RUTGERS

CS112 Data Structures

Recitation 07

Yu Yang
yy388@cs.rutgers.edu
Office: CoRE 331
Office Hour: 3-5pm, Wed.

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++;
}
```

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

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--;

}
```