=============================================================================

CSC 263 Tutorial 4 Winter 2019

=============================================================================

In this tutorial, will learn about **AVL-Delete** using various examples.

1. Consider the following AVL tree, what happens if we insert a node 5 into the tree? Which node is the lowest ancestor to become unbalanced? What type of rotations need to be performed to rebalance the tree? Perform the AVL-INSERT carefully step by step.

3

/ \

2 7

/ / \

1 4 8

\

6

answer:

Node 4 is the lowest ancestor that becomes unbalanced after inserting 5.  
Need to perform **double right-left rotation** in the subtree rooted at 4.

(因为这个是在在中间的，要先right rotation变成整体都right heavy，才可以整体都left rotation)

-Which node is the lowest ancestor to become unbalanced?

4 is the most unbalanced.

it has unbalanced factor of 2. Left-heavy

-What type of rotations need to be performed to re-balance the tree?

First attempt： Maybe right rotation around 4.（不行，违背了BST的property）

Second attempt（requires two rotations）：先right rotation变成4-5-6（上中下），再left rotation变成4-5-6（左中右）, still a log n time

Before insertion the height of the tree is 3, after insertion it is still 3.  
This is a special property for insertion, i.e., the heights before and after  
the insertion are always the same.

2. Consider the following AVL-tree, what happens if we delete the node 8 from

the tree? What type of rotations are needed to rebalance the tree? What's the

height of the tree before deletion? What's the height of the tree after

deletion?

6

/ \

4 7

/ \ \

2 5 8

/ \

1 3

answer:

After deleting 8, node 6 becomes unbalanced, a single right rotation fixes  
it. The height before deletion is 3, after deletion it becomes 2. Note that for deletion the height might change, which is different from insertion.

The root node is not balanced anymore after deletion.

-What type of rotations need to be performed to re-balance the tree?

4

2 6

1 3 5 7

-height of the tree before deletion? 3

-height of the tree after deletion? 2

3. Consider the following AVL-tree, what happens if we delete the node 2 from

the tree? What type of rotations are needed to rebalance the tree? What's the

height of the tree before deletion? What's the height of the tree after

deletion?

3

/ \

2 7

/ / \

1 5 8

/ \

4 6

answer:

After deleting 2, node 3 becomes unbalanced, and this one requires a double  
right-left rotation to fix (since the "middle part" is tall). The height before is 3, after is 2.

注意这里删除了2以后变成了right heavy但中间重的情况（左右的情况），就要在low level先right rotation一次变成整体right heavy，然后在统一left rotation。

3

1 5

* - 4 7

6 8

5

3 7

1 4 6 8

（requires two rotations）

4. Come up with an example AVL-tree for which deleting a node from the tree

causes two levels of double-rotations.

answer:

This example can be built in a bottom-up manner, i.e., first a small tree  
whose root becomes unbalanced and requires a double rotation to fix; then use  
this tree as a subtree and add another level of similarly structured tree above  
it. It is important that you take a systematical approach to construct this  
example, rather than doing random trial-and-errors.

5. Draw a picture of the structure of the AVL tree for which deleting a node

would cause O(log n) rotations, where n is the number of nodes in the tree.

answer:

Just repeat the similar structure to what's in the previous question. It's  
simpler in the sense that, for each level, we don't require double-rotation, so an example which requires a single rotation at each level would be good.  
  
Below is the sketch of a working example. Note that the number in this picture is not the key of a node, but the height of the subtree rooted at the node. For example, number "5" represents a subtree which is an AVL-tree with height 5.  
  
        /...  
       6  
      / \  
     4   5  
    / \  
   2   3  
  / \  
 0   1

6. Suppose in the AVL-INSERT operation, the tree looks like the following right after inserting the new node and before rebalancing.

1

\

2

\

3

\

4

Since node 2 is the lowest ancestor that is unbalanced, we do a left

rotation around 2, which gives us the following tree

1

\

3

/ \

2 4

This tree is still not AVL, which contradicts with what we said in the lecture

that AVL-INSERT requires only one level of rotation. What went wrong?

answer:

The chain of 4 nodes is an impossible state for an AVL-tree after insertion  
(before rebalancing) since the chain of 3 nodes is not an AVL tree. Therefore,  
the conclusions that we have about the properties of insertion does not apply  
here.