VORTO

Vorto Algorithmic Challenge

For this challenge, you will submit a program that solves a version of the Vehicle Routing Problem (VRP).

The VRP specifies a set of loads to be completed efficiently by an unbounded number of drivers.

Each load has a pickup location and a dropoff location, each specified by a Cartesian point. A driver completes a load by driving to the pickup location, picking up the load, driving to the dropoff, and dropping off the load. The time required to drive from one point to another, in minutes, is the Euclidean distance between them. That is, to drive from `(x1, y1)` to `(x2, y2)` takes `sqrt((x2-x1)^2 + (y2-y1)^2)` minutes.

As an example, suppose a driver located at `(0,0)` starts a load that picks up at `(50,50)` and delivers at `(100,100)`. This would take `2*sqrt(2*50^2) = ~141.42` minutes of drive time to complete: `sqrt((50-0)^2 + (50-0)^2)` minutes to drive to the pickup, and `sqrt((100-50)^2 + (100-50)^2)` minutes to the dropoff.

Each driver starts and ends his shift at a depot located at `(0,0)`. A driver may complete multiple loads on his shift, but may not exceed 12 hours of total drive time. That is, the total Euclidean distance of completing all his loads, including the return to `(0,0)`, must be less than `12*60`.

A VRP solution contains a list of drivers, each of which has an ordered list of loads to be completed. All loads must be assigned to a driver.

The total cost of a solution is given by the formula:

```
total_cost = 500*number_of_drivers + total_number_of_driven_minutes
```

A good program will produce a solution with a *low total cost*, but does not take too long to run (see Evaluation section below).

Program Requirements

You must provide a program that takes a text file path describing a VRP as a command line argument, and writes a solution to stdout.

We will accept solutions written in Go, Javascript, Typescript, Python, Java, C or C++. Please make sure your solution is version controlled using Git. **All work must be your own**. You may research and implement algorithms you find online (papers, blog posts, etc.), but you must provide references to all external sources. Please avoid using Google's OR-Tools.

The problem input contains a list of loads. Each load is formatted as an id followed by pickup and dropoff locations in (x,y) floating point coordinates. An example input with four loads is:

Unset
```
loadNumber pickup dropoff
1 (-50.1,80.0) (90.1,12.2)
2 (-24.5,-19.2) (98.5,1,8)
3 (0.3,8.9) (40.9,55.0)
4 (5.3,-61.1) (77.8,-5.4)
```

Your program must write a solution to stdout. The solution should list, on separate lines, each driver's ordered list of loads as a schedule. An example solution to the above problem could be:

Unset
```
[1]
[4,2]
[3]
```

This solution means one driver does load 1; another driver does load 4 followed by load 2; and a final driver does load 3. Do not print anything else to stdout in your submission.

For example, if your solution is a python script called "mySubmission.py", then your program should run with the following command:

```
python mySubmission.py {path_to_problem}
```

This should load the problem described in the file `{path_to_problem}`, run your algorithm, and print a solution to stdout in the correct format.

We have provided a python script to help evaluate your code. This will run your program on each provided training problem, check for solution validity, and print your score. We highly recommend you use it to guide your development. See `evaluationReadMe.txt` for instructions.

Evaluation

Upon completion email devgroup@vorto.ai and whoever sent you this challenge the source code for your solution along with instructions to build and run - preferably uploaded to a public Github repository.

Your solution will be evaluated by the following criteria, with an emphasis on minimizing total cost:

1. **Low total cost**
   We will run your program on a set of test problems. You will not have access to the test set, though we will provide a comparable set for you to test on during development. The solution quality is determined primarily by the total cost formula shown above.

2. **Run time**
   The program should not take longer than 30 seconds to run on a typical laptop for a problem containing up to 200 loads.

3. **Ease of deployment**
   Your program should be easy for one of our developers to get running.

4. **Code simplicity and readability**
   We'll do a subjective assessment of your codebase for simplicity and readability.

Problem Constraints

- All problems we provide will be solvable. That is, all loads are possible to complete within the duration of one 12-hour shift. Your program does not have to assess problem feasibility.
- No problem will contain more than 200 loads.
- This coding challenge is designed to assess your individual problem-solving skills and creativity. The use of AI-based tools, including ChatGPT or similar AI services, for assistance *in writing code* for this challenge is strictly prohibited. Your submission should be solely your own work. Engaging in such practices will result in immediate disqualification from the hiring process.