Dynamic Binary Search Trees

Rotations

Rotating right or left in a binary search tree is illustrated below. Note that the tree maintains its characteristics as a binary search tree.

```
    y       rotate (xy) R        x
   / \           ====>          / \
  x   3                        1   y
 / \        rotate (xy) L         / \
1   2            <====           2   3
```

```
           DOUBLE ROTATE (xyz) L
   x        1. rotate (yz) R
    \       2. rotate (xz) L       z
     y          =======>          / \
    /                            x   y
   z       DOUBLE ROTATE (xzy) R
            1. rotate (xz) R
            2. rotate (yz) L
                <=======
```

AVL Trees (height-balanced trees)

An AVL (Adelson-Velskii, Landis) tree is a binary search tree in which the heights of the right and left subtrees of each node differ by at most 1.

The height of a leaf (no children) is defined to be 0. The height of an empty tree (no nodes) is defined to be -1.

It can be shown by induction that the height, H, of an AVL tree with n nodes satisfies $\lg(n+1)-1 \leq H \leq 1.44 \lg(n+2)-1$.

The *balance factor* of a node is defined to be Height(RightSubtree) - Height(LeftSubtree). Thus, in an AVL tree, the balance factor of each node will be in $\{-1, 0, +1\}$.

Insertion into an AVL tree may change change the balance factors of some nodes on the path from the inserted node to the root. Any balance factor will change by at most 1 and the resulting value might become illegal (i.e., -2 or +2). After determining the deepest node that is "illegal", one single or one double rotation suffices to fix up the entire tree. Details.

Deletion from an AVL tree is handled similarly, but $O(\lg n)$ rotations may be required.

Weight-balanced Trees

Instead of height, weight (number of nodes) is balanced. Balance can be maintained using rotations in a similar manner.

Splay Trees

A splay tree is a binary search tree with no explicit balance condition, in which a special operation called a *splay* is done after each search or insertion operation. Splaying at node x causes node x to become the root of the binary search tree through a specific series of *rotations* as follows.

Three cases:

1. x has no grandparent (*zig*)
    - If x is left child of root y, then rotate (xy)R.
    - Else if x is right child of root y, then rotate (yx)L.

2. x is LL or RR grandchild (*zig-zig*)
    - If x is left child of y, and y is left child of z,
      then rotate at grandfather (yz)R and then rotate at father (xy)R.
    - Else if x is right child of y, and y is right child of z,
      then rotate at grandfather (yz)L and then rotate at father (xy)L.

   If x has not become the root, then continue splaying at x.

3. x is LR or RL grandchild (*zig-zag*)
    - If x is right child of y, and y is left child of z,
      then rotate at father (yx)L and then rotate at grandfather (xz)R.
    - Else if x is left child of y, and y is right child of z,
      then rotate at father (yx)R and then rotate at grandfather (xz)L.

   If x has not become the root, then continue splaying at x.