CS 150: Project I - Tesla Charging Stations

Version as of: 09:01 Saturday 9th September, 2017
Due: 11:55pm, Saturday October 14, 2017

1 Introduction

Your first project is to construct a program to determine how many charging stations would be needed in Northampton county to handle the needs of owners of Tesla vehicles. The goal of the project is to determine the number, density and size of the charging stations based on the number of Tesla vehicles present in Northampton county. Of course, you will have to make some general simplifying assumptions to have some reasonable way of calculating an answer.

2 Project Description

The goal of the program is to determine (approximately) the number and size (number of charging points per station) of charging stations necessary to support Tesla (or any electric) vehicles. One of the uses of the program is to also to project the growth of the number of charging stations as the number of electric vehicles grows - a form of complexity calculation. To do this, you have to make many simplifying assumptions about the problem. These assumptions will allow you to write a simple simulation program that will help you generate the answers you need. The program will simulate trips taken by the vehicles over a period of time and generalize from that data.

Simplifying Assumptions

The assumptions that we make to simplify the program are:

1. the frequency and length of trips is approximated by a Gaussian (normal) distribution
2. behavior of vehicles (frequency and length of trips) is not affected by the time of day
3. average distance of a vehicle from a charging station is approximated by a Gaussian (normal) distribution
4. all charging stations are identical
5. all Tesla vehicles are identical

The information that you will need for the simulation includes:

1. number of Teslas in the county,
2. average speed of a Tesla (distribution, median, *etc.,*)
3. distance traveled per unit charge of the battery
4. how many miles a Tesla will travel per trip (distribution, median, *etc.,*),
5. the time between trips (distribution, median , *etc.,*),
6. the threshold of battery charge before the driver will look to charge the battery,
7. the time to charge a unit of the Tesla battery,
8. the maximum capacity of a battery (total number of units),
9. distance to a charging station (distribution, median, *etc.,*),
10. number of charging stations,
11. number of charging points at each charging station
12. number of minutes in a time step

To see how to generate random numbers with a Gaussian (normal) distribution see `http://www.javapractices.com/topic/TopicAction.do?Id=62`.

Rules Governing Your Program

The assumptions underlying the program are fairly straightforward and are:

1. there are no charging stations in owners homes,
2. when a car exhausts its battery, it is removed from the simulation,
3. each time unit of the simulation represents a minute of elapsed time (wall clock time),
4. each car has a unique id (auto-generated),
5. each charging station has a unique id (auto-generated),
6. each charging point has a unique id unique to the station (auto-generated),

2.1 Program Behavior

We are going to simulate the behavior of the cars and the stations for some number of time steps. At each time step:

- if the vehicle is at home, determine if it will start a trip.
- if the vehicle is currently on a trip and not at a charging station:

- check the battery level. If it is above the threshold, then increment the miles traveled by the appropriate amount. If not, find the nearest charging station that can be reached given the number of miles that can be traveled in one time step. If no charging stations are within reach, continue on the journey. If the battery level is zero, then simulation for this vehicle terminates.
- if the trip can be completed in this time step, complete the trip and place the vehicle at home and not on a trip.
- the vehicle is at a charging station and not at a charging point:
  - if there is a charging point available and the vehicle is the first in line, move up to the charging point. All other vehicles in the line move up one position.
  - otherwise, remain in line.
- if the vehicle is at a charging station and charging point:
  if the battery is full, leave the charging station and continue on the journey.
  otherwise, continue charging the battery and increment the level by the specified amount for one time step.

At each step of the simulation, your program will accept and execute the following (one letter) commands from the console.

- *p*: print the state of the simulation. Print the charge of each car, the cars at each charging station,
- *t*: print the state of each car (h - home, t - traveling, w - at a charging station),
- *e*: print the state of each charging station, showing the cars (with ids) at each charging point (with ids)
- *s <integer n >*: continue n steps forward in the simulation,
- *c*: continue to the end of the simulation (number of time steps specified) without stopping,
- *x*: exit the simulation **now** and print all relevant statistics

Initially, all cars will be at home and so there will be some time before the program reaches a state when you can start collecting data. We also simplify the problem by having the same car usage profile throughout the program, *i.e.*, we will not take rush hour or weekdays vs weekends or day vs night into account.

The program/simulation will end if any of the following conditions are true:

- the program has executed the number of simulation steps specified at the beginning of the simulation,
- all the cars have exhausted their batteries,

2.2 Program Inputs

The following inputs will be through the $args[]$ array in $main()$ in the following order:

- the name of the config file
- the number of time steps for the simulation
- the config file will contain the information described in Section 2. The format of the file is that each line is a keyword/value pair. An example file is available on moodle in the section for Project 1.

Project Constraints

The following constraints apply to the project:

1. The project is to be completed individually. The only person you can consult is the instructor.
2. Each configuration of parameters should be run *at least* 5 times with different random seeds to obtain an ”average” value.
3. You are only to use containers from the Java Collections Framework. The only exception is when you use a method from an API that requires the use of arrays (e.g., main()).

3 Simplifications

You can simplify your project in one (or more) of the following ways:

1. limit the number of charging stations to 2
2. limit the number of charging points to 1
3. fixed time between trips

4 Report

Your simulation and report should try to answer questions (you should design your own questions) like the following:

1. the (average/total) additional miles driven to charge the Tesla,
2. the (average/total) number of cars that exhaust their battery,
3. the (average/total) time spent waiting at a charging station,
4. the (average/total) number of cars that waited more than the desired maximum waiting time,
5. how long (number of steps) before the simulation reaches a reasonable state that the data is valid,
6. what is a reasonable threshold level for the battery when the driver should look to recharge it?
7. if we kept the number of charging points at a

Grading

Your project will be graded on the following criteria (assuming the program compiles and runs):

1. ”the story” (2.5%)
2. class diagrams (2.5%)

3. correctness and functionality of the program (35%)
4. documentation (methods and classes) (10%)
5. unit testing (15%)
6. object oriented design (15%)
7. report (20% - 10% for the draft and 10% for the final)

5 Submission

The project submission is in 4 parts:

1. (2.5 pts) - due Wednesday September 27: the "story" of your program
2. (2.5 pts) - due Wednesday Octpber 4: class diagrams for your program
3. (85 pts) - due Saturday October 14: the rest including a draft report
4. (10 pts) - due Saturday October 22: the final report

s

Your submission for Part 3 will be composed of the following:

1. source files (*.java) that are commented and have javadoc directives
2. test files, one test file per class
3. a README.txt file that contains instructions on how to run your program
4. a draft of the project report (see project report guidelines `www.cs.lafayette.edu/~liew/courses/cs150/writeup-guidelines.pdf`) - 50% of the report grade is assigned to this. The remaining 50% will be given to a final report. The final report will be due several days after the draft is corrected and returned.