Problem Set 7 - Solution

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
    ...
}
```

SOLUTION

```
// Recursively fills height values at all nodes of a binary tree
public static <T extends Comparable>
void fillHeights(BSTNode root) {
   if (root == null) { return; }
   fillHeights(root.left);
   fillHeights(root.right);
   root.height = -1;
   if (root.left != null) {
      root.height = root.left.height;
   }
   if (root.right != null) {
      root.height = Math.max(root.height, root.right.height);
   }
   root.height++;
}
```

---

2. 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.

   SOLUTION

   Heights/Balance factors
   A:h+1/right, C:h/equal, E:h+2/left, G:h+1/equal, B:h+2/left, F:h+3/left, D:h+4/right
   Height of the tree is h+4

---

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

   SOLUTION

   1 and 2:

```
Node    Height    Balance factor
--------------------------------------
 B        0           -
 E        0           -
 C        1           -
 F        2           /
 H        0           -
 O        0           -
 S        0           -
 Q        1           -
```

```
L         0            -
N         2            \
V         0            -
X         1            /
T         3            /
J         4            \
```

3:

- After Inserting Z:

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

  Only the balance factors of Z and X are changed; others remain the same

- After inserting P (in the tree above):

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

  - Insert P as the right child of O
  - Set bf of P to '-'
  - Back up to O and set bf '\'
  - Back up to Q and set bf to '/'
  - Back up to N and stop. N is unbalanced, so rebalance at N.

  Rebalancing at N is Case 2.

  - First, rotate O-Q

```
       ---J---
      /        \
     F         T
   /   \     /    \
  C    H     N \   X -
 / \        / \   / \
B  E       L   O  V  Z -
                \
                 Q
```

```
  / \
- P  S
```

- Then, rotate O-N

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

- After inserting A (in the tree above):

```
          ---J---
         /        \
        F         T
      /   \     /    \
    /C    H     O-    X -
    / \        / \   / \
  /B  E      /N  -Q  V  Z -
  /          /   / \
-A         -L  - P  S
```

  - Insert A as the left child of B
  - Set bf of A to '-'
  - Back up to B and set bf to '/'
  - Back up to C and set bf to '/'
  - Back up to F and stop. F is unbalanced, so rebalance at F.

  Rebalancing at F is Case 1.

  - Rotate C-F

```
        ---J---
       /        \
     -C         T
    /  \      /    \
  /B  -F      O-    X -
  /   / \    / \   / \
-A   -E H- /N  -Q  V  Z -
           /   / \
         -L  - P  S
```

---

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.)

SOLUTION

Since the tree allows duplicate keys, only one comparison is needed at every node to turn right (>) or left (not >, i.e. <=) when descending to insert.

- To insert 1: 0 units

```
    1
```

- To insert 2: 1 comparison + 1 pointer assignment = 2 units

```
    1
     \
      2
```

- To insert 5: 2 comparisons + 1 pointer assignment:

```
    1
     \
      2
       \
        5
```

  Then rotation at 2-1, with 3 pointer assignments:

```
   root=2, 2.left=1, 1.right=null
```

  Total: 2+1+3 = 6 units, resulting in this tree:

```
    2
   / \
  1   5
```

- To insert 3: 2 comparisons + 1 pointer assignment = 3 units:

```
    2
   / \
  1   5
     /
    3
```

- To insert 4: 3 comparisons + 1 pointer assignment:

```
    2
   / \
  1   5
     /
    3
     \
      4
```

  Then a rotation at 4-3, with 3 pointer assignments:

```
    2
   / \
  1   5   Pointer assignments: 5.left=4, 3.right=null, 4.left=3
     /
    4
```

```
   /
  3
```

And a rotation at 4-5, with 3 pointer assignments:

```
    2
   / \
  1   4   Pointer assignments: 2.right=4, 4.right=5, 5.left=null
     / \
    3   5
```

Total: 10 units

Grand total: 21 units of work

---

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
 }
```

SOLUTION

```
public static <T extends Comparable<T>>
void rotateCase1(AVLTreeNode<T> x) {
   // r is root of taller subtree of x
   r = x.balanceFactor == '\' ? x.right : x.left;
   if (x.parent.left == x) { x.parent.left = r; } else { x.parent.right = r; }
   r.parent = x.parent;
   if (x.balanceFactor == '\') { // rotate counter-clockwise
      AVLTreeNode temp = r.left;
      r.left = x;
      x.parent = r;
      x.right = temp;
      x.right.parent = x;
   } else { // rotate clockwise
      AVLTreeNode temp = r.right;
      r.right = x;
      x.parent = r;
      x.left = temp;
      x.left.parent = x;
   }
   // change bfs of r and x
```

```
    x.balanceFactor = '-';
    r.balanceFactor = '-';
    // x's height goes down by 1, r's is unchanged
    x.height--;

}
```