Comp 251: Assignment 3

Instructor: Jérôme Waldispühl

Due on November 7th at 11h55:00 PM

- Your solution must be submitted electronically on MyCourses.
- 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, but this is at your own risk; helper methods could cause your methods to crash when called from other programs if not added responsibly. 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 give you an automatic 0.**
- The starter code includes a tester class. 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 set of examples. We therefore highly encourage you to modify that tester class, expand it and share it with other students on the myCourses 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 one of the files you must submit**. 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. Make sure to double-check your zip file.
- Do not submit individual files. Include all your files into a .zip file and, when appropriate, answer the complementary quiz online on MyCourses.
- **You will automatically get 0 if the files you submitted on MyCourses do not compile.**
- 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.
- 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.
- Multiple submissions are allowed before the deadline. We will only grade the last submitted zip 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 will receive a penalty of 20% per day. We will not accept any submission more than 72 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. However, such request must be submitted before the deadline, and justified by a medical note from a doctor, which must also be submitted to the McGill administration.
- 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@cs.mcgill.ca`) or on the discussion board on myCourses (recommended). Please, note that we reserve the right to make specific/targeted announcements affecting/extending these rules in class and/or on the website. It is your responsibility to monitor the course website and MyCourses for announcements.
- The course staff will answer questions about the assignment during office hours or in the online forum on MyCourses. 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 sent the day of the deadline.
- You should compile all your files directly from command line without using a package, by using the command `javac *.java`

1. (40 points) 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 is the similar to the file that you used in your previous assignment to build graphs. The only differences are the addition of setters and getters methods for the Edges and the addition of the parameters "source" and "destination". There is also an additional constructor that will allow the creation of a graph cloning a WGraph object. Graphs are also encoded using a similar format than the one used in the previous assignment. The only difference is that now the first line corresponds to two integers, separated by one space, that represent the "source" and the "destination" nodes. An example of such file can be found on the course website with 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 through a Depth First Search (DFS) between the nodes "source" and "destination" in the "graph" through non-zero weight edges. You must return an ArrayList of Integers with the list of unique nodes belonging to the path found by the DFS. If no path is found, return an empty ArrayList. 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" and the graph itself. The method `fordfulkerson` has a variable called `myMcGillID`, which must be initialized with your McGill ID number.
Once completed, compile all the java files and run the command line `java FordFulkerson ff2.txt`. Your program must use the function writeAnswer to save your output in a file. An example of the expected output file is available in the file `ff226000000.txt`. This output keeps the same format than the file used to build the graph; the only difference is that the a line has been added; the first line now represents the maximum flow (instead of the "source" and "destination" nodes). If the `fordfulkerson` method was unable to compute the maximum flow, it should output a result of -1 (and not throw an exception). The other lines represent the same graph with the weights updated with the values that represent the maximum flow. The file `ff226000000.txt` represent the answer of the example showed in [CLRS2009] page 727. You are invited to run other examples of your own to verify that your program is correct.

2. (40 points) We want to implement the Bellman-Ford algorithm for finding the shortest path in a graph where edge can have negative weights. This question extends the previous question on the implementation of the Dijkstra's algorithm done in the assignment 2. You will need to execute this program to use the same auxiliary class `Wgraph` used in question 1. Your task is to fill the method `BellmanFord(WGraph g, int source)` and `shortestPath(int destination)` in the file `BellmanFord.java`.
The method `BellmanFord` takes a object `WGraph` named `g` as an input (See Assignment 2) and an integer that indicates the source of the paths. If the input graph `g` contains a negative cycle, then the method should throw an exception. Otherwise, it will return an object `BellmanFord` that contains the shortest path estimates (the private array of integers `distances`), and for each node its predecessor in the shortest path from the source (the private array of integers `predecessors`).
The method `shortestPath` will return the list of nodes as an array of integers along the shortest path from the source to the node destination. If this path does not exists, the method should throw an exception.
Please take a look at the code, we defined some exceptions that you should use when appropriate if one of your methods fails to terminate.
Input graphs are available on the course webpage to test your program. Nonetheless, we invite you to also make your own graphs to test your program.

3. (20 points) You will complete this section through MyCourses. Note that you MUST use your own results to answer those questions. Answers to this quiz that would not conceptually match the output of your program will be considered plagiarism (refer to course outline).

You will submit FordFulkerson.java and BellmanFord.java in a single zip file.