COMP4500/7500
Advanced Algorithms and Data Structures

School of Information Technology and Electrical Engineering
The University of Queensland, Semester 2, 2016

Assignment 1

Due at 4:00pm, Friday 23 September 2016.
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 `DelegateFinder.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 7 days late will not be accepted without an extension having been granted.

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 enquiries@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: summit outcome analysis

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

A set of n delegates $\mathcal{D}$ will meet at a world summit to determine the outcome of an important decision. At the start of the summit, a subset of the delegates $\mathcal{I}$ are in favour of the decision, and the remaining delegates are against it.

There are a number of events that might take place at the summit. At each event, it is possible for the delegates to change their mind. The effect that an event e has upon each delegate's opinion is defined by a total mapping $e : \mathcal{D} \to \mathbb{P}\mathcal{D}$ from each delegate d in $\mathcal{D}$ to a (possibly empty) set $e(d)$ of delegates from $\mathcal{D}$. The idea is that a delegate d will be in favour of the decision directly after the event e if and only if at least one of the delegates in $e(d)$ was in favour of the decision directly before the event. For example, for $n = 4$ delegates $\mathcal{D} = \{d_0, d_1, d_2, d_3\}$ and event

$$e = \{d_0 \mapsto \{\}, d_1 \mapsto \{d_0, d_2\}, d_2 \mapsto \{d_2\}, d_3 \mapsto \{d_0, d_2, d_3\}\},$$

if the set of delegates $I_{init} = \{d_0, d_3\}$ were in favour directly before event e, then the set $I_{final} = \{d_1, d_3\}$ will be in favour directly afterwards.

The proceedings of the summit will be made up of a finite sequence of events, where each event takes place one at a time, in sequence order. The delegates who will be in favour of the decision at the end of the summit can be calculated by performing those events, one at a time in sequence order, starting from $\mathcal{I}$, the set of delegates who are initially in favour of the decision.

Before the start of the summit, the proceedings has not yet been finalised. However, the possible sequences of events that could occur at the summit are described by a directed graph $\mathcal{G}$ with a start vertex, v_{start}, and a finishing vertex v_{end}, such that each vertex in the graph is reachable from the start vertex, and can reach the end vertex. Each edge of the graph is labelled with an event (as described above) that takes place when making the transition from the source to the destination vertex. Each possible proceeding of the summit (a sequence of events) is described by a finite path through the graph starting at the start vertex and finishing at the end vertex. The graph may contain cycles, and paths need not be simple – that is, a path may visit the same vertex more than once. However you may assume that the graph does not contain self-loops (i.e. there cannot be an edge from a vertex v back to itself) or multiple edges (i.e. for any vertices u and v there can be at most one edge from u to v).

For example, for $n = 4$ delegates $\mathcal{D} = \{d_0, d_1, d_2, d_3\}$ and proceedings graph $\mathcal{G}$ with $v = 3$ vertices $V = \{v_0, v_1, v_2\}$, start vertex v_0, end vertex v_2, and $e = 3$ edges

$$E = \{(v_0, v_1, e_0), (v_1, v_2, e_1), (v_2, v_1, e_2)\}$$

where

$$\begin{aligned} e_0 &= \{d_0 \mapsto \{d_0\}, d_1 \mapsto \{d_1\}, d_2 \mapsto \{d_2\}, d_3 \mapsto \{d_3\}\}, \\ e_1 &= \{d_0 \mapsto \{\}, d_1 \mapsto \{d_0\}, d_2 \mapsto \{d_1, d_2\}, d_3 \mapsto \{d_3\}\}, \\ e_2 &= \{d_0 \mapsto \{d_0\}, d_1 \mapsto \{d_1\}, d_2 \mapsto \{d_2\}, d_3 \mapsto \{d_3\}\} \end{aligned}$$

we have that the paths v_0, v_1, v_2 and v_0, v_1, v_2, v_1, v_2 define two possible proceedings of the summit. In the first path, event e_0 occurs first, followed by e_1; in the second path event e_0 occurs first, followed by e_1, then e_2 followed by e_1 again. In fact, there are an infinite number of possible proceedings of the summit described by this graph: each one executes e_0 and then e_1, followed by zero or more executions of "e_2 followed by e_1".

If the set of delegates who are originally in favour of the decision is $\mathcal{I} = \{d_0\}$, then the delegates who will be in favour of the decision after the summit proceedings described by the first path, v_0, v_1, v_2, will be $\{d_1\}$. Those in favour of the decision after the proceedings described by the second path, v_0, v_1, v_2, v_1, v_2, will be $\{d_2\}$. There are no possible summit proceedings (from $\mathcal{I}$) such that either delegate d_0 or d_3 will be in favour of the decision at the end of the summit.

We say that a delegate may be in favour of the decision at the end of the summit if there is at least one summit proceedings (starting from $\mathcal{I}$) such that d is in favour of the decision at the end of that proceedings.

In our running example, the delegates who may be in favour of the decision at the end of the summit are d_1 and d_2.

For a given summit, our task is to find all the of delegates who may be in favour of the decision at the end of the summit:

For a given set of delegates $\mathcal{D}$, proceedings graph $\mathcal{G}$ with start vertex v_{start} and end vertex v_{end} describing the possible proceedings of the summit, and an initial set of delegates $\mathcal{I}$ who are in favour of the decision at the start of the summit, our task is to determine the set of delegates $\mathcal{X}$ who may be in favour of the decision at the end of the summit. (That is, $\mathcal{X}$ should contain a delegate d if and only if d may be in favour of the decision at the end of the summit).

(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 the number of delegates n, the number of vertices v in the graph and the number of edges e in the graph. 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.]

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 `DelegateFinder.findDelegates` from the `DelegateFinder` 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 `DelegateFinder` 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 `DelegateFinder`, or any of the files in package `graphs`, since we will test your code using our original versions of these other files. You may not change the class name of the `DelegateFinder` class or the package to which it belongs. You may not change the signature of the `DelegateFinder.findDelegates` 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.