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 **at most a grade of a C for the course**.

1 Project files

For this project you'll be simulating traffic in city. Specifically you need to complete "trafficSimulator.c". The places where you need to change the code are marked with TODOs. It is not recommended but, if you really want, you can change any of the files (other than "driver.c") or create new files but you will need to also submit them as well as your updated makefile.

2 Overview

The program you're writing will simulate cars traveling in a road network. Here is a quick overview of the road network:

- The road network is represented using a graph.
- Roads are represented as edges in your graph (see Section 3). Roads connect pairs of intersections. All roads are one way and have only a single lane (i.e. the cars all travel in the same direction and cannot pass each other). Each road has a traffic light associated with it. The traffic lights cycle between red and green (i.e. no yellow lights).
- Intersections are represented as vertices in your graph (see Section 3).
- Cars always take the shortest path to their destination (based on length of the roads it traverses). Cars are added to a starting road with Add Car Events (see Section 4).

Your ultimate goal is to find the average number of steps it takes for a car to reach it's destination in the road network as well as the maximum number of steps it took for a car to reach its destination. These numbers are a measure of the quality of the road network. This allows a user to quickly test the effects of building a new road on quality the road network.

3 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 it.
- 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 time step and repeats the specified pattern for the duration of the simulation. This is specified with the following three int inputs (see also Section 6):
    · `<green on>` - The light turns green after this many time steps (light starts as red)
    · `<green off>` - The light turns back to red after this many time steps
    · `<cycle resets>` - The light cycle resets (i.e. repeat the above checks).
- Example light cycle for $GreenOn = 1$, $GreenOff = 4$, and $CycleReset = 5$:

Time Step:	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Light State:	R	G	G	G	R	R	G	G	G	R	R	G	G	...

- 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. Otherwise, it remains at the front of its current road.

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.

4 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 the queue of 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 mergeQueues 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:
  - "ADD_CAR_EVENT - Cars enqueued on road from Y to Z"
  - The road of the event is from Y to Z.

Event - Closing Roads due to an Unresolved Accident

- The input file will specify a time step, a road, and a duration. Along with creating a "ROAD_ACCIDENT_EVENT" you will also create a "ROAD_RESOLVED_EVENT" which will occur after the specified time duration has passed since the time step of this event.
- Once the specified time step occurs, an unresolved accident is added to the road. As long as the number of unresolved accidents on a road is > 0, no car can make a move involving that road (including moving onto/off of that road).
- Print the following on the time step this event occurs:
  - "ROAD_ACCIDENT_EVENT - Adding accident to road from Y to Z"
  - The road of the event is from Y to Z.

Event - Resolving an Accident

- This event will be created a companion to a "ROAD_ACCIDENT_EVENT". It represents the time at which that accident on the road is resolved.
- Once the specified time step occurs, an unresolved accident is removed from the road.
- Print the following on the time step this event occurs:
  - "ROAD_RESOLVED_EVENT - Removing accident from road from Y to Z"
  - The road of the event is from Y to Z.

5 Output

- As you read the input file, print out the roads and events (example for “data-Merge2.txt”):

```
The roads:
Created road from 1 to 0 with length 6 (green=1; red=4; reset=5).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  R  G  G  G  R  R  G  G  G  R  R  G  G  G  R  R  G  G  G  R  R  G  G  G  R ...

Created road from 2 to 1 with length 3 (green=0; red=5; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  G  G  G  G  G  R  R  R  R  R  G  G  G  G  G  R  R  R  R  R  G  G  G  G  G ...

Created road from 3 to 1 with length 3 (green=5; red=9; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  R  R  R  R  R  G  G  G  G  R  R  R  R  R  R  G  G  G  G  R  R  R  R  R  R ...

Created road from 4 to 1 with length 3 (green=0; red=9; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  G  G  G  G  G  G  G  G  G  R  G  G  G  G  G  G  G  G  G  R  G  G  G  G  G ...

Created road from 0 to 5 with length 3 (green=8; red=9; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R ...

Created road from 0 to 6 with length 3 (green=8; red=9; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R ...

Created road from 0 to 7 with length 3 (green=8; red=9; reset=10).
Cycle number:  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 ...
Light state :  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R  R  R  R  G  R  R  R  R  R  R ...


The add car events:
ADD_CAR_EVENT       for time step   0 on road from   2 to   1.
Destinations of added cars: 5, 5, 5
ADD_CAR_EVENT       for time step   0 on road from   3 to   1.
Destinations of added cars: 6, 6, 6
ADD_CAR_EVENT       for time step   0 on road from   4 to   1.
Destinations of added cars: 7, 7, 7
ADD_CAR_EVENT       for time step   5 on road from   1 to   0.
Destinations of added cars: 0, 0, 0


The road accident/resolved events:
ROAD_ACCIDENT_EVENT for time step  23 on road from   1 to   0.
ROAD_RESOLVED_EVENT for time step  27 on road from   1 to   0.
ROAD_ACCIDENT_EVENT for time step  25 on road from   1 to   0.
ROAD_RESOLVED_EVENT for time step  28 on road from   1 to   0.
ROAD_ACCIDENT_EVENT for time step  13 on road from   0 to   5.
```

```
ROAD_RESOLVED_EVENT for time step  21 on road from   0 to   5.
```

- Each time step, you should print the contents of each road using the provided function "printRoadContents".
- Once a car reaches its destination you should print the following:
  - "Car successfully traveled from X to Y in Z time steps."
  - 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 is 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.

6 Input File Format

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

//1st road
<FROM: vertex> <TO: vertex> <length>      <green on> <green off> <cycle resets>

//2nd road
<FROM: vertex> <TO: vertex> <length>      <green on> <green off> <cycle resets>

...

//last road (one for each edge in the graph)
<FROM: vertex> <TO: vertex> <length>     <green on> <green off> <cycle resets>

<# of "ADD_CAR_EVENT" events>

//1st add car event
<"from" of edge> <"to" of 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 event
<"from" of edge> <"to" of 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 event
<"from" of edge> <"to" of 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>
```

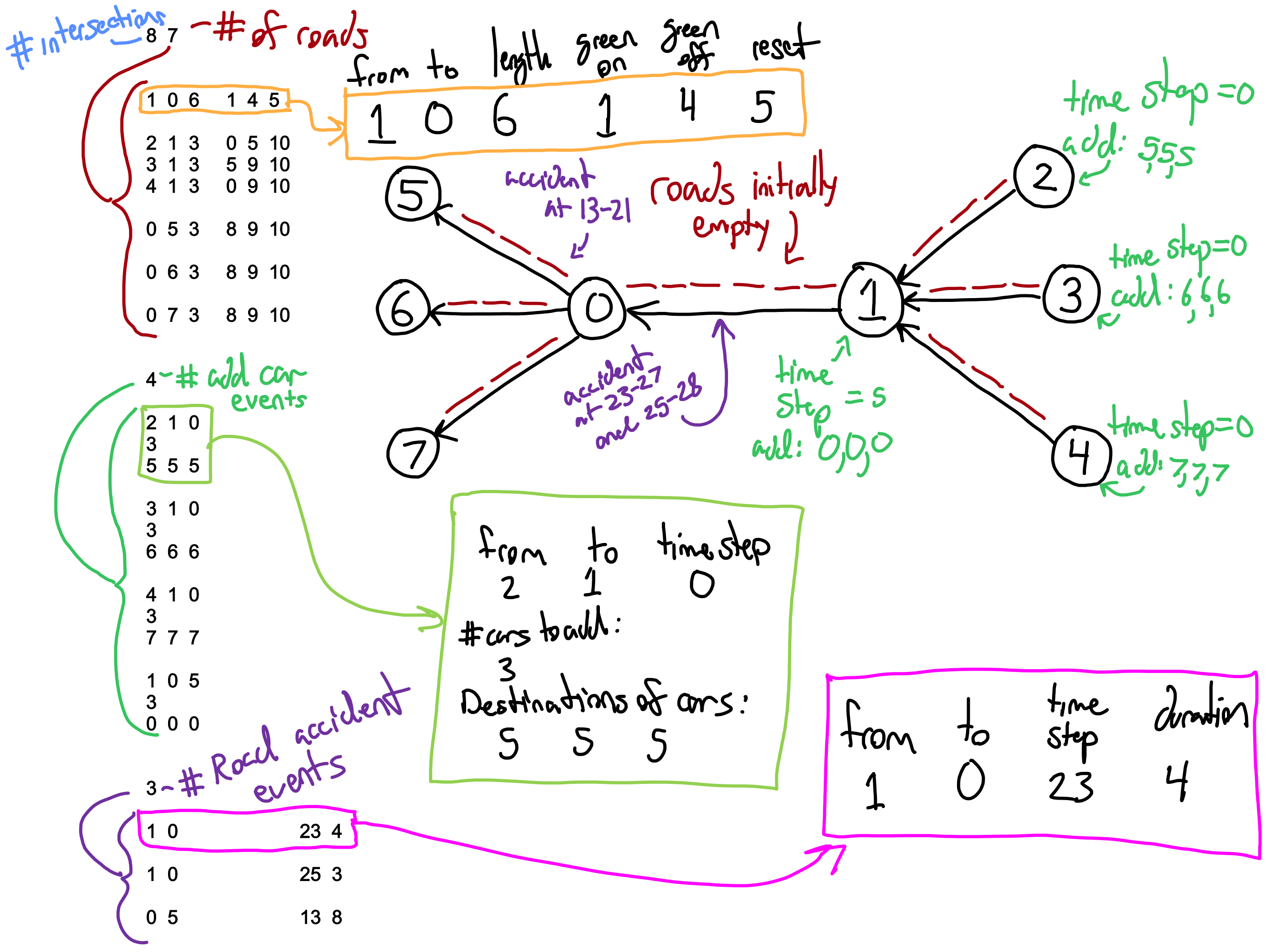
```
<# of "ROAD_ACCIDENT_EVENT" events>

//1st road accident event
<"from" of edge> <"to" of edge> <time step to add accident on> <duration of accident>

//2nd road accident event
<"from" of edge> <"to" of edge> <time step to add accident on> <duration of accident>

...

//last road accident event
<"from" of edge> <"to" of edge> <time step to add accident on> <duration of accident>
```



```
8 7
1 0 6   1 4 5
2 1 3   0 5 10
3 1 3   5 9 10
4 1 3   0 9 10
0 5 3   8 9 10
0 6 3   8 9 10
0 7 3   8 9 10
4
2 1 0
3
5 5 5
3 1 0
3
6 6 6
4 1 0
3
7 7 7
1 0 5
3
0 0 0
3
1 0         23 4
1 0         25 3
0 5         13 8
```

Fig. 1. Explanation of the file “data-Merge2.txt”

7 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) Print the contents of each road (use the provided function "printRoadContents")
(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.
(5) 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. (do not move cars which were just moved in the previous steps)

Repeat the above until all events have finished and all cars have reached their destination.

8 Deliverables:

Your solution should be submitted as "trafficSimulator.c". If you created any other files to solve the problem be sure to submit them (including possibly a new "makefile"). **Do not zip your files**. Be sure to double check the materials you submitted are the correct versions.

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

9 Grading Notes:

The project graded out of 20 points. Here's a rough breakdown of points awarded (each higher grade assumes all prior criteria are met):

- 10/20 - Correctly reading, printing, and storing all of the values in the input file
- 12/20 - Processing and printing the events
- 14/20 - Cars are correctly moving on their original roads
- 16/20 - Cars are correctly reaching their destinations
- 18/20 - Cars obey traffic lights
- 20/20 - Correctly printing summary

Additional deductions applied to the above scores:

- Code that does not compile on the CS Linux lab machines will receive a deduction of **at least** $2pts$.
  - · Code that does not compile with usually receive very few points (e.g. $< 5pts$).
  - · This because I cannot test your code.
  - · Be sure to check your code!
- Not mallocing your data will receive a deduction of **at least** $2pts$.
- Memory leaks will cause a reduction of grade by $1pt$ to $2pts$.
- Memory errors will cause a reduction of grade by $1pt$ to $2pts$.
  - · Be sure to look for these when you run Valgrind!

  - Examples: invalid reads, invalid writes, Conditional jump based on uninitiallized value, etc.

If your worked with a partner, remember that both you and your partner should submit your solution (even if they are completely the same).