CE204 Lab 8: AVL Trees

David Richerby

Week 24, 2020–21

The lab exercises are not assessed and you are not required to complete all of them. I recommend that everybody attempt exercise 1; exercise 2 is very optional. Feel free to work with others and talk about your answers.

Solutions to exercise 1 will be released on Moodle, on the Friday after the labs.

1 AVL Trees

Implement AVL trees in Java. I recommend the following steps:

1. Implement a binary search tree. If you wish to skip this step, you can download the file `BST.java` from Moodle.

2. Modify the BST so the nodes store the balance factor and the insert method updates balance factors. When a new node is created, it has no children, so its balance factor is 0.

   Test your code. If you downloaded `BST.java`, you can use the method `insertWithCommentary(int[] ints)` to insert a sequence of integers into the tree. At each stage, it prints out the contents of the tree, including balance factors. It won't show the shape of the tree, but you can check that the balance factors are correct by drawing the tree on paper and calculating them by hand.

3. Implement a method `void rotateLeft (Node x)` to do a left rotation (see slide 223). `x` is the node with balance factor +2. You'll need to identify the node `y` and the root of tree T_2. (We don't need to worry about T_1 and T_3, as these are the left child of `x` and the right child of `y` both before and after the rotation.) Move the nodes carefully, remembering to update parent references as well as children, and remember to update the balance factors.

   Test your code again. Inserting 1, 2, 3 in that order should cause a left-rotation at 1 and result in a completely balanced tree.

4. Implement `void rotateRight (Node x)` as a mirror image of `rotateLeft()` by swapping the roles of left and right. Test your code: inserting 3, 2, 1 should cause a right-rotation at 3.

5. Implement `void rotateLeftRight (Node x)` and its mirror-image `void rotateLeftRight (Node x)` (see slide 224). Note that a right-left rotation is a right-rotation at `y` followed by a left-rotation at `x`. Test your code again. Come up with a sequence of insertions that will cause a left-right rotation and one that will cause a right-left rotation.

2 Turing machines

This is more of a mini-project than a lab exercise. You are **not** required complete this or to know how Turing machines work for this course. However, if you like theory and/or like retrocomputing, this may appeal to you. Working with Turing machines is a lot like programming an old-time microprocessor in assembly language.

Find out how Turing machines work and write a Turing machine simulator. Your simulator should allow the Turing machine to be reprogrammed, either by storing the program in variables in the simulator source or by reading a program from a file. The simulator should allow the user to enter an input for the machine (the initial tape contents) and then should simulate the machine on that input.

Write a program for your machine that takes a string of 0s and 1s as input and tells you whether it's a palindrome. The program may overwrite the input as it works.