CSc 3102: AVL Trees

Supplementary Notes

- Introduction
- Balance Factor
- Out of Balance Conditions
- Rotations
- Some AVL Algorithms

1 Introduction

Definition 1. An AVL tree – named for its inventors, Adel'son-Vel'skii and Landis – is a balanced binary search tree. A binary search tree T has the AVL property if for each of its nodes N, with left subtree N_L, possibly empty, and right subtree N_R, possibly empty, $|Height(N_R) - Height(N_L)| \leq 1$.

2 Balance Factor

Definition 2. The balance factor of a node is the height of its left subtree minus the height of its right subtree. A node with balance factor less than -1 or greater than 1 is out of balance and requires a special transformation called **rotation** to restore its balance.

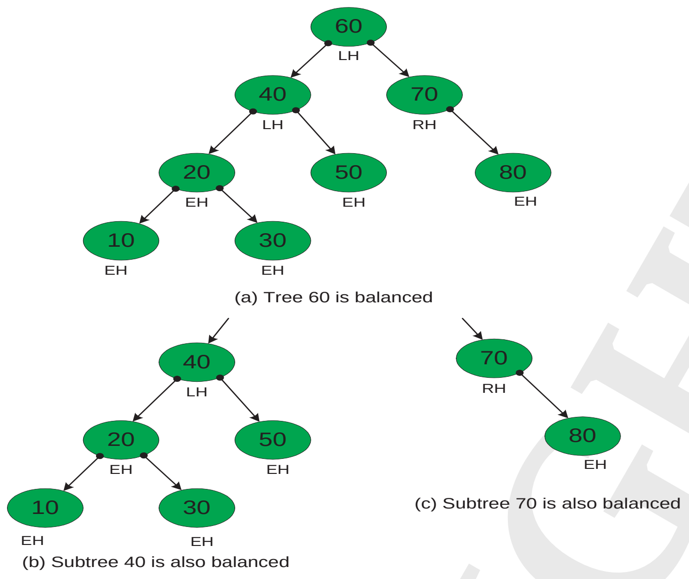


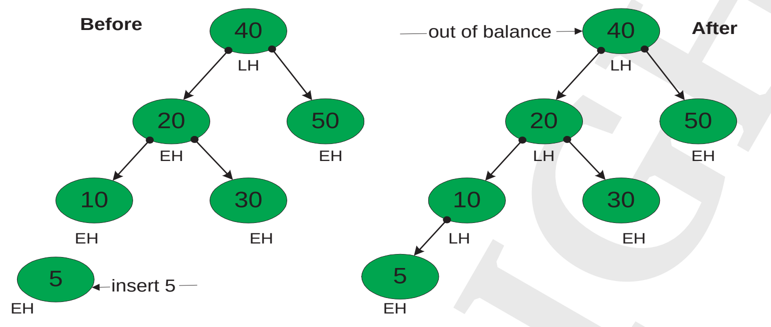
Figure 1: Determining the balance factor

3 Out of Balance Conditions

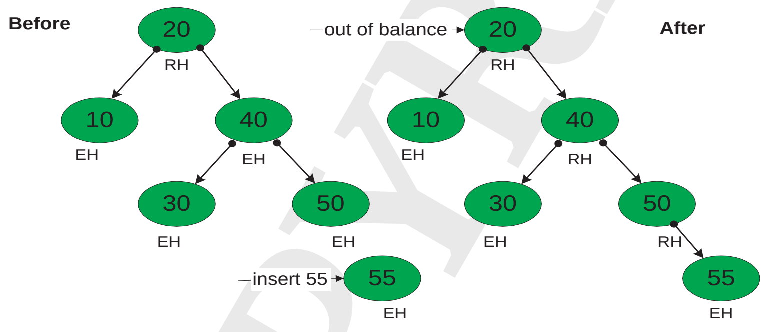
To efficiently ensure that a binary search tree has the AVL property, whenever an insertion or deletion causes the tree to be out of balance, its AVL property must be restored prior to any other insertion or deletion. There are four cases when a node is out of balance:

1. **Left of left**: A subtree of a tree that is left high becomes left high.
2. **Right of right**:A subtree of a tree that is right high becomes right high.
3. **Right of Left**:A subtree of a tree that is left high becomes right high.
4. **Left of Right**:A subtree of a tree that is right high becomes right left.

Here is a series of diagrams to illustrate these out-of-balance cases.

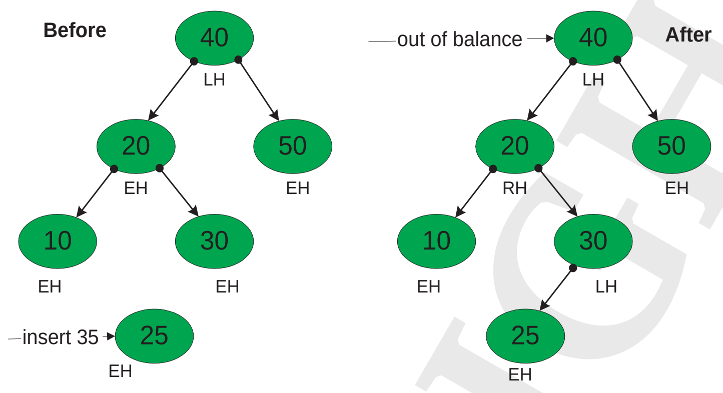


(a) case 1: left of left

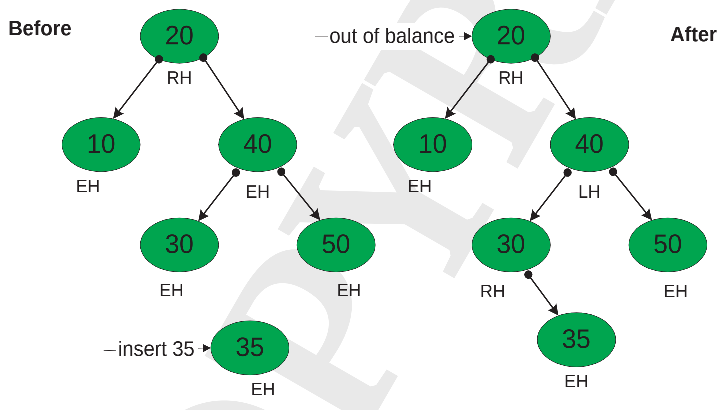


(b) case 2: right of right

Figure 2: Out-of-balance cases 1 and 2



(c) case 3: right of left



(d) case 2: left of right

Figure 3: Out-of-balance cases 3 and 4

4 Rotations

There are four types of rotations required to restore balance when the four out-of-balance conditions occur.

1. **Single right rotation**: This is used to restore balance when there is a left of left out of balance condition.

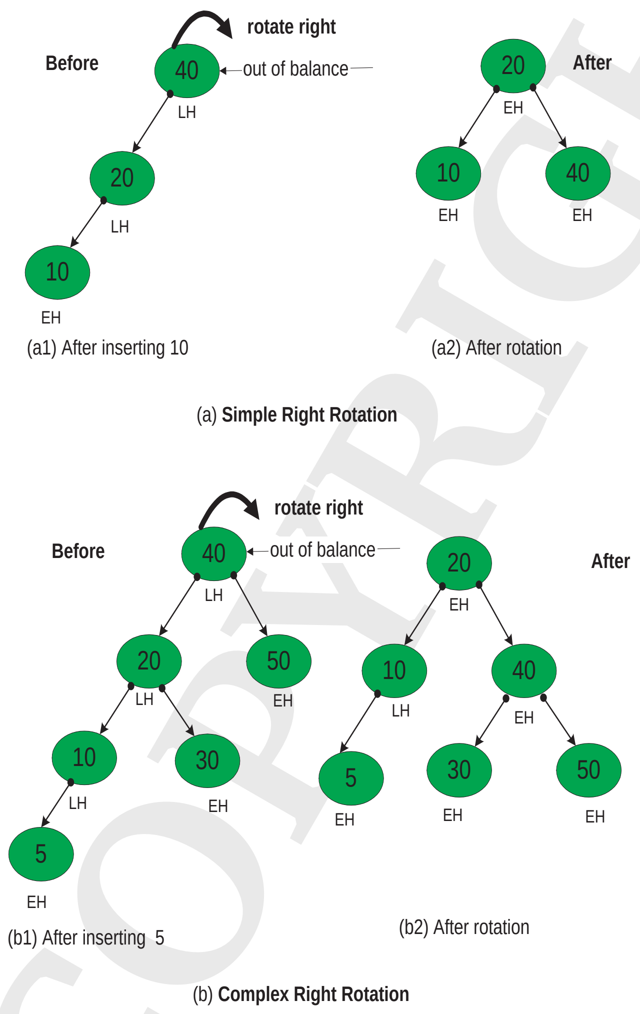
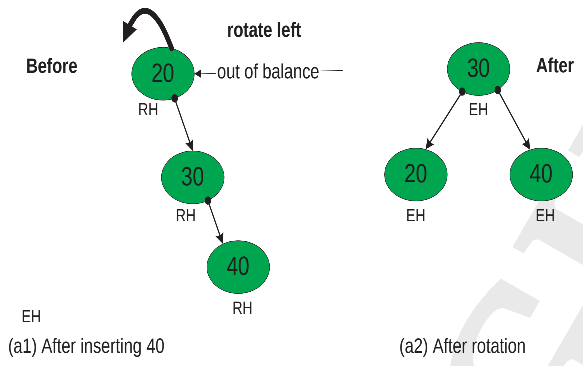
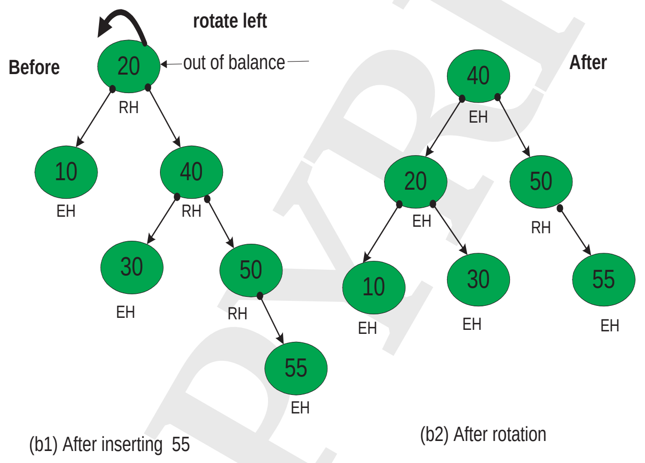


Figure 4: Single rotation right

2. **Single left rotation**: This is used to restore balance when there is a right of right out of balance condition.



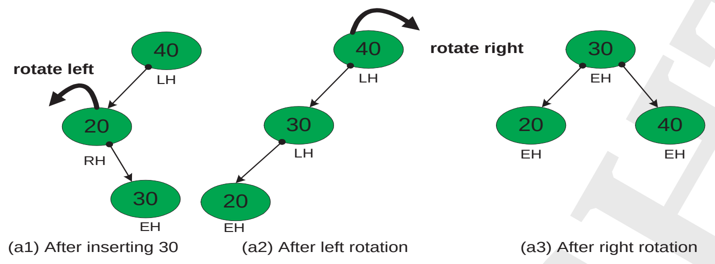
(a) **Simple Left Rotation**



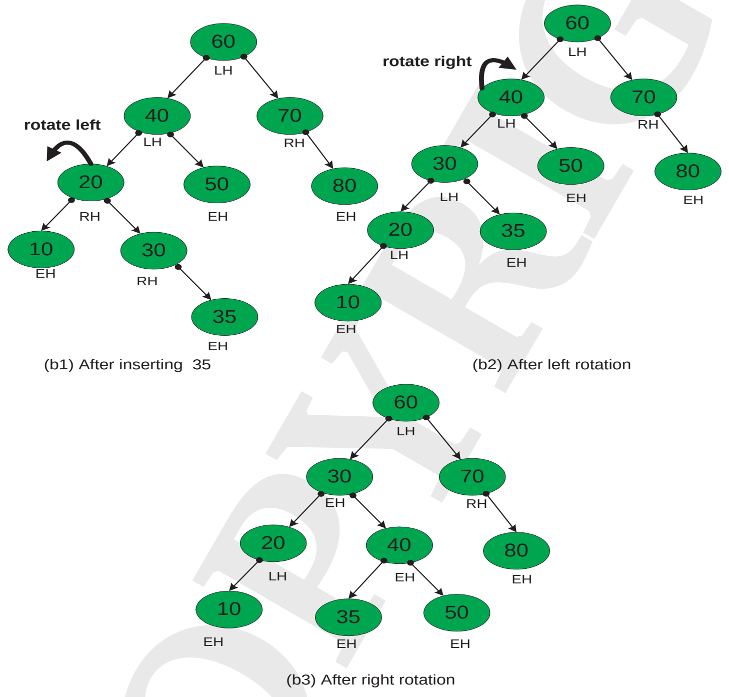
(b) **Complex Left Rotation**

Figure 5: Single rotation left

3. **Double rotation right**:This is used to restore balance when there is a right of left out of balance condition. First the subtree is rotated left and then the tree is rotated right.



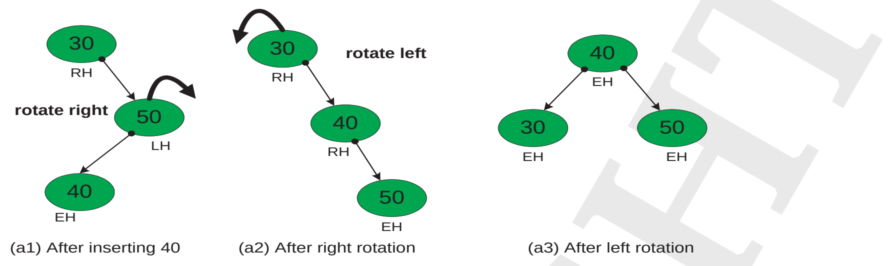
(a) Simple Double Rotation Right



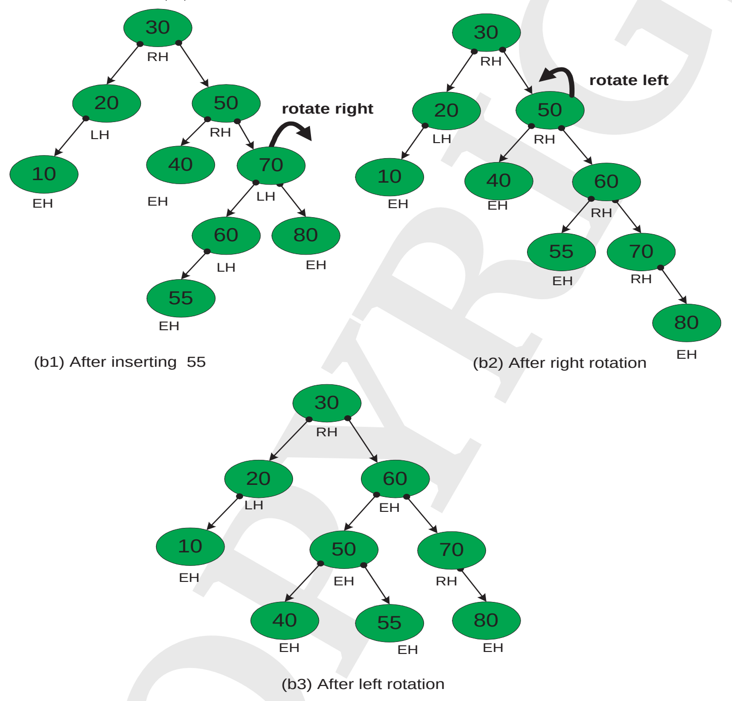
(b) Complex Double Rotation Right

Figure 6: Double Rotation Right

4. **Double rotation left**:This is used to restore balance when there is a left of right out of balance condition. First the subtree is rotated right and then the tree is rotated left.



(a) Simple Double Rotation Left



(b) Complex Double Rotation Left

Figure 7: Double Rotation Left

We will prove that insertion and deletion in an AVL tree are O($\lg n$). Starter code for an AVL implementation will be provided, with some subroutines omitted as project exercises for you.

5 Deleting from an AVL Tree

Finding the replacement node for one that is to be deleted from an AVL tree uses the same strategies that we discussed for a binary search tree. However, deleting from an AVL tree involves a lot more than that. After deleting a node, if the AVL property is lost, it must be restored by using a rotation transformation. If it is not, then no rotation is required. In either case, some bookkeeping may be required to properly handle future deletions that could affect the AVL property of the tree. The bookkeeping is critical to ensuring that deletion is $O(\lg n)$ since we do not want to visit every node in the tree when deleting a node. We must preserve the AVL property or restore it if it is lost and update the balance factors for nodes whose balance factors are affected by the deletion.

1. If a node is even high, after removing a node from its right or left subtree, the AVL property is preserved. However, its subtree becomes shorter and the node becomes left high, if the node was removed from the right subtree, or right high, if the node was removed from its left subtree. Some bookkeeping is required to properly handle future insertions and deletions. In particular, the balance factors of ancestor nodes may need to be updated.

2. If a node is currently left high and a node is deleted from its right subtree so that the right subtree becomes shorter, the AVL property of the tree is lost. A rotation is required to restore the AVL property. Also, some bookkeeping is required to properly handle future insertions and deletions. If its left subtree is even high or left high, a single rotation will restore the AVL property. If its left subtree is right high, a double rotation will restore the AVL property. After the rotation, the balance factors of ancestor nodes may require updates. The balance factor for the root of one of its subtree may also require update. An analogous situation occurs if the subtree from which the node is deleted is currently right high.

3. If the deletion of a node from the left subtree of a node that is left high causes the subtree to become shorter, the AVL property is preserved. No rotation is required. However, some bookkeeping will be required. The balance factors of ancestors of the deleted node must be updated to ensure that future deletions and insertions are properly handled. Deletion from the right subtree of a node that is right high that causes the right subtree to become shorter is analogous.

Problem 1. Answer problems 1 and 2 on the "Introduction to Binary Search Trees" handout but using an AVL tree rather than a simple binary search tree.

Problem 2. Answer problems 1 and 2 on the "Deleting from a Binary Search Trees" handout but using an AVL tree rather than a simple binary search tree.

Problem 3. Write the pseudocode description of an algorithm that returns the height of an AVL tree by tracing only one root-to-leaf path without investigating all the nodes in the tree.

.