Project 1, Traffic Light Simulator

Due: Please check due date on BlackBoard

Objectives

The objective of this programming assignment is to have you practice implementing a new data structure and also to gain some experience using a data structure from the Java Collections API.

Introduction

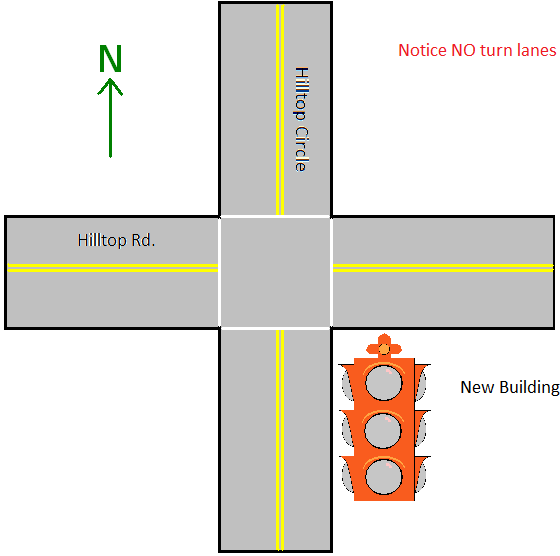
While playing Grand Theft Auto V, the unnamed instructor noticed that in many cases the traffic lights were not in sync with traffic movement. On many occasions, traffic did not move when they had the green light, or an errant vehicle would come out into the intersection when they had a red light. For a game that grossed 1 Billion dollars on the first day, this is sad. In reality, this should be easy to set up correctly with queues.

Intersection test setup

In many of the smaller 4 way intersections in GTA, there are no turn lanes (left or right), which will make this easier to code. For our test, we will use the Hilltop Circle and Hilltop Rd. as our example.

<https://maps.google.com/maps?q=39.256265,-76.716508&ll=39.256251,-76.716453&spn=0.000662,0.000871&num=1&t=h&z=21&iwloc=near>

But we will represent the intersection with the simple image below.



Four sensors (that look like cameras) are pointed in each direction to monitor if traffic exists on that side of the intersection.

Traffic Light Priority Setup

Our traffic light simulation will work in this fashion:

North/South

1. Primary – meaning we want this to be “green” as much as possible. If East/West does not have traffic, then North/South will STAY “green”
2. Will always be “green” together
3. Minimally “green” for 30 seconds each time it turns green
4. There is no maximum value for how long it stays green
5. The simulation will ALWAYS start with North/South being “green”

East/West

1. Maximally “green” for 30 seconds
2. Minimally “green” for 10 seconds
3. Will turn “red” early if no traffic on either East/West side

We will pretend the “yellow” light is “green” (which sounds about right) and ignore “yellow” as a whole. We will also assume there is no delay in car movement after their light turns green (wishful thinking). There are also NO turn lanes.

Flow rate

Each side will have a flow rate of cars and trucks. These are two separate values.To keep things simple, vehicle arrival time (when they should show up in the queue) will always be divisible by an even value in 60.

* 1. for example, the sample flow rates will be 1, 2, 3, 5, 10, 15, 20, 30.
  2. these are ALL per minute
  3. hence, a flow rate of 4 cars per minute will introduce a car every 15 seconds. NOT STARTING AT 0.

The flow rate (given by the file) will not change during your 2 minute simulation.

You will also be given a traffic flow data from a file. It will look as simple as below:

N: 3 3

S: 10 1

E: 15 3

W: 10 3

First token in the line is which direction they are bound for, second is the number of cars PER MINUTE, and the third token is the heavy trucks PER SIMULATED MINUTE. (You are not using the TIME function.) To make it easy, the number of cars per minute should be injected into the appropriate queue in even intervals. (2 cars per minute, then a car entered every 30 seconds, starting at second #1) The test file will not have errors, you will not need to validate.

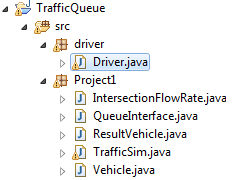
Vehicle Setup

It will take some vehicles longer to cross the intersection. You will be concerned with 2 types of vehicles, a car (1 second) and heavy trucks (2 seconds). Those vehicles will also have to track how long they waited for crossing the intersection. You will need to create a class “Vehicle” that will contain some of the data needed for this project. Also, there will be 2 cars in EACH side of the intersection when the simulation starts.

Coding and Implementation Requirements

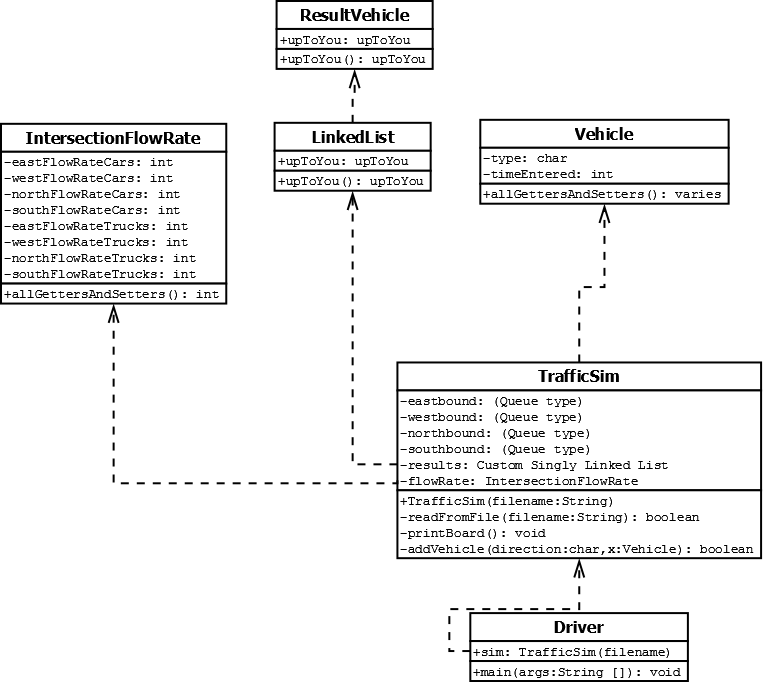
*Note: Running time is one of the most important considerations in the implementation of a data structure. Programs that produce the desired output but exceed the required running times are considered wrong implementations and will receive substantial deductions during grading.*

1. You will be required to use one of Java’s general-purpose Queue implementations (there are several you can use) for the intersection to hold the various vehicles. Failure to do so, or copying the data structure into another type (Linked List, Array) will have significant consequences.
2. You will also be required to create your OWN singly Linked List to store all Vehicles that exited the intersection to do some calculations described below. Failure to do so, or copying the data structure into another type (Linked List, Array) will have significant consequences.
3. The run time for the entire project should not be anything more than O(n).
4. Use a VERY basic class called *IntersectionFlowRate* that will contain the values retrieved from the input file for eastFlowRateCars, eastFlowRateTrucks, westFlowRateCars, westFlowRateTrucks, etc…
5. Another basic class named *Vehicle* that will contain the type of vehicle used (truck or car) and the time this vehicle entered the simulation.
6. Your main base class will be named *TrafficSim*. It will contain:
   1. the **queues** named *northbound, southbound, eastbound and westbound*
   2. a constructor that will accept the parameter of a filename to load flow data and
      1. initialize the queues and custom singly linked list
      2. call the function readFromFile to gather flow rates. ([sample File IO notes here](http://userpages.umbc.edu/~slupoli/notes/Java/FileIONotes.docx))
      3. call printBoard() at least once to show the start of the simulation
   3. In order to determine the results of the current simulation, you will also be required to STORE all vehicles that successfully passed the intersection to a SINGLELY Linked List of YOUR OWN DESIGN called “results”. You are not to use Java’s general-purpose Queue or Linked Lists implementations for this queue.
   4. a method readFromFile(filename) which reads the flow rate from the input files.
   5. a method printBoard() which prints the elaborate second by second account of the simulation described below
   6. a method addVehicle(char direction, Vehicle x) which will add the Vehicle to a specific queue depending on the direction.
7. Example input and output files are provided for you. Consider each set of files, input1.txt is an example flow rate file that should produce the results output1.txt, and so on. Please notice in the sample output, **WHY the traffic light changed is displayed**. This is another debugging tool to help.
8. Your traffic light class called TrafficSim and must be accessible from a main program in a different package. You can check that your code compiles correctly with this sample **main** (has the main() ) program: [Driver.java](http://userpages.umbc.edu/~slupoli/notes/DataStructures/projects/TrafficLightSimulatorQueuesS14/code/Driver.java). This test program must be placed in a separate directory named driver (since it belongs to the driver package). Your code must compile with Driver.java without alteration. Your overall setup (in Eclipse) should look like this.



1. Finally, your custom singly Linked List will be named LinkedList.java. We are leaving the implementation to you, **BUT**, you will need a new node to store the vehicles that did get through the intersection.

A UML diagram below may help you in the right direction.



To learn how to read this, visit: <http://www.youtube.com/watch?v=RdoXdreURLk>

Simulation Setup

The simulation will be tracking:

1. Traffic movement (displayed, explained below)
2. Will show the intersection after each SECOND is a display
3. Again, the simulation will start with 2 cars in EACH side (2 in eastbound, 2 in westbound, etc…)
4. The simulation will run for a total of 2 minutes (120 seconds)
   1. remove vehicles from “green” queues
   2. add new vehicles to all sides (if any)
      1. cars are always added first, if a car and truck “arrive” at the same time in a queue
   3. **it must be in this order to be able to replicate the examples given to you**
5. display results of the current setup with flow rate and green signal lengths

Result Setup and Custom Linked List

While your simulation is running, for debugging and grading purposes, the intersection will be printed to the screen after each “second” to show traffic movement. Think of it as a snap shot of the intersection after each second. Notice, the display is a simple set of characters. The explanation of each symbol is described below. Your resulting display should look like below with **VERY FEW** alterations. There should be 120 of these displays, separated by ONE blank line each time you run your simulation. Also notice that you will see the truck “t” twice, once when it JUST arrives and again after taking a second to get across. As ALL vehicles are being removed, they should be placed into ANOTHER CUSTOM singly linked list (that does not use any Pre-Built Java Linked List/Queue datatypes) called “results” that is holding all of the information that you need for the results below. The result queue will NOT have those still in the intersection after 120 seconds, or who is still left in the queues.

The intersection snapshot should look EXACTLY like below.

|  |  |  |
| --- | --- | --- |
| . . . | | The text to the left will be repeated 121 (from 0 to 120) times, with different values in the queues. |
|  | | |
|  | In the display, the MAXIMUM number of physical vehicles it can display is 6 for each side. (Orange) “c” is for a car is about to enter the intersection. The “x”s after are other vehicles waiting after the car. Also, since it is a queue, the vehicle type cannot be determined UNTIL is is the first position.  Also notice the symbols SB, EB, etc… The numbers beside them represent the TOTAL number of vehicles in the queue. This can be more than 6!  Notice (green), there could be more vehicles, but will only display the 6.  (Red) Also, you are to display what type of vehicle is at the head of each queue. This will also help you decide to pop, or wait one more sec for the truck, since it takes 2 seconds to clear the intersection. | |

Also, at the end of 120 seconds, you will print to the screen (IN ORDER) this information as shown below:

* total number of vehicles that went through the intersection (no matter type)
* Number of cars and trucks that went through the intersection
* Average wait time

The final results are:

The number of vehicles that passed through the intersection is: 84

The number of cars that passed through the intersection is: 70

The number of trucks that passed through the intersection is: 14

The average wait time for this intersection is: 11.309524

## What to Submit

Follow the [course project submission procedures](http://www.csee.umbc.edu/courses/undergraduate/341/fall13/projects/submission.shtml). You should copy over all of your Java source code with .java files in their own directories which are in turn under the src directory. You must also supply an Ant build file.

Make sure that your code is in the ~/cs341proj/proj1/ directory and not in a subdirectory of ~/cs341proj/proj1/. In particular, the following Unix commands should work. (Check this.)

cd ~/cs341proj/proj1

ant compile

ant run

ant clean

Don't forget the Project Submission requirements shown online!! One hint, **after you submit**, if you type:

ls ~/cs341proj/proj1/

and you see a bunch of .java files, this is WRONG. You should see:

build.xml src

instead. The java programs must be in package directories under the src directory. Your submissions will be compiled by a script. The script will go to your proj1 directory and run ant. This is required. You will be severely penalized if you do not follow the submission instructions.

Addendum

None yet!