Universidad Carlos III de Madrid

Algorithms and Data Structures (ADS)
Bachelor in Informatics Engineering
Computer Science Department

AVL Trees.

Authors: Harith Al-Jumaily
Isabel Segura-Bedmar

April 2011
Departamento de Informtica,
Laboratorio de Bases de Datos Avanzadas (LaBDA)
http://labda.inf.uc3m.es/

AVL Trees.

Definition.

An AVL tree is a self-balancing binary search tree, where the heights of the two child subtrees of any node differ by at most one. Insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Any binary search tree T that satisfies the height-balance property is said to be an AVL tree, named after the initials of its inventors: Adel'son-Vel'skii and Landis.

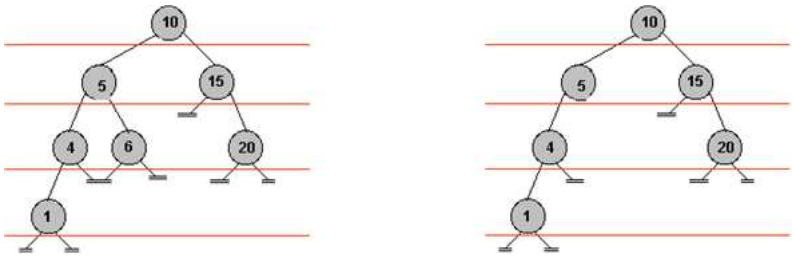

Figure 1: The second tree is not AVL because the Left-height and the Right-height of the node 5 differ by 2 nodes.

The balance factor of a node is the height of its left subtree minus the height of its right subtree (sometimes opposite) and a node with balance factor 1, 0, or 1 is considered balanced. A node with any other balance factor is considered unbalanced and requires rebalancing the tree. The balance factor is either stored directly at each node or computed from the heights of the subtrees. Formally, the balance factor of a node can be defined as follows:

$$Bf = Hr - Hl \tag{1}$$

where H_R is the height of its right subtree and H_L is the height of its left subtree.

Based on the possible values of Bf, we can define the following cases:

- If $Bf = 0$ the the left and the right subtrees of a node are the same height.

- If Bf =1 then the tree is balanced in height, but the right subtree is a higher level.
- If $Bf = -1$ then the tree is balanced in height, but the left subtree is a higher level.
- If $Bf \preceq 2$ or $Bf \succeq 2$ then the tree must be balanced.

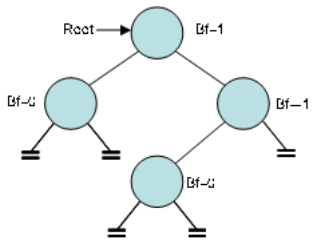

Figure 2: Balance factor of a node.

The insertion and removal operations for AVL trees are similar to those for binary search trees, but with AVL trees we must perform additional computations called rotations.

Operations

Insertion

An insertion in an AVL tree T begins as in an insert operation for a (simple) binary search tree. After inserting a node, it is necessary to check each of the node's ancestors for consistency with the rules of AVL. For each node checked, if the balance factor remains 1, 0, or +1 then no rotations are necessary. However, if the balance factor becomes 2 then the subtree rooted at this node is unbalanced.

For example, Figure 3 shows a sequence of integer nodes (40, 33, 46, 6, 8, 24, 18, 22, 25, 60) inserted in an empty AVL tree. The balance factor of the node 33 is $Bf = -2$, this means that the tree must be balanced.

There are four cases which need to be considered for balancing an AVL tree, of which two are symmetric to the other two. These cases are shown as follows:

Right-Right Simple Rotation (RR) In Figure 4, the node a has $Bf = -2$. In order to balance the tree, b becomes the new root, a becomes the left child of b, c becomes the right child of b.

Left-Left Simple Rotation (LL) In Figure 5, the node a has $Bf = 2$. In order to balance the tree, b becomes the new root, a becomes the right child of b, c becomes the left child of b.

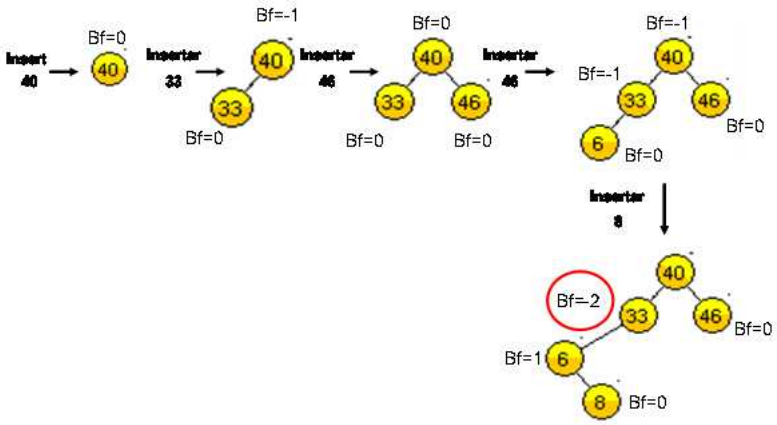

Figure 3: Insertion in an AVL tree.

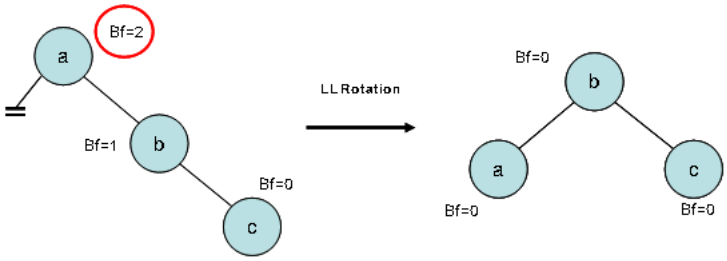

Figure 4: Right-Right Simple Rotation.

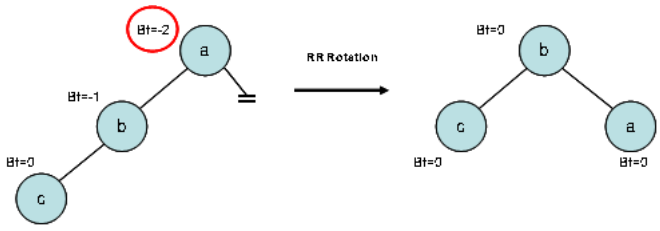

Figure 5: Left-Left Simple Rotation.

Right-Left Simple Rotation (Double RL) In Figure 6, it is necessary two rotations in order to balance the node a:

- First rotation: c becomes the right child of a, b becomes the right child of c.
- Second rotation: c becomes the new root, a becomes the left child of c.

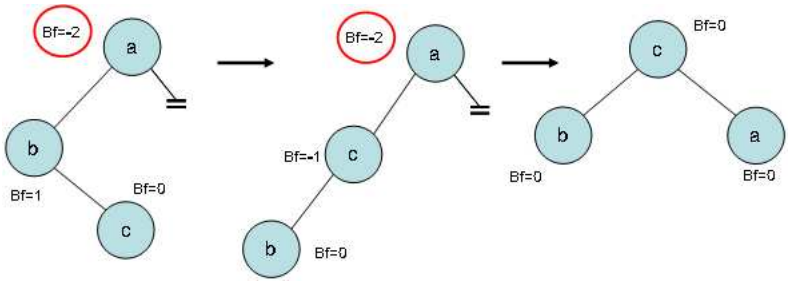

Figure 6: Right-Left Simple Rotation.

Left-Right Simple Rotation (Double LR) In Figure 7, it is necessary two rotations in order to balance the node a:

- First rotation: c becomes the left child of a, b becomes the left child of c.
- Second rotation: c becomes the new root, a becomes the right child of c.

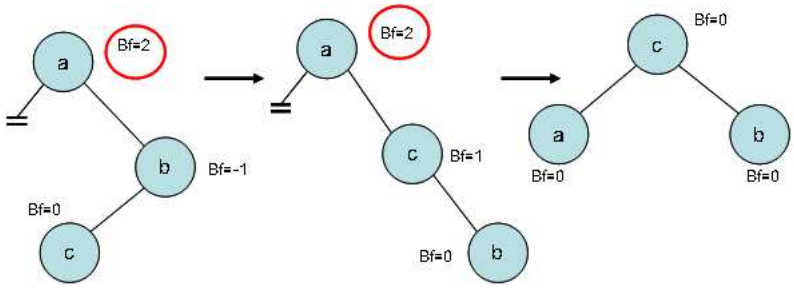

Figure 7: Left-Right Simple Rotation.

Deletion

If the node is a leaf or has only one child, remove it. Otherwise, replace it with either the largest in its left subtree (inorder predecessor) or the smallest in its right subtree (inorder successor), and remove that node. The node that was found as a replacement has at most one subtree. After deletion, retrace the path back up the tree (parent of the replacement) to the root, adjusting the balance factors as needed. Fox example, if the node whose value is 5 is deleted from the tree shown in Figure 8, the root has $Bf = 2$ and we should balance it by a Right-Right rotation.

An example

Check if the binary tree show in Figure 11 is AVL:

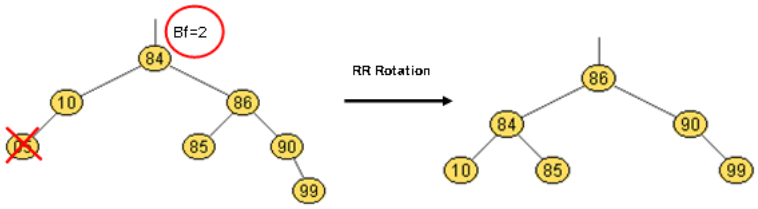

Figure 8: Deletion of a leaf node in an AVL tree requiring a RR rotation.

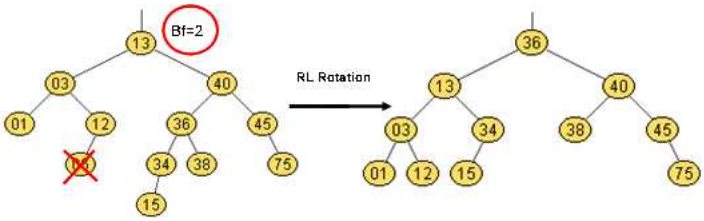

Figure 9: Deletion of a leaf node in an AVL tree requiring a RL rotation .

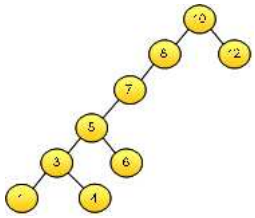

Figure 10: is this binary search AVL?.

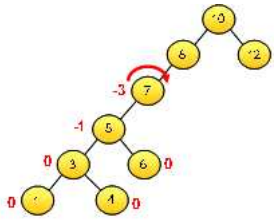

Figure 11: The tree is not an AVL tree because the balance factor of the node 7 is $Bf = -3$.

We must calculate the balance factors of each node, as follows:

Since tree is not an AVL tree (Bf of the node 7 is -3(, this means that the tree must be balanced as follows:

1. RR Rotation is applied on the node 7, (a=7, b=5, c=3) the result of the

new tree is shown in Figure **??**:

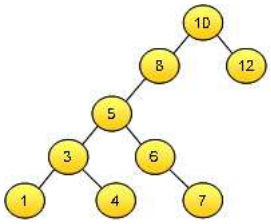

Figure 12: A RR rotation is applied on the tree in order to balance the node 7.

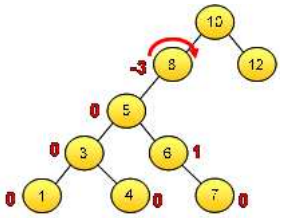

Figure 13: The tree is not an AVL tree because the balance factor of the node 8 is Bf = -3.

We must check checked the tree shown in Figure 12 to see if it is an AVL tree or not. The tree is not an AVL tree because the balance factor of the node 8 is $Bf = -3$, this means that the tree must be balanced. A RR Rotation is applied on the node 8, (a=8, b=5, c=3) the result of the new tree is shown in Figure 14. Once checked the new tree, it is an AVL

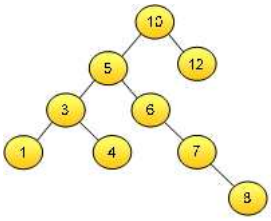

Figure 14: The tree is not an AVL tree because the balance factor of the node 8 is Bf = -3.

The tree shown in Figure 15 is not an AVL tree because the balance factor of the node 6 is Bf = 2 and, therefore, it must be balanced. A LL Rotation is applied on the node 6, (a=6, b=7, c=8) and the result of the new tree is shown in Figure 16. The previous tree is not an AVL tree because the balance factor of the node 10 is Bf = -2 (see Figure 17), this means that the tree must be

balanced. We can apply a RR Rotation on the node 10, (a=10, b=5, c=3) the result of the new tree is an AVL tree (see Figure 18).

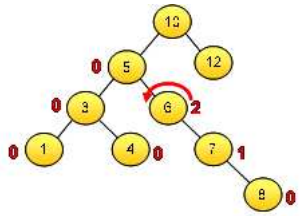

Figure 15: The previous tree is not an AVL tree because the balance factor of the node 6 is Bf = 2.

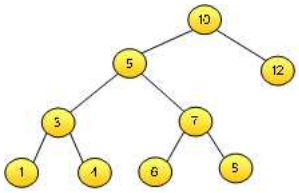

Figure 16: LL Rotation is applied on the node 6.

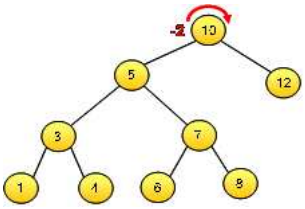

Figure 17: The previous tree is not an AVL tree because the balance factor of the node 10 is Bf = - 2.

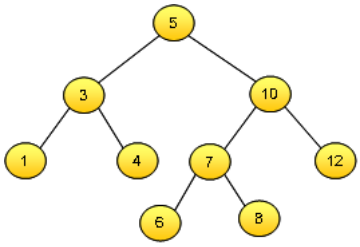

Figure 18: AVL Tree achieved by applying a RR rotation on the node 10.

Bibliography