AVL Tree

• An AVL tree is a binary search tree in which the heights of the left and right subtrees of the root differ by at most 1 and in which the left and right subtrees are again AVL trees.

• With each node of an AVL tree is associated a balance factor that is left higher, equal, or right higher according, respectively, as the left subtree has height greater than, equal to, or less than that of the right subtree.

• In each node structure there is an extra field: BalanceFactor bf;
Examples: Which one are AVL?

Insertion in AVL

• Usual Binary tree insertion should work.
  – Check if the new key will go left or right.
  – Insert it recursively in left or right subtree as needed.

• What about the Height?
  – Often it will not result in any increase of the subtree height, do nothing.
  – If it increases the height of the shorter subtree, still do nothing except update the BF of the root.
  – Only if it increases the height of the taller subtree then need to do something special.
Simple Insertion

InsertAVL

```c
TreeNode *InsertAVL(TreeNode *root, TreeNode *newnode, Boolean *taller)
{
    if (!root) {
        root = newnode;
        root->left = root->right = NULL;
        root->bf = EH;
        *taller = TRUE;
    }
    else if (EQ(newnode->entry.key, root->entry.key)) {
        Error("Duplicate key is not allowed in AVL tree.");
    }
    else if (LT(newnode->entry.key, root->entry.key)) {
        root->left = InsertAVL(root->left, newnode, taller);
        if (*taller)                        /* Left
            switch(root->bf) {
                case LH :                        /* Node was left high.
                    root = LeftBalance(root, taller); break;
                case EH :
                    root->bf = LH;  break;       /* Node is now left high.
                case RH :
                    root->bf = EH;          /* Node now has balanced
            */
            *taller = FALSE; break;
        }
    }
    else {
        (continued...)
    }
}
```
InsertAVL (continued..)

```c
TreeNode *InsertAVL(TreeNode *root, TreeNode *newnode, Boolean *taller) {
    (continued..)
}
```

else {
    root->right = InsertAVL(root->right, newnode, taller);
    if (*taller) /* Right subtree is taller. */
        switch(root->bf) {
            case LH:
                root->bf = EH; /* Node now has balanced height. */
                *taller = FALSE; break;
            case EH:
                root->bf = RH; break; /* Node is right high. */
                root = RightBalance(root, taller); break;
            case RH:
                /* Node was right high. */
                root = RightBalance(root, taller); break;
        }
    return root;
}

Balancing unbalanced AVL

- Problem:
  - Let us assume we have used InsertAVL
  - Now the right subtree height has grown one and the right subtree was already taller!
  - How to restore the balance?

- Solution:
  - There can be three situations:
    - The right subtree itself is now left heavy
    - The right subtree itself is now right heavy
    - The right subtree now has equal heights in both sides..
Case-1: Right Higher

- Left Rotation:

```c
TreeNode* RotateLeft(TreeNode* p)
{
    TreeNode* right child = p;
    if (!p)
        Error("It is impossible to rotate an empty tree in RotateLeft.");
    else if (!p->right)
        Error("It is impossible to make an empty subtree the root in RotateLeft.");
    else {
        right child = p->right;
        p->right = right child->left;
        right child->left = p;
    }
    return right child;
}
```

Case-2: Left Higher

- Double Left Rotation:
Behavior of Algorithm

- The number of times the function InsertSVL calls itself recursively a new node can be as large as the height of the tree.

- How many times the routine RightBalance or LeftBalance will be called?
  - Both of them makes the BF of the root EQ.
  - Thus it will not further increase the tree height for outer recursive calls.
  - Only once they will be called!
  - Most insertion will induce no rotation.
  - Even when, they usually occur near the leaf.

Case-3: Equal Height

- Can it Happen?
Example-1

Example-2
Deletion of a Node

- Reduce the problem to the case when the node $x$ to be deleted has at most one child.
- 2. Delete $x$. We use a Boolean variable shorter to show if the height of a subtree has been shortened.
- 3. While shorter is TRUE do the following steps for each node $p$ on the path from the parent of $x$ to the root of the tree. When shorter becomes FALSE, the algorithm terminates.

- 4. Case 1: Node $p$ has balance factor equal.
- 5. Case 2: The balance factor of $p$ is not equal, and the taller subtree was shortened.
- 6. Case 3: The balance factor of $p$ is not equal, and the shorter subtree was shortened. Apply a rotation as follows to restore balance. Let $q$ be the root of the taller subtree of $p$.
- 7. Case 3a: The balance factor of $q$ is
Deletion-1

No operation

Deletion-2

Single Rotation
Deletion-3

Double Rotation

Example
Example (continued..)

Adjust balance factors

Delete p

Rotate left:

Double rotate right around m:
The Height of AVL Tree (WC)

- Let $F_h$ be the minimum number of nodes that an AVL tree of height $h$ can have. Then:

$$|F_h| = |F_{h-1}| + |F_{h-2}| + 1$$

| $F_0$ | 1 | $F_1$ | 2 | Fibonacci Trees |

- Fibonacci vs. Our Series ($n=h+2$)

\[
F_{n+2} = F_{n+1} + F_n
\]

- $|F_h| + 1$ satisfies the definition of Fibonacci number.

\[
(|F_h| + 1) = (|F_{h-1}| + 1) + (|F_{h-2}| + 1)
\]

- By evaluation Fibonacci:

\[
(|F_h| + 1) = \frac{1}{\sqrt{5}} \left[ \frac{1 + \sqrt{5}}{2} \right]^{h+2} = \frac{(GR)^{h+2}}{\sqrt{5}}
\]

- By taking log in both sides: $h \approx 1.44 \log |F_h|

- In the worst case AVL will perform no more than 44% more of the perfect case!