Module Code
COM00016C

BEng, BSc, MEng and MMath Degree Examinations 2020-2021

Department Computer Science
Title Software 2 - Practical
Issued 09:00am on Monday 17th May 2021
Submission due 09:00am on Tuesday 18th May 2021
Feedback & marks due Tuesday 15th June 2021
Time Allowed 24 Hours (NOTE: papers late by up to 30 minutes will be subject to a 5 mark penalty; papers later than 30 minutes will receive 0 marks).

Notification of errors in the paper may be made up to **one hour** after the start time. If you wish to raise a possible error it must be done through ⟨`cs-exams@york.ac.uk`⟩ with enough time for a response to be considered and made within the first hour.

Time Recommended THREE hours

Word Limit Not Applicable

Allocation of Marks:
Question 1 is worth 50 marks, question 2 is worth 40 marks. The remaining 10 marks are allocated to style, clarity, and quality of code. The code must adhere to the Google Java Style Guide.

Instructions:
Candidates should answer **all** questions using Java 8 or above. Failing to do so will result in a mark of 0%. All questions are independent and can be answered in any order. Download the paper and the required source files from the VLE, in the "Assessment>Summer Exam" section. Once downloaded, unzip the file. You **must** save all your code in the `ClosedExamination` folder provided. **Do not** save your code anywhere else other than this folder.

Submit your answers to the Department's Teaching Portal as a single **.zip** file containing the `ClosedExamination` folder and its sub-directories. Failing to include the `ClosedExamination` folder will result in a loss of 10 marks. Other archival format such as .rar and .tar files will not be accepted and will result in a mark of 0%.

If a question is unclear, answer the question as best you can, and note the assumptions you have made to allow you to proceed. Please inform ⟨`cs-exams@york.ac.uk`⟩ about any suspected errors on the paper immediately after you submit.

A Note on Academic Integrity

We are treating this online examination as a time-limited open assessment, and you are therefore permitted to refer to written and online materials to aid you in your answers.

However, you must ensure that the work you submit is entirely your own, and for the whole time the assessment is live you must not:

- communicate with departmental staff on the topic of the assessment
- communicate with other students on the topic of this assessment
- seek assistance with the assignment from the academic and/or disability support services, such as the Writing and Language Skills Centre, Maths Skills Centre and/or Disability Services. (The only exception to this will be for those students who have been recommended an exam support worker in a Student Support Plan. If this applies to you, you are advised to contact Disability Services as soon as possible to discuss the necessary arrangements.)
- seek advice or contribution from any third party, including proofreaders, online fora, friends, or family members.

We expect, and trust, that all our students will seek to maintain the integrity of the assessment, and of their award, through ensuring that these instructions are strictly followed. Failure to adhere to these requirements will be considered a breach of the Academic Misconduct regulations, where the offences of plagiarism, breach/cheating, collusion and commissioning are relevant: see AM1.2.1 *(Note this supercedes Section 7.3 of the Guide to Assessment)*.

1 (50 marks) Object Oriented Programming

The code must be written in the provided file `SocialNetwork.java`. Failing to do so may result in a mark of 0%.
It is often asserted that social network analysis (SNA) can help to identify key players in a network who should be targeted in order to disrupt organisational terrorist activities. It is thought that measures of centrality might be useful as an investigative tool.The aim of this question is to compute such measures.
We have implemented and provided the class `User` which represents a user in a given network. You **must not modify** this class as it is complete, however you are advised to consult the class documentation before attempting the question.

Implement the class `SocialNetwork` representing connection between users. An instance of the class `SocialNetwork` has the following attributes (with package access modifier), public constructor, and public methods :

- a `String name` containing the name of the Network,
- an attribute `Map<String, User> users` containing all users from that network mapped by their `id`.

Note that the class `SocialNetwork` forms an implicit graph via users' relationships.

(i) [4 marks] implement the public constructor `SocialNetwork(String)` which takes the name of the Network as parameter and initialises `users` to an empty map.

(ii) [10 marks] implement the public method `User createUser(String, String)` which creates and returns a new `User` instance and adds it to the list of users in the social network. The first parameter is the user's `id` and the second the user's name. The method must throw an `IllegalArgumentException` if a user with the same `id` already exists in this network.

(iii) [4 marks] implement the public method `User getUser(String)` which takes a user's `id` as parameter and returns the `User` instance with this `id`. The method should throw an `IllegalArgumentException` if no such user exists.

(iv) [7 marks] implement the public method
`boolean addRelationship(String userOneID, String userTwoID)`
which adds the user with `userOneID` to `userTwoID` user's connection and vice versa. The method should return `True` if the connection was successful, `False` otherwise. The method must throw `IllegalArgumentException` if one or both of the users do not exist.

Continued.

(v) [15 marks] There are several centrality measures. The normalised closeness centrality (or closeness) of a node is the average length of the shortest path between the node and all other nodes in the graph. Thus the more central a node is, the closer it is to all other nodes. Closeness was defined by Bavelas in 1950, and its normalised form is given by Equation 1.

$$C(x) = \frac{N-1}{\sum_y d(y,x)} \tag{1}$$

where $C(x)$ is the closeness of vertex x, $d(y,x)$ is the shortest distance between vertices x and y, and N is the number of vertices in the graph. It is assumed that the graph is strongly connected, that is every vertex is reachable from every other vertex.

Implement the public method `double closeness(String userId)` which computes and returns the closeness measure of the user with ID `userID`. The method should throw an `IllegalArgumentException` if such a user does not exist.

(vi) [10 marks] Write the Java Doc for the entire class `SocialNetwork` following the Google Java Style Guide.

2 (40 marks) Recursion & Dynamic Programming

The code for this question must be written in the provided file `LevelChecker.java`.

Warning: if an implemented method in this question is not recursive where requested, a mark of 0 will be awarded for that part of the question.

The aim of the exercise is to check if a game level is feasible or not. The game is a simple 2D platform composed of springboards with power ups and deadly traps (represented by mines). The power ups' number below the springboard represents the player's maximum jump length from that board. The aim of the game is to depart from the first springboard and arrive to the last one without stepping on a mine. Figure 1 shows a game level that is feasible, whereas Figure 2 shows a game level that a player cannot win.

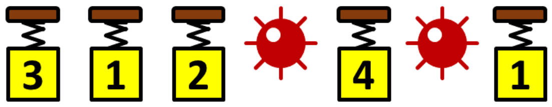

Figure 1: Example of a game level that is feasible. One solution is to jump 2 steps from index 0 to 2, then 2 steps from index 2 to 4 and finally another 2 steps to the last index. An alternative solution is to jump 1 step from index 0 to 1, then another step from 1 to 2 followed by 2 steps from index 2 to 4 and finally another 2 steps to the last index.

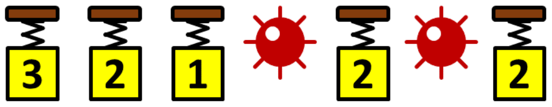

Figure 2: Example of a game level that is impossible to complete. Whatever the choice you make, you will always arrive at index 3 which is a deadly trap. It is therefore impossible to reach the last index.

To represent the level, we have chosen an array of non-negative integers. The starting point of the level is at the first index of the array, and the exit is at the last index of the array. The numbers in the array represent the power up of the springboard at that position. A mine is represented by the value 0. In this context, if the element at the last index is 0 then the level is not feasible. For example, the board shown in Figure 1 is represented by the array `{3,1,2,0,4,0,1}`. Similarly, the board shown in Figure 2 is represented by the array `{3,2,1,0,2,0,2}`.

(i) [10 marks] In the class `LevelChecker`, implement the **recursive** public static method `boolean check(int[] level)` that takes an array of positive integers representing a game level and returns `true` if the level is feasible, `false` otherwise. You **must** implement the brute force algorithm given in Algoritm 1.

Algorithm 1: Recursive Brute Force algorithm to check the feasibility of a level.

```
Function check(level: Array[int]): boolean
    if size(level) == 1 and level[0] > 0: then //landed at the exit
        return true;
    end
    if level[0] = 0: then //landed on a mine
        return false;
    end
    maxJump = min(level[0], size(level)-1);
    for i = 1 to maxJump do
        A = subArray(level, i, size(level));
        if check(A) then
            return true;
        end
    end
    return false;
end
```

The function `subArray(array, start, end)` returns the sub-array of `array`, starting at index `start` inclusive, and ending at position `end` exclusive.

(ii) [15 marks] In the class `LevelChecker`, implement the public static method `List<Integer> getJumps(int[] level)` that returns the shortest list of jumps to pass the level if the level is feasible, an empty list otherwise. For example, in the level shown in Figure 1, there are two possible solutions to pass the level, $(1,1,2,2)$ and $(2,2,2)$. The method should return the solution $(2,2,2)$. If there is more than one shortest solutions, the method should return only one of them (which one it is does not matter).

(iii) [15 marks] The recursive method `check(int[])` is inefficient. A dynamic programming approach to solve the problem would yield better performance. In the class `LevelChecker`, implement a public static method `boolean betterCheck(int[] level)` that takes an array of positive integers representing a game level and returns `true` if the level is feasible, `false` otherwise. Your solution **must implement a dynamic programming** approach to the solution, failing to do so will result in a mark of 0.

End of examination paper