COMP4500/7500
Advanced Algorithms and Data Structures

School of Information Technology and Electrical Engineering
The University of Queensland, Semester 2, 2017

Assignment 1

Due at 4:00pm, Monday the 18th September 2017.
This assignment is worth 20% (COMP4500) or 15% (COMP7500) of your final grade.

This assignment is to be attempted **individually**. It aims to test your understanding of graphs and graph algorithms. Please read this entire handout before attempting any of the questions.

Submission. Answers to each of the questions in parts A and B should be clearly labelled and included in a pdf file called `partAB.pdf`.

You need to submit (i) your written answers to parts A and B in `partAB.pdf`, as well as (ii) your source code file `LocationFinder.java` as well as any other source code files that you have written in the `assignment1` package electronically using Blackboard according to the exact instructions on the Blackboard website: `https://learn.uq.edu.au/`

You can submit your assignment multiple times before the assignment deadline but only the last submission will be saved by the system and marked. Only submit the files listed above. You are responsible for ensuring that you have submitted the files that you intended to submit in the way that we have requested them. You will be marked on the files that you submitted and not on those that you intended to submit. Only files that are submitted according to the instructions on Blackboard will be marked.

Submitted work should be neat, legible and simple to understand – you may be penalised for work that is untidy or difficult to read and comprehend.

For the programming part, you will be penalised for submitting files that are not compatible with the assignment requirements. In particular, code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks.

Late submission. Late assignments will lose 10% of the maximum mark for the assignment immediately, and a further 10% of the maximum mark for the assignment for each additional day late. Assignments more than 5 days late will not be accepted.

All requests for extension of assignments must be submitted on the UQ Application for Extension of Progressive Assessment form (`http://www.uq.edu.au/myadvisor/forms/exams/progressive-assessment-extension.pdf`) no later than 48 hours prior to the submission deadline. The application and supporting documentation (e.g. medical certificate) must be submitted to the ITEE Coursework Studies office (78-425) or by email to studentenquiries@itee.uq.edu.au (see Section 5.3 of the course profile for details).

School Policy on Student Misconduct. You are required to read and understand the School Statement on Misconduct, available at the School's website at:

`http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

This is an individual assignment. If you are found guilty of misconduct (plagiarism or collusion) then penalties will be applied.

Part A (35 marks total)

Question 1: Constructing SNI and directed graph [5 marks total]

(a) (1 mark) **Creating your SNI.** In this assignment you are required to use your student number to generate input.

Take your student number and prefix it by "98" and postfix it by "52". This will give you a twelve digit initial input number. Call the digits of that number $d[1], d[2], \ldots, d[12]$ (so that $d[1] = 9, d[2] = 8, \ldots, d[12] = 2$).

Apply the following algorithm to these twelve digits:

```
1  for $i = 2$ to 12
2      if $d[i] == d[i-1]$
3          $d[i] = (d[i] + 3) \mod 10$
```

After applying this algorithm, the resulting value of d forms your 12-digit SNI. Write down your initial number and your resulting SNI.

(b) (4 marks) Construct a graph S with nodes **all** the digits $0, 1, \ldots, 9$. If 2 digits are adjacent in your SNI then connect the left digit to the right digit by a directed edge. For example, if "15" appears in your SNI, then draw a directed edge from 1 to 5. Ignore any duplicate edges. Draw a diagram of the resulting graph. (You may wish to place the nodes so that the diagram is nice, e.g., no or few crossing edges.)

Question 2: Strongly connected components [30 marks total]

Given a directed graph $G = (V, E)$, a subset of vertices U (i.e., $U \subseteq V$) is a *strongly connected component* of G if, for all $u, v \in U$ such that $v \neq u$,

a) u and v are mutually reachable, and

b) there does not exist a set $W \subseteq V$ such that $U \subset W$ and all distinct elements of W are mutually reachable.

For any vertices $v, u \in V$, v and u are mutually reachable if there is both a path from u to v in G and a path from v to u in G.

The problem of finding the strongly connected components of a directed graph can be solved by utilising the depth-first-search algorithm. The following algorithm SCC(G) makes use of the basic depth-first-search algorithm given in lecture notes and the textbook, and the transpose of a graph; recall that the transpose of a graph $G = (V, E)$ is the graph $G^T = (V, E^T)$, where $E^T = \{(u, v) \mid (v, u) \in E\}$ (see Revision Exercises 3: Question 6). (For those who are interested, the text provides a rigorous explanation of why this algorithm works.)

```
SCC($G$)
1  call DFS($G$) to compute finishing times $u.f$ for each vertex $u$
2  compute $G^T$, the transpose of $G$
3  call DFS($G^T$), but in the main loop of DFS, consider the vertices in order of decreasing $u.f$
4  output the vertices of each tree in the depth-first forest of step 3 as a separate
   strongly connected component
```

(a) (15 marks) Perform step 1 of the SCC algorithm using S as input. Do a depth first search of S (from Question 1b), showing colour and immediate parent of each node at each stage of the search as in Fig. 22.4 of the textbook and the week 3 lecture notes. Also show the start and finish times for each vertex.

For this question you should visit vertices in numerical order in all relevant loops:

for each vertex $u \in G.V$ and

for each vertex $v \in G.Adj[u]$.

(b) (3 marks) Perform step 2 of the SCC algorithm and draw S^T.

(c) (12 marks) Perform steps 3, 4 of the SCC algorithm. List the trees in the depth-first forest in the order in which they were constructed. (You do not need to show working.)

Part B (40 marks total)

Question 3: Checking the post

[Be sure to read through to the end before starting.]

You have a job monitoring a postal network containing n postal *locations* $\mathcal{P} = \{P_0, P_1, \ldots, P_{n-1}\}$.

Packages are routed through the postal network by deliveries from one location in the postal network to another. Each *delivery* is represented by a quadruple (P_u, P_v, t_u, t_v) where t_u is the time that the delivery departed the source location $P_u \in \mathcal{P}$, and t_v is the time that the delivery arrived at the destination location $P_v \in \mathcal{P}$. For each such delivery we have that both the departure and arrival times are non-negative integers, and the departure time of the delivery is strictly less than its arrival time at the destination (i.e. $0 \leq t_u < t_v$).

Postal locations may simultaneously send and receive multiple deliveries. Zero or more packages can be moved from one location in the network to the other as part of the same delivery.

A *route* r from one postal location, P_s, to another, P_d, is a non-empty sequence of deliveries

$$\langle (P_{u0}, P_{v0}, t_{u0}, t_{v0}), (P_{u1}, P_{v1}, t_{u1}, t_{v1}), \ldots, (P_{u(x-1)}, P_{v(x-1)}, t_{u(x-1)}, t_{v(x-1)}) \rangle$$

where the number of deliveries in the sequence is x, such that the first delivery $r(0)$ departs P_s (i.e. $P_s = P_{u0}$), the last delivery $r(x-1)$ arrives at P_d (i.e. $P_d = P_{v(x-1)}$), and for each $i \in 0, \ldots, x-2$, we have that delivery $r(i)$ arrives at the same location that the next delivery, $r(i+1)$, departs from (i.e. $P_{vi} = P_{u(i+1)}$) at a time $t_{vi} < t_{u(i+1)}$. The time difference $t_{u(i+1)} - t_{vi}$ is referred to as the delay at postal location P_{vi} between delivery i and $i+1$: we say that a package is at postal location P_{vi} for $t_{u(i+1)} - t_{vi}$ units of time between those two deliveries.

A route may visit the same postal location (including P_s and P_d) more than once, i.e. it can contain cycles between postal locations.

You have been notified that a package departing postal location P_s at some time after t_s (i.e. at some time t such that $t_s < t$), and arriving at a different location P_d at some time before t_d (i.e. at some time t' such that $t' < t_d$), was *tampered with*. The package was *tampered with* at some time after t_s and some time before t_d. The tampering can only have taken place at a postal location, however, for the tampering to have taken place at a postal location, it must have been delayed there for at least d units of time between two deliveries while it was en route from P_s to P_d. Postal locations P_s and P_d are trusted, and so the tampering could not have taken place at those locations.

You need to identify the postal locations at which the tampering may have taken place, but you don't know which route was taken by the package. You do, however, have a log, l, of all of the deliveries that were made that departed after t_s and arrived before t_d. The log l is a sequence of m deliveries, and it is sorted in ascending order of the departure time of the deliveries, i.e., if $l(i) = (P_{ui}, P_{vi}, t_{ui}, t_{vi})$, $l(j) = (P_{uj}, P_{vj}, t_{uj}, t_{vj})$ and $i < j$, then $t_{ui} \leq t_{uj}$. The package may have taken any route from P_s to P_d (departing after t_s and arriving before t_d) that is made up of those deliveries.

Using your logs, you must now write an algorithm that returns the set of postal locations at which the tampering may have taken place. Your algorithm must be as efficient as possible.

Your algorithm should take as input the set of n locations $\mathcal{P}$ on the postal network; the location $P_s \in \mathcal{P}$ that the package was sent from; the location $P_d \in \mathcal{P}$ that the package was sent to; the times t_s and t_d such that $0 \leq t_s < t_d$; the log of m deliveries l that departed after t_s and arrived before t_d sorted in ascending order of the departure time of the deliveries, and the delay d, an integer greater than 0, representing the amount of time between two deliveries that was needed to tamper with the package at a postal location.

For example, given inputs $\mathcal{P} = \{P_0, P_1, P_2, P_3, P_4, P_5\}$, $P_s = P_0$ and $t_s = 2$, $P_d = P_4$ and $t_d = 20$, and log

$$\begin{array}{l}(P_0, P_1, 3, 4),\\(P_5, P_4, 4, 5),\\(P_1, P_2, 5, 7),\\(P_0, P_1, 10, 12),\\(P_0, P_3, 10, 16),\\(P_2, P_1, 10, 11),\\(P_0, P_3, 12, 14),\\(P_1, P_4, 13, 16),\\(P_3, P_4, 16, 19)\end{array}$$

and delay $d = 3$, we have that there are four possible routes from P_s to P_d that the package might have taken:

$$\begin{array}{l}\langle (P_0, P_1, 3, 4), (P_1, P_2, 5, 7), (P_2, P_1, 10, 11), (P_1, P_4, 13, 16) \rangle\\\langle (P_0, P_1, 3, 4), (P_1, P_4, 13, 16) \rangle\\\langle (P_0, P_1, 10, 12), (P_1, P_4, 13, 16) \rangle\\\langle (P_0, P_3, 12, 14), (P_3, P_4, 16, 19) \rangle\end{array}$$

In the first of these routes,

$$\langle (P_0, P_1, 3, 4), (P_1, P_2, 5, 7), (P_2, P_1, 10, 11), (P_1, P_4, 13, 16) \rangle$$

the package was at location P_1 for $5-4 = 1$ unit of time between the first and second deliveries; at location P_2 for $10-7 = 3$ units of time between the second and third deliveries; and back at location P_1 for $13-11 = 2$ units of time between the third and fourth delivery. That means that on that route, the package may only have been tampered with at P_2, since that is the only location where it was delayed for at least $d = 3$ units of time at between two deliveries. In the second route,

$$\langle (P_0, P_1, 3, 4), (P_1, P_4, 13, 16) \rangle$$

the package may have been tampered with at P_1, since it spent $13-4 = 9$ units of time there between deliveries, and $9 \geq d = 3$. In the last two routes, there is no postal location where at least $d = 3$ units of time are spent there between deliveries.

Overall then, the set of postal locations where the tampering may have taken place is:

$$\{P_1, P_2\}$$

and so this is the set that must be returned by your algorithm.

(a) (30 marks) Give an algorithm in pseudocode that answers the question above. You may use the programming constructs used in Revision solutions. Your algorithm should be as efficient as possible. Marks will be deducted for inefficient algorithms. Comment your code.

(b) (10 marks) Provide an asymptotic upper bound on the worst case time complexity of your algorithm in terms of parameters n and m. Make your bound as tight as possible and justify your solution. You must clearly state any assumptions that you make. [Make your analysis as clear and concise as possible – it should be no more than $\frac{3}{4}$ of a page using minimum 11pt font. Longer descriptions will not be marked.]

Hints: It may be inefficient to solve the problem by enumerating all of the possible postal routes from P_s to P_d. The problem *can* be solved in polynomial time! Algorithms similar to ones you are already familiar with from class could be used to determine if there was a route from one postal location to another within certain time bounds, or to find the earliest time that the package could reach a postal location etc.

Part C (25 marks total)

Question 4: Implement a solution for Question 3(a) (25 marks)

Implement your algorithm from Question 3(a) as efficiently as you are able to in the static method `LocationFinder.findLocations` from the `LocationFinder` class in the `assignment1` package that is available in the zip file that accompanies this handout.

The zip file for the assignment also includes some other code that you will need to compile the class `LocationFinder` as well as some junit4 test classes to help you get started with testing your code.

Do not modify any of the files in package `assignment1` other than `LocationFinder`, since we will test your code using our original versions of these other files. You may not change the class name of the `LocationFinder` class or the package to which it belongs. You may not change the signature of the `LocationFinder.findLocations` method in any way or alter its specification. (That means that you cannot change the method name, parameter types, return types or exceptions thrown by the method.)

You are encouraged to use Java 8 SE API classes, but no third party libraries should be used. (It is not necessary, and makes marking hard.) Dont write any code that is operating-system specific (e.g. by hard-coding in newline characters etc.), since we will batch test your code on a Unix machine. Your source file should be written using ASCII characters only.

You may write additional classes, but these must belong to the package `assignment1` and you must submit them as part of your solution – see the submission instructions for details.

The JUnit4 test classes as provided in the package `assignment1.test` are not intended to be an exhaustive test for your code. Part of your task will be to expand on these tests to ensure that your code behaves as required.

Your implementation will be evaluated for correctness and efficiency by executing our own set of junit test cases. Code that is submitted with compilation errors, or is not compatible with the supplied testing framework will receive 0 marks. A Java 8 compiler will be used to compile and test the code.