Project 3: Scientific Trees

DUE: Tuesday, November 5th at 11:59pm

Extra Credit Available for Early Submissions!

DO NOT SHARE "HOW" YOU DID YOUR PROJECT!

WE'RE NOT JUST SAYING "DON'T SHARE CODE"!

PART OF THIS PROJECT IS "DESIGN" AND "CREATIVITY". IF YOU SHARE YOUR DESIGN, THE OTHER PERSON WON'T CREATE IT *THEIR* WAY! EVERYONE NEEDS DESIGN PRACTICE, DON'T STEAL THAT FROM OTHER PEOPLE!

Project Quick Reference

You must:

- Pass the provided style checker from P0 on all your Java files.
- Pass the provided JavaDoc checker from P0 on all your Java files for this project.
- Have code that compiles with the command: **`javac -cp .;../libs.jar *.java`** in your user directory without errors or warnings.
- For methods that come with a big-O requirement (check the provided template Java files for details), make sure your implementation meets the requirement.
- Have code that runs with the commands: **`java -cp .;../libs.jar Display [filename1] [filename2]`**

You may:

- Add additional methods and class/instance variables, however they **must be PRIVATE** (no protected this time!)

You may NOT:

- Use arrays in **`LinkedTree`**.
- Use any built in Java Collections Framework classes in **`LinkedTree`**.
- Add additional class variables (of any type) in **`Convert`**.
- Alter provided classes that are complete (**`Display`**, **`SimGUI, ParentPointer, TreeNode`**).
- Add non-private methods or class/instance variables elsewhere (local variables != class/instance variables)
- Make your program part of a package.
- Alter any method signatures defined in this document of the template code. Note: "throws" is part of the method signature in Java, don't add/remove these.
- Add any additional import statements (or use the "fully qualified name" to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Create a folder for **`p3`**. This is your "project directory".
- Download the **`p3.zip`** and unzip it in your project directory. This will create a folder **`yourCodeHere`**. This is your "code directory".
- Move the style/Javadoc checker from P0 to P3 by taking the following files: **`checkstyle.jar`**, **`cs310code.xml`**, and **`cs310comments.xml`**, and put them in your project directory.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading.

Submission

Submission is on GradeScope (link will open sometime in the week of the 21st). See previous assignments for details on submitting through GradeScope and the feedback that may appear.

Project Sample Schedule

While this project is given to you to work on over about two and a half weeks, you are unlikely to be able to complete this is one sitting. We recommend the following schedule:

- Weekend 1 (by Sunday 10/20): Read the project, & Parts 0a, 0b, and 1
- Week 1 (by Thursday 10/24): Part 2, & get help understanding parent pointers if you need it!
- Weekend 2 (by Sunday 10/27): Part 3 and 4
- Week 2 (by Thursday 10/31): Part 5, & get help with parts 2 and 4!
- Weekend 3 (by Tuesday Nov 5th): Additional testing and debugging OR submit early for extra credit!

Overview

Part 0a: Do your JavaDocs

Annoyed about “wasting time” writing JavaDocs at then end of your project? That’s because you’re doing duplicate work! At the beginning of the project you have to go read and understand all the methods… at the end of the project you’re having to go write all that stuff down. If you do your JavaDocs *as you explore* the code base (at the beginning of the project), you’ll only need to document your private/helper methods at the end. This is much less painful!

Part 0b: Compile and Run the Provided Code

From your ***code*** directory, you can compile the provided code with this line (on Windows):

```
javac -cp .;../libs.jar *.java
```

or this line (on Mac/Linux):

```
javac -cp .:../libs.jar *.java
```

The main assignment can be run in one of three ways:

1) With no parameters, e.g.:
    a. On Windows: `java -cp .;../libs.jar Display`
    b. On Mac/Linux: `java -cp .:../libs.jar Display`
2) With one file parameter:
    a. On Windows: `java -cp .;../libs.jar Display ../sample-inputs/tree1.txt`
    b. On Mac/Linux: `java -cp .:../libs.jar Display ../sample-inputs/tree1.txt`
3) With two file parameters
    a. On Windows:
    `java -cp .;../libs.jar Display ../sample-inputs/tree1.txt  ../sample-inputs/tree2.txt`
    b. On Mac/Linux:
    `java -cp .:../libs.jar Display ../sample-inputs/tree1.txt  ../sample-inputs/tree2.txt`

No parameters opens the GUI, allowing you to visually see the output of parts 2-5 of this assignment.
One file parameter prints a simplified output of parts 2-4 of this assignment.
Two file parameters prints a simplified output of part 5 of this assignment.

You can compile and test these when you download the assignment. It is recommended you try:

1) Compile
2) Run with one file parameter (sample files that will work with this assignment are in the **`sample-inputs`** directory) and see the text output:

```
Parent Pointer Tree:
null
Load: ~-100.00%
Linked Tree:
null
Array Tree:
null
Load: ~-100.00%
```

Will change after Part 2 is done.

Will change after both Part 1 and 3 are done.

Will change after Part 4 is done (so will the other “load”

3) Run with two file parameters (sample files that will work with this assignment are in the **`sample-inputs`** directory) and see the text output:

```
Option 1 not chosen

Option 2 not chosen
```

4) Run with no parameters and see the GUI. If you use the file chooser, you can also see some other stuff.

Part 1: Linked Trees

There is a provided **`LinkedTree`** class that we will be using later in the project. Complete that class! It requires:

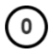
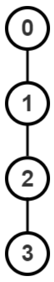
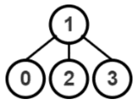
1. A constructor
2. An accessor for the root
3. A **`getSize()`** and a **`getHeight()`** method – best solved using recursion, btw
4. A override of **`toString()`**– instructions in the template file

Testing Part 1: Write a main method to test in **`LinkedTree`**. Construct trees by manually attaching nodes. Then get the size, height, and string output of your trees!

Part 2: Parent Pointer Trees

In class we've covered several ways to store trees. Linked structures (as in Part 1), array structures (root at index 0, children of root at index 1 and 2, grandchildren next, etc.). There is another type of tree storage where elements are stored with parent pointers in an array.

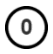
Here are some examples to give you an idea of how this tree storage works:

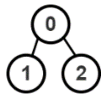
Array	Tree "Picture"	Notes
-1 0		A single node tree where the value 0 is at the root.
-1 | 0 | 1 | 2 0 1 2 3		A tree where the value 0 is at the root. The node with value 1 has a parent of value 0, the node with value 2 has a parent with value 1, and the node with value 3 has a parent with value 2.
1 | -1 | 1 | 1 0 1 2 3		A tree where value 1 is at the root and the other three nodes (containing values 0, 2, and 3) have a parent node with value 1.

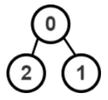
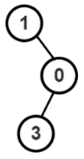
For our application, we have a (unspecified) scientific discipline which is storing their ***binary*** trees in files in this format. As you can see, normally there is no information about which child comes in which order, so the scientists have made a special format for their files that indicates this. They also have a special indicator for values that don't appear in the tree.

Below are some sample files and the trees they represent:

File	Tree "Picture"	Notes

`-1`
`x`
		A single node tree where the value 0 is at the root.
		x means it is not a right or left child

`-1 0 0`
`x L R`
		Parent pointers are stored space-separated on Line 1. Location indicators (R=right, L=left, x=N/A) are space separated on the line below.

-1 0 0 **x R L**		The same tree shape as above, but with the children reversed.
1 -1 x 0 **R x x L**		The value 1 is at the root. Value 2 is not in the tree because there is an "x" for the parent id at index 2.

The scientists want a number of different operations on their trees, Parts 3-4 will handle this, but before we can do that we must read their trees in from the files:

1. In the file **`Convert`** there is a method **`parentPointerFormat()`** which accepts a **`filename`** and returns an array of **`ParentPointer`** objects. The **`ParentPointer`** class is defined at the bottom of the **`Convert`** class and may not be changed.
2. Complete the **`parentPointerFormat()`** method.

<u>Testing Part 2:</u> You can run the one file parameter version of **`Display`** or the GUI, but it is likely better for debugging to write a main method in **`Convert`** and test out reading various trees into files. You can see how to do this by reading the **`Display`** main method, the first three lines of the "**`if(args.length == 1)`**" section should give you everything you need to accept filenames into **`main()`** and nicely print out your parent pointer arrays.

Part 3: Linked Trees Again

The scientists want to convert their parent pointer trees into linked-storage trees.

1. In the file **`Convert`** there is a method **`treeLinkedFormat()`** which accepts a **`filename`** and returns the root of an integer tree **`LinkedTree<Integer>`**.
2. Complete the **`treeLinkedFormat()`** method. Reuse your code from Part 2 for reading from the file! It is much easier to convert parent pointer arrays to linked trees than files to linked trees.
   - How do you do this? There are many ways, actually, and my solution uses arrays, but you have access to all of **`java.util`** so you could use a map if you would find that useful. Whatever you do, DO NOT JUST START CODING AND HOPE… you're going to want to draw pictures to design this part.

<u>Testing Part 3:</u> Expand your main method in **`Convert`** to test making your linked trees. You made a **`toString()`** for trees in Part 1, so use that! The one file parameter version of **`Display`** will also run your **`toString()`** and if you want to see your trees visually, you can use the GUI.

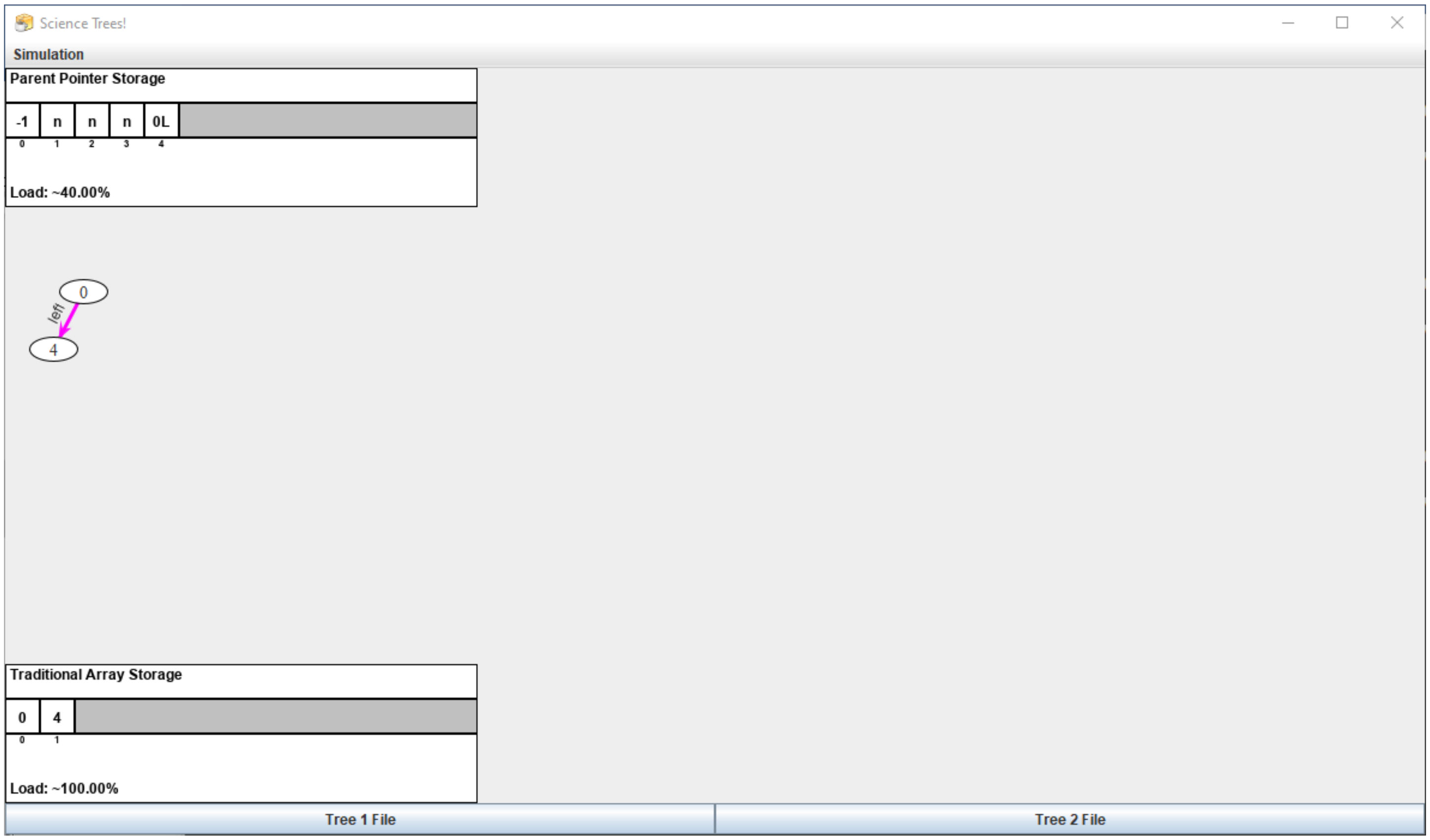
Part 4: Array Trees

The scientists want to convert their parent pointer trees into array-storage trees AND compare the storage efficiency.

1. In the file **`Convert`** there is a method **`treeArrayFormat()`** which accepts a **`filename`** and returns the traditional array storage for trees (level order storage).
2. Complete the **`treeArrayFormat()`** method. Reuse your code from earlier parts!
   - How do you do this? There are many ways, but you may want to look back at what you did in Part 1... but remember you have access to **`java.util`** here so your code can be different.
3. In the file **`Convert`** there is a method **`arrayLoad()`** which accepts a **`tree`** stored in any type of array format and returns the "load" on the tree. The "load" is the number of "used" array slots divided by the number of total array slots, ignoring "extra empty slots" at the end of the array.
4. Complete the **`arrayLoad()`** method.

Testing Part 4: Now you have all the parts to run the first part of our scientific application. The main method in `Display`, given a single filename, will run through parts 1-4 of this assignment on the provided file, and print out the results. You can also do more testing on your own in your `Convert` main method. At this point, you'll probably find the GUI more interesting as well. Using the buttons at the bottom, select a tree file to load it, displaying your parent pointer array, your linked trees, and your traditional array storage. (NOTE: The GUI does NOT display "extra" nulls at the end of your arrays two arrays!)

Here is `tree4.txt` in the GUI:



Parent Pointer Storage

0	1	2	3	4
-1	n	n	n	0L

Load: ~40.00%

Traditional Array Storage

0	1
0	4

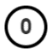
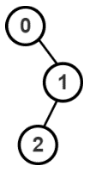
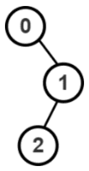
Load: ~100.00%

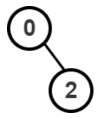
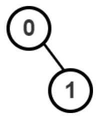
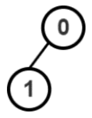
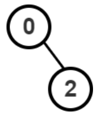
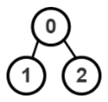
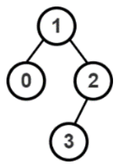
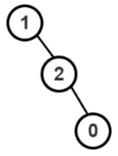
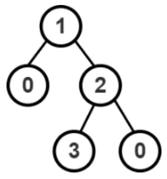
The GUI isn't great at displaying the linked trees (it's built on a graph library rather than a tree library) but you can drag the nodes around if you have trouble seeing edges. You can select a new tree using the "Tree 1 File" button again.

Part 5: Merging Trees

The scientists want to merge their trees and convert them back into parent pointer format, but there are some challenges here: (1) the trees may be strictly incompatible or (2) the trees may be incompatible with parent pointer storage.

Compatible vs. Incompatible Trees:

Tree 1	**Tree 2**	**Merged Tree (if compatible)**	**Notes**
		Compatible: 	Same root, no conflicts.

		Incompatible	Same root, but 1 and 2 are in the same location in the tree
		Compatible: 	Same root, and remaining nodes can be "laid on top of each other" without conflict.
		Compatible: 	While the merge tree does contain duplicate values, it is still considered "compatible". Duplicates will be handed later.

Once a tree is merged, if it contains duplicate values, it will not be compatible with parent pointer storage (because each index can only contain one parent pointer and indexes are the same as values in the tree). So you will have to determine if the merged trees contain duplicate values. If they do not, then you will be asked to convert them into the parent pointer storage.

To do all of the above you have TWO options and you only need to do ONE. [Note: if you do both options only ONE will be graded and the autograder gets to pick, not you! If you want to do both for your own edification, that's fine, but it is recommended that you submit only ONE of them for grading (and only run one option with the GUI).]

1. In the file `Convert` there is a method `merge()`, this method joins two trees together or returns `null` if the trees are incompatible.
2. In the file `Convert` there is a method `containsDuplicates()`, this method returns true if the tree cannot be stored in parent pointer format (i.e. there are duplicate values in the tree).
3. In the file `Convert` there is a method `toParentPointer()`, this method accepts a tree that *can* be stored in parent pointer format and converts it.

Option 1: Linked Storage Merge

4. Complete the three methods which accept `LinkedTree` parameters:
   - **`LinkedTree<Integer> merge(LinkedTree<Integer> tree1, LinkedTree<Integer> tree2)`**
   - **`boolean containsDuplicates(LinkedTree<Integer> tree)`**
   - **`ParentPointer[] toParentPointer(LinkedTree<Integer> tree)`**

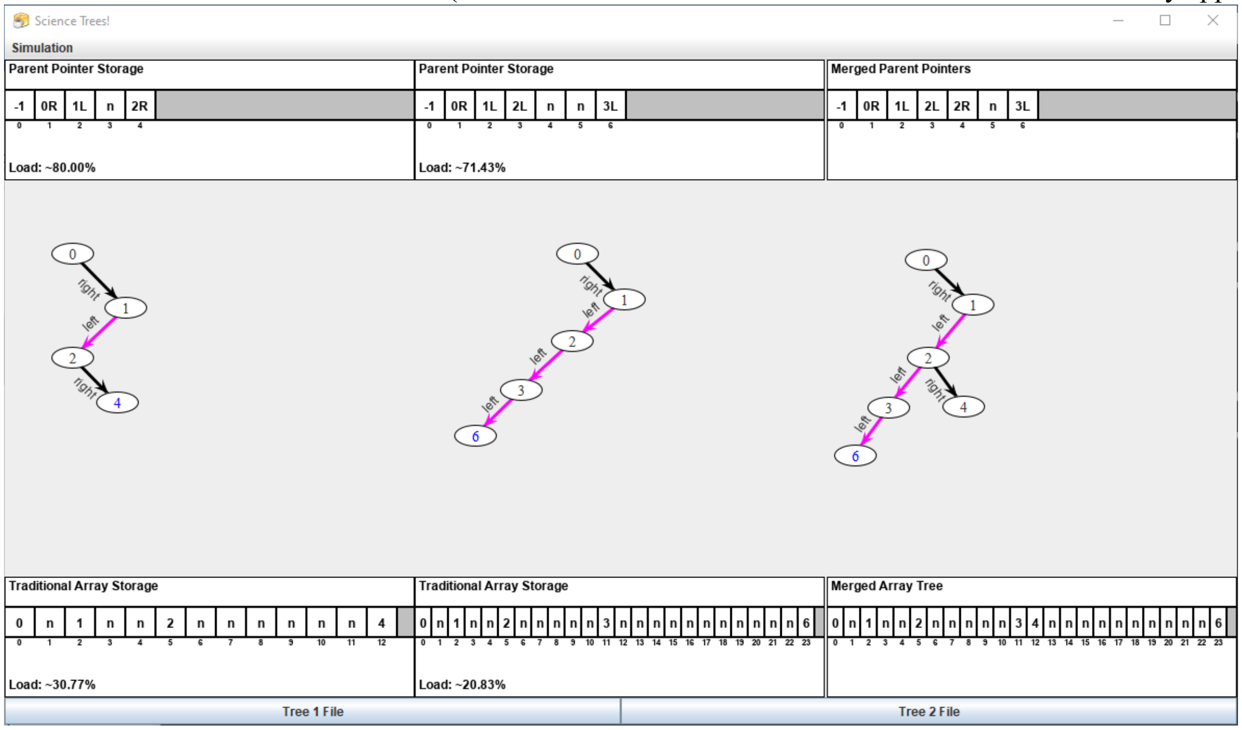
Option 2: Array Storage Merge

4. Complete the three methods which accept `Integer[]` parameters:
   - **`Integer[] merge(Integer[] tree1, Integer[] tree2)`**
   - **`boolean containsDuplicates(Integer[] tree)`**
   - **`ParentPointer[] toParentPointer(Integer[] tree)`**

Testing Part 5: You have all the parts to run the second part of our scientific application. The main method in `Display`, given two filename arguments, will run part 5 of this assignment (merging the trees in the two files), and print out the results of merging, checking for duplicates, and converting back to parent pointers. You can also do more testing on your own in your `Convert` main method.

Additionally, at this point, you can see the GUI do a little more for you as well. If you select a second file it will display the output of the merge, the array or linked storage (if you chose that option), and the parent pointers (if there were no duplicates). If it is able to display the parent pointers, it will also try to convert those into the format of the option you didn't choose so you can see the full working application:

Here is **`tree1.txt`** and **`tree2.txt`** (note: I've moved the linked nodes around to be more visually appealing):



Parent Pointer Storage

-1	0R	1L	n	2R
0	1	2	3	4

Load: ~80.00%

Parent Pointer Storage

-1	0R	1L	2L	n	n	3L
0	1	2	3	4	5	6

Load: ~71.43%

Merged Parent Pointers

-1	0R	1L	2L	2R	n	3L
0	1	2	3	4	5	6

Traditional Array Storage

0	n	1	n	n	2	n	n	n	n	n	n	4
0	1	2	3	4	5	6	7	8	9	10	11	12

Load: ~30.77%

Traditional Array Storage

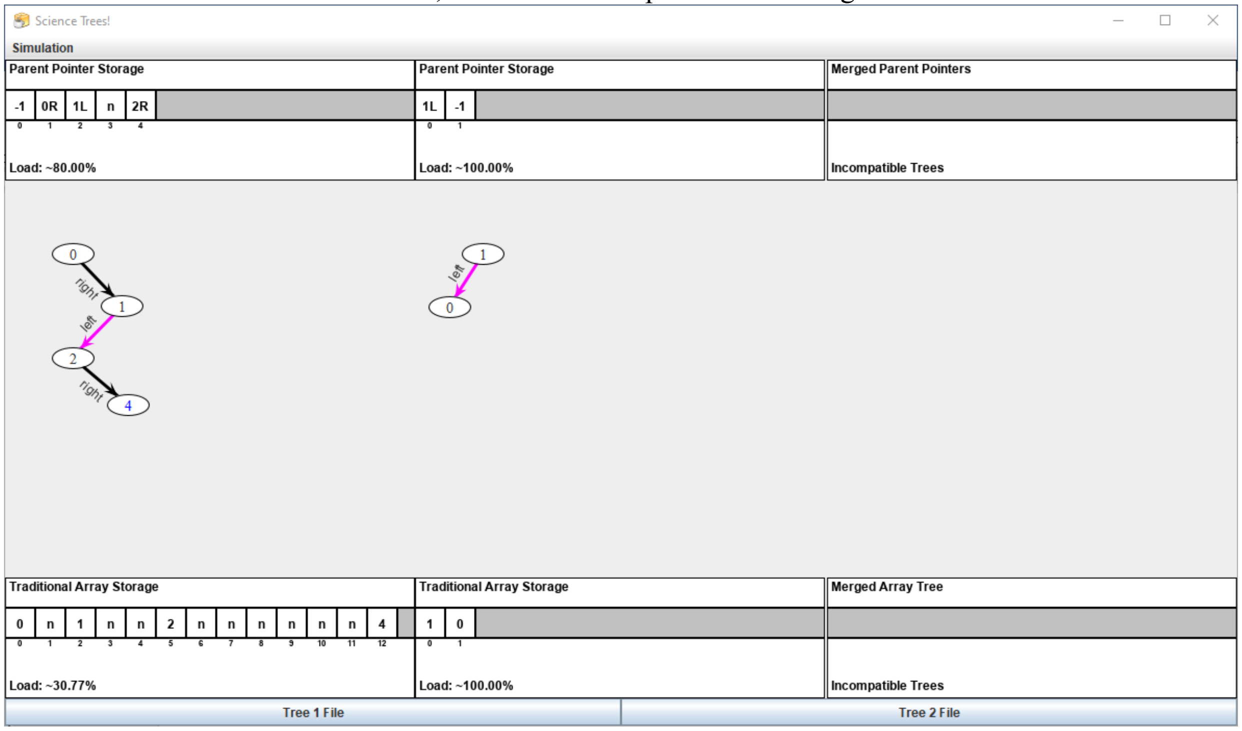
0	n	1	n	n	2	n	n	n	n	n	3	n	n	n	n	n	n	n	n	n	n	n	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

Load: ~20.83%

Merged Array Tree

0	n	1	n	n	2	n	n	n	n	n	3	4	n	n	n	n	n	n	n	n	n	n	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

This is **`tree1.txt`** and **`tree3.txt`**, which are incompatible for a merge:



Parent Pointer Storage

-1	0R	1L	n	2R
0	1	2	3	4

Load: ~80.00%

Parent Pointer Storage

1L	-1
0	1

Load: ~100.00%

Merged Parent Pointers

Incompatible Trees

Traditional Array Storage

0	n	1	n	n	2	n	n	n	n	n	n	4
0	1	2	3	4	5	6	7	8	9	10	11	12

Load: ~30.77%

Traditional Array Storage

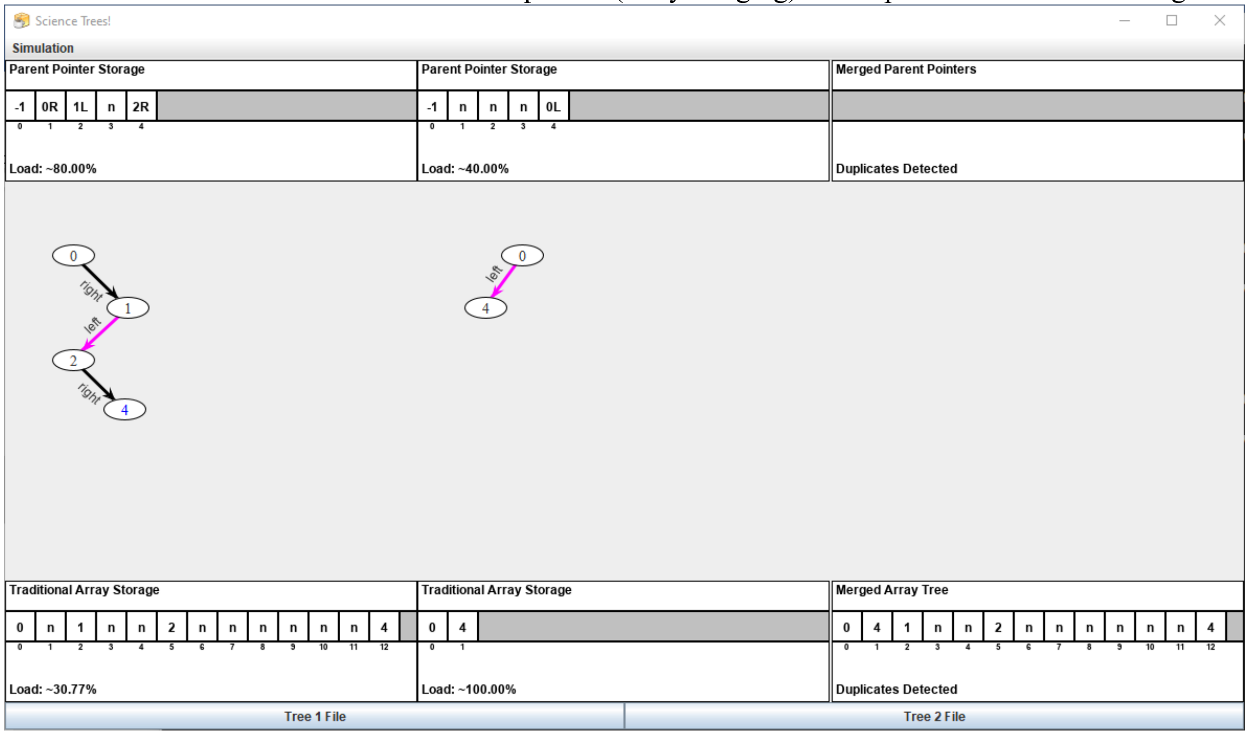
1	0
0	1

Load: ~100.00%

Merged Array Tree

Incompatible Trees

This is **`tree1.txt`** and **`tree4.txt`** with Option 2 (array merging) and duplicate values in the merged trees:



Parent Pointer Storage

0	1	2	3	4
-1	0R	1L	n	2R

Load: ~80.00%

Parent Pointer Storage

0	1	2	3	4
-1	n	n	n	0L

Load: ~40.00%

Merged Parent Pointers

Duplicates Detected

Traditional Array Storage

0	1	2	3	4	5	6	7	8	9	10	11	12
0	n	1	n	n	2	n	n	n	n	n	n	4

Load: ~30.77%

Traditional Array Storage

0	1
0	4

Load: ~100.00%

Merged Array Tree

0	1	2	3	4	5	6	7	8	9	10	11	12
0	4	1	n	n	2	n	n	n	n	n	n	4

Duplicates Detected

<u>Command Reference</u>

All commands assume you haven't been moving files around. See "Common Questions" if the commands don't work before messaging a professor/TA.

From *in* your CODE directory (e.g. "**`yourCodeHere`**"):

```
Compile:      javac -cp .;../libs.jar *.java          (Windows)
              javac -cp .:../libs.jar *.java          (Mac/Linux)
Run the GUI:  java -cp .;../libs.jar Display          (Windows)
              java -cp .:../libs.jar Display          (Mac/Linux)
Run Part 1-4: java -cp .;../libs.jar Display path-to-file     (Windows)
              java -cp .:../libs.jar Display path-to-file     (Mac/Linux)
Run Part 5:   java -cp .;../libs.jar Display path-to-file1 path-to-file2   (Windows)
              java -cp .;../libs.jar Display path-to-file1 path-to-file2   (Mac/Linux)
```

From *in* your PROJECT directory (e.g. "**`p3`**"):

```
Style Checker:     java -jar checkstyle.jar -c cs310code.xml yourCodeHere/*.java
Comments Checker:  java -jar checkstyle.jar -c cs310comments.xml yourCodeHere/*.java
```