CSCI 2110 Data Structures and Algorithms
Fall 2024
Laboratory No. 7
Week of October 28 – November 1

Due (on Brightspace): Sunday, November 3, 11.59 PM

BINARY TREES

The objective of this lab is to help you to continue to get familiar with the Binary Tree Data Structure. Your task is to write, compile and run each program using an Integrated Development Environment (IDE). You may use any IDE of your choice – IntelliJ, Eclipse, NetBeans, VSCode, etc.

Marking Scheme: Throughout this course, we will be focusing on the complexity of algorithms and consequently, the efficiency of programs. In your first-year courses, the focus was on creating runnable code to specifications. This course builds on previous knowledge, but also focuses on efficiency. This will involve reducing unnecessary coding and designing or choosing good algorithms.

Each exercise will list the test cases that you need to test the code. Ensure that the output that you generate covers all the test cases.
Each exercise carries 10 points.
Your final score will be scaled down to a value out of 10. For example, if there are three exercises and you score 9/10, 10/10 and 8/10 on the three exercises, your total is 27/30 or 9/10.

Commenting your code: Your code should be clear and easy to understand. While extensive comments are not required, you are expected to include brief, meaningful comments to explain complex logic or any non-obvious sections of your code. Use inline comments to clarify key parts of the code. For methods and classes, use Javadoc to briefly describe their purpose, parameters, and return values. This ensures others can understand your code's intent quickly.

Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.

Submission: One zip file containing the source codes (.java files) for all classes, and sample outputs. Please see details at the end of this document.

Submission Deadline: Sunday, November 3, 2024, 11.59 PM

Grace Time: Submissions will be accepted until 4.59 AM on Monday, November 4, 2024 without late penalty.

Late Penalty: Submissions received after the grace time will be subject to a 10% per day late penalty. The submission portal will close on Wednesday, November 6, 4.59 AM.

Dropping of Labs: Up to 2 labs can be dropped during the semester. No SDA submission is required.

The objective of this lab is to help you get familiar with the binary tree data structure. You will need the following files to complete your work:

BinaryTree.java (Generic BinaryTree Class)

BinaryTreeDemo.java (A class demonstrating the methods of the BinaryTree class)

Download the above files provided along with this lab document.

Exercise 1 (Binary Tree Methods)

For this exercise, you will complete a number of methods in the BinaryTree.java file.

1. Complete the recursive method called countNodes in the BinaryTree.java file. This method should return the number of nodes in a binary tree.

To determine the number of nodes in a binary tree you can apply the following rules:

- if the binary tree is empty, then the number of nodes is zero.
- otherwise, the number of nodes is equal to one plus number of nodes in the left subtree plus number of nodes in the right subtree.

The header for the method is

```
public static <T> int countNodes(BinaryTree<T> t)
```

You will be writing this as a static method so that it can be called from the main method by passing in the root of the binary tree as the input parameter.

2. Complete the recursive method called findHeight in the BinaryTree.java file. This method should return the height of a binary tree.

To determine the height of a binary tree you can apply the following rules:

- if the binary tree is empty, then the height of the binary tree is -1.
- otherwise, the height is equal to 1 plus the height of either the left subtree or the right subtree, whichever is greater.

The header for the method is

```
public static <T> int findHeight(BinaryTree<T> t)
```

You will be writing this as a static method so that it can be called from the main method by passing in the root of the binary tree as the input parameter.

3. Complete the recursive isBalanced method in the BinaryTree.java file. This method should return a Boolean reflecting whether or not a binary tree is height balanced.

A binary tree is height balanced if, for every node in the tree, the height of its left subtree differs from the height of its right subtree by no more than one. In other words, either the left and the right subtrees are of the same height, or the left is one higher than the right, or the right is one higher than the left.

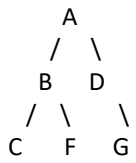
The header for the method is

```
public static <T> boolean isBalanced(BinaryTree<T> t)
```

You will be writing this as a static method so that it can be called from the main method by passing in the root of the binary tree as the input parameter.

4. Complete the levelorder method in the BinaryTree.java file.

Note: A level order traversal goes level by level, from left to right, starting at the root.
The level order traversal of the tree:



Is A B D C F G

This static method should perform a level order traversal of a binary tree passed to it as an argument.

The header for the method is

public static <T> void levelOrder(BinaryTree<T> t)

You will be writing this as a static method so that it can be called from the main method by passing in the root of the binary tree as the input parameter.

To accomplish this, you will implement a simple breadth first search. You can use an ArrayList of type BinaryTree and add the root node, as follows:

```
ArrayList<BinaryTree<T>> q = new ArrayList<BinaryTree<T>>();
      q.add(tree);
```

Then traverse the tree by removing the first element from q, printing, and adding the children of the node removed to q. Repeat until q is empty.

After you complete the methods, test it on BinaryTreeDemo.java that is provided to you. You should see the following output:

Height of the tree is: 3

Number of nodes in the tree is: 6

Inorder:	F	D	B	E	A	C
Preorder:	A	B	D	F	E	C
Postorder:	F	D	E	B	C	A
Level order: A	B	C	D	E	F	

The height of the tree is: 3

And is it height balanced... No.

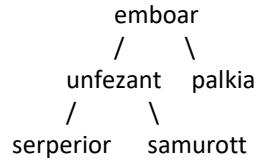
Now make changes to the BinaryTreeDemo.java file and create another binary tree of your choice. Test it and collect the results.

Repeat for three different sample runs. Cut and paste the sample outputs onto a text file.

Exercise 2 (Binary Tree Demo)

For this exercise you will develop a program that can build a binary tree from user input. Create an Exercise2 class (Exercise2.java). You may reuse and expand upon code provided in the BinaryTreeDemo.java file, but your program must **accept input from a user** (an arbitrary number of Strings) and build a binary tree with String data from that input.
The most straightforward approach is to create single node binary trees from each String captured, storing them in an ArrayList of type BinaryTree. You can then quickly build a larger binary tree by assigning the first tree in your ArrayList to be the root node, and attaching subsequent elements to that root.
After building a binary tree, your program should display the height of the tree, the number of nodes in the tree, the inorder, preorder, postorder and level order traversals. You may reuse the test code from the BinaryTreeDemo.java file to this end.

Note: You are building a complete binary tree. It will, by definition, be height balanced. If the Strings provided by the user are emboar, unfezant, palkia, serperior, and samurott, the resulting tree will look like this:



A sample run of your program should look like this:

```
Enter name or done: emboar
Enter name or done: unfezant
Enter name or done: palkia
Enter name or done: serperior
Enter name or done: samurott
Enter name or done: done

Height of the tree is: 2
Number of nodes in the tree is: 5

Inorder:        serperior   unfezant    samurott    emboar      palkia
Preorder:       emboar      unfezant    serperior   samurott    palkia
Postorder:      serperior   samurott    unfezant    palkia      emboar
Level order:    emboar      unfezant    palkia      serperior   samurott
```

Test your program for at least three different sets of inputs. Cut and paste the sample outputs onto a text file.

What to submit:
One zip file containing the following:
1. BinaryTree.java, BinaryTreeDemo.java, Exercise2.java
2. One text/word/pdf document containing sample inputs/outputs. Please ensure your name and Banner ID is on this document.

You MUST SUBMIT .java files that are executable by your TAs. TAs will run your program on the RandomNames.txt file and check your solution. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers.