lab_avl – Awful AVL Trees
Week #8 – March 7-9, 2018

Welcome to Lab AVL!

Course Website: https://courses.engr.illinois.edu/cs225/labs

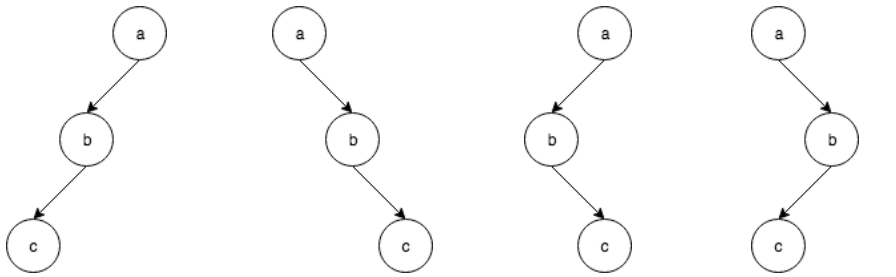
Overview

In this week's lab you will work on implementing important features for an AVL tree. An AVL tree is a dynamically balancing BST (Binary Search Tree). It maintains a height of at most lg(n) where n is the number of nodes in the BST. This is important since the runtime of searching for an element in a balanced BST is O(lg(n)).

AVL Tree Rotations

In lecture, you learned an AVL tree has four kinds of rotations that it can perform in order to balance the tree: L, R, LR, and RL. Left (L) and right (R) rotations are singular rotations used on "sticks" in order to turn them into "mountains". Left-right (LR) and right-left (RL) rotations are combinations of the previous two rotations that are used to turn "elbows" into "mountains".

Exercise 1: Consider the four possible "sticks" and "elbows":



Right	Left	Left-Right	Right-Left
Q1) Right			
Q2) Root BF = -2	Root BF = 2	Root BF = -2	Root BF = 2
Q3) Child BF = -1	Child BF = 1	Child BF = 1	Child BF = -1

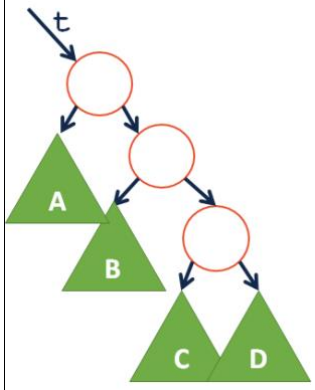
For each tree:
Q1: What type of rotation fixes each of these trees?
Q2: Identify the balance factor of the root node in each tree.
Q3: Identify the balance factor of the node below the root in each tree.

Left Rotation

Let's focus on writing the code for a single rotation.

Exercise 2: Complete the pseudo-code for a left rotation

Initial structure:



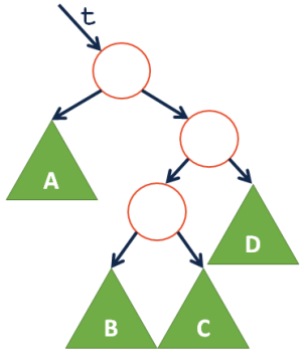
Q1: Source code to perform a left rotation:

```
TreeNode *& t = /* point of imbalance */
TreeNode * y = t->right;
t->right = y->left;
y->left = t;
t = y;
Don't forget to update the heights!
```

Q2: How does a right rotation differ?

Almost the same as the left pseudocode, just swap lefts and rights.

Initial structure:



Q3: Source code to perform a RL rotation:

```
TreeNode *& t = /* point of imbalance */
// Use a combination of single rotations
rotateRight (t->right);
rotateLeft (t);
```

Q4: How does an LR rotation differ? *(You may want to draw a picture and check!)*

rotateLeft(t->left); rotateRight(t);

Removing from an AVL Tree

Another operation that you will implement is being able to correctly remove a node from an AVL tree while maintaining a balanced BST.

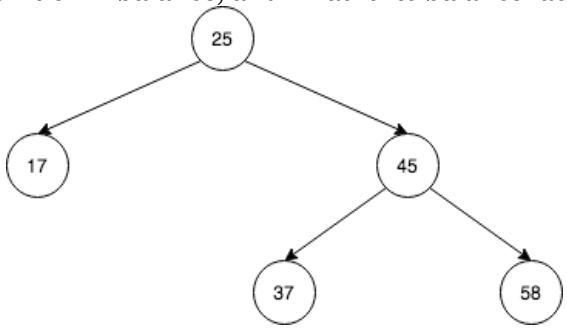
Exercise 3: Fill in the blanks for the pseudocode

```
1: Find the node t that needs to be removed
2: if t is a leaf:  delete the node
3: if t has only a left child: Delete and set t to its
left child
4: if t has only a right child: Delete and set t to its
right child
5: if t has a left and a right child:
   Swap t with its inorder predecessor
   Remove t
6: As the recursion unwinds, rebalance the tree
```

AVL Tree Insertion

Inserting into an AVL tree is very similar to the process of inserting into a BST. The difference between the two is the insertion operation of an AVL tree also rebalances the tree. The pseudocode was given in class, so make sure to look back at your notes for understanding the algorithm of inserting into an AVL Tree.

Exercise 4.1: Insert the element 29 to the Binary Search Tree below. After inserting, find the height for each of the nodes in the tree. Identify the lowest point of imbalance, and what is its balance factor?



Add 29 as 37's left child.
Heights: 29 -> 0, 58 -> 0, 17 -> 0, 37 -> 1, 45 -> 2, 25 -> 3
Lowest point of imbalance is 25 since height(45) – height(17) = 2

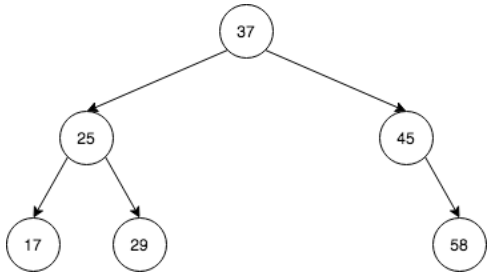
The following nodes are in the elbow: 25, 45, 37

The following nodes are in the stick after the first rotation: 25, 37, 45

Exercise 4.2: Based on the lowest point and its balance factor, and the balance factor of its child node, what is the type or rotation that needs to be used to rebalance the tree?

Rotation Type: Right-Left

Exercise 4.3: Draw the state of the tree after the rotation is done, and it has been rebalanced.



In the programming part of this lab, you will:

- Get familiar with AVL Trees
- Practice writing the implementations of rotations
- Implement the functions to rebalance, insert, and remove an AVL Tree

As your TA and CAs, we're here to help with your programming for the rest of this lab section! ☺