CSU22010: Data Structure and Algorithms - HT: Assignment 2

Note: Please read the specification of the problem CAREFULLY. The solution is relatively short and simple, but relies on understanding the problem and the constraints correctly.

In this assignment you will work with shortest path algorithms to solve a problem of a number of people meeting up in a city at a random location.

Total points for this assignment: 200 (100 automatic, 100 marked by demonstrators)

The following will be marked:

1. Correctness of your results, JUnit tests, and test code coverage – automatic mark through web-cat, 100 points
2. Correct and efficient implementation of your algorithms and supporting data structures, 100 points by demonstrators.

Submission and automatic marking is through https://webcat.scss.tcd.ie/cs2012/WebObjects/Web-CAT.woa. Submission of final version both through Web-CAT and Blackboard**. PLEASE NOT FORGET TO SUBMIT THE ASSIGNMENT ON BLACKBOARD TOO!**

Deadline: Wednesday April 8thth 2020 23:45

Late assignments will be deducted 40 points per day

Please submit only CompetitionDijkstra.java, CompetitionFloydWarshall.java and CompetitionTests.java and any additional files you created in a single zip file.

Problem description:

A Contest to Meet (ACM) is a reality TV contest that sets three contestants at three random city intersections. In order to win, the three contestants need all to meet at any intersection of the city as fast as possible.

The contestants may arrive at the intersections at different times, in which case, the first to arrive can wait until the others arrive.

From an estimated walking speed for each one of the three contestants, ACM wants to **determine the minimum time that a live TV broadcast should last to cover their journey regardless of the contestants' initial positions and the intersection at which they finally meet**. You are hired to help ACM answer this question.

You may assume the following:

- Each contestant walks at a given estimated speed.
- The city is a collection of intersections in which some pairs are connected by one-way streets that the contestants can use to traverse the city. Two intersections can be connected by two one-way streets allowing travel in opposite directions of each other.

Input:

As input you are given the following parameters:

- A filename (String) containing the details of the road network, as follows:
    - Line 1 = integer N representing the total number of intersections
    - Line 2 = integer S representing the total number of streets in the city. All streets are one-way, they connect two intersections, and there is at most one street connecting any pair of intersections in any one direction (at most two streets, one in each direction)
    - Lines 3 onwards – each line represents a street and consists of 2 integers (2 intersections that the street is connecting) and a double (representing the street length in kilometers), separated by a space.

      A sample input file might look like:

      3

      2

      1 2 0.5

      2 3 0.75

      Representing a city with 3 intersections and 2 streets, in which the length of the street leading from intersection 1 to 2 is 0.5km, and the length of the street leading from intersection 2 to 3 is 0.75.
- Three integers representing the average walking speed of each of the three contestants, in meters per minute *sA*, *sB*, *sC where* (50 ≤ *sA*, *sB*, *sC* ≤ 100)

Assignment specification

You need to solve the above problem using two different shortest path algorithms and discuss the differences between their implementation and performance.

1. **Dijkstra version**

   Download CompetitionDijkstra.java file.

Write a java class CompetitionDijkstra in CompetitionDijkstra.java file (please do not use custom packages as web-cat will give an error) which should implement at least the following methods:

- CompetitionDijkstra (String filename, int sA, int sB, int sC) – constructor for this class should take the four parameters as specified in the input, and create and populate the most appropriate data structure in which to hold the city road network in this example. Walking speeds should be stored in member variables.

- public int timeRequiredforCompetition()- this method should return an integer indicating the minimum number of minutes that will pass before the three contestants can meet in the city generated in your constructor, if they start to walk immediately after the show starts. Remember that the contestants can be originally located at any random (unknown) intersection and can decide to meet at any random unknown intersection: you need to account for the worst case scenario. The answer should be given rounding decimals to the next integer (e.g., 2.9 minutes rounds up to 3 minutes and 3.2 minutes rounds up to 4 minutes). If it is not possible to run the given competition in a given city represented by the

map you generated (i.e., if there are 2 random locations in a city between which no path exists), the method should return -1, as should for any other errors (eg input walking speeds outside the specified range). To implement this method you have to use Dijkstra's shortest path algorithm.

2. Floyd-Warshall version

Download CompetitionFloydWarshall.java file.

Write a java class CompetitionFloydWarshall in CompetitionFloydWarshall.java file. The class should have the exact same functionality and the same Constructor parameters and method signature as specified in part 1 but use Floyd-Warshall's algorithm to generate shortest paths.

3. Testing

Download CompetitionTests.java file.

Write a java class CompetitionTests in CompetitionTests.java file, which should implement JUnit tests for your CompetitionDijkstra and CompetitionFloydWarshall classes

Your goal is to write enough tests so that:

- Each method in in the classes is tested at least once,
- Each decision (that is, every branch of if-then-else, for, and other kinds of choices) in is tested at least once,
- Each line of code in all classes is executed at least once from the tests.

The submission server will analyse your tests and code to determine if the above criteria have been satisfied.

You are given 2 test files to test your implementations with (download from Blackboard).

You also might need to create additional test files yourself, to ensure your code is fully tested and works for all graph characteristics. Please submit these additional files together with the rest of the assignment in a zip file. List the names of the additional files you created in the comment at the top of CompetitionTests file, and for each file explain the characteristics of the graph/which case does this particular file test.

For Fun (not marked)

Implement the same competition using Bellman-Ford algorithm and compare the performance to Dijkstra and Floyd-Warshall for different type/size of input graphs. Download a sample file containing 1 milion vertices and 15 million edges from https://algs4.cs.princeton.edu/44sp/ and try to test the performance. Modify input files to contain negative weights and run the competition. You will need to extend the code to first detect negative cycles. Observe what happens to Dijkstra version.

Appendix: reminder of general assignment instructions from Semester 1

Please see a walkthrough on how to submit an assignment on Web-CAT.

When you upload code for an assignment to the submission server, it compiles it and runs your JUnit tests, giving you back an automatic score. This score is part of your marks for this assignment; the other part is given manually by the teaching staff. You can improve and reupload your code to improve your score. The only limit is the submission deadline.

Students are allowed to discuss assignments but **not to share code!** Sharing code will result in reduced marks for all students involved and the consequences described in the College rules. If you discuss an assignment with fellow students then **you must write the names of the students in your submission**. All students must complete the College's online seminar about plagiarism before submitting any assignment.

- Write your name next to @author at the beginning of each file
- Write the names of people you discussed this assignment with under the @author line. Do not share code and do not write code for others!
- You need to adequately test each method in your source code by adding sufficient jUnit tests
- Do not import data structures from the java libraries unless specified you are allowed do so.
- The submission server for this assignment will open shortly.