COMP4500/7500
Advanced Algorithms and Data Structures

School of Information Technology and Electrical Engineering
The University of Queensland, Semester 2, 2019

Assignment 1

Due at 4:00pm, Monday the 16th September 2019.
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 part A and Question 4(a) and 4(b) from part B should be clearly labelled and included in a pdf file called `a1.pdf`.

You need to submit (i) your written answers to parts A and Question 4(a) and 4(b) from part B in `a1.pdf`, as well as (ii) your source code file `TransferFinder.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. Unless you have been approved to submit an assignment after the due date: late assignments will lose 10% of the marks allocated to the assignment immediately, and a further 10% of the marks allocated to the assignment for each additional calendar day late. Assignments more than 5 calendar days late will not be accepted.

If there are medical or exceptional circumstances that will affect your ability to complete an assignment by the due date, then you can apply for an extension as per Section 5.3 of the electronic course profile (ECP). Requests must be made at least 48 hours prior to the submission deadline. Assignment extensions longer than 7 calendar days will not be granted.

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 (65 marks total): Warehouse shuffle

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

You are in charge of storing *items* of different possible *types* in a set of *warehouses*.

Each warehouse has a capacity, denoting the maximum number of items that can be stored in the warehouse, and a defined set of types of items that are able to be stored in the warehouse. The capacity of a warehouse, and the set of types of items that can be stored in a warehouse cannot change. At any point in time, a warehouse keeps track of the number of each different type of item it has stored. The total number of items stored in a warehouse at any point in time cannot exceed the capacity of the warehouse, and the warehouse can only store items of types that can be stored in that warehouse. If the total number of items currently stored in the warehouse equals the capacity of the warehouse, then we say it is *full*.

A *transfer* of one item of stock of a given type x can be made from a source warehouse s to a destination warehouse d, if warehouse s currently contains at least one item of type x, and warehouse d is not currently full, and it is able to store items of type x.

From time to time a new item of stock is purchased and needs to transferred from a *depot* to one of the warehouses that you are in charge of. (For simplicity, a depot is a warehouse that is not in the set of warehouses that you are in charge of.) If you are in charge of a warehouse that is not full that can store the type of the new item, then this can be achieved easily by performing a single transfer of the new item of stock from the depot, to the warehouse. On the other hand, if all the warehouses that are able to store items of the given type are currently full, then it may be necessary to first transfer items between the existing warehouses, in order to make room for the new item, before transferring the new item from the depot to an available warehouse. In some circumstances, there may be no way to rearrange the items in the existing warehouses (by making transfers between them) in order to make room for the new item.

Example 1 For example, imagine that you are in charge of the following set of warehouses:

$$
\begin{array}{l}
W1 : 2 : (T8, 0) \\
W2 : 2 : (T3, 0), (T6, 1), (T7, 1) \\
W3 : 1 : (T1, 0), (T2, 1), (T9, 0) \\
W4 : 2 : (T4, 1), (T6, 1) \\
W5 : 3 : (T1, 1), (T3, 2), (T6, 0), (T8, 0) \\
W6 : 3 : (T4, 0), (T8, 1) \\
W7 : 4 : (T8, 4) \\
W8 : 3 : (T2, 0), (T7, 0), (T8, 0), (T9, 3)
\end{array}
$$

where each warehouse above is described first by its name, then its capacity, and thirdly, a list of the types of items that can be stored in the warehouse and the number of items of that type that are currently stored in the warehouse. E.g. warehouse $W6$ has a capacity of 3 and can store items of type $T4$ or $T8$. It currently stores zero items of type $T4$ and one item of type $T8$.

If a new item of stock of type $T8$ was purchased and needed to be transferred from the depot

$$W0 : 1 : (T8, 1)$$

to one of the warehouses that you are in charge of, then this could be accomplished (easily) by performing the following sequence of transfer operations:

$$\langle (W0, W1, T8) \rangle$$

where each transfer in the list above is being represented by a tuple of the source warehouse, followed by the destination warehouse, and then the type of the item that is transferred. Other sequences of transfers operations would also work, e.g. $\langle (W0, W6, T8) \rangle$.

Example 2 If you are still in charge of the same set of warehouses (in the same initial state) from Example 1, and a new item of stock of type $T1$ was purchased and needed to be transferred from the depot

$$W0 : 1 : (T1, 1)$$

to one of the warehouses that you are in charge of, then this cannot be achieved by one transfer operation, because all the warehouses that can store items of type $T1$ are currently full. The item of stock can, however, be transferred from the depot to one of the warehouses by performing the following sequence of transfer operations (in the order they are given):

$$\langle (W4, W6, T4), (W2, W4, T6), (W5, W2, T3), (W0, W5, T1) \rangle$$

That is, an item of type $T4$ can first be transferred from $W4$ to $W6$; and then an item of type $T6$ can be transferred from $W2$ to $W4$; then an item of type $T3$ can be transferred from $W5$ to $W2$; and then finally the new item of type $T1$ can be transferred from the depot, $W0$, to warehouse $W5$. There are other sequences of valid transfer operations that would also work.

Example 3 If you are now in charge of the following set of warehouses (note that these differ from those in the above examples only in the current contents of $W6$);

$$\begin{array}{l} W1 : 2 : (T8, 0) \\ W2 : 2 : (T3, 0), (T6, 1), (T7, 1) \\ W3 : 1 : (T1, 0), (T2, 1), (T9, 0) \\ W4 : 2 : (T4, 1), (T6, 1) \\ W5 : 3 : (T1, 1), (T3, 2), (T6, 0), (T8, 0) \\ W6 : 3 : (T4, 3), (T8, 0) \\ W7 : 4 : (T8, 4) \\ W8 : 3 : (T2, 0), (T7, 0), (T8, 0), (T9, 3) \end{array}$$

a new item of stock of type $T1$ was purchased and needed to be transferred from the depot

$$W0 : 1 : (T1, 1)$$

to one of the warehouses that you are in charge of, then this cannot be done. Not only are all of the warehouses that can store items of type $T1$ currently full, but there is also no way to rearrange the items in the existing warehouses (by making transfers between them) in order to make room for the new item.

You task is to design, implement and analyze an algorithm that takes as input:

- A depot that is non-null and contains exactly one item of a particular type. The depot may not contain any items of any other type and must have a capacity of one,
- A set of warehouses that is non-null, does not contain null warehouses, and does not include the depot,

and, *without modifying the state of the depot or any of the warehouses in any way (e.g. items should not be added or removed from the depot or any of these warehouses)*, it returns as output either:

(a) A non-null list of transfers such that

  (i) each transfer is non-null and takes place either between two warehouses in the given set of warehouses, or between the given depot and a warehouse in the set of warehouses;

  (ii) given the initial state of the depot and warehouses, all the transfers can be performed in the order in which they appear in the list; and

  (iii) the depot is empty when all transfers in the list are completed (in order, starting from the initial state of the depot and warehouses).

  OR

(b) the value null if and only if no such list of transfers, as described in part (a), exists.

You algorithm must be designed and implemented as efficiently as possible.

Question 3: Design and implement a solution (50 marks)

Design and implement an algorithm that answers the question above. Your algorithm should be as efficient as possible. Marks will be deducted for inefficient algorithms (e.g. a brute-force approach is not appropriate). Clearly structure and comment your code. Use meaningful variable names.

- Your algorithm should be implemented in the static method `TransferFinder.findTransfers` from the `TransferFinder` 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 `TransferFinder` 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 `TransferFinder`, since we will test your code using our original versions of these other files.

- You may not change the class name of the `TransferFinder` class or the package to which it belongs. You may not change the signature of the `TransferFinder.transferFinder` 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.

Question 4: Worst-case time complexity analysis (15 marks)

This question involves performing an analysis of the worst-case time complexity of your algorithm from Question 3.

(a) (5 marks) For the purpose of the *worst-case time complexity* analysis in Q4(b), provide clear and concise pseudocode that summarizes the algorithm you used in your implementation from Question 3. You may use the programming constructs used in Revision solutions, and assume the existence of common abstract data types like sets, maps, queues, lists, graphs, as well as basic routines like sorting etc.

Clearly structure and comment your pseudocode.

[It should be no more than one page using minimum 11pt font. Longer descriptions will not be marked.]

(b) (10 marks)

Let w be the number of warehouses that you are in charge of; t be the number of different possible types of items that can be stored in the warehouses that you are in charge of (e.g. in Example 1 the types of items that can be stored in the warehouses that you are in charge of are $\{T1, T2, T3, T4, T6, T7, T8, T9\}$, and so there are $t = 8$ of them); n be the total number of items currently stored in the warehouses that you are in charge of (e.g. in Example 1 there are $n = 16$ items).

Provide an asymptotic upper bound on the worst case time complexity of your algorithm in terms of parameters w and t and n. Make your bound as tight as possible and justify your solution using your pseudocode from Q4(a).

You must clearly state any assumptions that you make (e.g. on the choice of implementations of any data structures that you use, and their running time etc.).

To simplify your analysis, you should make the (incorrect but simplifying) assumption that $HashSet$ (or $HashMap$) operations that have expected-case time complexity $O(1)$ actually have worst-case time complexity $O(1)$. E.g. checking for set-membership in a $HashSet$ has expected-case time complexity that is $O(1)$.

[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.]

Evaluation Criteria

Question 1

- **Question 1 (a) (1 mark)**

  1 : correct answer to question given

  0 : answer not given or contains one or more mistakes

- **Question 1 (b) (4 marks)**

  4 : The answer to question 1(a) is 100% correct, and the correct graph is given.

  0 : If the answer to question 1(a) is not 100% correct, or the answer contains at least one mistake.

Question 2

If the graph produced for Question 1 is given and 100% correct, then the following marking scheme applies. If the graph produced for Question 1 is mostly correct (but contains a minor error), then the student will receive 2/3 of the marks obtained using the following marking criteria. Else, if the graph produced for Question 1 contains more than a minor error, zero marks will be given for all aspects of this question.

- **Question 2 (a) (15 marks)**

  15 : Correct answer given.

  12 : All stages of the traversal shown, including relevant features for each vertex (colour, immediate parent and start and finish times), but there are one or two minor mistakes.

  9 : All stages of the traversal are shown, however at most one of the relevant features for each vertex (e.g. start-time) may not be included and there may be up to three mistakes.

  6 : Most stages of the traversal are shown, however at most two of the relevant features for each vertex (e.g. start-time) may not be included and there may be up to four mistakes.

  3 : Most stages of the traversal are shown, however at most two of the relevant features for each vertex (e.g. start-time) may not be included and there may be up to five mistakes.

  0 : Otherwise.

- **Question 2 (b) (3 marks)**

  3 : correct answer to question given

  0 : answer not given or contains one or more mistakes

- **Question 2 (c) (12 marks)**

  This part of the question will be marked correct with respect to the finishing times for each vertex calculated in Q2(a). If those finishing times are not given in Q2(a) then no marks will be awarded for this section. Otherwise, the following marking criteria applies.

  12 : All the trees in the depth-first forest are listed in the order in which they were constructed. All aspects of the depth-first forest are correct.

  9 : All the trees in the depth-first forest are listed in the order in which they were constructed, but there was an error in the traversal that produced the forest.

6 : All of the trees in the depth-first forest are listed, however the order in which the trees were created may not be clear, or there may be one or two errors in the traversal that produced the forest.

3 : Either: (i) all of the strongly connected components of the graph are correctly listed, but the trees (from the depth-first forest) from which these connected components were derived are not clearly listed, or (ii) all of the trees in the depth-first forest are listed, however the order in which the trees were created may not be clear and there may be up to three errors in the traversal that produced the forest.

0 : Otherwise.

Question 3 (50 marks)

Your implementation will be evaluated for correctness and efficiency by executing our own set of junit test cases.

50 : All of our tests pass

45 : at least 90% of our tests pass

40 : at least 80% of our tests pass

35 : at least 70% of our tests pass

30 : at least 60% of our tests pass

25 : at least 50% of our tests pass

20 : at least 40% of our tests pass

15 : at least 30% of our tests pass

10 : at least 20% of our tests pass

5 : at least 10% of our tests pass

0 : less than 10% of our test pass or work with little or no academic merit

Note: 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.

Question 4

For any marks to be received for this section, a plausible solution to Question 3 must have been presented in Q3. If a plausible solution to Q3 has been attempted, the following marking criteria applies.

- **Question 4(a) (5 marks)**

  For this question, the pseudocode given must be no longer than one page using minimum 11pt font. Longer solutions will receive 0 marks, otherwise the following marking criteria applies.

  5 : Clear and concise pseudocode that summarizes the algorithm you used in your implementation from Question 3. Clearly commented, and clear correspondence between the implementation from Q3 and the pseudocode.

3 : Mostly clear and concise pseudocode that summarizes the algorithm you used in your implementation from Question 3. Mostly clearly commented, and mostly clear correspondence between the implementation from Q3 and the pseudocode.

2 : Pseudocode given that summarizes the algorithm you used in your implementation from Question 3. Potentially more verbose than necessary. May not be commented, or have a very clear correspondence between the implementation from Q3 and the pseudocode.

1 : Pseudocode given that attempts to summarize the algorithm you used in your implementation from Question 3. Aspects are unclear, or overly verbose, or the correspondence to the actual implementation may be confusing.

0 : Work with little or no academic merit

- **Question 4(b) (10 marks)**

  For this part of the question, the analysis should be no more than 3/4 of a page using minimum 11pt font. Longer solutions will receive 0 marks. Also, Q4(a) must have been answered and received a mark of at least 2. Otherwise the following marking criteria applies.

  10 : A correct asymptotic upper bound on the worst-case time complexity of the algorithm from Q3 is given in terms of parameters w, t and n. The asymptotic upper bound should be as tight as reasonably possible for the algorithm at hand. The worst-case time complexity analysis is clearly justified with respect to the pseudocode from Q4(a). Any assumptions made in the analysis are reasonable and clearly stated. Asymptotic notation should be used correctly and the asymptotic time complexity given has been simplified to remove lower order terms and unnecessary constant factors.

  7 : A correct asymptotic upper bound on the worst-case time complexity of the algorithm from Q3 is given in terms of parameters w, t and n. The asymptotic upper bound should be reasonably tight for the algorithm at hand. The worst-case time complexity analysis is mostly clearly justified with respect to the pseudocode from Q4(a). Any assumptions made in the analysis are mostly reasonable and clearly stated.

  5 : A reasonable attempt has been made to give a reasonably-tight asymptotic upper bound on the worst-case time complexity of the algorithm from Q3 in terms of parameters w, t and n, however either it contains some minor mistakes but is otherwise reasonably justified, or the justification of the analysis with respect to the pseudocode from Q4(a) is unclear or lacking. Assumptions made in the analysis may be unclear.

  3 : An attempt has been made to give an asymptotic upper bound on the worst-case time complexity of the algorithm from Q3 in terms of parameters w, t and n, however it contains either a major mistake, or many mistakes, or gives an unreasonably loose upper bound. The justification for the analysis with respect to the pseudocode from Q4(a) may be unclear or lacking.

  0 : Work with little or no academic merit.