AVL TREES

AVL Tree

- Two Russian Mathematicians , G.M Adelson-Velskii and E.M. Landis created the balanced tree known as the **AVL tree.**
- An AVL tree is a search tree in which the **heights of the subtrees differ by no more than 1.**
- **It is thus a balanced tree.**
- **An AVL tree** is a binary tree that is
  - **Either empty or**
  - **Consists of 2 AVL subtrees T_L and T_R whose heights differ by no more than 1**

    | H_L and H_R | <=1

    Where H_L: height of the left subtree and

    H_R : height of the right subtree

Difference between Binary Search Tree and AVL Tree

- The BST is not a balanced tree
- The search effort is O(n)

- The AVL tree is a balanced tree
- The search effort is O(logn)

Example:

Descriptive identifiers for the balance factors

- LH : Left High(+1) :
  - Indicates that the **left subtree is higher than the right subtree**
- EH:Even High(0):
  - Indicates that the **left subtree is equal to the right subtree**
- RH: Right High(-1):
  - Indicates that the **left subtree is Shorter than the right subtree**

Balancing Trees

- Whenever a node is inserted/deleted into/from a tree respectively, the resulting tree may become unbalanced.
- Therefore we need to rebalance it.
- **Basic Balancing Algorithms:**

4 cases that require rebalancing

- **Left of Left:**
  - – A subtree of a tree that is **left high has also become left high**
- **Right of Right:**
  - – A subtree of a tree that is **right high has also become right high**
- **Right of left:**
  - – A subtree of a tree that is **left high has become right high**
- **Left of right:**
  - – A subtree of a tree that is **right high has become left high**

Left of Left: When a out-of-balance condition has been created by a left high subtree, balance the tree by **rotating the out-of-balance node to the right.**

Algorithm rotateRight(root)

1. set left subtree = right subtree of left subtree
2. Make left subtree new root

Example : 1.Simple right rotation
2. Complex right rotation

```
AVLNode *
    rotateRight(AVLNode
    *root)
AVLNode *tempptr
tempptr=root->left
root->left= tempptr->right
tempptr->right=root
root=tempptr
return
```

Right of right: When a out-of-balance condition has been created by a right high subtree, balance the tree by **rotating the out-of-balance node to the left.**

Algorithm rotateLeft(root)

```
    set right subtree = left
     subtree of right subtree
    Make right subtree new
     root
```

```
AVLNode *
   rotateLeft(AVLNode *root)
AVLNode *tempptr
tempptr=root->right
root->right= tempptr->left
tempptr->left=root
return
```

Example : 1.Simple left rotation
2. Complex left rotation

Right of left: when a out-of-balance condition is created in which the **root is left high and the left subtree is right high,first rotate the left subtree to the left and then rotate the root to the right, making the left node the new root**

Pseudocode for balancing left high

```
Algorithm leftBalance(root)
left_subtree=root->left
If(left_subtree high)
    1. rotateRight(root)
else
    1. rotateLeft(left_subtree)
    2. rotateRight(root)
```

Examples:

- **Left of right**: when a out-of-balance condition is created in which the **root is right high and the right subtree is left high,first rotate the right subtree to the right and then rotate the root to the left, making the right node the new root**

Pseudocode for balancing right high

```
Algorithm rightBalance(root)
right_subtree=root->right
If(right_subtree high)
    1. rotateLeft(root)
else
    1. rotateRight(right_subtree)
    2. rotateLeft(root)
```

Examples:

- **Note: the Search and retrieval algorithms are the same for any binary tree.**

- Algorithm :Insert into AVL Tree

Insert into AVL tree

```
Algorithm AVLInsert(
    root, newData)
1. if(subtree empty)
    1. Insert newdata at
       root
    2. return root
2. If(newdata<root)
    1. AVLInsert(
       left_subtree,newdata)
    2. If(left_subtree taller)
        1. leftBalance(root)
   else
    1. AVLInsert(
       right_subtree,newdata)
    2. If(right_subtree taller)
        1. rightBalance(root)
3. return root
```

leftBalance algorithm

```
Algorithm leftBalance
   (AVLNode *root)
1.  leftTree=root->left
2.  If(leftTree left-high)
        //case 1:Left of left
        1.  rotateRight(root)
        2.  Adjust balance
            factors
    else
        1. rightTree=
                 leftTree->right
        2. Adjust balance factors
        3
3. rotateLeft(root)
4. rotateRight(root)
```