AVL Trees

AVL trees are balanced binary trees (also mention 2-3 Trees) they allow a tree to be rebalanced when inserts and/or deletes cause it to become unbalanced. The rebalancing operations make use of "rotate right" and "rotate left" operations.

Rotate right:

```
               23
             /   
           17     37
          /   
         8     28
```

```
               17
             /   
           8     23
          /     
         21     37
        /       
       28       49
```

21 > 17, so its child must move to right subtree

Rotate left:

```
      17
     /   
    8     21
```

```
      23
     /   
    17     37
   /     / 
  8     21  28
```

1) right child becomes root
2) 28 (and its subtree) must move to left subtree
Defining balance of a tree (or subtree):

balance of a node is \( \text{height of left subtree} - \text{height of right subtree} \)
balance of tree = balance of root

AVL trees will maintain balance = \(-1, 0, \text{or } 1\)
at every node, i.e., left/right heights never differ by more than one, at every node in tree
(this is sufficient to guarantee \( O(\log n) \) search time in an AVL-based BST)

Balance example:

There are 2 ways a tree could become unbalanced after an insert:

1) inserted node can be a left descendant of a node w/ balance = 1
   increases height of subtree
2) inserted node is right descendant of a node w/ balance = -1
   increases height of subtree
there are two possible sub-cases for each of these main cases

Consider Case 1:

focus on the nearest ancestor to become unbalanced after insert (A in below figure)

before insert -

in Case 1a, B's balance becomes 1, A's balance becomes 2 after insert

in Case 1b, B's balance becomes -1, A's balance becomes 2 after insert

Case 1a:

to rebalance, just do a right rotation of tree rooted at A
after right rotation:

So, A and B are now balanced, other subtrees are still balanced.

also, B's height = n+2 and A's height before insertion = n+2

since B replaced A, all ancestors of B are unaffected

Case 1b (with inserted node in left subtree of C):

to rebalance, do a left rotation at B followed by a right rotation at A

after left rotation at B:
after right rotation at \( A \):

\[
\begin{array}{c}
\text{Subtree of height } n \\
\text{Subtree of height } n \\
\text{Subtree of height } n-1 \text{ (formerly C's right subtree)} \\
\end{array}
\]

again, height of \( C = n + 2 \) so ancestors of \( C \) unaffected

**Case 1b (w/ inserted node in right subtree of \( C \))**:

after left rotation at \( B \):

\[
\begin{array}{c}
A \\
\text{Subtree of height } n \\
\text{Subtree of height } n \text{ (w/ inserted node)} \\
\text{Subtree of height } n-1 \text{ (formerly C's left subtree)} \\
\end{array}
\]

after right rotation at \( A \):

\[
\begin{array}{c}
\text{Subtree of height } n \\
\text{Subtree of height } n-1 \text{ (formerly C's right subtree)} \\
\end{array}
\]