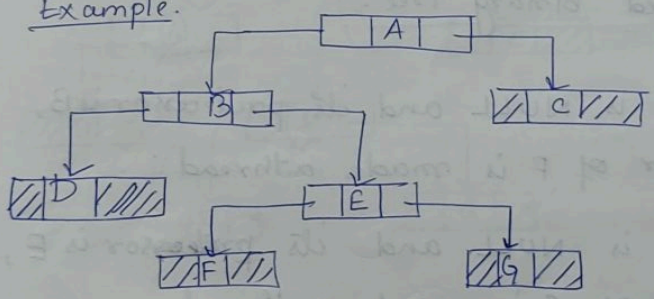


fig: <u>Binary tree without threads.</u>

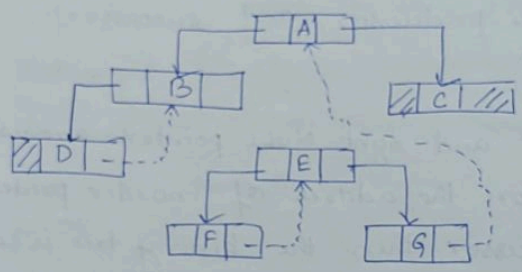
Here, Inorder traversal of the tree is DBFEGAC.

D's right link is NULL and its successor is B, so right pointer of D is made a thread.

F's right link is NULL and its successor is E, so right pointer of F is made a thread.

G's right link is NULL and its successor is A, so right pointer of G is made a thread.

Right in-threaded binary tree.



Inorder successor : D B F E G A C.

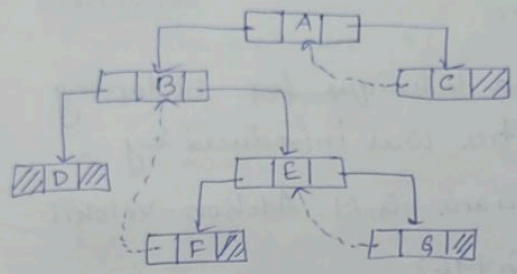
Left in-threaded binary tree.

F's left link is NULL and its predecessor is B, so left pointer of F is made a thread.

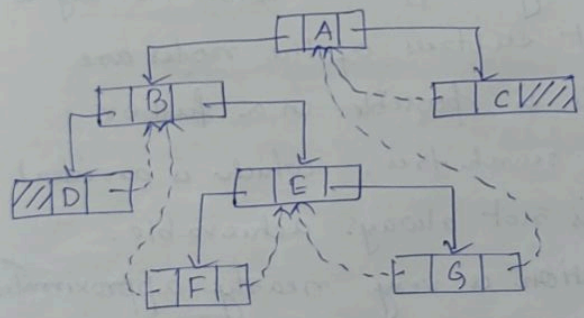
G's left link is NULL and its predecessor is E so left pointer of G is made a thread.

C's left link is NULL and its predecessor is A, so left pointer of C is made a thread.

Inorder predecessor : D B F E G A C



Fully in-threaded Binary tree.



D's left link is NULL but it has no predecessor and C's right link is NULL but it has no successor so both these pointer fields are NULL in the fully threaded tree.

06/05/2020

AVL Tree

An AVL Tree is a technique for balancing a binary search tree was introduced by Russian Mathematicians G.M. Adelson-Velskii and E.M. Landis in 1962.

The main aim of AVL tree is to perform efficient search, insertion and deletion operations. Searching is efficient when the heights of left and right subtrees of the nodes are almost same. This is possible in a full or complete binary search tree, which is an ideal situation and is not always achievable. This ideal situation is very nearly approximated by AVL trees.

An AVL tree is a binary search tree where the difference in the height of left and right subtrees of any node can be atmost 1. Each node of an AVL tree has a balance factor, which is defined as the difference between the heights of left subtree and right subtree of a node.

Balance factor of a node = Height of its left-subtree − Height of its right subtree

From the definition of AVL-tree, it is obvious that only possible values for the balance factor of any node are $\boxed{-1, 0, 1}$

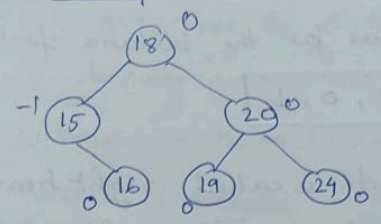
<u>Right heavy</u> → A node is called right heavy or right high if height of its right subtree is one more than height of its left subtree.

<u>Left heavy</u> → A node is called left heavy or left high if height of its left subtree is one more than height of its right subtree.

<u>Balanced</u> → A node is called balanced if the heights of its right and left subtrees are same.

The balance factor is 1 for left high.
" " " " −1 for right high.
" " " " 0 for balanced node

For example, (1)



For leaf nodes 16, 19, 24, left and right subtrees are empty so difference is 0

For node 15, height of left subtree is 0 and height of right subtree is 1,
So difference is $(0-1) = -1$

For node 20, height of left subtree is 1 and height of right subtree is 1,
So difference is $(1-1) = 0$

For root node 18, height of left subtree is 2 and height of right subtree is 2.
So difference is $(2-2) = 0$.

Therefore, the balance factor of each node is -1, 0 or 1.

Therefore the binary search tree is an AVL tr

Example ②

Let us take some trees that are binary search trees but not AVL trees.

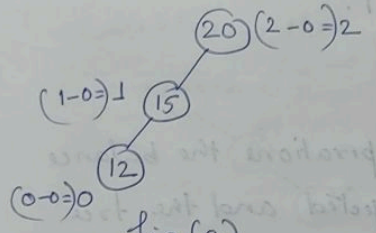


fig (a)

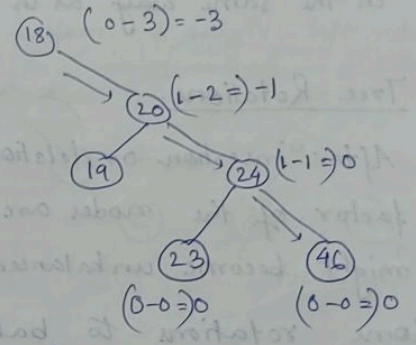


fig (b).

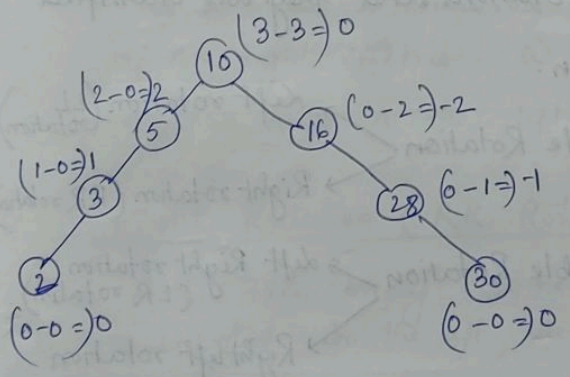


fig (c).

Since the balance factor ~~does not lies~~ of each node is n bet (-1, 0, 1) so the binary search trees a not AVL tree.

07/05/2020

Searching and Traversal in AVL tree.

Searching and traversal in AVL tree is done in the same way as in BST.

Tree Rotations.

After insertion or deletion operations the balance factor of the nodes are affected and the tree might become unbalanced. So we need to do some rotations to balance the tree.

There are four rotations and they are classified into two types.

- Rotations
  - Single Rotation
    - Left rotation (LL rotation)
    - Right rotation (RR rotation)
  - Double Rotation
    - Left Right rotation (LR rotation).
    - Right left rotation (RL rotation).

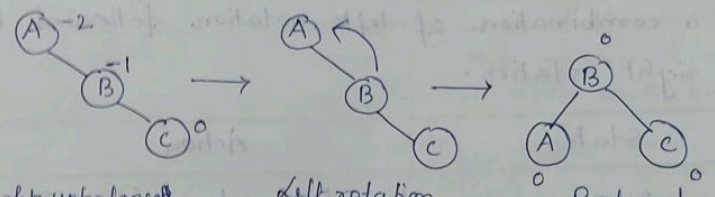
When these rotations can take place?

Let the newly inserted node be w.

1. Perform standard BST insert for w.
2. Starting from w, travel up and find the first unbalanced node.

Left Rotation (LL Rotation)

If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation.

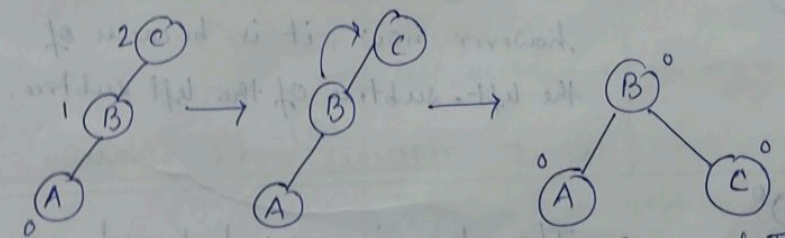


Right unbalanced tree. Left rotation. Balanced

In our example, node A has become unbalanced as a node is inserted into the right subtree of A's right subtree. We perform the left rotation by making A the left subtree of B.

Right Rotation (RR Rotation)

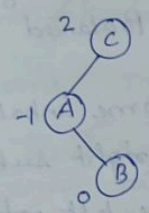
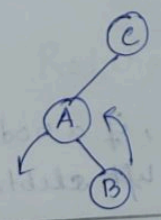
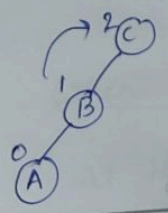
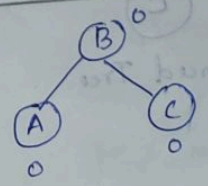
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.



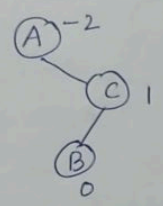
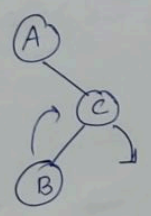
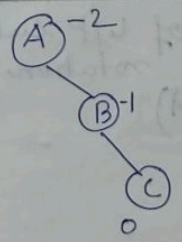
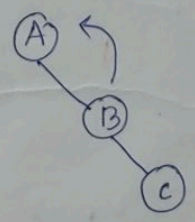
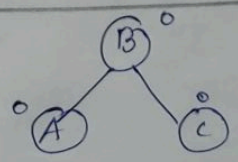
Left unbalaned tree Right rotation. Balanced Tree

As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

<u>Left-Right Rotation</u> : A left-right rotation is a combination of left rotation followed by right rotation.

State	Action.
	A node has been inserted into the right subtree of the left subtree. This makes C an unbalanced node. These scenarios cause AVL tree to perform left-right rotation.
	We first perform the left rotation on the left subtree of C. This makes A, the left subtree of B.
	Node C is still unbalanced, however now, it is because of the left-subtree of the left subtree. So it requires right rotation
	The tree is now balanced.

Right-Left Rotation : It is a combination of right rotation followed by left rotation.

State	Action.
	A node has been inserted into the left subtree of the right subtree. This makes A, an unbalanced node with balance factor 2
	First we perform the right rotation along C node, making C the right subtree of its own left subtree B. Now, B becomes the right subtree of A.
	Node A is still unbalanced because of the right subtree of its right subtree and requires a left rotation.
	A left rotation is performed by making B the new root node of the subtree. A becomes the left subtree of its right subtree
	The tree is now balanced.

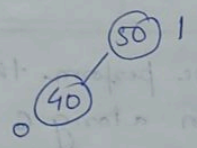
8/5/2020

Example: Let us take some numbers and construct an AVL tree from them.

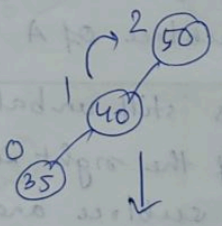
50, 40, 35, 58, 48, 42, 60, 30, 33, 25.

Soln. Insert 50.

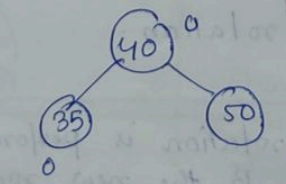
Insert 40



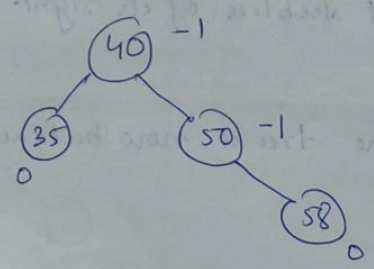
Insert 35



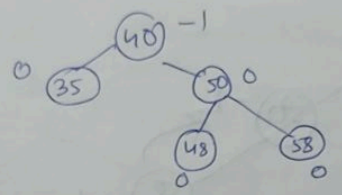
(Since insertion in left subtree of left subtree of root so right rotation has to be performed)



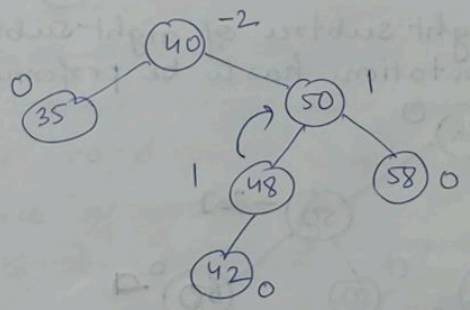
Insert 58.



Insert 48.

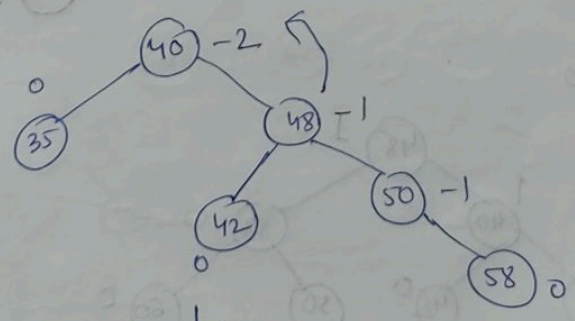


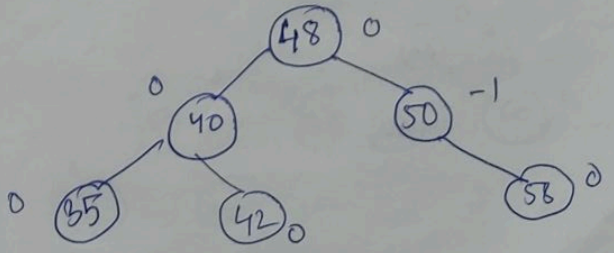
Insert 42.



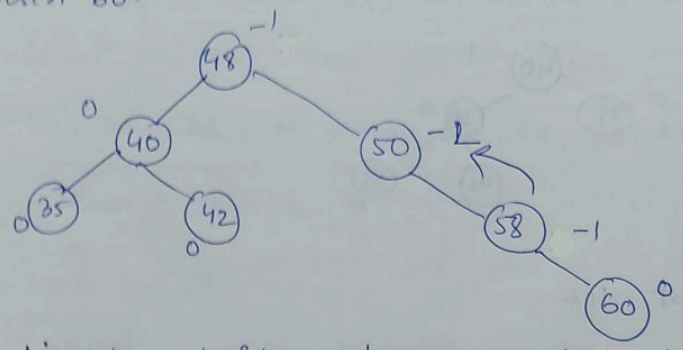
[since insertion in left subtree of right subtree of root so Right-left rotation has to be performed]

① Right right rotation
② Left left rotation.

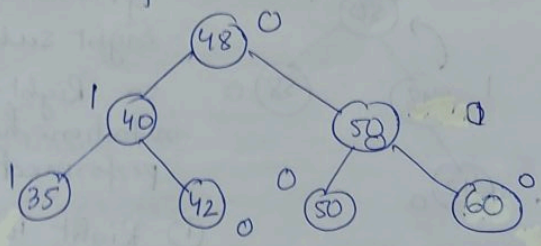




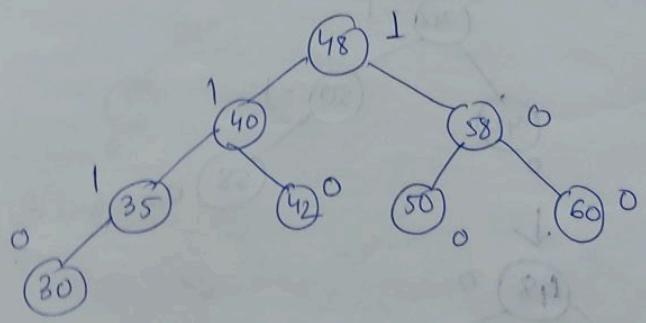
Insert 60.



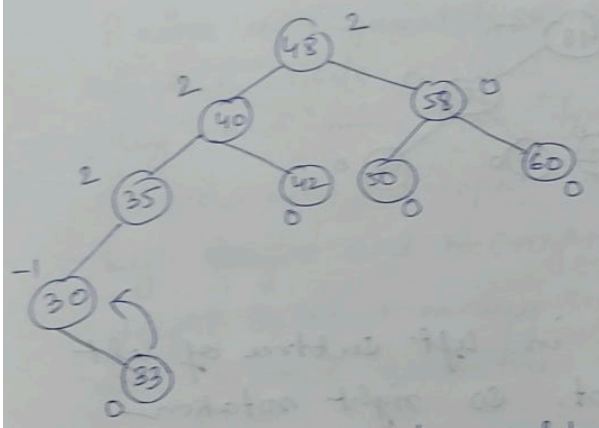
Since Insertion on right subtree of right subtree of root. So left rotation has to be performed.



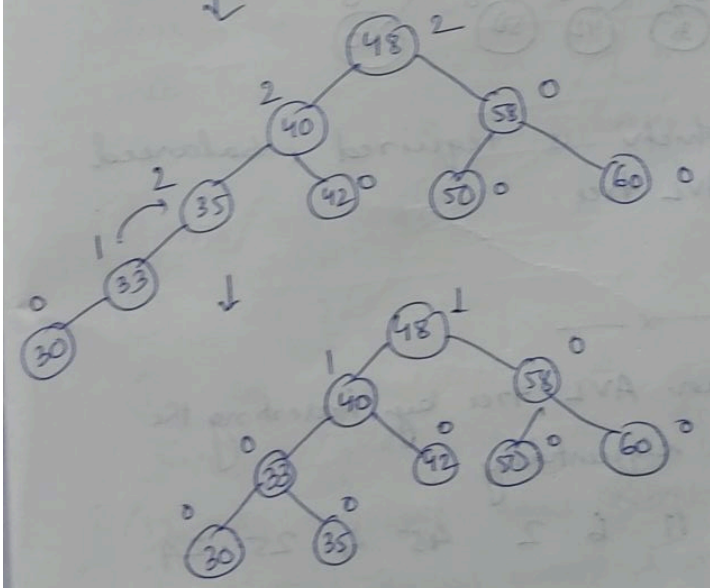
Insert 30.



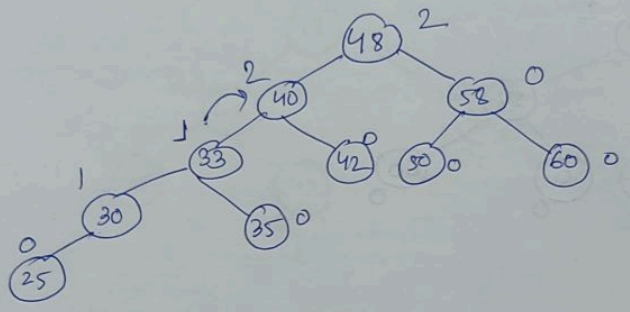
Insert 33.



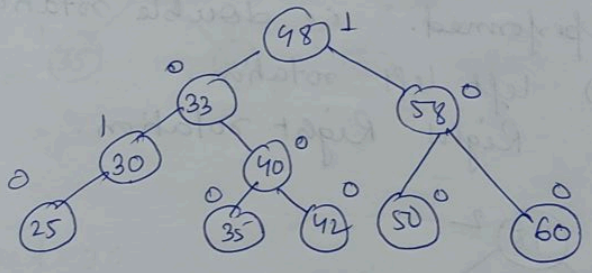
Since insertion in right subtree of left subtree of root node, so left-Right rotation has to be performed. i.e double rotation occurs. (1) left left rotation.
(2) Right Right rotation.



Insert 25.



Since insertion in left subtree of left subtree of root, so right rotation has to be performed.



which is required balanced AVL tree.

—x—

H.W. Construct an AVL tree by inserting the following values sequentially.

23 34 12 11 6 2 45 4 25 24.