Project 2 AVL Tree Report

**Author:** Gabriel Bentley

**NetID:** Geb190000

**Course:** CS 3345.502

**Instructor:** Kamran Z. Khan

Project 2 Timeline

* **3/16/21**: Started project, created report outline, created program outline, copied provided code from eLearning, created Book text file, created function to read text from file and create an array list of AVLNodes, modified AVLNodes to have the key be a String variable and take book objects.
* **3/17/21:** Worked on Modifying AVLTree class to work with the AVLNode, insert all AVLNodes into an AVLTree using the AVLTree class, output all rotations as depicted in the project document. Technically completed Part A but will come back to further improve the program after Completing part B
* **3/23/21:** Created a randomBT method that generates a random Binary Tree with x nodes and key sizes 0-y. Created a printBT method that outputs the created binary tree by performing a depth first traversal of the tree. I tested both methods to confirm that they are functioning properly and documented the results of the tests.
* **3/24/21:** Created a isBST method that checks to see if the Binary tree is a BST. Created an isAVL method that checks if a BST is balanced. Created a height method that gets the height of a binary tree.
* **3/25/21:** Created a GUI that puts all the methods together for repeatable and selected testing of the methods. Tested the methods and added results to the report showcasing how the program works as a whole and proving requirements for both Part A and Part B of the project are satisfied.
* **3/26/21:** Finalized the report for Project 2

Introduction

Project 2 Involving the AVL tree abstract data type, is divided into two parts; Part A where data is read from a text file and an AVL tree is constructed from the keys in the file, with all rotations being reported to the console as the tree is being constructed, and Part B where a program is created to determine if a randomly created binary tree is a Binary search tree, and if it is, is it an AVL tree. For both Part A and Part B the program should be capable of handling 25 entries, and other methods may need to be created to show that the tasks in Part A and Part B are completed.

Approach

I started working on part A of the project first, I made a list of 25 book items in a file, within project 2. The book titles, and author names for each item were made up on the spot since all that really matters is that the title and author names have string values to be put into the book object. I modified the AVL node class provided to hold a book object as one of its variables, and to have the key be a string, so when the ISBN number is read from the text file it will be put into the key variable of the AVL node object. Then I made a method that reads in text from a file, puts the values read into book and AVL node objects, and adds the AVL nodes to an array list that will later be used to build the AVL tree.

The next day I modified the given AVL tree code, so that it would run with the changes I made to the AVLNode class. The only big change that happened from using the new node is that the key variable is a string instead of an integer, so everywhere in the code where there is a comparison between two integer key values, I replaced those with the string key values of the node. Since Java does not have a simple way to compare to string values, I converted the key strings to integers with the parseInt() method. I also added print statements in the AVL code so that when a rotation takes place, that rotation is outputted to the console.

For part B of the project I made a method that creates a random binary tree to test if they were BST or AVL trees. The method made keys for each node, by using the random class to get random integers that can be up to the max key size, provided by the user. The number of nodes in each subtree is randomly decided through the random class, and the function is recursively called for the left and right branch of the binary tree node until the number of nodes in a subtree is equal to zero. I also made the printBT method that travels through a binary tree by visiting each node, through the use of recursive calls. When a node is visited depth, key, and if it was a left or right child node is printed, which makes it easier to visualize the binary tree.

The last thing to do for part B was to make the methods that would determine if the given binary tree was a BST and if it is, is it an AVL tree. Checking if a binary tree was relatively easier after making a random one, all I needed to do was visit each node in the tree recursively and check that the right child key has a greater value than the current nodes key, and the left child key has a lesser value. Checking if a BST was an AVL tree was a little trickier, but once I split the process of checking into two different methods it became much simpler. The isAVL method checks if the balance at each node is <= 1, by visiting all nodes in the tree recursively, and the height method gets the height of a node, which was passed to the isAVL method for balance checking purposes.

Finally, to finish up the project I created the gui method to show the results of the programs methods more neatly and make testing each method easier. I gave the gui method a switch statement with 5 options, the first two call the methods related to parts A and B of the project, the next two provide validity to the isBST and isAVL methods correctness, and the last one exits the program which runs in a while loop until the user decides to exit.

Challenges

* I was not used to interacting with files in java, as I did most of my reading and writing programs in C++, so I researched online file reading for the java language.
* Creating a random binary tree was difficult since the branches of the tree have to be randomly decided. If I went with a fixed branch structure going down row by row, then whenever the binary tree created was a BST it would also be an AVL tree. First I tried to use iteration to create the random tree, but found an easier solution through the use of a recursive construction of the binary tree. This further convinced me that recursion was worthwhile to learn in previous courses.
* Creating a method to check if a binary tree was an AVL tree was difficult so I listed out the conditions needed for the tree to be and AVL tree, and determined that creating a separate method to check the height of a trees node would be needed to check if the binary tree was balanced.

Part A

**Methods used**

* All methods in the AVL tree code provided by the instructor were used in the creation of the AVL tree, with some modifications to the code, to accept an AVL node when making the tree
* The readFile method, was used to read in all lines of text, and add the books found into book objects that are put into AVL nodes, that are then added to an array list of AVL nodes.
* The gui method was used to build the tree, by inserting AVL nodes into the AVL tree object in a for loop for each item in the Array List

**Final AVL tree after all ISBNs from the bookList file have been inserted into the Binary tree**

* **Diagram

  Description automatically generated**
  + I didn’t enter the full numbers but since none of the ISBN keys were within 10 digits of each other it does not affect the actions of building the AVL tree
  + I checked at each insertion to make sure my program outputted the right rotations when building the tree, and found that the program performed rotations the same as the visualization of an AVL tree did, thus showing that the book items where made inputted into the AVL tree correctly

**Showing the rotations that took place when inserting new elements into the AVL tree**

* Text

  Description automatically generated

Part B

**Methods used**

* randomBT: creates a random binary tree with x nodes, and key sizes of up to y
* printBT: prints out the contents of the binary tree in a depth first search
* isBST: checks to see if the binary tree satisfies the conditions of a binary search tree
* isAVL: checks to see if a binary search tree satisfies the conditions of an AVL tree
* gui: prompts the user for input on number of nodes and key size for the binary tree to be created, and calls all other methods

**Test: program output of random binary tree with 10 nodes of random keys 0-10**

* **Text

  Description automatically generatedDiagram

  Description automatically generated with medium confidence**
  + Visual representation of binary tree, numbers written represent the order nodes were visited in the traversal.
* This has verified the creation of a random binary tree

**Test: program output of random binary tree with 25 nodes of random keys 0-25**

* Text

  Description automatically generatedA picture containing timeline

  Description automatically generated
  + Visual representation of binary tree, numbers written represent the order nodes were visited in the traversal
* This has verified the creation of a random binary tree.

**Test: checking to see if the isBST method correctly determines if a binary tree is a BST or not**

* **Graphical user interface, text

  Description automatically generated**
  + Correct output
* **Graphical user interface, text

  Description automatically generated**
  + Correct output
* **Text

  Description automatically generated**
  + Correct output

Example of Code Execution

**Proving Program satisfies the conditions for Part A of Project 2**

Graphical user interface, text

Description automatically generated

**Proving Program satisfies the conditions for Part B of Project 2**

Text

Description automatically generated

Text

Description automatically generated

**Proving program can detect BST’s**

Text

Description automatically generated

**Proving program can detect AVL Binary trees**

Text

Description automatically generated

**Contents of bookList file**

18074 Hello Greater

19481 Good Great

70274 Bad Criminal

63378 Cool Shade

45529 Hot Sun

43637 Fast Bullet

40568 Slow Snail

27065 Up High

78637 Down Low

62913 Front Honest

10371 Back Lier

83821 Book Tablet

98528 Koob Telbat

63318 Giant Tall

68758 Tiny Small

79564 Candy Sweet

27703 Fruit Health

74602 Fun Smile

39308 Work Busy

23487 Fish Shark

88589 Insect Fly

45709 Bird Eagle

18172 Mammal Mouse

24338 Reptile Snake

33074 Amphibian Frog