7SENG010W Data Structures and Algorithms
Week 7
Tutorial Exercises: Tree Properties and AVL Trees

These exercises cover: tree properties and AVL trees.

Before you attempt these exercises you should complete Tutorial 5, because for some of these tutorial exercises you should re-use & modify the Binary Search Tree (BST) tree drawings and code from Tutorial 5 and any code you created yourself as part of that tutorial.

Exercise 1.

(a) Using either pen and paper or an online drawing tool, draw (or re-use your previous drawings of) the BSTs that were constructed using the 4 test arrays from the Week 5 Tutorial:

```
int[]  treeA = { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 } ;

int[]  treeB = { 55, 50, 45, 40, 35, 30, 25, 20, 15, 10, 5 } ;

int[]  treeC = { 30, 25, 20, 15, 10, 5, 35, 40, 45, 50, 55 } ;

int[]  treeD = { 30, 15, 45, 10, 40, 20, 50, 5, 35, 25, 55 } ;
```

(b) For each of these BSTs you have drawn label each node of the tree with its:

- node level,
- height,
- balance factor.

See the Week 7 Lecture on Balanced Trees, slides 9, 14, 15 for examples.

Exercise 2.

(a) Based on one of your *Pre-Order, In-Order* or *Post-Order* traversal algorithms you developed for the Week 5 Tutorial (or by using one of the sample solutions), modify it to define C# methods that calculate and display the following properties of a tree:

(I) **Level** of all the nodes in a tree and prints out the list of: *node, level* pairs.

(ii) **Height** of all the nodes in a tree and print out the list of: *node, height* pairs.

You can add these methods to the Binary Search Tree code from the Week 5 Tutorial.

(b) For each of the above algorithms add code to your test program from the Week 5 Tutorial to test each of them on the 4 BSTs.

(c) Compare the results from your test program with the node level and tree heights you added to your drawn BSTs in Exercise 1. What does this mean for their efficiency with respect to the standard tree operations?

Exercise 3.

(a) Using either pen and paper or an online drawing tool, use the `treeC` array values to construct the AVL tree for these values by based on the examples in the Week 5 and 7 Lecture notes and by repeating the following steps:

(1) Insert a value into the AVL tree in the order they occur in the array using the standard BST insertion operation.

(2) Check if the tree is balanced or unbalanced. Do this by following the path from the *inserted node* up to the r*oot node* checking the *balance factor of* each node along the path.

(3) If a node on the path is not balanced then re-balance it using one of the 4 AVL tree rotations: *Left, Right, Left-Right* or *Right-Left* to re-balance it. Then continue up the path to the root.

(4) Draw the re-balanced tree.

(b) After you have finished constructing the AVL tree for `treeC` values, to check that it is correct by labelling each node with its **balance factor** and **height** of the tree with the node as the root. Also check that it satisfies the **BST property**.

Exercise 4.

Construct the AVL trees for the 4 test trees from **Exercise 1** into David Galles AVL tree visualization tool at: https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

It is recommended that you use a separate browser tab for each of the 4 trees.

(a) Compare the **structure** of the AVL tree version with the BST version of each tree. What differences are there? For example, how many **comparisons** would each require to find the values stored in their respective leaf nodes, or the maximum number of comparisons they would needed to find a value, their heights. Put your results in a table.

(b) Compare your drawn AVL tree for the **`treeC`** array, are they the same?

OPTIONAL Exercise 5.

(a) Modify the given C# TreeNode.cs class by adding code to:

- add “height” data member that holds the tree height for the node,
- a set and get methods for it.
- modify the node print method to include the height value.

(b) Using and modifying any of your existing traversal algorithms or solutions to **Exercise 2,** develop a C# method `UpdateHeights` that calculates and updates all of a BST node's height values with its tree height. Add this method to the Binary Search Tree code from the **Exercise 2**.

(c) Add code to your test program to test the `UpdateHeights` method on the 4 BSTs.