COMP 251: Assignment 3

Answers must be returned online by April 8^{th} (11:59:59pm), 2025.

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.

- This assignment is due on April 8^{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.

- 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 (`comp251.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 that you must submit or the method headers in these files**. Files with the wrong name will not be graded. Make sure that 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 the command line on linux.**

Homework

Exercise 1 (50 points)**.** ***Graph Algorithms - Sensitive species protection***

Certain species are particularly sensitive to the effects of humans "progress". In particular, construction projects affect food patterns for animals in many ways, including changes to food availability, habitat loss, and changes in diet. When the government's environmental agencies notice that a certain species are endangered, they run simulations to identify the change in food patterns. In Quebec, the chorus frog is one such example[1]. Suppose that conservation biologists have canvassed different areas of the greater-Montreal on a grid made of 1km x 1km squares. See Figure 1 for an example of the grid. At every location of the grid, they indicated the following information:

- A "dot" (i.e., ".") represents the presence of food that the chorus frog consumes.
- A "space" (i.e., " ") represents the absence of food that the chorus frog consumes.
- A capital "ex" (i.e., "X") represent a human or natural barrier that can not be crossed by the chorus frog.
- Capital letters A-W are entrances to the areas. In this example (A,B,C,D) are entrances.

```
XXXXXXXAXXXXXXXBXXXX
X.. ..X.X..... ...X
X.XXX...X.X.XXXXXX.X
X.X.XXXXX.X.X....X.X
X.X... ...X.X.XX.X.X
X.X.X.XXXXXXX.XX.X.X
X.X.X.X...X...X....X
X.X.X.XXXXXXX.XXXX.X
X...X.X X.. ..X..X.X
XXXXXXXDXXXXXXXXCXXX
```

Figure 1: Example of a grid representing a Montreal area

The simulation has the following constrains (or simplifications).

- Each chorus frog enter the area at a different entrance.
- The chorus frogs always try to consume all the food (i.e., dots in the grid) as quickly as possible.
- Chorus frogs can only move down, up, right and left.
- Chorus frogs can only move through spaces and dots (not barriers).

[1] https://www.cbc.ca/news/canada/montreal/western-chorus-frog-longueuil-road-1.7179384

- If two entrances are adjacent, the chorus frogs are not allowed to move from one to the other (as you may know, frogs are very territorial).
- The border of the analyzed area (rows $1, n$, and columns $1, m$) are guaranteed to consists only of capital letters $A \ldots X$.
- All the entrances are guaranteed to be in the border of the grid.

You (as a COMP251 student) have been commissioned (by David :P) to design an **efficient and correct** graph algorithm that computes the minimum number of chorus frogs necessary to eat all the reachable food (i.e., the dots in the grid). Please notice that the simulations analyze different areas of the greater-Montreal (in other words, your algorithm will be tested with different grids). Then, the minimum number of frogs needed to consume all of the food can vary, and even some food might not be reachable at all. So, your algorithm will need also to return the number of food spaces (i.e., dots in the grid) that are not reachable (because they are blocked by the human constructions).

Your job for this part of the assignment is to complete the function `saving_frogs`, which takes an `int[][]` 2d-array that represents the grid and returns a 1d-array of length 2 that reports (in the index 0 and as an integer) the minimum number of frogs necessary to eat all the food (or zero if no food is reachable), and (in the index 1 and as an integer) the number of food spots that could not be reached. This function is located in the java file `A3Q1`. Note: For your reference, your algorithm must return `[2,3]` as the answer to the problem shown in the Figure 1. In this grid, all the reachable food can be reached from 2 entrances (`A` and `C`, or `B` and `C`) and there are three food spots that can not be reached.

Exercise 2 (50 points)**.** ***Graph Algorithms***

Do you know that there was one time when United Airlines sold lifetime passes?[2]. After reading that news article (be sure to check the link in the footnote - it will not help with the assignment, but it is a great read! :P), I haven't been able to stop thinking about how, with that pass, you wouldn't need to rent or own a place to live. On top of that, you wouldn't even have to cook your own meals! You just need to keep booking consecutive flights where you can sleep and eat free - what a deal! :O. The only caveat to this plan is that we must ensure continuous access to flights as long as necessary to not get stuck in an airport. Given a list of daily United Airlines flights between pairs of cities and a list of city names used in the United itinerary, you have been tasked (once again by David) with implementing an **efficient and correct** algorithm. This algorithm must determine, for each city on the list, whether my plan will succeed or if I will end up stranded at an airport.

Your job for this part of the assignment is to complete the function `time_pass`, which takes (i) an `String[][] itinerary` 2D-array representing the United daily itinerary and (ii) an `String[] cities` 1D-array representing the list of (some) cities used in the itinerary. `time_pass` must return a `String[]` 1D-array reporting for each city, in the same order than `String[] cities`, whether my plan "succeed" or "failed" in that city.

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

[2]https://www.theguardian.com/travel/2023/jun/25/new-jersey-man-lifetime-united-airlines-pass

`String[][] itinerary = { {Arlington, San_Antonio}, {San_Antonio, Baltimore}, {Baltimore, New_York}, {New_York, Dallas}, {Baltimore, Arlington} }`

`String[] cities = {San_Antonio, Baltimore, New_York }`

A call to the function `time_pass(itinerary, cities)` must return the array: `[succeed, succeed, failed]`.

Please check the following notes about our previous example:

- The first row of the array `itinerary` implies that there is a **one-way** flight from `Arlington` to `San_Antonio` every day.
- The array returned as the answer by your algorithm (i.e., `[succeed, succeed, failed]`), can be read as follows: There are infinitely long sequences of cities David can flight to making one flight each day from `San_Antonio` and `Baltimore`. On the contrary, David will find himself stranded at an airport (at some point) if he flights from `New_York`.

Exercise 3 (60 points)**.** ***Graph Algorithms***

This year, I attended a hands-on Arduino workshop at McGill, where we learned the basics of Arduino, electrical circuits, and Arduino coding. The workshop focused on assembling various electronic components - including pushbuttons, light-emitting diodes (LEDs), potentiometers, and resistors - onto a breadboard. Our goal was to logically connect these components to create a functional circuit. At one point, the instructor handed out wires for the connections and asked me, `"David, what is the minimum length of wire I need to provide you to ensure that you create a closed circuit?"` In the context of this workshop, a closed circuit meant that every electronic component on the breadboard was interconnected through a continuous sequence of wires. I didn't have an exact answer at that moment, but I told the instructor that my COMP251 students would be more than happy (wouldn't you?) to design an **efficient and correct** algorithm to solve it. In particular, I would give you all the `(x,y)` breadboard locations that need to be connected and the job of your algorithm will be to connect the locations (using straight lines of wire) so as to minimize the length of wire used.

For this question, you will need to complete the function `arduino` which receives as parameter an 2-D array of `double` numbers called `locations`. The output of the function `arduino` must be a single `double` number to two decimal places representing the minimum total length of wire that can connect all the locations.

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

`double[][] locations = { {1.0, 1.0}, {2.0, 2.0},{2.0, 4.0} }`

A call to the function `arduino(locations)` must return the double value: `3.41`.

Please check the following notes about our previous example:

- The answer (i.e., `3.41`) is truncated to have two decimal values.
- The answer correspond to wire (in straight lines) the following locations:
  - `{1.0, 1.0}` with `{2.0, 2.0}` for a length of 1.4142.

– `{2.0, 2.0}` with `{2.0, 4.0}` for a length of 2.00

HINT: Make sure that you're considering the graph sparsity when working on your solution.

Exercise 4 (60 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 MyCourses. 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 MyCourses 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 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 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.