A4 – Going Viral

Table of Contents

1. Introduction

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

Content —posts, memes, videos— can be shared on social networks. Some content is only posted once. Other content is reposted over and over; it goes viral! A4 uses trees to model how content spreads through a social network. The root of the tree represents the first person to post the content, and the children of each node represent the people who saw the post by 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 that simulates the propagation of a shared post 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, as soon as possible —and certainly 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 and the other accepts.

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 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 another person in the class.

Getting help: If you don't know where to start, if you don't understand testing, if you are lost, etc., please SEE SOMEONE IMMEDIATELY—a course instructor, a TA, a consultant, the Piazza for the course. Do not wait.

2. Sharing in Social Networks

Social networks have become a pervasive component of modern life. They connect friends and professional acquaintances, and they are fun! Social networks also have a growing economic impact. Ad-driven social networks are now some of the most valuable companies in the United States. And influential content-drivers can build careers off of a social network presence. Understanding how content spreads through a social network is therefore a priority for economists, sociologists, and psychologists.

In A4, we use an extremely simple model of how content is shared in a social network. We model a social network as a set of cliques —subsets of the network corresponding to social groups in which almost everyone is friends with almost everyone else— connected by a few links between cliques — friendships between people with different hobbies, backgrounds, or languages. At each time-step, an individual can choose to share content they have seen with some probability; this probability can be different depending how interesting the original post is and on whether the person they saw it from is in the same clique or in a different clique. (In the real world, most content is more likely to be shared within a single

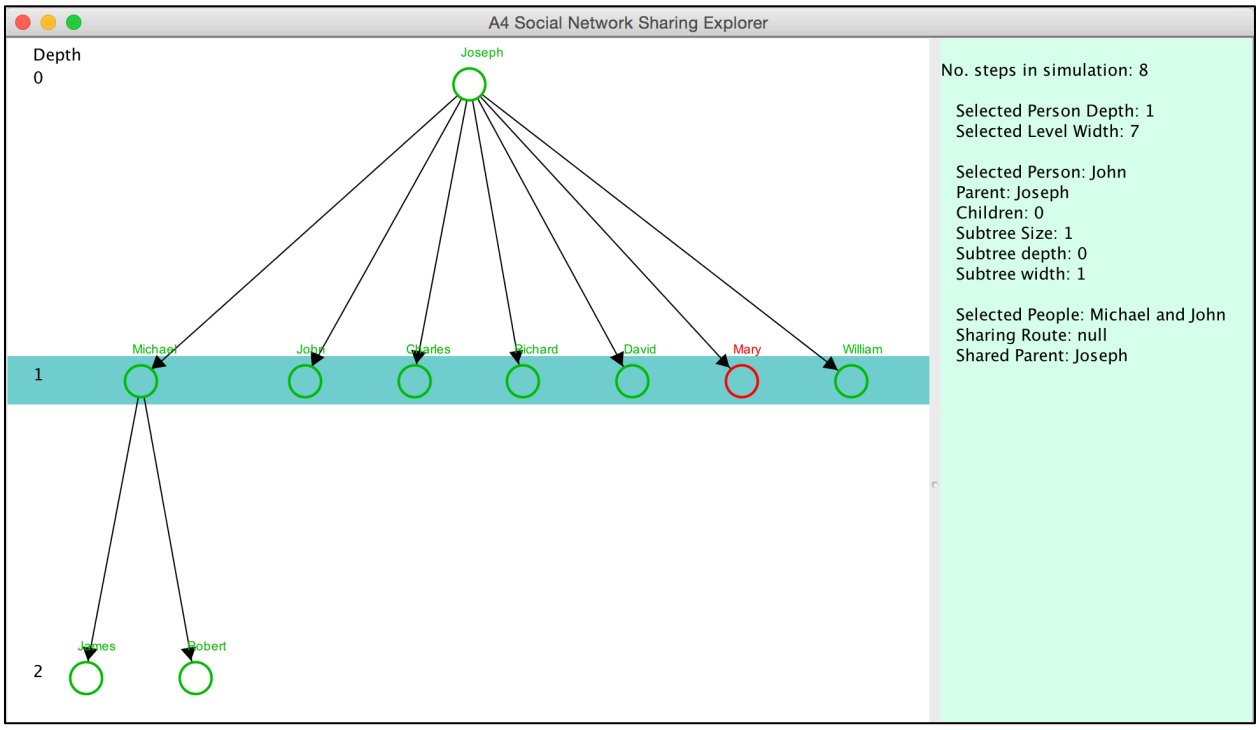
clique, but not across cliques. Viral content, however, is universally appealing and is equally likely to be shared within a clique or across cliques.) This model provides a real world example of the general tree data structure.

You are not responsible for writing any of the code that implements the simulation of how content spreads through a social network.

The model is initialized with a population of people —1, 2, 10, 50, however many you choose— divided into some number of cliques—1, 2, 5, however many you choose. At the beginning of the simulation, a graph is constructed with people as nodes and with random edges between people, indicating they are friends in the social network. You supply a probability indicating that two people in the same clique are friends and a probability that two people in different cliques are friends. A probability of 1 means the two people definitely are friends, a probability of 0 means they definitely are not friends.

When you start the program, you supply three other numbers: the interest score of the content, in 0..10; the probability that a person shares content they saw from a friend in the same clique; and the probability that a person shares content they saw from a friend in a different clique. Once you have given this input, the program starts by making one random person post the content and making that person the root of a new sharing tree. Initially, that is the only node in the tree.

A series of time-steps follows. In each time-step, a couple different things happen. (1) Each person who has seen the content decides whether to share it with their friends. (2) The interest score of the content —to people who have already seen it— goes down by one. These time steps continue until everyone in the network has either seen the content or has lost interest (the local interest score has gone down to zero). At that time, the results are shown in a GUI on your monitor.

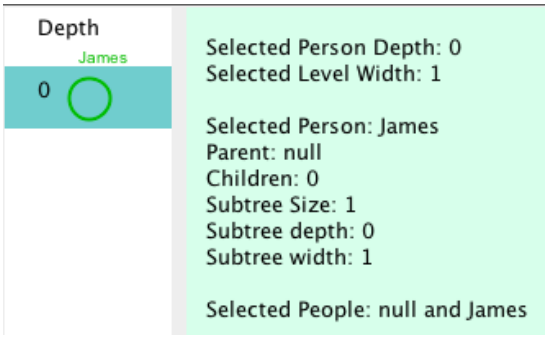


Above is an example of the output in the GUI. At depth 0 (the root) is Joseph; at depth 1 are Michael, John, Charles, Richard, David, Mary, and William. At depth 2 are James and Robert.

Joseph generated the original post; the people at level 1 saw Joseph's post; the green nodes re-shared the post, and James and Robert saw the re-shared post from Michael. Most people were interested enough to re-share the post; only Mary got bored before deciding to re-share it.

To the right in the image above is data that comes from selecting (with your mouse) Michael and then John. It shows: John's depth, 1; the width at (number of nodes at) that level, 7; John's parent, number of children, subtree depth, and subtree width; and the shared ancestor of Michael and John.

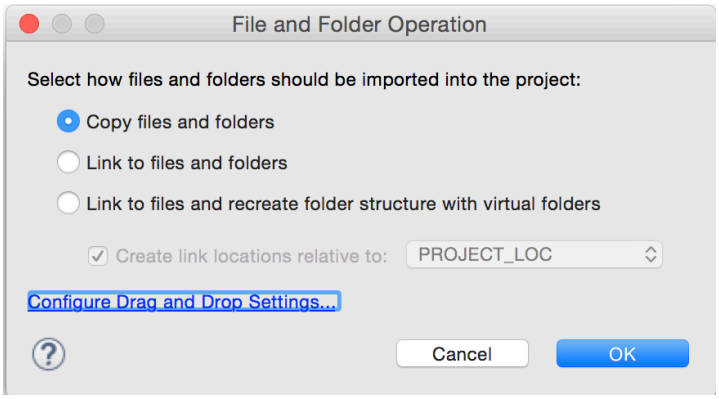
A run may result in a tree with one node, as shown to the right. This happens when the original post isn't seen by anyone because the original poster doesn't have any friends in the social network.



3. Installation

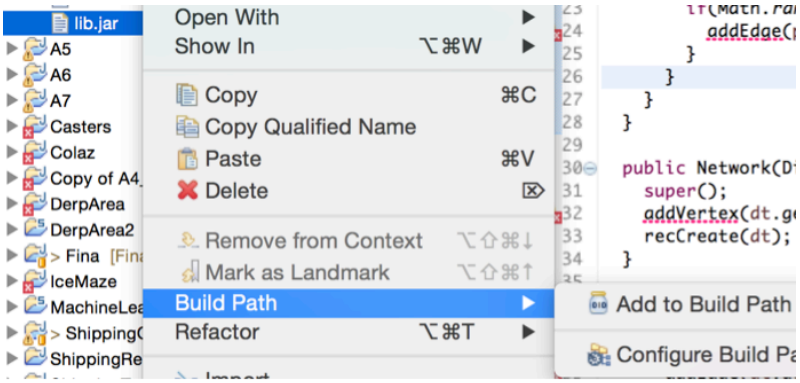
1. Download the A4 assignment zip file from the CMS or the course website.
2. Unzip the downloaded file. The folder should contain two folders, data and src, and a file lib.jar.
3. Create a new Java project (called A4, for example).
4. Copy-paste all three items in the downloaded folder into the root of the new Java project in Eclipse (i.e. data, src, lib.jar). When asked how to copy, select "Copy files and folders", then OK.

When asked what you want to do about the folder already named src there, click "Yes" or "Yes to All"



5. Right-click the project root and select refresh (F5). Many of the files (as seen below) will have errors on them. This is resolved in steps 6..7.

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



7. Add JUnit4.jar to the build path. One way is to create a new JUnit testing class, as usual, and make sure JUnit 4 is add. Instead, you can do this:

a. (1) Select the project, right click, and select Build Path -> Add libraries.

b. (2) In the window that opens, select JUnit and click Next; in the window that opens, select JUnit 4 and click Finish.

c. (3) Refresh your project root (f5) again.

4. Running

The executable class is Post.java. To run the project, open Post.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 people to model. A positive integer. A higher number may result in a larger tree.
2. Number of cliques: the number of cliques in the modeled social network. A higher number may result in a sparser tree.
3. Probability of connection within a clique: how likely two random people in the same clique are to be friends. (On page 2, we call them neighbors). A float in the range [0,1].
4. Probability of connection across cliques: how likely two random people in different cliques are to be friends. A float in the range [0,1]. This should probably be lower than the previous number.
5. Interest of the modeled content: A number in 0..10. The higher the number, the more likely it is to be shared.
6. Sharing probability within a clique: the relative probability that a person shares content they saw from someone else in the same clique. A float in [0,1].
7. Sharing probability across cliques: the relative probability that a person shares content they saw from a friend in a different clique. A float in [0,1]

A nice set of starting values is: (50, 5, 0.8, 0.2, 5, 0.5, 0.1)

If you want to run Post.java repeatedly with the same parameters, you can 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).

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 simulation will start modeling how a post would be shared —starting with a randomly chosen poster and spreading across that person's connections until everyone who has seen the post has either re-shared it or has become bored.

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

5. Your Tasks

The only file you have to edit is SharingTree.java. Before jumping into the methods, be sure to thoroughly read the javadoc description of the class at the top of the file.

In order to complete the assignment, complete each method marked with a //TODO comment to the specification listed above the method. There are 7 such methods. You may assume that all preconditions to these methods are respected. In the case that they are not, any behavior (even non-deterministic) is acceptable. It's best to leave the //TODO comments in —note that they are marked in blue on the right of the text in Eclipse.

Recursion is your friend! The bulk 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: Many of the functions you are asked to write can be written very simply or even trivially (one or two lines) simply by relying on previous functions you have already written or ones that we wrote. The order in which the //TODOs are given (top to bottom) and numbered in SharingTree.java is the order in which to implement them.

Warning: Every time application Post (i.e. method main in Post.java) is called, a new, random, unrepeatable SharingTree 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.

Dos and Donts:

- Do read all the specifications of methods in SharingTree.java very thoroughly. You may choose to read specs in other files, but they should not be too important. Read the files outside package LinkedList only if you are particularly interested in them —you shouldn'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 SharingTree.java must work with unaltered versions of the other files.
- Do not change the method signatures of any method in SharingTree. The name and types of parameters should not be changed.
- Do not have println statements (which you added to help debug) in your code when you submit. Comment them out or delete them before submitting.
- You may add new methods to SharingTree 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 a javadoc comment (/** … */) specification.

6. Debugging

JUnit testing class SharingTreeTest, has a testing procedure for each method you have to write. We urge you to test thoroughly, adding more test cases to each of the methods in order to systematically test the functionality of SharingTree. Use the same testing practices you learned since A1 and used in A2 and A3. You won't submit SharingTreeTest.java, so using it isn't required. But, since Post.java is entirely probabilistic, a JUnit file is the only way to systematically ensure that your SharingTree is correct.

Debugging your tree method can be difficult. It's harder than with linked lists, where we were easily able to test *all* fields using toString(), toStringRev(), and size().

Pinned Piazza note A4 FAQs contains some help to get started on testing and debugging.

7. What to submit

Complete the information at the top of file SharingTree.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.

Submit file SharingTree.java on the CMS.