COMP3506/7505 Homework 4 – Graph Algorithms

Weighting: 15%

Due date: 23rd Oct 2020, 11:55 pm

Overview

The purpose of this assignment is to gain practice converting worded scenario problems into graph problems, implementing graph data structures, and implementing various graph algorithms to solve such graph problems.

Marks

This assignment is worth 15% of your total grade. Both COMP3506 and COMP7505 students will be marked on questions 1 to 3 out of 30 marks.

Submission Instructions

- Your solutions to Q1, Q2 and Q3 will be submitted via Gradescope to the `Homework 4` submission. You should only submit the following Java files:
  - `ErdosNumbers.java`
  - `FactChecker.java`
  - `ContactTracer.java`
- No other submitted files will be marked.
- ZIP files (or other compressed file formats) won't be marked - you should only submit the relevant Java files to Gradescope.
- No marks will be awarded for non-compiling submissions, or submissions which import non-supported 3rd party libraries. If one of your files isn't compiling, then it will cause you to receive 0 for the entire assignment. If you can't manage to get a class compiling, don't submit that broken file.
- You should follow all constraints laid out in the relevant questions and in the next section.

Late Submissions and Extensions

Late submissions will not be marked. It is your responsibility to ensure you have submitted your work well in advance of the deadline (taking into account the possibility of internet or Gradescope issues). Only the latest submission before the deadline will be marked. See the ECP for information about extensions.

Academic Misconduct

This assignment is an individual assignment. Posting questions or copying answers from the internet is considered cheating, as is sharing your answers with classmates. All your work (including code) will be analysed by sophisticated plagiarism detection software. Students are reminded of the University's policy on student misconduct, including plagiarism. See the course profile and the School web page: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`.

Support Files Provided

The following provided support files have been provided to you:

- `ErdosNumbers.java` - You will implement question 1 in this file and submit it.
- `Fact.java` - A support file for question 2. **Do not modify or submit this file.**
- `FactChecker.java` - You will implement question 2 in this file and submit it.
- `Trace.java` - A support file for question 3. **Do not modify or submit this file.**
- `ContactTracer.java` - You will implement question 3 in this file and submit it.

Notes and Constraints

- You may use code (or implement the pseudocode) provided in lecture slides and tutorials.
- Your solution will be automatically marked using Java 11.
- You may use anything from the Java standard library (unless otherwise stated in the question).
- You may add private helper methods and inner classes.
- You should not create additional files in your solution - these won't be marked.
- Do not modify any of the provided method or class signatures. In particular, do not add a package declaration. Doing so will mean receiving a mark of 0 for your submission as it won't work with the automated tests.
- Do not add **static** variables to any classes.
- Sample tests have been provided. However, these are far from exhaustive and your actual submissions will be marked on a much larger set of test cases. You should write your own tests while working on your assignment.
- You may be penalised for bad code style and formatting, or inefficient algorithm implementations.

Questions

1. (10 marks) The famous Hungarian mathematician Paul Erdős was particularly renowned for his social practice of mathematics, in which he engaged with more than 500 different collaborators on research papers. His collaboration was so prolific in the mathematics community that it prompted the creation of the "Erdős number", which describes the "collaborative distance" between Paul Erdős and another person.

   The calculation of Erdős numbers can be aided through the construction of a graph, where each node or vertex represents a distinct person, and edges between persons indicate a paper co-authorship. The length of the shortest path between a person and Paul Erdős is that person's Erdős number.

   For this question, you will implement the following parts in the class `ErdosNumbers.java`.

   (a) Implement the constructor for `ErdosNumbers`, which takes as input a list of papers for use in the subsequent parts. In addition, implement

   - The method `getPapers`, which returns the set of papers an author has written.
   - The method `getCollaborators`, which returns the set of unique co-authors an author has written a paper with before.

   (b) In `isErdosConnectedToAll`, implement an efficient algorithm to determine if every author has an Erdős number. Consult the Javadoc for more specific details.

   (c) In `calculateErdosNumber`, implement an efficient algorithm to determine the Erdős number of a given author. Consult the Javadoc for more specific details.

   (d) In `averageErdosNumber`, implement the code to determine the average Erdős number of all the co-authors on a given paper.

   (e) Multiple variants of the Erdős number exist that consider more information about co-authorships. One variant, which we will call here the "weighted Erdős number", weights each edge in the graph between two authors a and b as $w(a,b) = \frac{1}{c(a,b)}$, where $c(a,b)$ is the number of papers the two authors have published together.

   Then, the "weighted Erdős number" is the shortest weighted path between a person and Erdős in this graph - and will hence be a real number instead of an integer.

   In `calculateWeightedErdosNumber`, implement an efficient algorithm to calculate the weighted Erdős number of a given author. Consult the Javadoc for more specific details.

   Notes and Hints:

   - A constant has been defined in `ErdosNumbers.java` for you to test against for Erdős's name in the datasets.
   - Authors and papers have unique names (e.g. you can consider authors with the exact same name to be the exact same author).
   - It may make more sense to do some pre-processing in your constructor in order to make other methods more efficient (as they may be called multiple times).

2. (10 marks) Over the mid-semester break, Kristian spent an entire day baking some delicious triple chocolate chip cookies. He then hid them in his cupboard so no one would eat them during a (COVIDSafe^{TM}) party he was hosting later that night for the other tutors. It was a great night with many people (all 1.5 meters apart from each other) coming and going throughout.

   Later that week, he went to his secret stash of delicious cookies that would help him get through a COMP3506 marking filled night. However, to his dismay and shock, they were all gone! Only crumbs were left at the scene of the crime. Immediately, he interrogated the other tutors who attended the party to attempt to narrow down the potential suspects.

   After interviewing them, he collated a bunch of information about the attendees at the party. Each fact he collected was of the following type:

   - **Type I**: Person P_a left the party before person P_b arrived.
   - **Type II**: Person P_a and P_b were both at the party at the same time at some point.

   for persons P_a and P_b at the party.

   Before he investigates further, Kristian first needs to figure out if the facts he's collected are "consistent" - that is, can they all be true at the same time... or is someone lying?

   The number of facts he has collected makes this task too difficult to do manually. However, he luckily knows some COMP3506 students who he thinks will be able to devise an algorithm to determine this task automatically for him!

   For this task, you will need to implement `areFactsConsistent` in `FactChecker.java`, which will take as input a list of such facts, and returns `true` if they are all internally consistent (or otherwise `false`).

   Your algorithm should run in worst case $O(|P| + |F|)$ time, where $|P|$ is the number of people who were at the party, and $|F|$ is the number of facts to process.

   (Hint: Can you make a graph out of the information you are given?).

3. (10 marks) The year is 3506 and a very contagious new virus, named COMP-3506, is spreading rapidly. Queensland Health has finally decided to invest in some new advanced computer programs for contact tracing, as the current ones haven't been upgraded since the last pandemic, COVID-19.

   Currently, they have a database storing a list of person-to-person contact traces of the form of a tuple (P_i, P_j, t_k), where P_i and P_i and P_j are the two people involved in the interaction that happened at time t_k. Note that traced interactions are directionless, as in the trace (A, B, t_k) is the same as the trace (B, A, t_k).

   They have hired you to improve their algorithms for contact tracing by extending this current system into `ContactTracer.java`.

   Note that it would be wise to understand the entire question before attempting to do the individual parts (as your design decisions may change).

   (a) Implement the constructors for `ContractTracer`, which takes as input a list of check-ins of the form described above. Additionally, implement `addTrace` which adds a new contact trace datum to the internal data structure.

   (b) Implement `getContactTimes`, which returns a list of times that two people have been in direct contact in the tracing data. This list should be returned in ascending order of time.

   (c) Implement the `getContacts` and `getContactsAfter` methods, which take as input a person and then return the set of people that person has come into **direct** contact with in the tracing data. For the `getContactsAfter` method, it only returns contacts that have happened at or after the given timestamp.

   (d) Now that we have a data structure which can help us store and query the tracing information, we want to implement an algorithm to help find potential infectees from a given source.

   That is, given a person P who become contagious with COMP-3506 at exactly time t, we want to find all the people who now may have contracted COMP-3506 from person P (and therefore need to be contacted for testing and tracing), or may have contracted it from someone else who P has infected (and so on...).

   For this question, we will assume the following:

   - If a person comes into contact with COMP-3506 at time t and they do not have it already, they might contract the virus and become contagious at exactly $t + 60$ minutes.
   - If a person is *contagious* with COMP-3506 at time t, then they may instantly spread it to anybody they come into contact with at or after time t.

   In `contactTrace`, implement an efficient algorithm that takes a person P, who became contagious with COMP-3506 at time t, and return the set of all other people in the dataset who may now have contracted COMP-3506 under the above assumptions.

   Small example: For example, suppose your contact tracing database has the following information (note: this is not necessarily how you may choose to store this information but just a representation of it).

Person 1	*Person 2*	*Time*
Anna	Sanni	1st October 3PM
Anna	Matt	2nd October 8PM
Matt	Kristian	3rd October 9PM
Kristian	Sanni	3rd October 9:30PM
Kristian	Kenton	3rd October 11PM
Kristian	Max	3rd October 11PM
Kenton	Kristian	4th October 10AM

Table 1: Representation of example database

   If Anna become contagious with COMP-3506 at 1st October 3:30PM, then Matt, Kristian, Kenton, and Max would need to be contacted for tracing due to possible transmission.