BEAT

Challenge Accepted!

Senior Backend Engineer Take-home Exercise (Golang)

Intro

Beat drivers perform thousand of rides per day. In order to ensure that our passengers always receive the highest quality of service possible, we need to be able to identify mis-behaving drivers that overcharge passengers and block them from our service.

Our servers keep detailed record of drivers' position throughout the progress of each ride. Due to the high volume of daily rides we need to implement an automated way of estimating the fare for each ride so that we may flag suspicious rides for review by our support teams.

Moreover, our drivers sometimes use low-cost devices that frequently report invalid/skewed GPS coordinates to our servers. The fare estimation algorithm should be capable of detecting erroneous coordinates and remove them before attempting to evaluate the ride fare.

In this exercise you are asked to create a **Fare Estimation script.** Its input will consist of a list of tuples of the form (`id_ride, lat, lng, timestamp`) representing the position of the taxi-cab during a ride.

Two consecutive tuples belonging to the same ride form a segment **S**. For each segment, we define the elapsed time **`Δt`** as the absolute difference of the segment endpoint timestamps and the distance covered **`Δs`** as the Haversine distance of the segment endpoint coordinates.

Your first task is to filter the data on the input so that invalid entries are discarded before the fare estimation process begins. Filtering should be performed as follows: consecutive tuples **`p1, p2`** should be used to calculate the segment's speed **U**. If **`U > 100km/h`**, **p2** should be removed from the set.

Once the data has been filtered you should proceed in estimating the fare by **aggregating the individual estimates** for the ride segments using rules tabulated below:

State	Applicable when	Fare amount
MOVING (U `> 10km/h`)	Time of day `(05:00, 00:00]`	`0.74` per km
	Time of day `(00:00, 05:00]`	`1.30` per km
IDLE (U `<= 10km/h`)	Always	`11.90` per hour of idle time

At the start of each ride, a standard 'flag' amount of `1.30` is charged to the ride's fare. The minimum ride fare should be **at least** `3.47`.

Input Data

The sample data file contains one record per line (comma separated values). The input data is guaranteed to contain **continuous** row blocks for each individual ride (i.e. the data will not be multiplexed). In addition, the data is also pre-sorted for you in ascending timestamp order.

Deliverables

We expect you to deliver:

1. A **Golang 1.8+** program that processes input as provided by the sample data, filters out invalid points and produces an output comma separated value text file with the following format:

   ```
   id_ride, fare_estimate
   ```

   Note that your solution should make good use of Golang's concurrency facilities and result in a high-performance script that is space and time efficient in its processing of the data.

2. A brief document with an overview of your design

3. A comprehensive unit and end-to-end test suit for your code.

Notes

- In order to calculate the distance between two (`lat,lng`) pairs you can use the Haversine distance formula.
- Your code should be capable of ingesting and processing large datasets. **Assume we will try to pipe several GBs worth of samples to your script.**
- Follow best-practices and be well documented throughout.

Good luck!