

Section 12: Binary Trees

Pair up with anyone who is agreeable to pairing up with you, pick the first driver, and let's get to work!

PART I: Expression Trees and Traversals

In class Monday we did a walk-through of the expression tree creation algorithm. Being able to follow an algorithm by hand is an important skill for a computer scientist ... and the postfix algorithm question on the last midterm showed that the class could use some more practice with following algorithms.

1. On the class web page, in the Class Handouts section, is the Expression Tree Creation Algorithm. Open a web browser and open that document.
2. Dig out a sheet of paper and a pen/pencil. Walk through that algorithm on the following expression, and show your final expression tree. (Your SL will want to see it.)

$$s = u * t + 1 / 2 * a * t * t$$

3. Now that you have created a binary tree, we might as well make use of it to practice tree traversals. We learned Monday about these three traversals:
 - (a) Preorder: Visit, Recurse Left, Recurse Right
 - (b) Inorder: Recurse Left, Visit, Recurse Right
 - (c) Postorder: Recurse Left, Recurse Right, Visit

Perform each on your expression tree, and write down the results.

4. Take a look at the inorder traversal and the original expression that you used to create the tree. What do you notice?
5. Take a look at the postorder traversal and the original expression. Does the traversal remind you of anything?

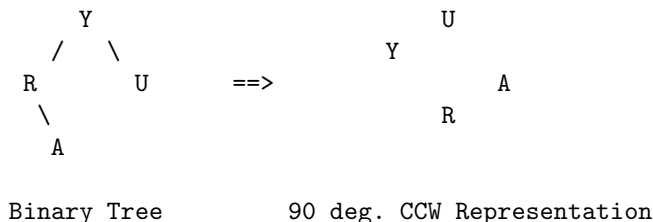
**CHECKPOINT 1**

Raise your hand. Your SL will come over and ask about your answers.

PART II: Printing a Tree Using Indentation

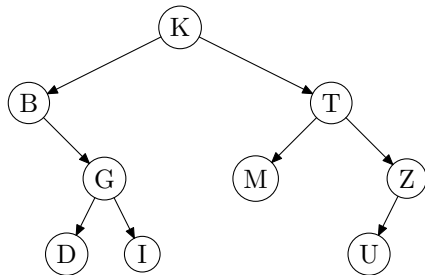
Imagine rotating your expression tree from Part I 90 degrees counter-clockwise. Now imagine printing the data held by each node, one data value per line, with indentation that helps show the depth of each node. If you could do this, you'd have a crude but useful way to display the content of a tree, a technique that would be useful for debugging tree creation algorithms. Bonus: This is easy to do recursively!

- Here's an example of this '90 degrees counter-clockwise' tree drawing using an indentation of four spaces for each level of depth (thus, zero spaces of indentation for Y, the root) and one line of output for each node of the tree:



Look at this carefully until you are confident that you see how the 90 degree CCW representation corresponds to the original tree.

- Your turn! Shown below is a different binary tree. Draw its 90 degree CCW representation.



- Time to write a method that produces such representations. On the class web page are the files `Sec12PartII.java` and `BinaryNode.java`. Load them both into DrJava.
- `Sec12PartII.java` has a `main()` method that builds three binary trees composed of `BinaryNode` objects, and tries to call a method named `printTree()` on each. At the moment, `Sec12PartII.java` cannot be compiled because `printTree()` doesn't exist.

Your job: Bring it into existence! `printTree()` is a recursive static void method that accepts two arguments: A reference to the root node of a tree or subtree, and the depth of that node. Essentially, `printTree()` is a modified inorder traversal. You need to figure out how to modify it to produce the four-spaces-per-level, one-node-per-line output shown in the example above. Remember to think recursively!

The expected output of `main()` is provided in the documentation of `Sec12PartII.java`. When your method is producing that output, you're done with this part.

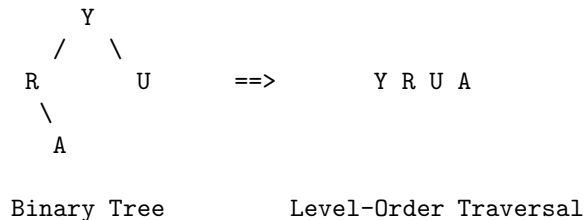
✓ CHECKPOINT 2 Raise your hand. Your SL will come over and verify that your method is working correctly.

(Continued ...)

PART III: Level-Order Traversal

Preorder, inorder, and postorder are closely related, naturally recursive tree traversal algorithms. A fourth type of traversal, known as a level-order traversal, is much easier to do iteratively, because it depends on a queue instead of a stack. In this part you'll write a method that outputs the content of a binary tree, one level at a time.

1. To help you understand what a level-order traversal does, here's our small example binary tree from Part II, this time next to the level-order traversal's output:



Hopefully, this is pretty easy to follow. We are just outputting the content of each level of the tree, with each level's content being displayed left to right. You can also see that a level-order traversal doesn't help reconstruct the tree – we can't tell from the output much more than that Y is the root and that there are four nodes in the tree.

2. Here's a pseudocode algorithm for creating a level-order traversal. As mentioned above, performing this traversal requires a queue:

```
1  If the tree has no nodes, tell the user there is tree to traverse.
2  Otherwise:
3      Create a queue able to hold node references
4      Enqueue a reference to the root node
5      Loop so long as the queue is not empty:
6          Dequeue the front node reference from the queue
7          Output its data value
8          Enqueue its children (if any) into the queue, left child first
9      End loop
```

Walk through this algorithm by hand on our four-node sample tree, to make sure that you understand how it works.

3. On the class web page is the file `Sec12PartIII.java`, which is a lightly modified version of `Sec12PartII.java`. Load it into DrJava.
4. Add a static void method named `levelOrder()` to `Sec12PartIII.java`. It is to be an implementation of the level-order algorithm shown above, subject to the following:
 - (a) `levelOrder()` accepts one argument, a reference to the root node of the tree.
 - (b) Use a JCF (Java Collections Framework) class as your queue representation. Which class is up to you, except that the class must implement the `Deque` interface.
 - (c) There are several required output details:
 - i. If the tree has no nodes, display the message, "This tree has no nodes."
 - ii. The traversal output is to be comma-separated; specifically, between each node value is to be a comma and space.
 - iii. There must not be a trailing comma-space pair after the last value.
 - iv. The traversal's output is to be enclosed in double quotes and followed by a space and the phrase "has *n* characters.", where *n* is the number of characters in the level-order traversal.


(Continued ...)

For example, the expected output from the four-node example tree is:

"Y, R, U, A" has 10 characters.


Expected output is shown in the documentation of `Sec12PartIII.java`.

5. When the output of `Sec12PartIII.java` is matching the expected output, you're ready to raise your hand.

 **CHECKPOINT 3** Raise your hand. Your SL will come over and see how well you followed the directions.

PART IV: Clean Up!

1. Log out of your computer; pick up your papers, writing implements, cell phones, trash, etc.; push in your chair(s).

 **CHECKPOINT 4** Raise your hand. Your SL will come over and look for evidence of moist gum under your desktop.

You're free to go! But, if you have time, we recommend that you use it to work some more on the programming assignment (individually, of course), if you're not already done. (We really hope you were done before Thanksgiving break!)