2017 Career Fair Programming Competition

Problem Revealed: Sunday, 5 March 2017
Submissions due: **Sunday, 17 March 2017** at **10pm** (on Courseware)

Contents

Problem Description

You have been provided with a log (in a CSV file) of the movement of one of Ashesi's shuttle buses. The file contains the bus's location, recorded several times each minute, over a period of a month. Your challenge is to process this data file to perform the following tasks. The tasks are listed in increasing level of difficulty. You are encouraged to participate even if you cannot solve all three tasks.

Task 1 *(Difficulty Level: 1)*
Determine the approximate location (latitude and longitude, approximated to 2 decimal places each), of where the vehicle is usually parked each night.

Task 2 *(Difficulty Level: 2)*
You have been provided with another file of "interest points" in which each line has a GPS location of interest and a distance value in kilometres (km). For each of the GPS locations specified in the file, imagine a circular region around the given GPS location (a "geofence"), with a radius of the given distance. Compute the number of times the vehicle enters the region, and the average time spent within the region each time it enters. You will do this for each GPS location listed in the file.

Task 3 *(Difficulty Level: 3)*
Identify any "unusual" trips made by the bus (i.e. trips that do not fall in the usual pattern of most trips in the log file). In doing this, you should ignore "trips" in which the bus is effectively stationary.

Input File Formats

The **vehicle log file** is a CSV (comma-separate-value) file. Note that CSV files are simply plain text files in which the data values on each line are separated by commas. You can read such files in your Java or Python program like you would read any text file. You can view such files with a text editor (such as Notepad). You can also use Excel to view such files in a nice tabular format.

The vehicle log file is structured as follows. You may not need all the data in the file.

- Line 1 is the title/summary of the log
- Line 2 has the time zone
- Line 3 has the column headers
- Line 4 onwards has the data. The columns are as follows:
    - Trip ID – a unique integer for each "trip"
    - Date/Time – the date and time this line of data was recorded
    - Latitude – the latitude of the data point
    - Longitude – the longitude of the data point
    - Heading – the direction the vehicle is facing
    - Speed (kph) – The speed of the vehicle at this instant, measured in km/hr
    - Ignition – Whether the ignition was turned on
    - Type – "T" to indicate "track/trip" – needed for visualizing using the gpsvisualizer tool (see Tips section)
    - New_Track – "1" for the beginning of a new trip (for the gpsvisualizer tool)

The **interest points file** is also a CSV file, that has the following structure:

- Line 1 has the column headers
- Line 2 onwards has the data. The columns are as follows:
    - Interest Point ID – a unique integer for each gps location of interest
    - Latitude – the latitude of the point of interest
    - Longitude – the longitude of the point of interest
    - Distance – the radius of the region around the point of interest (the "geofence")

Note that the data files provided are simply samples. Your code will be tested on these and other data files with the same format, but which could have fewer or more data points than the sample input files that have been provided. As such, you need to write your code to be general enough to work with any number of rows of data.

What to submit

You should submit:

1. Your code (see specification below)
2. A readme.txt file indicating which tasks your code supports and any additional information needed to run your code.
3. A 1-page document briefly describing your approach, and listing any references (see rules below).

Code Structure

Java

If your solution is in Java, the structure should be as follows:

- There should be a class called `VehicleLogAnalysis`. The constructor of this class should take the vehicle movement log file name as an input parameter.

- The `VehicleLogAnalysis` class should have a method called `task1` (with no parameters) to solve task 1. Calling this method should generate a file called *output-task1.txt* containing the solution (format specified in next section).
- The `VehicleLogAnalysis` class should have a method called `task2` to solve task 2. This method should take as a parameter the name of the file with the GPS interest points. Calling the method should generate a file called *output-task2.txt* containing the solution. If you don't solve Task 2, this method should simply print "Unsupported Task".
- The `VehicleLogAnalysis` class should have a method called `task3` to solve task 3. Calling the method should generate a file called *output-task3.txt* containing the solution. If you don't solve Task 3, this method should simply print "Unsupported Task".

You are free (and are encouraged) to have other classes and methods to ensure a well-organized and modular solution approach. However, you **must** have the `VehicleLogAnalysis` class described above. A skeleton version of this class is provided, along with the evaluation file that will be used to test your code.

Python
If your solution is in Python, the structure should be as follows:

- There should be a file called *VehicleLogAnalysis.py*.
- The *VehicleLogAnalysis.py* file should have a function called `initialize` that takes the vehicle movement log file name as an input parameter.
- The *VehicleLogAnalysis.py* file should have a function called `task1` (with no parameters) to solve task 1. Calling this function should generate a file called *output-task1.txt* containing the solution (format specified in next section).
- The *VehicleLogAnalysis.py* file should have a function called `task2` to solve task 2. This function should take as a parameter the name of the file with the GPS interest points. Calling the function should generate a file called *output-task2.txt* containing the solution. If you don't solve Task 2, this method should simply print "Unsupported Task".
- The *VehicleLogAnalysis.py* file should have a function called `task3` to solve task 3. Calling the method should generate a file called *output-task3.txt* with the solution. If you don't solve Task 3, this method should simply print "Unsupported Task".

You are free (and are encouraged) to have other files & functions, to ensure a well-organized and modular solution approach. However, you **must** have the *VehicleLogAnalysis.py* file described above. A skeleton version of this file is provided, along with the evaluation file that will be used to test your code.

Output File Formats

For task 1, the first line of the output file *output-task1.txt* should have the column headers (latitude and longitude). The second line of the file should have the computed latitude and longitude of the location where the bus is usually parked overnight.

For task 2, the format of the output file *output-task2.txt* should be similar to that of the input "interest points" file, with two additional columns – one with the count of the

number of times the vehicle entered the specified region, and one with the average amount of time the vehicle spent in the region.

For task 3, the first line of the output file *output-task3.txt* should have the number of "unusual" trips that were identified. The trip IDs of these trips should then be listed, one per line.

Rules of the Competition

1. Participation in the competition is by individual Ashesi students. Collaboration is not allowed. Individuals who are not Ashesi students may not participate.
2. Solutions must be coded in Python or Java. Standard built-in classes and libraries can be used (e.g. the math library in Python and classes in the java.util package in Java). No external/third-party libraries can be used in either language.
3. All submitted code must be your own. Code may not be copied from any source.
4. You are allowed to do research if needed (e.g. to figure out how to compute distance between two GPS locations). However, you **must** cite **all** resources you consult.

Evaluation Criteria

We reserve the right to disqualify submissions that do not follow the specified code structure or do not comply with the submission instructions.

Solutions will be evaluated according to the following criteria. The order below will be used to rank the various submissions.

1. Completion & correctness (the number of tasks solved correctly).
2. Efficiency (the time it takes to execute your solution approach). Accommodation will be made for the inherent difference in speed between Java and Python
3. Code formatting and structure: code should be modular, well-structured, well-formatted, and well-commented.
4. Clarity of 1-page documentation of approach

Tips

- If being able to visualize the data will help in your brainstorming process, you can consider using the tool http://www.gpsvisualizer.com/. The vehicle movement log is in a format that can be displayed using this tool. You have also been provided with a file called "COMBINED (for vis).csv". This file contains both the vehicle log and the interest points, for visualization. Simply click the "choose file" button in the box labelled "Get started now!" and click "Map it"