Comp 251: Assignment 3

Answers must be returned online by April 1^{st} (11:59:59pm), 2022.

General instructions (Read carefully!)

- **Important:** All of the work you submit must be done by only you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. For Comp251, we will use software that compares programs for evidence of similar code. This software is very effective and it is able to identify similarities in the code even if you change the name of your variables and the position of your functions. The time that you will spend modifying your code, would be better invested in creating an original solution.

  Please don't copy. We want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

  Never look at another assignment solution, whether it is on paper or on the computer screen. Never share your assignment solution with another student. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web, or get code from a private tutor, that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in CS involve students who have never met, and who just happened to find the same solution online, or work with the same tutor. If you find a solution, someone else will too. The easiest way to avoid plagiarism is to only discuss a piece of work with the Comp251 TAs, the CS Help Centre TAs, or the COMP 251 instructors.

- Your solution must be submitted electronically on ed-Lessons. Here is a short **tutorial** to help you understand how the platform works.

- To some extent, collaborations are allowed. These collaborations should not go as far as sharing code or giving away the answer. You must indicate on your assignments (i.e. as a comment at the beginning of your java source file) the names of the people with whom you collaborated or discussed your assignments (including members of the course staff). If you did not collaborate with anyone, you write "No collaborators". If asked, you should be able to orally explain your solution to a member of the course staff. At the end of this document, you will find a check-list of the behaviours/actions that are allowed during the development of this assignment.

- This assignment is due on April 1^{st} at 11h59:59 pm. It is your responsibility to guarantee that your assignment is submitted on time. We do not cover technical issues or unexpected difficulties you may encounter. Last minute submissions are at your own risk.

- This assignment includes a programming component, which counts for 100% of the grade, and an optional long answer component designed to prepare you for the exams. This component will not be graded, but a solution guide will be published.

- Multiple submissions are allowed before the deadline. We will only grade the last submitted file. Therefore, we encourage you to submit as early as possible a preliminary version of your solution to avoid any last minute issue.
- Late submissions can be submitted for 24 hours after the deadline, and will receive a flat penalty of 20%. We will not accept any submission more than 24 hours after the deadline. The submission site will be closed, and there will be no exceptions, except medical.
- In exceptional circumstances, we can grant a small extension of the deadline (e.g. 24h) for medical reasons only.
- Violation of any of the rules above may result in penalties or even absence of grading. If anything is unclear, it is up to you to clarify it by asking either directly the course staff during office hours, by email at (`cs251-winter@cs.mcgill.ca`) or on the discussion board on our discussion board (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on one of the communication channels used in the course. It is your responsibility to monitor MyCourses and the discussion board for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum. We urge you to ask your questions as early as possible. We cannot guarantee that questions asked less than 24h before the submission deadline will be answered in time. In particular, we will not answer individual emails about the assignment that are sent the day of the deadline.

Programming component

- You are provided some starter code that you should fill in as requested. Add your code only where you are instructed to do so. You can add some helper methods. Do not modify the code in any other way and in particular, do not change the methods or constructors that are already given to you, do not import extra code and do not touch the method headers. The format that you see on the provided code is the only format accepted for programming questions. **Any failure to comply with these rules will result in an automatic 0.**
- Public tests cases are available on ed-Lessons. You can run them on your code at any time. If your code fails those tests, it means that there is a mistake somewhere. Even if your code passes those tests, it may still contain some errors. We will grade your code with a more challenging, private set of test cases. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the discussion board. Do not include it in your submission.
- Your code should be properly commented and indented.
- **Do not change or alter the name of the files you must submit, or the method headers in these files**. Files with the wrong name will not be graded. Make sure you are not changing file names by duplicating them. For example, main (2).java will not be graded.
- **Do not add any package or import statement that is not already provided**
- Please submit only the individual files requested.
- **You will automatically get 0 if the files you submitted on ed-Lessons do not compile, since you can ensure yourself that they do. Note that public test cases do not cover every situation and your code may crash when tested on a method**

that is not checked by the public tests. This is why you need to add your own test cases and compile and run your code from command line on linux.

Homework

Exercise 1 (30 points). ***Graph Traversal***

I just got an email from Dr. Robert Bruce Banner showing his appreciation for your help during the coding of his training program (i.e., question 2 of our previous assignment). He is impressed with your algorithmic skills and he wants to ask you for help again.

During his mission in the planet Titan, he got captured by the army of Thanos. Dr Banner is currently trapped in a 3D jail and he needs your help to find the **quickest** way out. In his email, Dr Banner describes the jail to be composed by unit cubes which may or may not be filled with indestructible rock. Dr Banner can only move one unit in the following directions: east, west, north, south, up or down. He emphasizes that he is **not** able to move diagonally. Dr Banner says in his email that it takes one minute to move one unit in one of the allowed directions.

Given the description of the 3D jail made by Dr Banner, I have been able to code a representation of it. In particular, the jail will be represented by a 3D String array (`jail[level][rows][columns]`) of one-character String. Each character describes one cell of the jail. A cell of indestructible rock is indicated by a "#" and empty cells are represented by a ".". The current position of Dr Banner (i.e., the starting point) is represented by "S" and the exit of the jail by the letter "E".

Lets see an example of how (I believe) the jail looks like:

```
\\The following jail has 3 levels, each level has a grid of 4 rows and 5 columns.

S....
.###.
.##..
###.#

#####
#####
##.##
##...

#####
#####
#.###
####E
```

From the example shown above, please notice that Dr Banner is originally located in the cell `jail[0][0][0]` (i.e., where the "S" is) and needs to go to the cell `jail[2][3][4]` (where the "E" is). The starting position of Dr Banner is **not** always [0][0][0]. Please notice that the following is the (quickest) sequence of unit moves that Dr Banner has to perform to find the exit.

1. (start) `jail[0][0][0]`
2. (west) `jail[0][0][1]`

3. (west) `jail[0][0][2]`
4. (west) `jail[0][0][3]`
5. (west) `jail[0][0][4]`
6. (south) `jail[0][1][4]`
7. (south) `jail[0][2][4]`
8. (east) `jail[0][2][3]`
9. (east) `jail[0][3][3]`
10. (down) `jail[1][3][3]`
11. (west) `jail[1][3][4]`
12. (down) `jail[2][3][4]`

Then, your algorithm needs to return the integer 11 as the solution of this puzzle (i.e., Dr Banner performed 11 moves to find the exit). Let's see now, an example where Dr Banner will (unfortunately) be trapped in the jail.

```
\\The following jail has 1 levels, the level has a grid of 3 rows and 3 columns.

S##
#E#
###
```

Please notice that for the above example, Dr Banner will not be able to find the exit. In this case your algorithm must return the number -1.

For this assignment, you will complete the function `public static int find_exit(String[][][] jail)`, which receives as a parameter the (`jail[level][rows][columns]`) and returns an integer representing the **shortest** time it takes for Dr Banner to escape from the 3D jail. If it is not possible to escape, the function must return -1. I forgot to mention, please produce correct and efficient code, Dr Banner (and David) can get green of anger if your implementation is not good

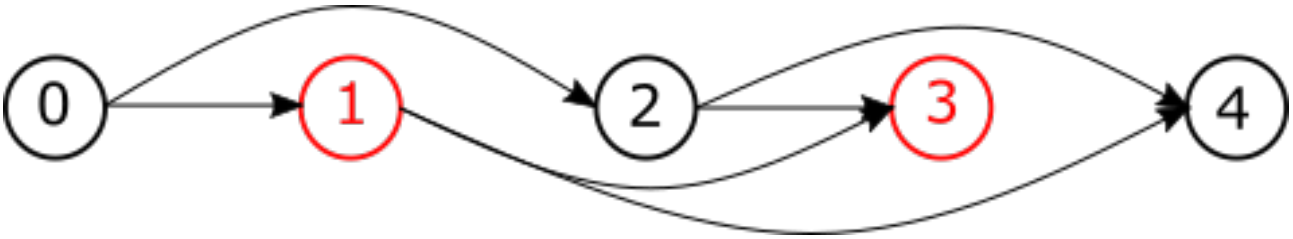
Exercise 2 (40 points). ***Topological Sort***

You will not believe this, but I also got an email from Shang-Chi (yes, go and google the name). He told me that during his last battle, one of his powerful rings got damaged. He needs to repair it as soon as possible; however fixing the ring is not an easy task. The reparation of the ring needs to be conducted in two different planets (planet Earth and planet Asgard). The reason for that is that each planet has a very specialized laboratory. The repair process has been divided into several stages; in which we know which planet each of the stages will take place on. Please notice that the reparation of the ring can begin in any of the two planets.

Transporting the ring (between the two planets) introduces additional risk (e.g., Thanos can try to steal it); therefore, it should be avoided whenever possible. **Ideally, all the reparation**

work in the first planet will be done, and then the ring would be moved to the second one. Unfortunately, there are several dependencies between the reparation stages (i.e., some of them need to be completed before the others may begin). Your job for this question (and to help Shang-Chi) is to find an ordering of reparations stages that **minimizes** the number of times the ring needs to be moved from one planet to the other. In particular, you will need to complete the function `public static int rings(Hashtable<Integer, ArrayList<Integer>> graph, int[] location)`. This function receives as a parameter a hash table that represent the graph. The key of the hash table is a node, and the value of that key is a list of nodes that can not be completed before the key node is completed. A node in our problem represents one of the possible repair stages. The repair stages are enumerated from 0 to $n-1$. The function also receive the integer array `location`, which contains n integers - where the $i-th$ index has value of 1 if the $i-th$ reparation stage will take place in the planet Earth, and 2 otherwise (i.e., it will take place in Asgard). Your function must return an integer representing the **minimal number of times** the ring needs to be transported between the two planets.

Let's see now an example to make sure that the task is clear. Please consider the following graph.



This graph has 5 nodes (enumerated from 0 to 4) which represent five different stages that need to be performed to repair the ring. These five stages can happen in two different planets; nodes 0, 2 and 4 (which are black in my draw) happens in the planet Earth. Nodes 1 and 3 (which are red in my draw) happens in the planet Asgard. Please notice that the edges of the graph impose the order on which the reparations need to be executed. For example, the reparation stages represented by node 1 and 2, can not be executed before the reparation of node 0 is done. This graph is represented in a hash table, where each slot in the table represents a node. The value of a key (i.e., each node) is an array list of the nodes that depends on the key. For example, if node 0 is the key, the value will be a list containing the values 1 and 2. The information of the planet where each reparation takes place is coded in an array, where a value of 1 represent the planet Earth, and a value of 2 represent Asgard. For the provided example, the array will look like this `location = [1,2,1,2,1]`.

The optimal order to perform the reparations in the example is as follows.

1. Start the reparation in planet Earth.
2. Perform the reparations in node 0 .
3. Perform the reparations in node 2.
4. Transport the ring to planet Asgard.
5. Perform the reparations in node 1.
6. Perform the reparations in node 3.

7. Transport the ring to planet Earth.

8. Perform the reparations in node 4.

Please notice that this order of reparations respect the dependencies and the location of the stages. Furthermore, this sequence minimize the number of times that the ring needs to be transported between the two planets. In particular, the ring is only transported between the planets two times. Your algorithm should then return as answer the integer 2.

HINT1: Remember that this question is in the topological sort section
HINT2: Remember that ideally, all the (possible) reparation work in one planet needs to be done, before the ring is moved to the other planet

Exercise 3 (30 points). ***Flow Network***

In this exercise, we will implement the Ford-Fulkerson algorithm to calculate the Maximum Flow of a directed weighted graph. Here, you will use the files WGraph.java and FordFulkerson.java, which are available on the course website. Your role will be to complete two methods in the template FordFulkerson.java.
The file `WGraph.java` implements two classes `WGraph` and `Edge`. An Edge object stores all informations about edges (i.e. the two vertices and the weight of the edge), which are used to build graphs.

The class `WGraph` has two constructors `WGraph()` and `WGraph(String file)`. The first one creates an empty graph and the second uses a file to initialize a graph. Graphs are encoded using the following format: the first line corresponds to two integers, separated by one space, that represent the "source" and the "destination" nodes. The second line of this file is a single integer n that indicates the number of nodes in the graph. Each vertex is labelled with a number in $[0, \ldots, n-1]$, and each integer in $[0, \ldots, n-1]$ represents one and only one vertex. The following lines respect the syntax "n_1 n_2 w", where n_1 and n_2 are integers representing the nodes connected by an edge, and w the weight of this edge. n_1, n_2, and w must be separated by space(s). It includes setter and getter methods for the Edges and the parameters "source" and "destination". There is also a constructor that will allow the creation of a graph cloning a `WGraph` object. An example of such file can be found on the course website in the file `ff2.txt`. These files will be used as an input in the program `FordFulkerson.java` to initialize the graphs. This graph corresponds to the same graph depicted in [CLRS2009] page 727.
Your task will be to complete the two static methods `fordfulkerson(Integer source, Integer destination, WGraph graph, String filePath)` and `pathDFS(Integer source, Integer destination, WGraph graph)`. The second method `pathDFS` finds a path via Depth First Search (DFS) between the nodes "source" and "destination" in the "graph". You must return an ArrayList of Integers with the list of unique nodes belonging to the path found by the DFS. The first element in the list must correspond to the "source" node, the second element in the list must be the second node in the path, and so on until the last element (i.e., the "destination" node) is stored. The method `fordfulkerson` must compute an integer corresponding to the max flow of the "graph", as well as the graph depicting this max flow. Once completed, compile all the java files and run the command line `java FordFulkerson ff2.txt`. Your program will output a String containing the relevant information. An example of the expected output is available in the file `ff2testout.txt`. This output keeps the same format than the file used to build the graph; the only difference is

that the first line now represents the maximum flow (instead of the "source" and "destination" nodes). The other lines represent the same graph with the weights updated to the values that allow the maximum flow. There are a few other open test cases you can access on Ed-Lessons. You are invited to run other examples of your own to verify that your program is correct. There are namely some examples in the textbook.

What To Submit?

Attached to this assignment are java template files. You have to submit only this java files. Please DO NOT zip (or rar) your files, and do not submit any other files.

Where To Submit?

You need to submit your assignment in ed - Lessons. Please review the tutorial 2 if you still have questions about how to do that (or attend office hours). Please note that you do not need to submit anything to myCourses.

When To Submit?

Please do not wait until the last minute to submit your assignment. You never know what could go wrong during the last moment. Please also remember that you are allowed to have multiple submission. Then, submit your partial work early and you will be able to upload updated versions later (as far as they are submitted before the deadline).

How will this assignment be graded?

Each student will receive an overall score for this assignment. This score is the combination of the passed open and private test cases for the questions of this assignment. The open cases correspond to the examples given in this document plus other examples. These cases will be run with-in your submissions and you will receive automated test results (i.e., the autograder output) for them. You MUST guarantee that your code passes these cases. In general, the private test cases are inputs that you have not seen and they will test the correctness of your algorithm on those inputs once the deadline of the assignment is over; however, for this assignment you will have information about the status (i.e., if it passed or not) of your test. Please notice that not all the test cases have the same weight.

Student Code of Conduct Assignment Checklist

The instructor provides this checklist with each assignment. The instructor checks the boxes to items that will be permitted to occur in this assignment. If an item is not checked or not present in the list, then that item is not allowed. The instructor may edit this list for their case. A student cannot assume they can do something if it is not listed in this checklist, it is the responsibility of the student to ask the professor (not the TA).

Instructor's checklist of permitted student activities for an assignment:

Understanding the assignment:

- ☒ Read assignment with your classmates
- ☒ Discuss the meaning of the assignment with your classmates
- ☒ Consult the notes, slides, textbook, and the links to websites provided by the professor(s) and TA(s) with your classmates (do not visit other websites)
- ☐ Use flowcharts when discussing the assignment with classmates.
- ☒ Ask the professor(s) and TA(s) for clarification on assignment meaning and coding ideas.
- ☐ Discuss solution use code
- ☐ Discuss solution use pseudo-code
- ☐ Discuss solution use diagrams
- ☐ Can discuss the meaning of the assignment with tutors and other people outside of the course.
- ☐ Look for partial solutions in public repositories

Doing the assignment:

- Writing
  - ☒ Write the solution code on your own
  - ☒ Write your name at the top of every source file with the date
  - ☒ Provide references to copied code as comments in the source code (e.g. teacher's notes). Please notice that you are not allowed to copy code from the internet.
  - ☐ Copied code is not permitted at all, even with references
  - ☐ Permitted to store partial solutions in a public repository
- Debugging
  - ☒ Debug the code on your own
  - ☒ Debugging code with the professor
  - ☒ Debugging code with the TA
  - ☒ Debugging code with the help desk
  - ☒ Debugging code with the Internet. Please notice that this is allowed to debug syntax errors, no logic errors.
  - ☐ You can debug code with a classmate
  - ☐ You can debug code with a tutor or other people outside of the course
- Validation
  - ☒ Share test cases with your classmates
- Internet

☒ Visit stack-overflow (or similar). Please notice that this is allowed only to debug syntax errors, no logic errors.
☐ Visit Chegg (or similar)

- Collaboration
  - ☐ Show your code with classmates
  - ☐ Sharing partial solutions with other people in the class
  - ☐ Can post code screenshots on the course discussion board
  - ☒ Can show code to help desk

Submitting and cleaning up after the assignment:

- ☐ Backup your code to a public repository/service like github without the express written permission from the professor (this is not plagiarism, but it may not be permitted)
- ☐ Let people peek at your files
- ☐ Share your files with anyone
- ☐ ZIP your files and upload to the submission box
- ☒ Treat your work as private
- ☐ Make public the solutions to an assignment
- ☐ Discuss solutions in a public forum after the assignment is completed