# CS2124 Data Structures
## Project 2: Simulating Traffic

**Reminder:** This project is like an exam. The program you submit should be the work of only you and, optionally, one other partner. You are not allowed to read, copy, or rewrite the solutions written by anyone other than your partner (including solutions from previous terms or from other sources such as the internet). Copying another person's code, writing code for someone else, or allowing another to copy your code are cheating, and can result in a grade of zero for all parties. If you are in doubt whether an activity is permitted collaboration or cheating, ask the instructor.

Every instance of cheating will be reported to UTSA's Student Conduct and Community Standards office (SCCS). At minimum, this will result in a zero for the project/exam and may result in failing the class.

### 1 Project files

For this project you'll be simulating traffic in city. Specifically you need to complete "trafficSimulator.c", "trafficSimulator.h", "road.c", "road.h", "car.c", "car.h", "event.c", and "event.h". You can change any of the other files or create new files but you will need to also submit them as well as your updated makefile. Here is brief description of the functionality of your simulation:

### 2 Intersections, Cars, and Traffic Lights

#### Intersections and Roads

- Intersections are the vertices of your graph.
- Each vertex is represented by a unique integer value.
  - These range from 0 to $size - 1$ where $size$ is the number of vertices.
- The roads connecting them are edges of graph.
- Each directed edge is represented by a triple of integers.
  - The first integer is the starting vertex.
  - The second integer is the ending vertex.
  - The third integer is the weight of the edge (i.e, the length of this road).
- Cars traverse the edge based on order of arrival (i.e., no passing). You should use an array to represents the current contents of the road. During each time step every car with an empty space in front of it moves forward one position. Example:

Suppose that − is an empty space on the road and the numbers represent 3 cars on the road.

| | | | | |
|---|---|---|---|---|
| − | 1 | − | 2 | 3 |

After one time step only cars 1 and 2 are able to move forward. Car 3 is blocked from moving due to car 2.

| 1 | – | 2 | – | 3 |
|---|---|---|---|---|

**Traffic Lights and Road Length**

- An intersection permits cars from its adjacent roads to pass through at a time.
- Each road has traffic light associated with it. When the light is green, the car on the front of the road may attempt to pass through the intersection towards its destination.
- The times in which the light is green/red and allows traffic through is specified in the input (we omit yellow lights for the sake of simplicity). The light operates on a cycle and repeats the specified pattern for the duration of the simulation. This is specified with the following three int inputs (see also Section 5):
  - `<green on>` - The cycle the light turns green (light starts as red)
  - `<green off>` - The cycle the light turns back to red
  - `<cycle resets>` - The cycle resets goes back to 0
- The order the roads should be processed in is same as the order in which they were added to the graph. **Hint:** It will be useful to store the roads in an array based on this order.
- The length of a road also denotes the maximum number of cars which can be on it.
- A car can only pass through the intersection if the next road on its shortest path has an empty space on the end of its car array.

**Cars**

- The destination of each car is an intersection.
- A car is removed from the simulator once it moves off the front of the road at its destination. As with any move through an intersection this requires the traffic light to be green.
- Cars always follow a shortest path to their destination.
  - Note that this shortest path is based on the lengths of the roads and **not** on how many cars are currently on the roads.
  - You should call the graph.c function "getNextOnShortestPath" to find the next intersection on a shortest path.
- You will want to track the number of time steps the car took to reach its destination in order to report your results at the end of the simulation.

## 3 Events

It is recommended that you store the events in a priority queue based on the time step it is supposed to occur on.

**Event - Adding Cars**

- The input file will also specify time steps in which more cars should enter the simulation.
- Initially, those cars should be stored in event struct. A dynamically array would work but an even simpler approach is to just create and store them in a queue in the event struct.
- Once the specified time step occurs, these cars should be added to the end of a waiting queue associated with the specified edge (**Hint:** the merge function in queue.c may

come in handy here). For each time step remove the first car in the queue and place it at the end of the road array of the edge if possible.

- Print the following on the time step this event occurs:
  - "CYCLE $X$ - ADD_CAR_EVENT - Cars enqueued on road from $Y$ to $Z$"
  - $X$ is the time step this event occurred on. The road of the event is from $Y$ to $Z$.

**Event - Printing Road Contents**

- The input file will also specify time steps in which the contents of all of your roads should be printed. See provided output for examples of what this will look like.
- Print the following on the time step this event occurs:
  - "CYCLE $X$ - PRINT_ROADS_EVENT - Current contents of the roads:"
  - $X$ is the time step this event occurred on.
- For each car on the road print its destination. For empty spaces on the road print "–".
- At the end of the above line also print either "(GREEN light)" or "(RED light)" to indicate the current state of the traffic light.
- Example format for cycle 8 of "data-Merge.txt":

```
CYCLE 8 - PRINT_ROADS_EVENT - Current contents of the roads:
Cars on the road from 1 to 0:
- 5 - 6 - 6 (GREEN light)
Cars on the road from 2 to 1:
5 5 - (RED light)
Cars on the road from 3 to 1:
6 - - (GREEN light)
Cars on the road from 4 to 1:
7 7 7 (GREEN light)
Cars on the road from 0 to 5:
- - - (GREEN light)
Cars on the road from 0 to 6:
- - - (GREEN light)
Cars on the road from 0 to 7:
- - - (GREEN light)
```

## 4 Output

- Once a car reaches its destination you should print the following:
  - "CYCLE $W$ - Car successfully traveled from $X$ to $Y$ in $Z$ time steps."
  - $W$ is the time step the car reached its destination. $X$ and $Y$ are respectively the starting intersection and destination of this car. $Z$ is the number of time steps since the car was added to the simulation.
- The simulation ends once all of the cars reach their destination and there are no more events left to process. You should print the following:
  - "Average number of time steps to the reach their destination is $X$."
  - "Maximum number of time steps to the reach their destination is $Y$."
  - $X$ is average number of time steps taken by the cars to reach their destination. $Y$ is maximum time steps taken by any car to reach its destination.
- The above time step calculations should be based on when the car entered the waiting queue from the add car event. It will be handy to store this time step in the cars

associated with the event.

- Alternatively, if the simulation goes through a full cycle of every traffic light with no car being able to move then the city is in gridlock and the simulation halts (**Hint:** to determine this it will helpful to track the longest traffic light cycle as well as when a car was last able to move).
  - Output "CYCLE $Z$ - Gridlock has been detected."
  - $Z$ is the current time step.

## 5 Input File Format

```
<size = # of vertices> <# of edges>

//Incoming roads for vertex 0
<number of incoming roads>
<FROM: 1st vertex> <length>      <green on> <green off> <cycle resets>
<FROM: 2nd vertex> <length>      <green on> <green off> <cycle resets>
...
<FROM: last vertex> <length>     <green on> <green off> <cycle resets>

//Incoming roads for vertex 1
<number of incoming roads>
<FROM: 1st vertex> <length>      <green on> <green off> <cycle resets>
<FROM: 2nd vertex> <length>      <green on> <green off> <cycle resets>
...
<FROM: last vertex> <length>     <green on> <green off> <cycle resets>

...

//Incoming roads for vertex size-1
<number of incoming roads>
<FROM: 1st vertex> <length>      <green on> <green off> <cycle resets>
<FROM: 2nd vertex> <length>      <green on> <green off> <cycle resets>
...
<FROM: last vertex> <length>     <green on> <green off> <cycle resets>

<# of "add car" commands>

//1st add car command
<"from" of starting edge> <"to" of starting edge> <time step to perform "add car" on>
<number of cars to add to this edge>
<dest. vertex of 1st car> <dest. vertex of 2nd car> ... <dest. vertex of last car>

//2nd add car command
<"from" of starting edge> <"to" of starting edge> <time step to perform "add car" on>
<number of cars to add to this edge>
<dest. vertex of 1st car> <dest. vertex of 2nd car> ... <dest. vertex of last car>

...

//last add car command
```

```
<"from" of starting edge> <"to" of starting edge> <time step to perform "add car" on>
<number of cars to add to this edge>
<dest. vertex of 1st car> <dest. vertex of 2nd car> ... <dest. vertex of last car>


<# of "print road" commands>
<time step to print roads> <time step to print roads> ... <time step to print roads>
```

## 6 Simulation

Below is the order of operations that your program should go through for each time step:

(1) Dequeue and execute any and all events associated with the current time step.
(2) For each road, attempt to move the car on the front of the road through the intersection and to the end of the next road on its shortest path. Cars that have reached their destination intersection are removed from the simulation.
(3) For each road, Attempt to move a car from the add car queue for that road onto the last position on the road.
(4) For each road, cars which had empty spaces in front of them move forward one space (do not move cars which were just moved in either of the previous steps).

Repeat the above until all events have finished and either all cars have reached their destination or gridlock has occurred.

## 7 Deliverables:

Your solution should be submitted as "trafficSimulator.c", "trafficSimulator.h', "road.c", "road.h", "car.c", "car.h", "event.c", and "event.h". You can skip files you didn't modify but be sure to double check the materials you submitted are the correct versions. If you created any other files to solve the problem be sure to submit them (including possibly a new "makefile").

To receive full credit, your code must compile and execute. You should use valgrind to ensure that you do not have any memory leaks.