Assignment 3: Congested Intersection [1]

Due date: *Wednesday, May 8, 2019 at 22:00*

This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. You might receive a bonus for an outstanding solution.

Problem Statement

You are asked by the mayor of a large city to determine the level of congestion of all the intersections (or squares) in the city. The city consists of rectangular blocks arranged in a grid, horizontal and vertical streets intersecting at the corners of each block.

Given a list of car *trips*, your implementation must be able to determine how many cars have gone through a particular intersection in a given time interval. A *trip* is defined by the following information:

- the identifier of the car, which is initially parked at an intersection
- destination, which is another intersection
- departure time, the exact time at which a car starts moving from the starting point to the destination

Cars stop moving after they reach their destinations, and they stay there until the beginning of the next trip. There are unlimited parking slots, so cars immediately find a parking spot at their destination and therefore do not create more traffic *after* reaching their destination. Each car requires one time-unit to pass one city block, moving from one intersection to the next. This time is always the same and is independent of traffic.

The city consists of a rectangular grid of streets, with R horizontal streets (rows) and C vertical streets (columns). Figure 1 shows an example city map with 3 horizontal streets and 4 vertical streets. Street are identified by consecutive integer numbers starting from 0. Therefore each intersection is identified by the numbers of the two intersecting streets, horizontal and vertical, respectively. For example, (r, c) identifies the intersection of the r-th horizontal and the c-th vertical street ($0 \leq r < R$, $0 \leq c < C$).

There are F cars in the city. Initially, all cars are at intersection $(0, 0)$. There is no limit to how many cars can be in the same intersection.

The routing is deterministic for all trips and for all cars, and is defined by the following rules:

- A car follows a minimal path to the destination,
- A car first drives left or right, and then up or down.

The distance between two intersections is defined as the minimum total number of city blocks (cells in the grid) that a car has to pass in any direction to get from one intersection to the other. That is, the distance between intersection (a, b) and intersection (x, y) is equal to $|x-a|+|y-b|$. Figure 2 illustrates an example trip, in which a car with the initial position of (a, b) starts a trip to the destination (x, y) at time t_i. Both the initial position and the final destination are included in the traffic inducted by this trip. Therefore, this trip imposes an extra traffic to 6 intersections.

[1] Idea is partially barrowed from the Google Hash Code competition 2018; https://codingcompetitions.withgoogle.com/hashcode/

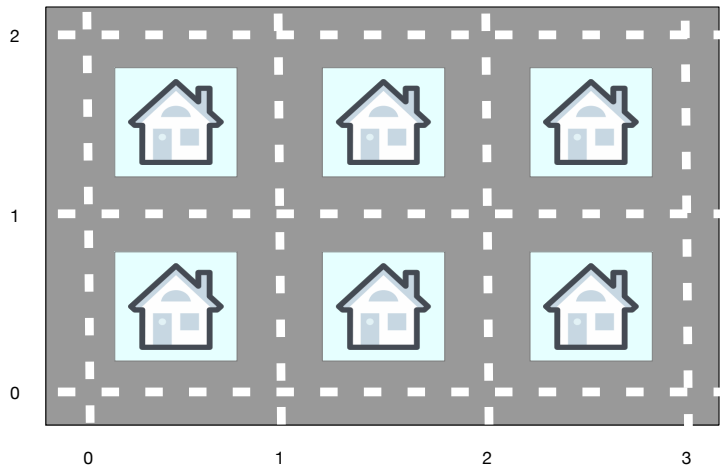

Figure 1: Example city of 3 horizontal and 4 vertical streets

Implementation and Input Format Specifications

In a source file called *busyintersection.c*, you must implement a library that simulates traffic and therefore that can be used by the city mayor to determine the congestion levels at intersections. Your library may not use any external library other than the standard C library. The library must define (i.e., implement) all the declarations in the following header file:

busyintersection.h

```c
#ifndef BUSY_INTERSECTION_H_INCLUDED
#define BUSY_INTERSECTION_H_INCLUDED

struct simulation;

/* Constructor: Reads the set of trips from the given file;
*  return a null pointer in case of failure.
*/
struct simulation *si_new(char * trips_filename);

/* Destructor: clear all memory allocated for the given simulation. */
void si_delete(struct simulation *s);

/* Return the number of times a car has passed through
* the intersection (x,y), since time "start_t" up to
* the time "end_t". It returns -1 if the given intersection
* doesn't exist.
*/
int si_get_congestion(struct simulation * s, unsigned start_t, unsigned end_t,
    unsigned x, unsigned y);

#endif
```

The library must read the specifications of set of the trips from a file. The name of this file is given as a parameter to the constructor function `si_new`. The file contains exclusively ASCII characters with lines terminated with a single end-of-line character. All data points are separated by a single space. The first line of the file contains four integers:

- C ($1 < R < 10^5$), the number of horizontal streets;
- R ($1 < C < 10^5$), the number of vertical streets;

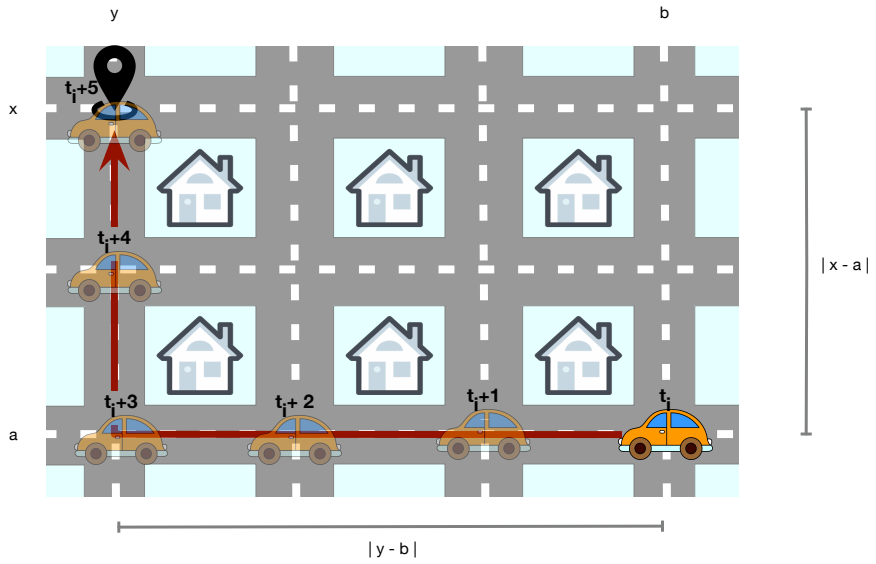

Figure 2: An example trip. The car starts at time t_i from the position (a, b) toward the destination (x, y). This trip generates extra traffic for 6 squares in the city.

- F ($0 < F < 1000$), the number of cars in the city;
- N ($1 < C < 10^5$), the total numbers of trips.

This is followed by N lines, where each line contains a description of a trip. The description of a trip contains the following four integers:

- id ($0 \leq id < F$), the car identifier;
- t ($0 \leq t < 10^9$), the starting time of the trip;
- x ($0 \leq x < R$), the row of the destination intersection;
- y ($0 \leq y < C$), the column of the destination intersection.

Example

Below is an example of a trip database file with $R = 4, C = 4$ number of horizontal and vertical streets, respectilvely. $F = 2$ cars having $N = 4$ trips during the simulation.

tripset1.in

```
4 4 2 4      // C=4, R=4, F=2, and N = 4
0 0 0 3      // Car-0 goes to (0,3) starting at t=0
0 4 3 0      // Car-0 goes to (3,0) starting at t=4
1 5 2 3      // Car-1 goes to (2,3) starting at t=5
1 12 3 0     // Car-1 goes to (3,0) starting at t=12
```

Next, in `usercode.c` you see how your library is used in order to read the DB file and to obtain the congestion level of an intersection.

usercode.c

```c
#include "busyintersection.h"

int main(){
  struct simulation *s = si_new("tripset1.in");
  if(s == NULL)
    return 0;

  si_get_congestion(s, 0,0, 0,0);       // =1
  si_get_congestion(s, 0,5, 0,0);       // =2

  si_get_congestion(s, 1,3, 0,1);       // =1
  si_get_congestion(s, 0,6, 0,1);       // =3

  si_get_congestion(s, 0,5, 3,0);       // =0
  si_get_congestion(s, 11,20, 3,0);     // =1

  si_delete(s);
}
```

Submission Instructions

Add comments to your code to explain sections of the code that might not be clear. You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file,* such as project description files. Submit the source file *busyintersection.c*, and nothing else, through the iCorsi system.