# Problem Set 7

## AVL Tree

---

1. * Each node of a BST can be filled with a height value, which is the height of the subtree rooted at that node. The height of a node is the maximum of the height of its children, plus one. The height of an empty tree is -1. Here's an example, with the value in parentheses indicating the height of the corresponding node:

```
     P(3)
    /  \
   M(1)  V(2)
  /    / \
 A(0)  R(1) X(0)
        \
         S(0)
```

   Complete the following recursive method to fill each node of a BST with its height value.

```
public class BSTNode<T extends Comparable> {
   T data;
   BSTNode<T> left, right;
   int height;
   ...
}

// Recursively fills height values at all nodes of a binary tree
public static <T extends Comparable>
void fillHeights(BSTNode<T> root) {
   // COMPLETE THIS METHOD
   ...
}
```

---

2. **WORK OUT THE SOLUTION TO THIS PROBLEM, AND TURN IT IN AT RECITATION**

   In the AVL tree shown below, the leaf "nodes" are actually **subtrees** whose heights are marked in parentheses:

```
          —— D ——
         /        \
        B          F
      /   \       /   \
     A     C     E     G
    / \   / \   / \   / \
   T1  T2  T3  T4  T5  T6  T7  T8
 (h-1) (h) (h-1) (h-1) (h+1) (h) (h)  (h)
```

   1. Mark the heights of the subtrees at every node in the tree. What is the height of the tree?
   2. Mark the balance factor of each node.

---

3. Given the following AVL tree:

```
        —J—
       /    \
```

```
      F     T
     / \   / \
    C   H  N   X
   /\     /\  /
  B  E   L  Q V
         /\
        O  S
```

1. Determine the height of the subtree rooted at each node in the tree.
2. Determine the balance factor of each node in the tree.
3. Show the resulting AVL tree after each insertion in the following sequence: (In all AVL trees you show, mark the balance factors next to the nodes.)
   - Insert Z
   - Insert P
   - Insert A

---

4. Starting with an empty AVL tree, the following sequence of keys are inserted one at a time:

   1, 2, 5, 3, 4

   Assume that the tree allows the insertion of duplicate keys.

   What is the total units of work performed to get to the final AVL tree, counting only key-to-key comparisons and pointer assignments? Assume each comparison is a unit of work and each pointer assignment is a unit of work. (Do not count pointer assignments used in traversing the tree. Count only assignments used in changing the tree structure.)

---

5. * After an AVL tree insertion, when climbing back up toward the root, a node x is found to be unbalanced. Further, it is determined that x's balance factor is the same as that of the root, r of its taller subtree (Case 1). Complete the following rotateCase1 method to perform the required rotation to rebalance the tree at node x. You may assume that x is not the root of the tree.

```
public class AVLTreeNode<T extends Comparable<T>> {
  public T data;
  public AVLTreeNode<T> left, right;
  public char balanceFactor;   // '-' or '/' or '\'
  public AVLTreeNode<T> parent;
  public int height;
}

public static <T extends Comparable<T>>
void rotateCase1(AVLTreeNode<T> x) {
    // COMPLETE THIS METHOD
 }
```