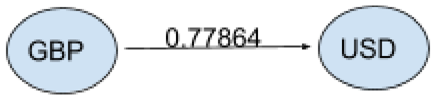
HW7
Due Friday December 7th, 11:59pm
100 points

Java Program - Graph Traversal, Foreign Exchange Currencies

Different currencies have different exchange rates. For example, the exchange rate from US Dollars (USD) to British Pounds (GBP) is 0.77864. The price is quoted as a “currency” pair, and USD/GBP-0.77864 can be read as: in order to buy one unit of USD, it costs 0.77864 British Pounds.

On rare occasion, there exists a scenario where you can exchange one currency to another, and then another, and then back to the original currency for more money than you started with. This is similar to a negative cycle in a graph.

You will create a directed weighted graph, where the nodes represent currencies, and the edges represent the exchange rate between currencies. For example USD/GBP-0.77864 would be represented in a graph as:



You will use the starter code provided to create a graph that represents the currency pair exchange rates, and search for a scenario where you can exchange one currency for another, then another, and have more money in the original currency when you are done. To accomplish this you will need to search for a cycle in the graph, where the path from a source node to a node in the cycle, is a different weight, than the path from the node back to the source node. Dijkstra’s algorithm will give you the shortest path from a source node to all other nodes in the graph. You can use this to detect if there exists a path from the source node to another, that is a different weight from the other node back to the source.

To simulate, let your program start with $100.00 USD, and print to the screen if you have found a cycle that leaves you with more than $101.00 .

You will be provided the following files:

currency_adjacency.csv - this file represents the major currency pairs and their exchange rates. The file can be read as- one unit of the currency on the left(row), can be purchased for this amount of the currency on the top (column). See USD/GBP-0.77864 for an example.

Node.java - A class for a Node, to be used in Graph.java

Graph.java - A class for a Graph, to be used in Dijsktra.java

Dijkstra.java - You are provided with an example of dijkstra's algorithm. The Main method in this follow creates nodes, initializes the graph, and solves the shortest path using dijkstra's algorithm.

Note: You will need to change Dijkstra.java to fit your purposes here.
Hint: Get Dijkstra.java running in your environment, understand how it works, and then move on from there.

Submit to Canvas:
Submit one zip file containing your src folder with your .java files and any other files your program needs to run. Please make your zip file well organized for the grader. You can either zip one src folder with two packages (one for 7.49a and one for 7.50), or you can zip together two separate src folders. Just make it clear to the grader what you did.

Program Grading Criteria	**% Points**
Program(s) fulfill all the requirements. All .java files are included and declare the necessary classes. Code is well organized, and easy to follow (especially for the grader). Coding stye is well utilized, including well named variables and methods. Comments are included, well written and descriptive.	100%
Program(s) fulfill almost all of the requirements. All .java files are included and declare the necessary classes. Code is fairly well organized, and somewhat easy to follow (especially for the grader). Some comments are included.	80%
Program(s) fulfill most of the requirements. All .java files are included and declare the necessary classes. Code is fairly well organized, and somewhat easy to follow (especially for the grader). Some or no comments are included.	60%

Program(s) fulfills some of the requirements, or does not run at all. Some .java files are included and declare some of the necessary classes. Some or no comments are included.	40%
Program(s) does not run at all. Some .java files are included and declare some of the necessary classes. Some or no comments are included.	20%
Either no attempt was made, or the attempt made shows no progress toward solving the problem.	0%