Data Structures and Algorithms

Assignment v0.9

Semester 2, 2019

Department of Computing
Curtin University

1 Introduction

In practicals you have implemented and learned about a number of algorithms and ADTs and will be implementing more of these in the remaining practicals. In this assignment, you will be making use of this knowledge to implement a system to explore and compare a variety of ADT implementations. Feel free to re-use the generic ADTs from your practicals. However, remember to self-cite; if you submit work that you have already submitted for a previous assessment (in this unit or any other) you have to specifically state this. Do not use the Java implementations of ADTs – if in doubt, ask.

2 The Problem

This assignment requires the extension of your graph code to apply it to social network analysis. You will be simulating the spread of information (or disease) through a social network. In the network, at each timestep, there will be a probability of “liking” a post, which will expose your followers to the information. Liking a post also gives a probability of “following” the original poster - making a direct connection to the original source.

Your program should be called **SocialSim**, and have three starting options:

- No command line arguments : provides usage information
- "-i" : interactive testing environment
- "-s" : simulation mode

When the program starts in interactive mode, it should show the following main menu:

(1) Load network
(2) Set probabilities
(3) Node operations (find, insert, delete)
(4) Edge operations (like/follow - add, remove)
(5) New post
(6) Display network
(7) Display statistics
(8) Update (run a timestep)
(9) Save network

You can structure the menu/UI differently, just make sure all the options are there.

When running in simulation mode, you will give the input file and probabilities on the command line, then output the simulation state and statistics for each timestep to a file (unique filename).

Once you have a working program, you will investigate the behaviour of social networks, and the impact of their structure on their performance. This investigation will be written up as The Report.

Remember: think before you code!

3 Submission

Submit electronically via Blackboard.

You should submit a single file, which should be zipped (.zip) or tarred (.tar.gz). Check that you can decompress it on the lab computers. These are also the computers on which your work will be tested, so make sure that your work runs there. The file must be named DSA_Assignment_<id> where the <id> is replaced by your student id. There should be no spaces in the file name; use underscores as shown.

The file must contain the following:

- **Your code.** This means all .java/.py files needed to run your program. Do include code provided to you as part of the assignment if that is required to run your program.
- **README** file including short descriptions of all files and dependencies, and information on how to run the program.
- Your **unit test harnesses**. One of the easiest ways for us to be sure that your code works is to make sure that you've tested it properly. Our test harnesses may not work for some of your classes and you may have classes that we're not specifically asking for, so make it easy for us to test your work. A test harness for class X should be called UnitTestX.
- **Documentation and Report** for your code, as described in Section 2.1.
- A signed and dated **cover sheet**. These are available from Blackboard with the assignment specification. You can sign a hard copy and scan it in or you can fill in a soft copy and digitally sign it.
- **Java Students:**
    - Do not include .class files or anything else that we do not need. We will recompile .java files to ensure that what we're testing is what we're reading.
    - You may include a Makefile, but this is not required. We will use javac *.java to compile your files and run the unit tests by their expected names.

Make sure that your file contains what is required. Anything not included in your submission will not be marked, even if you attempt to provide it later. It is your responsibility to make sure that your submission is complete and correct.

3.1 Documentation and Report

You need to submit documentation in docx or pdf format.

Your **Documentation** will be minimum 2-4 pages (excluding UML and Javadocs) and should include the following:

- An **overview** of your code, explaining any questions that the marker may have. This is supplemented by the comments in your code. In general, if the marker is looking at your code and isn't sure why you have done something in that particular way, they will check your documentation to see if they can find an explanation. Using an automated documentation system like Javadocs may be very helpful. It is not required, though.
- A **UML Class diagram** of your code
- A description of any **classes** you have, you need to let us know not only what the purpose of that class is but why you chose to create it. As part of this, also identify and justify any places where it was possibly useful to create a new class but you chose not to, especially when it comes to inheritance.
- **Justification** of all major decisions made. In particular, when you choose an ADT, underlying data structure or an algorithm, you need to justify why you chose that one and not one of the alternatives. <u>These decisions are going to be of extreme importance in this assignment.</u>

The **Report** will follow the structure of a standard academic report or paper. It should be at least 4-6 pages long. Required sections are:

- **Abstract**: Explain the purpose of the report and state what you are investigating, and the outcomes/recommendations.
- **Background**: Discuss your approach to developing the simulation code, and the aspects of the simulation that you will be investigating.
- **Methodology**: Describe how you have chosen to profile and compare multiple runs of the simulation, and why. Include the commands, input files, outputs – anything needed to reproduce your results.
- **Results**: Present the results of your simulations, and what you discovered.
- **Conclusion and Future Work:** Give your conclusions and what further investigations could follow.

3.2 Marking

Marks will be awarded to your submission as follows:

- **[10 marks] Documentation**. This is mainly things like choosing and justifying your structure, ADTs, and algorithms. Note that you will only get these marks if you adequately justify your decisions and properly implement them. So, for example, if you choose an algorithm and adequately justify it but aren't actually able to implement it you'll only get part of the marks.
- **[30 marks] Code testing.** We'll have a number of tests to run, and you get marks proportional to how many tests your code passes. If your code passes all of the tests, then you will get all of these 30 marks.
- **[30 marks] Implementation**. We will use unit testing as well as looking at code quality; your test harnesses will have a big impact on these marks. Students are encouraged to use testing frameworks for their harnesses.

- **[30 marks] Report.** As described in section 3.1, with the majority of marks for the methodology and results sections.
- **[Bonus Marks]** There will be bonus marks available exceptional/additional features.
- Marks will be deducted for not following specifications outlined in this document, which includes incorrect submission format and content and using built-in Java ADTs.
- If the cover sheet isn't provided with your submission, your submission will not be marked and you will be awarded zero (0) marks. If you forget to submit the cover sheet you will be allowed to submit it separately to the unit coordinator (by e-mail or in person) but will lose 5 marks.

The aims of this marking breakdown is as follows:

- To reward you for good design decisions, but not unless you're able to implement them. This is the core aim of the unit.
- To let you score some marks if you get the program working but make with poor decisions. If all of your decisions are based only on ease of implementation you can still score the 30 marks from the implementation category and will score most of the marks from testing, meaning you can pass if all of your code works perfectly and is of sufficient quality. It's taking a risk, though.
- To promote good testing. Industry is repeatedly telling us they are really looking for students who can properly test their code.
- To teach you to follow specifications carefully.
- To reward students who have been serious about doing their practical work

3.3 Requirements for passing the unit

Students must submit an assignment worthy of scoring 15% to pass the unit.

This assignment has many correct solutions so plagiarism will be easy for us to detect (and we will). For information about plagiarism, please refer to http://academicintegrity.curtin.edu.au.

In the case of doubt, you may be asked to explain your code and the reason for choices that you have made as part of coding to the unit coordinator. A failure to adequately display knowledge required to have produced the code will most likely result in being formally accused of cheating.

Finally, be sure to secure your code. If someone else gets access to your code for any reason (including because you left it on a lab machine, lost a USB drive containing the code or put it on a public repository) you will be held partially responsible for any plagiarism that results.

3.4 Late Submission

Late submissions will incur a 10% deduction per day.

3.5 Clarifications and Amendments

This assignment specification may be clarified and/or amended at any time. Such clarifications and amendments will be announced via Blackboard. These clarifications and amendments form part of the assignment specification and may include things that affect mark allocations or specific tasks.