MREN 178 - Lab 4

AVL Trees

Jano Fu

Matthew Pan, PhD

A. Introduction

A.1 Objectives

In the previous lab, we implemented binary search trees (BST) that stored ultrasonic sensor data as well as strings. As we talked about in the lectures, one weakness with BSTs is the possibility for their search performance to degrade significantly. For example. if the data inserted into a BST is somewhat ordered, then the tree begins to resemble a long linked list, with a few branches off to the side. Performance on searches of this type of tree degrades significantly, moving from O(log N) towards O(N).

In this lab, we will be experimenting with AVL trees, a type of self-balancing BST tree that keeps search performance at O(log N). The self-balancing part is a bit of a misnomer, in that you will have to implement algorithms to achieve tree balance whenever an item/node is inserted in the AVL tree. Recall from class that these algorithms will include single and double rotations. Selection of which rotation to use depends on where the new item/node is inserted in the tree. This may be a good time to pull out your class notes on AVL rotations.

Now to introduce the sensor of the week; for generating our data, we will be using a GY-273 HMC5883L triple-axis magnetometer in this lab (Figure 1). The HMC5883L (the black chip on the center of the board) is an ultra-low-power (3.6V at 100 microamps), high-performance 3-axis digital magnetic sensor. It measures magnetic field in all three directions: X, Y, Z, with the units being milli-Gauss. Magnetometer sensors such as this one are typically used in smartphones, laptops, robot navigation, and virtual reality devices.

Figure 1. GY-273 HMC5883L Triple-Axis Magnetometer Module. What is likely to be found in your kit will look like this, but will actually be a QMC5883L sensor that is visually identical (it will even have identifier 'HMC5883L 'silkscreened on the board).

To learn more about the sensor, you can examine the datasheet included as part of the lab documentation, or alternatively find here:

https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf

With all that being said about the HMC5883L, there's something that I need to come clean about: it is very likely that what you have in your kits is not a HMC5883L module. Unfortunately, the HMC5883L (made by Honeywell) has been out of production for a number of years. However, to capitalize on the post-production demand of this sensor in consumer and hobby electronics, a chip manufacturer (QST) has flooded the market with a near-clone of the HMC5883L named the QMC5883L. Unfortunately, the genuine HMC5883L and QMC5883L modules have the exact same markings – boards with the QMC5883L are also incorrectly labeled as 'HMC5883L'. Thus it is impossible to tell one from the other by visual inspection. The wiring of these sensors are the same; however, the source code needed to read data from each module is different. Additionally, the QMC5883L is more sensitive (to noise) and more difficult to calibrate; though, for the purposes of this lab, the QMC5883L will work fine.

The data sheet for the QMC5883L can be found in the OnQ folder for this lab. Welcome to the shady world of electrical components.

A.2 New Topics Covered

- GY-273 HMC5883L/QMC5883L Magnetometer
- AVL Tree
- Comparing the performance of BST (from Lab 3) and AVL trees.

A.3 Pre-Requisite Knowledge

This lab assumes that you have completed the pre-lab for Lab 4 and have reviewed the general concepts/method of AVL search tree. You will also be using C programming knowledge for this lab exercise.

A.4 Equipment and Supplies

Most everything that you need to complete this lab is included.

From your MREN 178 lab kit, you will need:

- Arduino UNO R3 board
- GY-273 HMC5883L or QMC5883L Magnetometer
- Magnet
- Solderless Breadboard
- Male-Male Jumper Wires
- Male USB-A to Male USB-B Cable

From the OnQ MREN 178 course site, you will need:

- Lab 4 Instructions (this document)
- The zip file containing source code for the lab.

On your windows-based computer, you'll need:

- A C programming IDE (e.g., CLion, VSCode, etc.)
- Arduino IDE
- Internet Access

For this lab (at least for the first exercise), you will need to use a Windows-based computer as the serial communications source code used depends on Windows-specific headers.

A.5 Other Resources

https://www.best-microcontroller-projects.com/hmc5883l.html

QMC5883L

https://github.com/mprograms/QMC5883LCompass

https://www.circuitbasics.com/how-to-setup-a-magnetometer-on-the-arduino/

HMC5883L

https://www.electronicwings.com/arduino/magnetometer-hmc5883l-interfacing-with-arduino-uno

B. Lab Setup

B.1 Connecting the Arduino UNO R3 and GY-273 Board

The pins on the GY-273 HMC5883L or QMC5883L Magnetometer are labeled on the back of the board.

Pin	Description
VCC	This is the power in pin, give it 3-5VDC. It is reverse-polarity protected. Use the same voltage as you do for the controlling microcontroller's logic. For an Arduino, 5V is best.
GND	This is the common power and data ground pin.
SDA and SCL	These are the I2C data and clock pins used to send and receive data from the module to your microcontroller. I2C is the communications protocol (you don't need to know the details – we'll be using a library to communicate with the sensor)
DRDY	This is the 'data ready' pin output. If you want to stream data at high speed (higher than 100 times a second) you may want to listen to this pin for when data is ready to be read. Check the datasheet for more details. We aren't going to use this pin for this lab.

Connect the GY-273 HMC5883L or QMC5883L magnetometer sensor to the Arduino using the wiring diagram in Figure 2. Double check your wiring connections before connecting the Arduino to your computer.

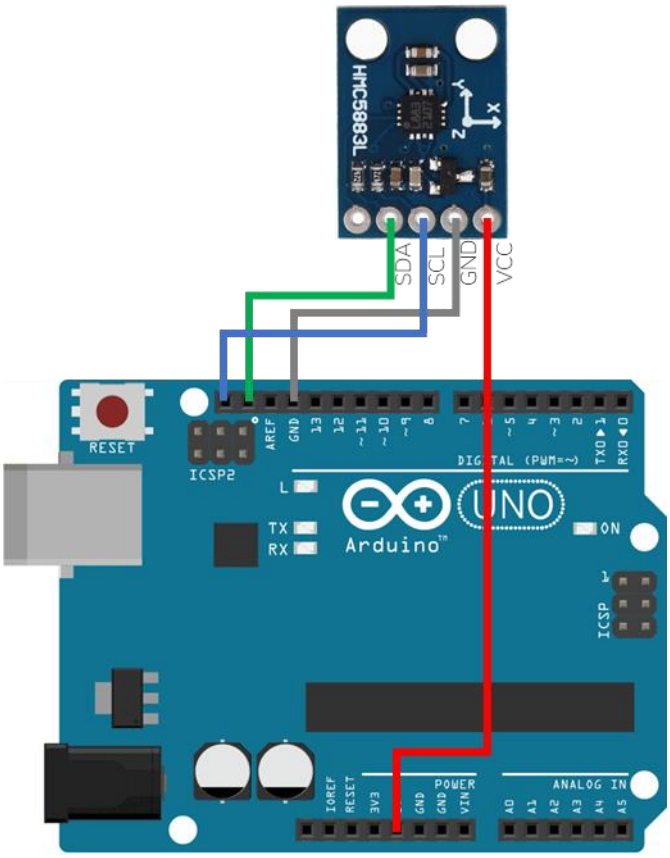

Figure 2. Wiring schematic for GY-273 to the Arduino.

Unfortunately, some boards may have had the pins soldered on incorrectly, leaving the pins shorter than what is necessary in order for female-ended jumper wires to directly connect to the GY-273 HMC5883L or QMC5883L board. Thus, you may need to plug it into the solderless breadboard to make a successful connection.

B.2 Loading Arduino Code

In order to read from the magnetometer, we need to install the relevant library, just like what we did in labs 1 and 2. As mentioned earlier, it is extremely likely (but not guaranteed) that you have the QMC5883L as opposed to the HMC5883L. Given this uncertainty, we'll try installing the library the QMC5883L first, and testing if it works. If not, we can proceed to installing the libraries needed for the HMC5883L.

QMC5883L

In your Arduino IDE, and go to “Sketch” -> “Include Library” -> “Manage Libraries…”; or simply use the short cut “Ctrl+Shift+I”. This should open up the Arduino library manager.

Search for QMC5883L, and install the newest version of “QMC5883LCompass” as shown in Figure 3.

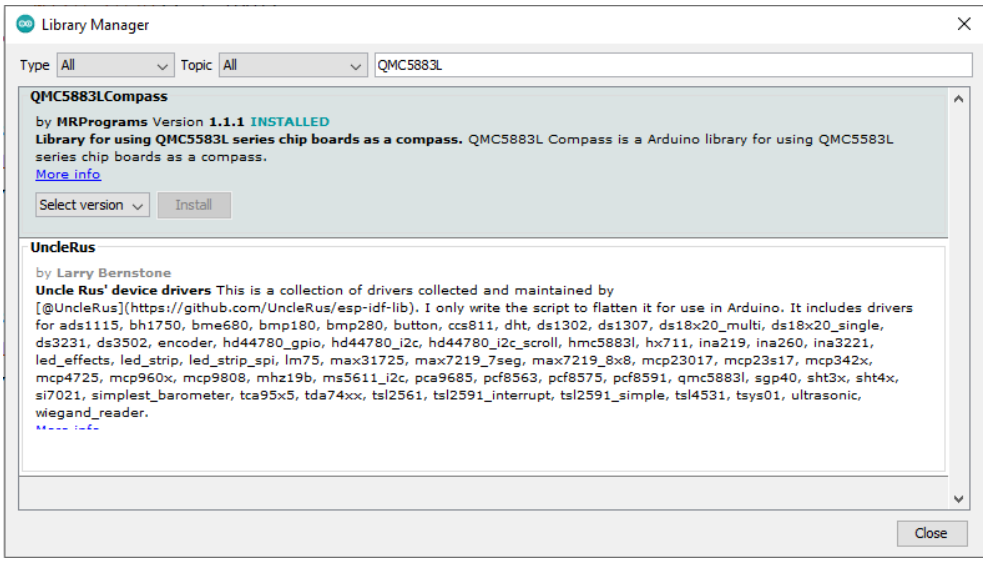

Figure 3. Arduino Library Manager showing QMC883LCompass.

Once installed, plug in your Arduino (wired to the magnetometer) into the computer using the USB cable.

In the Arduino IDE, go to “File” -> “Examples” -> “QMC5883LCompass” (at the very end of the menu) and open up the “xyz” example sketch. Compile and upload this code to the Arduino.

Open up the Serial Monitor (Ctrl+Shift+M) and set the baud rate to 9600. XYZ values showing the field strength in microTeslas (μT) should be streamed to the monitor. If that works, congratulations! You have a working QMC5883L sensor. Try exploring some of the other examples in the QMC5883LCompass folder to understand the capabilities of the QMC5883L sensor. You can also refer to the code repository/documentation here: https://github.com/mprograms/QMC5883LCompass

If nothing is displayed, try resetting or unplugging/reconnecting the Arduino. If still nothing is displayed, move onto the next section, where we will try installing the HMC5883L library.

HMC5883L

In your Arduino IDE, and go to “Sketch” -> “Include Library” -> “Manage Libraries…”; or simply use the short cut “Ctrl+Shift+I”. This should open up the Arduino library manager.

Search for HMC5883L, and install the latest version of “Adafruit HMC5883 Unified”.

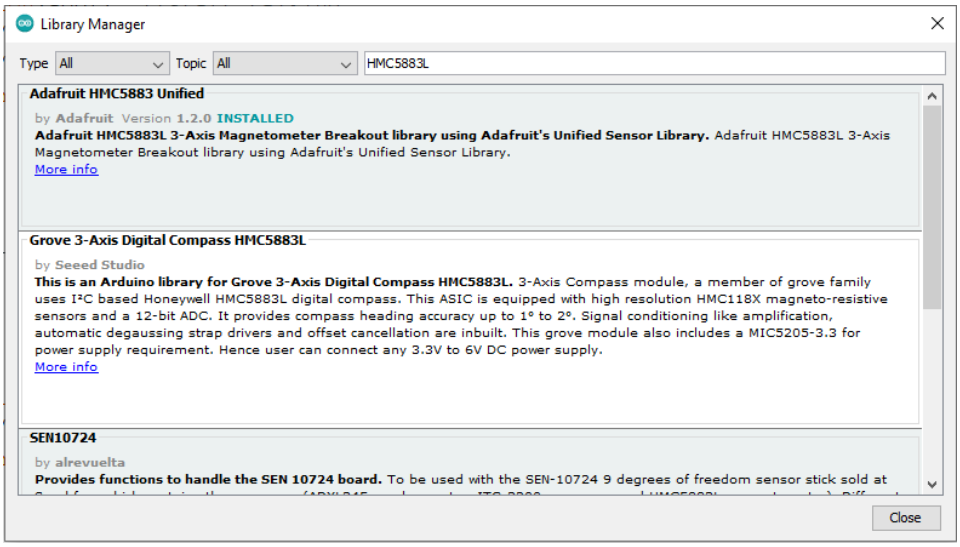

Figure 4. Arduino Library Manager showing Adafruit HMC5883 Unified.

You’ll need to agree to install Adafruit dependencies. Once installed, plug in your Arduino (wired to the magnetometer) into the computer using the USB cable.

In the Arduino IDE, go to “File” -> “Examples” -> “Adafruit HMC5883 Unified” (at the very end of the menu) and open up the “magsensor” example sketch. Compile and upload this code to the Arduino.

Open up the Serial Monitor (Ctrl+Shift+M) and set the baud rate to 9600. Sensor details like the following will be displayed:

```
------------------------------------
Sensor:       HMC5883
Driver Ver:   1
Unique ID:    12345
Max Value:    800.00 uT
Min Value:    -800.00 uT
Resolution:   0.20 uT
------------------------------------
```

This will be followed by the a continuous stream of x,y and z values showing the magnetic field strength detected by the sensor in microTeslas (μT) should be displayed. If this occurs, congratulations! You have a working HMC5883L sensor.

If nothing except the sensor details is displayed (i.e., xyz data is not streamed to the monitor), try resetting or unplugging/reconnecting the Arduino. If still nothing is displayed, you may have a defective sensor (this happens sometimes…these are cheap sensors). Try using your lab partner's sensor and run the example code for QMC5883L and HMC5883L; one of these should work. If not, consult a TA.

B.3 QMC5883L Calibration (Optional)

If you have determined that you have a QMC5883L, you can perform a calibration such that the sensor provides 'more correct' readings.

1. Ensure that your QMC5883L chip is connected to the Arduino.
2. Locate the included calibration sketch under Examples -> QMC5883LCompass -> Calibration.
3. Upload the calibration sketch to your Arduino and then open the serial monitor.
4. Follow the directions on the screen by moving your sensor around when the calibration process starts.
5. Once all calibration data has been collected, the sketch will tell provide you with some code that will look like `compass.setCalibration(-1537, 1266, -1961, 958, -1342, 1492);` Copy this code. You can insert this line of code when you upload the Lab 4 sketch in B.4.

B.4 Load Arduino Code

Similar to Lab 3, the Arduino with magnetometer will act as a stand-alone data generator – the sole function of the Arduino is to read data from the magnetometer and output it to serial at 9600 baud.

Reading data from serial means accessing data that are printed into the Serial port through the command "Serial.print()". To have the Arduino correctly reading the sensor data, we first need to upload the code.

1. Go into the folder "ArduinoCode" and open the folder "QMC5883L" or "HML5883L" depending on the sensor that you have. Then open the .ino file inside the folder.
2. If you have the QMC5883L sensor and you performed the calibration in B.3, you can paste the calibration code (e.g., `compass.setCalibration(-1537, 1266, -1961, 958, -1342, 1492);`) in the setup function, after `compass.init();`

3. Connect the Arduino to your computer using the USB cable.
4. Finding the correct board and port of your connection.
5. Upload the code to the Arduino.
6. Read through the code to understand what's happening.

The code itself should be pretty easy to understand. If you have difficulties understanding the code, read the HMC5883L or QMC5883L data sheet, or ask a TA.

Once the code is uploaded, open the Arduino Serial Monitor to see what gets outputted (ensure that the communication rate selected is 9600 Baud). The values in the x, y, and z fields represent the magnitudes of the magnetic field along each axis (axes shown on the module itself) and are outputted are in units of microtesla (μT).

To test your setup, you can use the magnet: bring the magnet close to the sensor along each of the axes. If you have a bar magnet and place the north pole of the magnet in front of the x axis and you should see the x value decrease significantly. If you put the magnet in front of the y and z axes, the values should decrease too. However, even without the magnet, the sensor is sensitive enough to detect the earth's own magnetic field meaning that you can convert the readings to get a compass heading which is another use for this sensor.

When you're done uploading the code to the Arduino and have tested that data is being outputted by the Arduino over serial, you can close the Arduino IDE.

Make a note of the COM port that the Arduino uses. This will be needed as part of the first lab exercise below.

C. Lab Exercises

For the exercises in this lab, you'll need to use a C programming IDE and compiler of your choice. Because this project contains multiple .c files, you'll need to ensure that your compiler is set up to compile and link all source code contained in the folder. You should have already done this as part of Lab 3 (see Lab 3 Instructions - Appendix A).

C.1 First AVL Tree Program – Storing Measured Data

Description

The code for this part of the lab is found in the folder 'lab4_1'. There will be a few source code files inside:

- *lab4_1_avl.c* - This .c file contains the implementation of the AVL tree and associated algorithms. Some of these functions are incomplete, and will require you to fill it in as part of this lab.
- *lab4_1_avl.h* – This is the associated header file for lab4_1_avl.c. It contains function prototypes for manipulating the AVL tree.
- *bintree.c* – Contains an implementation of a binary tree, and includes functions to initialize a tree and node, and a function to find a node with a specified key value.
- *bintree.h* – This is the associated header file for bintree.c. It contains function prototypes and struct definitions needed for the implementation of binary trees.
- *lab4_1_main.c* – This is where the main function is. You may need to change some the `#define`'ed globals such as COMPORT (the COM port the Arduino is on – you can obtain this from the Arduino IDE), `DATAREADGAP` (default is 1000 = 1 sec, should be aligned with delay in the .ino sketch).
- *SerialClass.c* – The functions in this file are used to read incoming data on the serial port serial port. It's pretty much the same file used in Lab 3, except it has one additional function (`ReadMagnetometer`) that will be used to read the magnetometer XYZ values from the sensor.
- *SerialClass.h* – the associated header file for SerialClass.c.

A little overwhelming with all of these files, right? However, once you dig into what these files contain, you'll realize that they only contain functions that are related to the name of the file. For example, bintree.c/.h only has functions that relate to the creation and finding of nodes in a binary tree. lab4_1_avl.c/.h has AVL specific functions that are required to maintain an AVL tree, and uses functions from the previously mentioned files. Lastly, SerialClass.c/.h handles getting data from serial.

Objectives

The objective of this portion of the lab is to read in magnetometer data published by the Arduino over a serial connection and to store the data in an AVL tree (parts of which you'll need to implement. lab4_1_main.c reads the magnetometer data from serial - the default is that 10 readings will be taken spaced one second apart. Each data point of the x,y,z

magnetic field strengths is stored in a tree node. The Euclidean norm of the field readings is used as the node key, and is calculated via:

$$key = \sqrt{x^2 + y^2 + z^2}$$

In this exercise, you'll implement code to calculate the height and balance factor of any node in an AVL tree, and you will also implement functions to rebalance an AVL node after an insertion (using rotations). From the node definition (in bintree.h), you can see that each node has a field recording its own height. Start by assuming that the heights of the left and right subtrees are correct. Calculate the height of the node root using the heights of its subtrees. No recursion is needed here.

The balance factor of a node root can be calculated using:

$$Balance\ Factor\ =\ H_L\ -\ H_R$$

with H_L and H_R being the height of left branch/ right branch, respectively. After a value is inserted in the tree, the tree may be unbalanced. If the balance factor of a node is less than -1 or greater than 1, then the node is unbalanced and needs rebalancing.

As discussed in class, there are four (4) types of rotations:

Situation	Rotation
Ancestor is L and insertion is to its left subtree	Right Rotation
Ancestor is R and insertion is to its right subtree	Left Rotation
Ancestor is L and insertion is to the left subtree of ancestor's right subtree	Left Rotation on Left subtree then Right rotation (LLR)
Ancestor is R and insertion is to the right subtree of ancestor's left subtree	Right rotation on Right subtree then Left rotation (RRL)

You should review what does each rotation does, and the circumstances in which each rotation is used.

Deliverables

You will notice that the following functions in lab4_1_avl.c require operational code.

```
Node* rotateRight(Node* root)
// Rotate to right.  Returns new root pointer.

Node* rotateLeft(Node* root)
// Rotate to left.  Returns new root pointer.

int getBalanceFactor(Node* root)
// Get balance factor - difference between left height and right height

int calcHeight(Node* root)
// Calculate height of this node by adding 1 to maximum of left, right
// child height.

Node* rebalance(Node* root)
// Check balance factor to see if balancing required (bf > 1 or bf < -1).
// If balancing required, perform necessary rotations.|
```

Fill in the code. Test your code using lab4_1_main.c and ensure the generated tree is correct. You will be submitting the file lab4_1_avl.c

C.2 Deleting Nodes from AVL Trees

Description

Here, we can set aside the Arduino and magnetometer for the remainder of the lab. The second part of the lab involves creating a delete function to delete a node from the AVL tree. The code for this exercise in the lab is located in the folder 'lab4_2'. There will be a few files inside:

- *lab4_2_avl.c* - This .c file contains the implementation of the AVL tree and associated algorithms. A few of these functions are incomplete, and will require you to simply copy your code snippets from lab4_1_avl.c into this file; lab4_1_avl.c and lab4_2_avl.c are almost the same files except for some code designed specifically to work with the data in the text files. Additionally, two additional delete node functions are included – deleteNodeAVL is incomplete and will need to be filled out as part of this exercise.
- *lab4_2_avl.h* – This is the associated header file for lab4_2_avl.c. It contains function prototypes for manipulating the AVL tree. It is almost the same as lab4_1_avh.h.
- *bintree.c* – Contains an implementation of a binary tree, and includes functions to initialize a tree and node, and a function to find a node with a specified key value. In this version of bintree.c, findParent and findParentHelper functions are implemented to help you code deletion of a node from the AVL tree.
- *bintree.h* – This is the associated header file for bintree.c. It contains function prototypes and struct definitions needed for the implementation of binary trees.

- *lab4_2_main.c* – This is where the main function is. This main() includes an interface for deleting items from an AVL tree.
- *data_a.txt* – A text file with data values to be stored in a tree. The data is terminated with a '-1' used to denote end of file.
- *output.txt* – A text file that has the correct output for when the data in data_a.txt is stored in an AVL tree.

To make this part easier, we will not require you to collect data using magnetometer to create the tree. Instead, data will be read directly from a given file (data_a.txt). Because the data being read is different in this part of the lab compared to the previous exercise (lab4_1), you'll need to copy the code snippets that you wrote in lab4_1_avl.c, and insert them in the appropriate places in lab4_2_avl.c.

Objectives

In this exercise, you are required to implement the function deleteNodeAVL, which deletes a node from an AVL tree. As mentioned earlier, the functions deleteNode() in lab4_2_avl.c and findParentHelper() in bintree.c have already been implemented to assist your efforts. Hint: the structure for deleteNodeAVL is somewhat similar to insertNodeAVL.

To check if your AVL tree implementation and delete code is working correctly for this part of the lab, we have included a sample data input file (data_a.txt) which is read by default when the compiled executable (probably named lab4_2_main.exe) is called. lab4_2_main.c contains function calls to test the functionality of your delete. output.txt contains the proper output of the program if your AVL implementation and node delete functionality is working correctly – use this to check whether your AVL code is correct.

Deliverables

Submit the file lab4_2_avl.c contained the completed deleteNodeAVL function.

C.3 Comparing AVL and BST Performance

Description

The third exercise of this lab involves comparing the performance of AVL and BST on identical data (stored in files). The code for this exercise in the lab is located in the folder 'lab4_3'. There will be a few files inside:

- *lab4_2_bst.c* – This .c file contains a basic the implementation of the binary search tree and associated algorithms (i.e., it only contains an insert function).
- *lab4_2_bst.h* – This is the associated header file for lab4_2_bst.c. It contains function prototypes.

- *bintree.c* – Contains an implementation of a binary tree, and includes functions to initialize a tree and node, and a function to find a node with a specified key value.
- *bintree.h* – This is the associated header file for bintree.c. It contains function prototypes and struct definitions needed for the implementation of binary trees.
- *lab4_3_main.c* – This is where the main function is.
- *data_a.txt* – A text file with data values to be stored in a tree. The data is terminated with a '-1' used to denote end of file.
- *output.txt* – A text file that has the correct output for when the data in data_a.txt is stored in an AVL tree.
- *data_b*.txt – A large text file containing many single value data points., also terminated with a '-1' to denote end of file. This is the file that will be used to generate BST and AVL trees in this lab, and to compare performance.

Before we begin, copy 'lab4_2_avl.c' and 'lab4_2_avl.h' from the 'lab4_2' folder into 'lab4_3' and rename the files to 'lab4_3_avl.c' and 'lab4_3_avl.h'. Remember to change the `#include lab4_2_avl.h` in 'lab4_3_avl.c' to `#include lab4_3_avl.h.`

Similar to in lab4_2, to check if your AVL tree implementation is working correctly for this part of the lab, we have included a sample data input file (data_a.txt) which is read by default when the compiled executable (probably named lab4_2_main.exe) is called without any arguments. output.txt contains the correct output of the program – use this to check whether your AVL code is correct.

You will notice that the supplied collection of code contains a version of the BST but is missing some important components. This is because both BST and AVL trees are binary trees, and the code to create a tree descriptor and a node are the same for both. The code to do a find for a specified key is also common to both BST and AVL. Where the code differs is in the Insert and Delete routines. The supplied BST code does not include a delete because it is not required for the lab. Also note that print routines could have been placed in the generic bintree.c module (but were not).

There is a file containing a large set of numbers (data_b.txt) that has been deliberately constructed to exploit a BST weakness – its tendency to become quite imbalanced if its data is inserted somewhat in order. I have provided the code for the data file generator (MakeRandom.c) – it produces a large number of random numbers (which, in theory, are supposed to be evenly distributed through the range) but then takes the 9th decade of the numbers and sorts them. So, about 10 percent of the numbers get inserted in one long

chain, with some of the remaining 10 percent of the numbers forming small branches off the chain.

Objectives

The BST code provided (carried over from Lab 3) does not have the same interface for insert as the AVL code. The BST code only takes a pointer to a node (making it easier to handle the recursion) and the AVL code takes a pointer to a tree – and deals with the recursion by having a second recursive routine. Write additional code in lab4_3_bst.c to make the BST interface match the AVL interface. When looking at the AVL version, can you see a reason why getting a pointer to the tree is preferable to getting a pointer to a node?

Review the file lab4_3_main.c and see how it stores data and how it runs a test. Modify it so that it stores the data in a BST as well as the AVL Tree. Further modify lab4_2_main.c so that it runs a comparable timing test using the BST as is run on the AVL.

After compiling the code successfully, run the executable using the command line command:

```
lab_4_3_main.exe data_b.txt
```

Where 'lab_4_3_main.exe' should be replaced with the name of the compiled executable. Ensure the executable is in the same folder as data_b.txt, otherwise an error will be thrown. Once the two trees are built, finds are performed on the trees. You will notice in lab4_3_main.c that the last 10 numbers read in are saved and used for searches. The code searches for these 10 numbers a million times – a total of 10 million searches. Both the BST and AVL tests should be able to complete this in a few seconds (or less). However, the AVL tree should take a shorter time than the BST. Ensure this is what happens.

Deliverables

Submit the following files for grading: lab4_2_main.c, lab4_2_bst.c and lab4_2_best.h.

D. Grading Rubric

Component	Item	Mark	Max
Lab4_1	lab4_1_avl.c		30
	`rotateRight`		5
	`rotateLeft`		5
	`getBalanceFactor`		5
	`calcHeight`		5
	`rebalance`		10
Lab4_2	lab4_2_avl.c		5
	`deleteNodeAVL`		5
Lab4_2	lab4_2_bst.c/.h		10
	`insertBST`		5
	lab4_2_main.c		
	Code to Build BST Tree		2
	Code to Test BST Tree		3
	Total		45

Congratulations! This concludes the labs for MREN 178! Time to really get cracking on that project.