# Snapp!™ Box Challenge

For **Algorithm Arena**
course participants
and
Computer Engineering Students
@ Amirkabir University of Technology

دانشگاه صنعتی امیرکبیر
( پلی تکنیک تهران )

انجمن علمی دانشجویی
دانشکده کامپیوتر
دانشگاه صنعتی امیرکبیر (پلی تکنیک تهران)

Snapp!™ Box

**SnappBox** couriers perform thousands of deliveries per day. To ensure that our customers receive the best service, we need to identify any delivery agents who might need to be overcharging for deliveries or reporting incorrect delivery times. This is crucial for maintaining trust and efficiency in our service.

Our servers maintain a detailed log of each courier's location during their delivery routes. Given the high volume of daily deliveries, we need to develop an automated solution to estimate the delivery fare accurately and flag any suspicious deliveries for review by our support teams.

Additionally, our couriers sometimes use devices that may provide inaccurate GPS data due to hardware limitations. The fare estimation algorithm should detect these erroneous GPS coordinates and filter them out before calculating the delivery fare.

In this exercise, you are required to create a **Delivery Fare Estimation Script**. The input will be a list of tuples in the format **(id_delivery, lat, lng, timestamp)** representing the courier's location during a delivery.

Two consecutive tuples of the same delivery form a segment S. For each segment, the elapsed time **Δt** is defined as the absolute difference between the segment endpoint timestamps, and the distance covered **Δs** is calculated using the Haversine distance between the segment endpoint coordinates.

Your first task is to filter the input data to discard any invalid entries before starting the fare estimation process. Filtering should be done as follows: for consecutive tuples **p1** and **p2**, the segment's speed U is calculated. If **U > 100 km/h**, p2 should be removed from the dataset.

Snapp! Box

Once the data has been filtered, you will proceed to estimate the delivery fare by **aggregating the individual segment** estimates using the rules provided 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 delivery, a standard 'flag' amount of 1.30 is charged.
- The minimum delivery fare should be **at least** 3.47.

## Input Data

The sample data file contains one record per line in CSV format. The input data is guaranteed to contain continuous row blocks for each individual delivery (i.e., the data is not multiplexed) and is pre-sorted in ascending timestamp order.

## Outputs and Results

**1.** A Golang, Java program that processes input as provided by the sample data, filters out invalid points, and produces an output CSV file with the following format:

**id_delivery, fare_estimate**

Your solution should leverage concurrency features to create a high-performance, space- and time-efficient script.

**2.** A brief document outlining your design approach.

**3.** A comprehensive suite of unit and end-to-end tests for your code.

Snapp! Box

## Notes

- Use the Haversine distance formula to calculate the distance between two `(lat, lng)` pairs.
- Test your code with other data to handle edge cases.
- **Your code should be capable of ingesting and processing large datasets. Assume we will test it with several gigabytes of sample data.**
- Follow best practices and ensure your code is well-documented.

**Good luck!**

Snapp! Box