A4 – Modeling Bugs with Trees
Developed by Michael Patashnik, revised by Gries

Table of Contents

1. Introduction
2. Bug-infection propagation
3. Installation
4. Running
5. Your tasks
6. Debugging
7. What to submit

1. Introduction

Note: Keep track of the time you spend on this assignment. You have to give it to us when you submit.

We all get bugs, or infections, from time to time. Sometimes they are just an annoyance. Sometimes they are dangerous —like the zika virus. For a dangerous one, the CDC (US Center for Disease Control and Prevention) may track how it is spreading and try to determine where it will spread next, so preparations can be made to fight it. People develop programs to simulate the spread of bugs in order to learn about them, just as they write programs to simulate weather. If you are interested in learning more about using Math/CS to model and understand infectious bugs, check out some of these resources:

- http://idmod.org/home
- https://en.wikipedia.org/wiki/Mathematical_modelling_of_infectious_disease
- http://ocw.jhsph.edu/courses/epiinfectiousdisease/pdfs/eid_lec4_aron.pdf

Assignment A4 uses trees to model the spread of an infectious bug. The root of the tree represents the first person to get the bug; the children of a node represent the people who were infected by contact with the person represented by that node.

Most of the code you write for A4 involves using recursion to explore a tree. We have supplied you with starter code, which simulates the propagation of a bug, as well as a GUI (Graphical User Interface) that allows you to set parameters and visualize the results on the screen. But the simulation won't work until you write recursive methods to process the tree.

Learning objective: Become fluent in using recursion to process data structures such as trees.

Collaboration policy: You may do A4 with one other person. If you are going to work together, then, as soon as possible —well before you submit the assignment— visit the CMS for the course and form a group. Both people must do something to form a group: one person proposes; the other accepts. Don't group and ask us later to group you! That's a five-point deduction. Need help with the CMS? Ask a 2110 staff member or visit www.cs.cornell.edu/Projects/CMS/userdoc/.

If you do this assignment with another person, you must work together. It is against the rules for one person to do some programming on this assignment without the other person sitting nearby and helping. You should take turns driving —using the keyboard and mouse— and navigating —reading and reviewing the code on the screen.

Academic Integrity: With the exception of your CMS-registered partner, you may not look at anyone else's code from this semester or from a similar assignment in a previous semester, in any form, or show your code to anyone else, in any form. You may not show or give your code to others until after the last deadline for submission.

Getting help: If you don't know where to start, if you don't understand, if you are lost, etc., See someone IMMEDIATELY—a course instructor, a TA, a consultant, the Piazza for the course. Don't wait.

2. Bug propagation

In A4, we use a simple model of an infectious bug. This model provides a real world example of the general tree data structure. You are not responsible for writing any of the code that implements the simulation of the bug spreading.

At the beginning of the simulation, you supply 4 parameters:

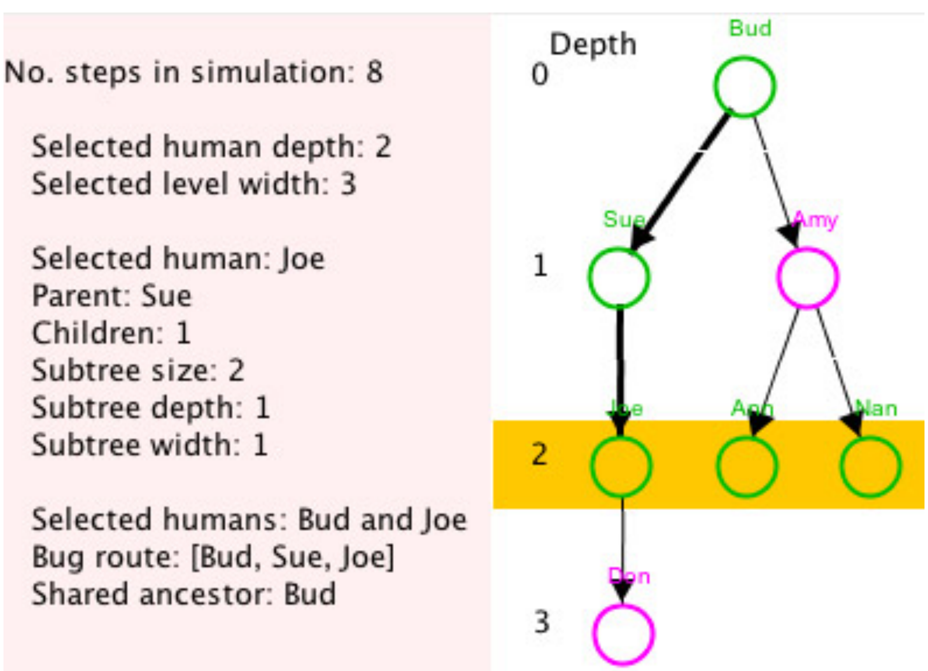
(1) The number of people in the simulation. 10 is a good first choice.
(2) The maximum amount of health, a positive integer. Suppose you supplied 10. 10 means very healthy, 0 means very sick —actually, dead. At each time step during the simulation, an infected human either becomes healthy or has their health decrease by 1. The larger the health range, the longer the program will run since there are more possible states of the simulation.
(3) The probability that two people are neighbors. In the range 0 <= number <= 1, where 0 means no one is a neighbor with another and 1 means everybody is a neighbor of everyone else. Neighbors can infect each other.
(4) The probability that a sick human becomes immune and thus gets well immediately.

At the beginning of the simulation, a graph is constructed with humans as nodes and random edges between them, indicating they are neighbors, depending on the probability that you gave ((3) above). The simulation then starts by making one random human sick and making that human the root of a new bug tree. Initially, that is the only node in the tree.

A series of time steps follows. In each time step, two things happen.

(1) Each sick human may infect one neighbor, who is added to the bug tree as the sick human's child.
(2) Each sick human either gets well (and thus immune from the bug) or gets sicker (their health decreases). A sick human whose health decreases to 0 dies.

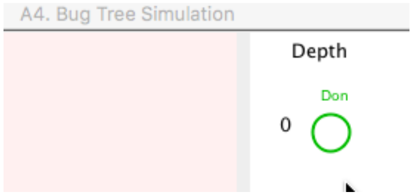
These time steps continue until everyone in the bug tree is either healthy or dead. At the end of the simulation, the results are shown in a GUI on your monitor.



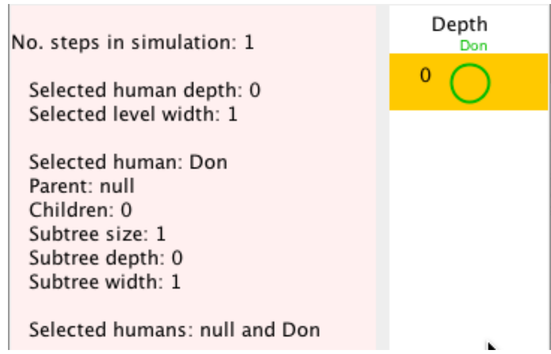
The example to the right shows the output in the GUI. Bud is at depth 0 (the root); Sue and Amy are at depth 1; Joe, Ann, and Nan are at depth 2, and Don is at depth 3.

Bud got the bug first; he infected Sue and Amy; Sue infected Joe, Amy infected Ann and Nan, and Joe infected Don. All seven got ill and two of them died (Amy and Don).

To the left in the image is data that comes from selecting (with your mouse) Bud and then Joe. It shows: Joe's depth, 2; the width (number of nodes) at that level, 3; Joe's parent; number of children; subtree size, depth, and width; and the shared ancestors of Bud and Joe.



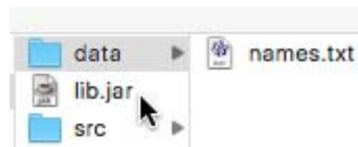
WATCH OUT FOR THIS! A run may result in a tree with one node, as shown to the left. This happens when the one human with the bug doesn't infect anyone, either because they have no neighbors or because when generating a random number to test whether a neighbor got the bug, the result is always "no". **Click the green node to have the data appear to the right!**



3. Installation

1. Download the A4 assignment zip file from the pinned A4 FAQs note on the Piazza.

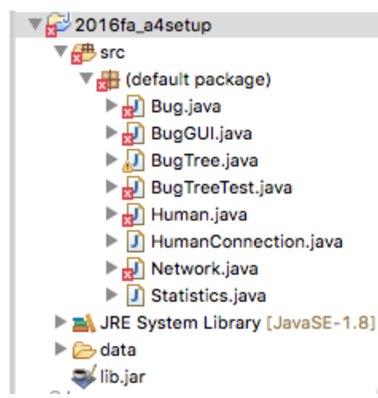
2. Unzip the downloaded file. The folder should contain two folders (data and src) and a file lib.jar. Folder data should look as shown to the right.



3. Create a new Java project (called A4). **IMPORTANT** – you must use java 1.8 for this project. Some of the code relies on functionality specific to java 1.8.

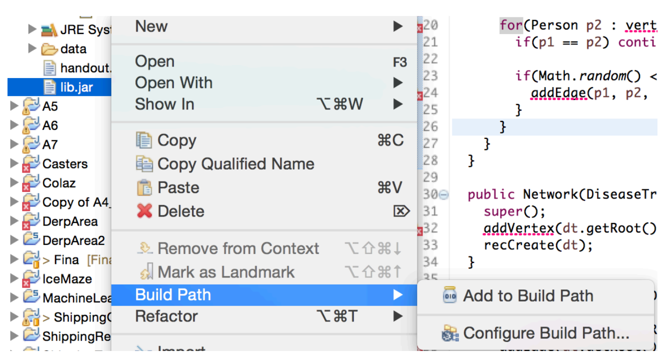
Copy-paste all three items in the downloaded folder into the root of the new Java project in Eclipse (i.e. data, lib.jar, and src). When asked how to copy, select "Copy files and folders", then OK. Also, if asked to replace src, do it.

To the right, you see what folder src should be. Note that many of the files have syntax errors that will be fixed in a moment.



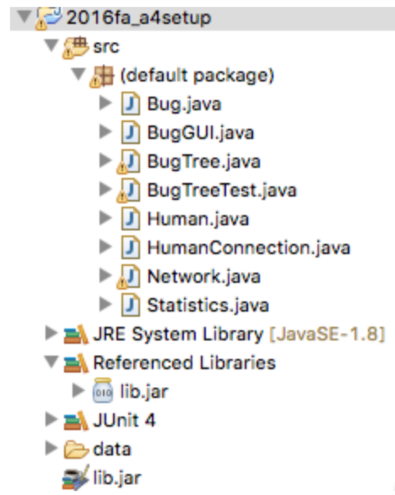
4. Add lib.jar to the build path by right clicking lib.jar and selecting Build Path → Add to build path.

After this, the only file that should show an error is file BugTreeTest.java.



5. Create a new JUnit testing class as usual (File → New → JUnit Test Case), making sure to say “yes” when it asks whether JUnit 4 should be added to the build path. Then delete the newly added JUNit test case file (because you don’t need it).

Your project should now look as shown to the right, and you’re ready to go!



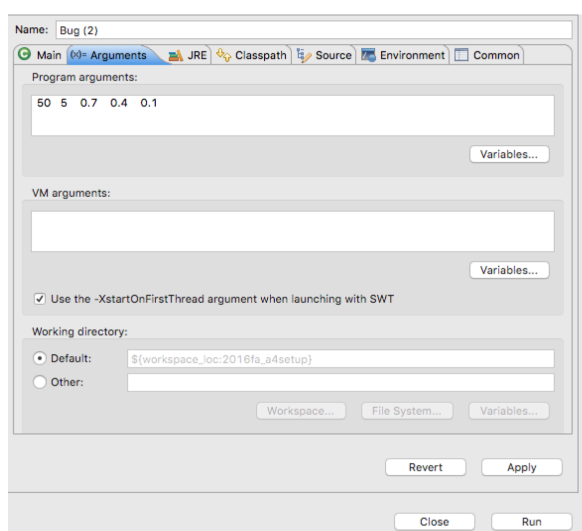
4. Running

The executable class of this project is Bug.java. To run the project, open Bug.java and click run (green button with white arrow) or use menu item Run -> Run. You will be prompted for a few seeding values via the console at the bottom of Eclipse. These are:

1. Size of population: how many humans to model. A positive integer. A higher number may result in a larger tree.
2. Amount of health per human: how long a human can have the bug before dying. A positive integer. A higher number may result in a larger tree. 10 is reasonable.
3. Probability of connection: how likely two random humans are to come into contact with one another. (On page 2, we called them neighbors). A float in the range [0,1]. A higher number may result in a larger tree.
4. Probability of getting the bug: how likely a human is to get the bug when in contact with an infected human. A float in the range [0,1]. A higher may number result in a larger tree.
5. Probability of becoming immune: how likely a human with the bug will become immune (fight off the bug). A float in the range [0,1]. A *lower* number may result in a larger tree.

A nice set of starting values is: (50, 5, 0.7, 0.4, 0.1)

If you want to run Bug.java repeatedly with the same five arguments, put them in the program arguments instead of having to type them into the console every time you run the program. This is done the usual way (Run Configurations … → arguments). For example, the run configurations shown to the right would run the program with the above arguments every time the run button is clicked.



If there is any issue with the arguments provided, either though the console or the program arguments (health is less than 0, for example), you will be re-prompted to enter the arguments through the console. Thus, if you have entered arguments in the arguments tab but are still prompted to enter arguments via the console, there may be something wrong with the arguments you entered.

From there, the bug infection will run —starting with a randomly chosen patient and spreading across that human's connections until no one is left alive and infected.

After the simulation has finished, the full tree is printed out by calling toStringVerbose(); then the tree explorer GUI pops up.

5. Your Tasks

All the code you write is in BugTree.java. In order to complete the assignment, complete each method marked with a //TODO comment (in Eclipse, they are marked in blue on the right of the text). Complete and test them in the order in which they are numbered. Do not delete the //TODO comments. If a precondition is false, any behavior is acceptable. Do the following before you code:

1. Read the comments at the top of BugTree.java.
2. Study the already written methods in BugTree.java. You will learn a lot by that.
3. You *must* read the Piazza pinned A4 FAQs note. It continues useful and necessary info.

Recursion is your friend! Most of these functions are best and most easily written using recursion. Iteration (using loops instead of recursive calls) may be possible but will certainly be more work both to reason through and to debug.

Hint: Some of the functions you are asked to write can be written very simply or even trivially (one or two lines) simply by relying on functions that we or you previously wrote and deubgged.

Warning: Every time application Bug (i.e. method main in Bug.java) is called, a new, random, unrepeatable BugTree is created, so you cannot debug the methods you are writing using that application. It is best to use a JUnit testing class to test your methods and to run the program only when you know your methods are correct. See the pinned Piazza A4 FAQs note.

Dos, Mays, and Don'ts:

- Do read all the Javadoc in BugTree.java thoroughly. You may choose to read the Javadoc in other files, but it should not be too important. Read the files outside the default package only if you are particularly interested in them —you don't need to know more than their javadocs in order to complete the assignment.
- Do not alter any of the other files given to you. You won't be able to submit them, so your BugTree.java must work with unaltered versions of the other files.
- Do not change the method signatures of any method in BugTree. The name and types of parameters should not be changed.
- Do not leave println statements (which you may have added to help debug) in your code when you submit. You will lose up to 5 points. Comment them out or delete them before submitting.

- You may add new methods to BugTree to help complete the required functions (some are only workable with the use of helper methods). Make sure that methods you add are **private** and have good javadoc (/** … */) specifications.

6. Debugging

Class BugTreeTest already contains some testing procedures. Look at them, and also study what we say about testing on the Piazza A4 FAQs note. THIS IS IMPORTANT! We will not look at your tests cases, as we did for A3. All we require is that your A4 methods are correct.

7. What to submit

Complete the information at the top of file BugTree.java: your netid(s), the hours and minutes that you spent on this assignment, and any comments you would like to make on this assignment. Please be careful doing this. We run a program to extract the times and compute statistics. If you are not careful, we have to manually go in and fix you comment. Save us work: Be careful.

Submit file BugTree.java on the CMS.