Homework #2

RELEASE DATE: 03/11/2014

DUE DATE: 04/01/2014, **17:30** in CSIE R217 and on CEIBA

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

2.1 More from the Class and Reading

(1) (10%) The following BINSEARCH algorithm has been introduced in the class. Modify the algorithm such that when there are multiple elements that matches *value* within the array, the one with the largest index would be returned.

```
BINSEARCH(record array $arr$, integer $len$, integer $value$)
// $arr$ ordered such that $arr[0] \le arr[1] \le arr[2] \le \ldots \le arr[len-1]$,
// where the order $\le$ is defined on some property of the record
$left \leftarrow 0, right \leftarrow len$
while $left < right$ do
  $mid \leftarrow \left\lfloor \frac{left+right}{2} \right\rfloor$
  if $arr[mid] = value$ then
    return $arr[mid]$
  else if $arr[mid] > value$ then
    $right \leftarrow mid$
  else if $arr[mid] < value$ then
    $left \leftarrow mid + 1$
  end if
end while
return NOTFOUND
```

There are many different solutions of answering this question, and please note that the *quality* of your solution would be evaluated, too. In addition to the algorithm, we do not need a formal proof in this problem, but please write down a few lines to illustrate your key ideas.

(2) (10%) Do Exercise R-3.2 of the textbook. Now that you have learned about insertion in an ordered array, you should be able to learn the *insertion sort* algorithm on your own, and answer this question.

2.2 Asymptotic Complexity

In all the questions below, the notation $f(n) = O(g(n))$ applies to functions $f(n)$ and $g(n)$ from $\mathbb{N}$ to $\mathbb{R}^+ \cup \{0\}$, and $f(n) = O(g(n))$ if and only if there exists $n_0 > 0$ and $c > 0$ such that $\forall n \geq n_0$, $f(n) \leq cg(n)$.

(1) (10%) Write down the proof of a simple quantization law in asymptotic notations. Let $f(n) = \lceil g(n) \rceil$. Prove that $f(n) = O(g(n))$ if $g(n)$ is bounded away from 0 (i.e. all $g(n) \geq \epsilon$ for some constant $\epsilon > 0$).

(2) (10%) Write down the proof of the multiplicative law in asymptotic notations. Assume that $f_1(n) = O(g_1(n))$ and $f_2(n) = O(g_2(n))$. Prove that $f_1(n) \times f_2(n) = O(g_1(n) \times g_2(n))$.

(3) (10%) If $g(n) > 0$ for all $n \in \mathbb{N}$, if there exists $c' > 0$ such that $\forall n \geq 1$, $f(n) \leq c'g(n)$, prove that $f(n) = O(g(n))$. (*Hint: This is almost a give-away question, and you should follow the definition and list what* (n_0, c) *can be used to match the definition.*)

(4) (15%) If $g(n) > 0$ for all $n \in \mathbb{N}$, if $f(n) = O(g(n))$, prove that there exists $c' > 0$ such that $\forall n \geq 1$, $f(n) \leq c'g(n)$. (*Hint: This is a bit harder than the previous question, and now you need to list what* c' *you want to use. After you prove the claims in the two problems above, you show that* n_0 *can be chosen as* 1 *when* $g(n)$ *is always positive.*)

(5) (15%) Do Exercise C-4.16(b) of the textbook. Note that Horner's method is an important tool to learn when you want to evaluate a polynomial somewhere.

(6) (Bonus 10%) Given any two functions $f(n)$ and $g(n)$ from $\mathbb{N}$ to $\mathbb{R}^+ \cup \{0\}$, is either $f(n) = O(g(n))$ or $g(n) = O(f(n))$ true? Prove your answer.

(7) (Bonus 10%) Given any function $f(n)$ from $\mathbb{N}$ to $\mathbb{R}^+ \cup \{0\}$, is $f(n) \in O(f(n)^2)$? Prove your answer.

2.3 Playing with Big Data

Background

(Total 120%) In this homework, we'll play with a toy version of ETC (Electronic Toll Collection) scenario, where you are to implement various operations that help process monitor data.

The ETC system currently implemented in Taiwan majorly consists of two parts: the e-Tag detection system and the plate-number recognition system, where the plate-number recognition system is only resorted to if a valid e-Tag is not present.

As you can see from the news, many reported error for the current ETC system comes from faulty detection of an e-Tag when it shouldn't be tolled. Therefore, if we could better exploit the plate-number recognition system and extract information from it, some errors can perhaps be avoided. This is where you come in.

Problem Specification

Assume for this problem that there is a single highway that linearly spans from south to north. There are a total of $N = 100$ tolling checkpoints, numbered $1, 2, \ldots, N$ from south to north. A pair of cameras monitor each checkpoint, the camera with ID "xSN" monitors cars passing from south to north at checkpoint x, and the camera with ID "xNS" monitors cars passing from north to south at checkpoint x. So for example, a car passing through checkpoint 7 toward south would be captured by camera"7NS".

A plate number, for simplicity, is of the form of 6 alpha-numerics. So "123456", "ABCDEF", "1X2Y3Z", "A7B8C9" are all valid plate numbers.

The problem with the plate-number recognition system is that, despite of the recent advances of digit-recognition techniques, the accuracy can never be perfect due to practical reasons (e.g. shades and dirt on the plate). We must therefore assume some of the digits could be wrongly identified when performing queries.

We assume there are *at most* 2 digits that are erroneous (i.e. might in fact be any other digit). For example, the plate number "DSA014" may be erroneously identified as "NBA014".

One single record captured by a camera is of the form:

```
<date> <time> <plate-number>
```

An example would be:

```
2014/03/11 13:00:00 DSA014
```

You'll be provided with the full records each camera captured (which is simply the collection of all records listed one in a line in chronological order), and you must answer queries of the two forms:

- **possible(*id*, t_1, t_2)**
  For a vehicle with plate number *id*, retrieve all records between time $t1$ and $t2$ (inclusive) that may be associated with it in all cameras, sorted in chronological order. By "*may be*" we mean all records that may be an errorneous identification of the plate number *id*.

- **anomaly(*id*)**
  For a particular plate number *id*, consider all its *exact* matches in the records in chronological order. Find and report "suspicious records".

  Note that if vehicles only travel using this single highway, their locations should be somewhat "tractable"—a car that just passed through camera 5 toward north must be located somewhere between camera 5 and 6. In this sense it is suspicious if a car is seen passing through camera 5 toward north, but next immediate record is neither (6, S→N) nor (5, N→S).

  Strictly speaking, a record of a car passing checkpoint x from south to north would imply it moved from some where between checkpoint $x - 1$ and x, to somewhere between checkpoint x and $x + 1$, and vice versa in the other direction. By *"suspicious records"* we mean if some two consecutive records (in chronological order) for a car provide conflicting positional information; i.e. the former record saying the car ended up being somewhere between checkpoint x and $x + 1$, while the latter indicating it being somewhere else before passing through.

For the ease of this problem, you *can assume* that no two records will ever has the exact same date-time in any records.

Please be aware that the data set is huge—several Gegabytes. Thus, if you are not careful with your implementations, your program can easily crash. Also, if you are programming on the R217 machines, **DO NOT copy the data set to your own directory**. Your own directory is on the NFS system and copying it there would only slow down your program (and other users' programs).

The input/output/data formats are in the following sections. We only provide sample output for a few smaller test cases this time to help you validate your code, but you are responsible to think about how you can make sure that your code is correct with larger test cases.

Input Format

There are a total of $201 = 100 \times 2 + 1$ files. Two for each camera, and one that stores all the queries. Camera record files are described above. The query file contains many lines, each being either `possible(id,t1,t2)` or `anomaly(id)`.

Output Format

You should answer the queries in the query file line-by-line. For a `possible` query, output all the lines in chronological order in the following format.

```
<date> <time> <plate-number> <camera-id>
```

An example output for a query "DSA014" might be:

```
2014/03/11 13:00:00 DSA014 3SN
2014/03/11 13:54:32 NBA014 4SN
2014/03/11 13:55:55 NBA014 5NS
```

For an `anomaly` query, for a pair of conflicting adjacent records, output both occurence in a single line in the following format:

```
<date1> <time1> <camera-id1> - <date2> <time2> <camera-id2>
```

An example may be:

```
2014/03/11 13:54:32 4SN - 2014/03/11 13:55:55 5NS
```

Again, you must output all anomalies in chronological order.
Finally, after each query (even if the result is empty), output "`--`" in a single line as delimiter.

Judging Criteria

Your output will be judged automatically, so please be sure to follow the format exactly. Specificially, do not insert unnecessary whitespace, and do not output unnecessary lines either. You can checkout utilities such as "cmp" and "vimdiff" for comparing (large) files.

We will try to be friendly on tolerating minor format errors, but matching your output with the standard could greatly save our effort, which we'll very much appreciate.

There will be a total of 12 test case, with the data size ranging from small to large. You will be provided with some of them as "public", and the others will be kept as "private" by the TAs. Each test case shares an equal amount of points (i.e. 10 points). Your credit for a test case will depend on two things: **correctness** and **efficiency**. Your will be deducted points if your output is not entirely correct. Note that properly separating test case with delimiter "`--`" is extremely important here, otherwise we may not be able to identify the corresponding section of output for a specific query. There is also a fixed amount of time we could afford to run each test case. So should your program time out, you will be given credit based on your (partial) output before the program is terminated.

We will try to give some reference on the cut-off time for the public test cases later, so you could know if your program's performance is optimized enough.

Submission File

Please submit your written part of the homework on all problems together to the box in R217 before the deadline. Also, you need to upload your coding part as a single ZIP compressed file to CEIBA. The ZIP file should contain the following items:

- all source files that you implement
- a Makefile (perhaps modified from the TAs) such that typing "make" results in a single executable file `hw2_3` that takes the data directory as the input, and writes to the standard output to solve Problem 2.3
- an optional `README`, anything you want the TAs to read before grading your code

Please make sure that your code can be compiled and run with your Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!