Lab: AVL Trees
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
March 10, 2021

Goal

The objective of this lab is to insert elements into an AVL Binary Search Trees (BST) while maintaining the height balance of AVL nodes. Recall that 4 kinds of insertion may violate the balance property. Restoring the balance of the AVL trees requires single or double rotations. This lab focuses on restoring the AVL tree balance for two out of four cases of inserting elements into the AVL tree.

From the following path, import files to an Eclipse project:

/home/hajiamini/courses/CSC207/code/avl/

AVL Nodes

An AVL tree consists of AVL nodes that are defined in the AVLNode class. Each node has 3 fields: iData, for storing an element, leftChild and rightChild that reference the left and right child nodes. In addition, an AVLNode object has two other fields:

1. height: it keeps track of the height of a subtree rooted at this node.
2. balance: it keeps track of the balance of the node.

AVL Tree

An AVLTree can be viewed as a collection of AVLNodes that satisfy the balance property in addition to the binary search order property. For an AVLNode object, its balance is defined as:

balance = height of right subtree – height of left subtree

An AVLNode is balanced if its balance is +1, -1, or 0. Otherwise, that node is imbalanced.

The AVLTree class has a public find method. This method accepts an element as a parameter and checks if that element exists in the tree. If the element is found, the find method returns an AVLNode that contains the element. Otherwise, it returns null.

Use the binary search order property to implement the find method as a non-recursive (iterative) method.

The insert method performs the following steps to place an element into the tree:
1. Given that the AVLTree is not empty, it checks if the element is already in the tree. If so, it returns IllegalArgumentException. Otherwise, it creates a new AVLNode. The insert method places the given element into an appropriate position using a search path. The search path is a sequence of nodes, starting at the root, that are visited before reaching null. To store the search path, the insert method uses an AVLStack that is implemented as an array.

(2 points) A parent node is used to identify the parent of the new node. Use the parent node to place the new node at the appropriate position. The insert method code has a placeholder for specifying the insertion point.

2. After inserting the new element, the insert method walks back along the search path to check the balance of AVLNodes as defined above.

3. If a node is imbalanced, the insert method performs a single or double rotation at the imbalanced node.

4. After performing an appropriate rotation, the insert algorithm walks back along the search path to update the balance and height fields of AVLNodes that are in AVLStack. These nodes are on top of the recently inserted node.

Complete Step 4 when AVLTree is imbalanced for two cases of insertions: 1) The new element is inserted at the left subtree of the left child of the imbalanced AVLNode, 2) The new element is inserted at the right subtree of the left child of the imbalanced AVLNode.

To restore balance, you need to call either rotateWithLeftChild method for case 1 or the doubleRotateWithLeftChild method for case 2. (These methods are already implemented in the AVLTree class.) After returning from these methods, AVLNode(s) whose balances are changed after doing the rotations need to be pushed into AVLStack. Eventually, the height and balance fields of the imbalanced nodes will be corrected. The insert method has a placeholder for implementing Step 4.

Finally, test your program by building three AVLTrees with these sequences of integers:

- 20, 15, 10
- 20, 15, 17
- 20, 15, 17, 10, 8

(3 points) For each sequence, call the displayTree method to display the AVLNodes of an AVLTree after inserting an integer. For this part of the lab, you receive the points for correctly restoring the AVLTree balance using only the single rotation.