Comp 251: Assignment 3

Answers must be returned online by April 4^{th} (11:59:59pm), 2023.

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.

- 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 4^{th} 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**
- **Do not create new nested classes or static variables**
- 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 (35 points)**.** ***Graph Algorithms***

During reading week, my kid was playing (again) with legos. For one particular game, my kid had one lego mini figure of Spider-Man located in a $N \times M$ grid. My kid told me that the grid represented the city of New York (which was being attacked by Dr. Otto Gunther Octavius). The goal of the game is to find the fastest route to escape the city walls. The rules of the game are as follows:

- It takes 1 unit of time to swing from one building to another (each building corresponds to a cell in the grid).
- Spider-Man can swing horizontally or vertically.
- Spider-Man can not pass through burning buildings. These buildings are represented by the `String` "1".
- Safe buildings (which Spider-Man can pass through) are represented by the `String` "0".
- There are some buildings that can only be accessed from a specific direction. In particular:
  - A building represented by the `String` "U" can only be accessed from the building above.
  - A building represented by the `String` "D" can only be accessed from the building below.
  - A building represented by the `String` "R" can only be accessed from the right building.
  - A building represented by the `String` "L" can only be accessed from the left building.
- The initial position of Spider-Man is represented by the `String` "S".
- If Spider-Man escapes at or before a given time `t`, then he is safe. Please notice that Spider-Man will be safe once he reaches any building in the border of the city.
- Here are the restrictions of the game:
  - $(1 \leq t \leq 200)$
  - $(1 \leq N \leq 100)$
  - $(1 \leq M \leq 100)$

For this assignment, you will complete the function `public static int find_exit(int time, String[][] board)`, which receives as a parameter the time `t` that Spider-Man has to try to escape, and the $M \times N$ grid (called `board`) that represents the city of New York. If it is possible for Spider-Man to escape in the provided time, your function must output the minimum number of

buildings that must be visited to escape. If it is not possible to escape, your function must return the integer -1.

Lets see an example of how the city of New York looks like:

```
\\This is a 4X4 grid with a total of 16 buildings.
1111
1S01
1011
0U11
```

From the example shown above, please notice that Spider-Man is originally located in the cell `board[1][1]` (i.e., where the "S" is) and needs to go to any building located on the boarder of the grid. The following is the quickest sequence of unit moves that Spider-Man has to perform to be safe.

1. (start) `board[1][1]`
2. (south) `board[2][1]`
3. (south) `board[3][1]` //Please notice that I can access this cell (which is marked with the string "U") because I came from the building above. Also notice that this building is located on the border of the city, so Spider-Man is safe.

Then, (if the time $t \geq 2$) your algorithm needs to return the integer 2 as the solution of this puzzle (i.e., Spider-Man performed 2 swings to be safe).

Let's see now, an example where Spider-Man will unfortunately be trapped in the city.

```
\\This is a 4X4 grid with a total of 16 buildings.

1111
1S01
1011
0L11
```

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

Exercise 2 (35 points)**.** ***Graph Algorithms***

The following is a real world problem that affects many parents who buy Lego for their kids. Kids build with Lego to later take apart all the pieces. The reality is that all the different Lego pieces (that you bought one day) end in a huge box. The problem arises when kids want to build with Lego again. In particular, kids have the building instructions, but they need to look for the pieces in the huge box (making the Lego building process super inefficient). Today, we will find a solution to this terrible problem affecting so many parents around the world. Specifically, given the instructions to build a Lego toy, we want an algorithm to extract (before starting the construction) all the required Lego pieces from the huge box.

As you may know (did you play with Lego?), some complex Lego modules can be created by combining simpler ones. As it turns out, some of those simpler modules might also be built with even simpler modules and so on. For example, to build a Lego-arm of a robot, we need a

Lego-hand. To build each Lego-hand, we would need 5 Lego-fingers. To build each Lego-finger, we need 3 Lego-phalanx. Then, our total list of modules (to build two arms of our robot) would be 2 Lego-arms, 2 Lego-hands, 10 Lego-fingers, and 30 Lego-phalanx. Given how many of each modules the kid needs, (two Lego-arms in our example), please help me to find out how many of the other simpler modules (Lego-hands, Lego-fingers, and Lego-phalanx in our example) we would need to build them.

For this question, you will need to complete the function `num_pieces` which receives as parameters an array of `long` numbers called `pieces`, and a 2-D array of `integers` called `instructions`. The array `pieces` stores the amount of each Lego module we need to built our toy. In particular, the integer a_i stored at the index i, specifies the amount of the i-th module we need, where $0 \leq a_i \leq 3$. The size N of the array `pieces` is between the following limits $2 \leq N \leq 50$. The array `instructions` has dimensions $M \times 3$ and it stores all the module dependencies. The limits for M are as follows: $N - 1 \leq M \leq \frac{N(N-1)}{2}$. Each row of the array `instructions` stores three integers (i.e., `u`, `v`, and `w`, stored in columns 1, 2, and 3, respectively) that indicates that there is an instruction that takes w quantities of module u to produce **one** material v. You can safely assume that there will never be any cycles in the building instructions. The limits for u, v, and w are as follows: $0 \leq u, v < N$, and $0 \leq w \leq 3$. The output of the function `num_pieces` must be an N-length array storing the amount of modules needed to build the toy.

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

`pieces` = `[0,0,0,0,3]`

`instructions` = `[[0,1,3], [1,4,1], [2,4,1], [3,4,2]]`

From those arrays we can extract the following information:

- From the `pieces` array:
  - We need to build 5 modules (i.e., the length of the array `pieces`) to assemble our toy. Initially, we know that we would need 3 times the module 4 (i.e., because the 3 stored in the index 4 of the array `pieces`).
- From the `instructions` array:
  - We need 3 modules 0 to build the module 1 (i.e., first row of the array `instructions`).
  - We need 1 module 1 to build the module 4 (i.e., second row of the array `instructions`).
  - We need 1 module 2 to build the module 4 (i.e., third row of the array `instructions`).
  - We need 2 modules 3 to build the module 4 (i.e., fourth row of the array `instructions`).

We can now update the information of `pieces` to reflect the total number of modules that we will need to build our toy.

- For module 4.
  - As stated in the definition of the problem, we need **3** modules 4 to complete the toy.
- For module 3.

  - As stated by the `instructions` array, we need 2 modules 3 to build one module 4. We need to build 3 modules 4, then, we will need **6** modules 3 to complete the toy.

- For module 2.
  - As stated by the `instructions` array, we need 1 module 2 to build one module 4. We need to build 3 modules 4, then, we will need **3** modules 2 to complete the toy.

- For module 1.
  - As stated by the `instructions` array, we need 1 module 1 to build one module 4. We need to build 3 modules 4, then, we will need **3** modules 1 to complete the toy.

- For module 0.
  - As stated by the `instructions` array, we need 3 module 0 to build one module 1. And we already know that we need 3 modules 1 to complete the toy. Then, we will need **9** modules 0 to complete the toy.

Based on the above-mentioned information, the array that your function must return is `[9,3,3,6,3]`.

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(WGraph graph)` 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 encoding the assignment associated with 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 these 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