CSE 326 – Data Structures, Winter 2004
Wet assignment #2

Due dates.

Team formation: by 5pm on Tuesday, Feb 17, 2004
Full assignment: by class time on Wednesday, Feb 25, 2004 (electronic)
in class on Wednesday, Feb 25, 2004 (on paper; *see instructions*)

Logistics.

For this wet (programming) assignment, you need to work in a team with one other person in the class. It is up to you to decide who that person will be. It does not have to be the person whom you worked with on the first assignment, although you are free to pair up with the same person if you like.

We require that one of the two students in each team email both TAs (and CC their teammate) *by 5pm on Tuesday, Feb 17,* letting us know the following:

(a) who the team members will be – names and email addresses for both; and
(b) what implementation language (Java or C++) both of you are going to be using.

Note: If you fail to notify us on time, we will deduct 10% of your score on this assignment.

Problem area description.

Cities A and B are connected by several highways, which do not intersect anywhere between A and B. Each highway is named using a unique positive integer from an unbounded range. The highways serve trucks that travel from A to B.

There are tunnels along the highways, each of which has a limit on the maximum height of a truck that can pass through it. It is known that each highway connecting A and B has between 0 and N tunnels (for some known N).

The *bottleneck height of a highway* is the maximum height of a truck that can travel on that highway and successfully pass through all the tunnels. For example, in Figure 1, the bottleneck height of highway 5200 is 13.2 feet – trucks higher than that won't be able to clear the tunnel with the corresponding limit.

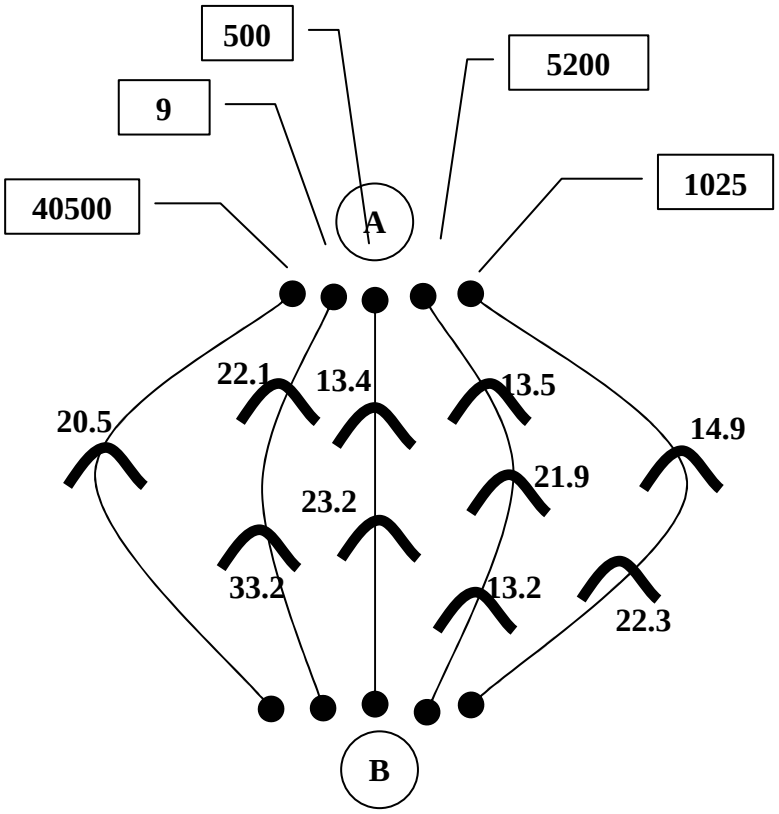

Figure 1. A highway system with tunnels

The *bottleneck height of the whole highway system* is defined as the maximum of the bottleneck heights of all highways. In the example in Figure 1 this value is 22.1 feet, i.e., any truck whose height is at most 22.1 feet can travel from A to B (on highway 9); also, no truck whose height is over 22.1 feet can travel from A to B (on any highway).

What you need to do.
For this project, you will aid designers and planners of such highway systems by writing software and doing some analysis.

1. *Warm-up questions (include the solutions in your submissions)*

Consider the highway system in Figure 1.

1) Where can the planners add a tunnel in order to change the maximum allowed height (the result of `get_system_bottleneck()`) of a truck that travels from A to B?
2) List all possible ways whereby removing exactly one tunnel would change the result of `get_system_bottleneck()`.
3) Define analytically the result of `get_system_bottleneck()` as a function of the heights of the tunnels.

2. *Design and Implementation*

Propose data structures and suggest algorithms for "simulating" a highway system under the set of constraints outlined below.
Note: Be sure to carefully read the constraints (e.g., for required time and space complexities, method interfaces, etc.) and think over your design *before* you start implementing it!

Write a class `HighwaySystem` with (at least) the following operations:

1) `init (highway_names: array of integers; m: integer): void`
   Initializes the system to have `m` highways, whose names are given in the array `highway_names` (whose length is `m`). Since there are no tunnels in the system initially, for this operation you do not need to worry about them.
   The required time complexity is O(`m`).
2) `add_tunnel (height: double; highway: integer): void`
   Adds a tunnel whose height is `height` (the first parameter) to the highway whose name is `highway` (the second parameter). If such a highway does not exist, or if `height` cannot be a valid height (i.e., 0 or a negative number), nothing should be done.
   The required time complexity is O(log `n` + log `m`), where `n` is the total number of currently existing tunnels.
3) `remove_bottleneck_tunnel (highway: integer): void`
   Removes one tunnel with minimum height along the highway whose name is `highway`. (Note: If there are two or more tunnels with the same (minimum) height along that highway, only one of them (chosen arbitrarily) should be removed; after this the highway bottleneck will remain unchanged.) If there is no such highway, or if there are no tunnels on that highway, nothing should be done.

The required time complexity is O(log `n` + log `m`), where `n` is the total number of currently existing tunnels.

4) `get_system_bottleneck (): double`
   Returns the maximum height of a truck that can travel from A to B. If there exists a highway with no tunnels, any truck can travel from A to B and so the system's bottleneck is undefined; in this case the returned value should be –1.
   The required time complexity is O(1).

The total space complexity of your data structures should be O(`N*m`), where `N` is the maximum number of tunnels along a single highway, and `m` is the number of highways in the system.

3. *Input and Output*
   1) Your program should take a single command-line argument which will be a file name. **If your program does not run properly with a single command-line argument which is a filename, points may be deducted.** That file will include a series of commands, one command per line, corresponding to each of the four parts above. Your program should correctly perform each command listed in the file. Each command takes the form of a single character followed by a series of numbers, all separated by spaces. You may assume that all commands are properly formatted (the add command always has two numbers, the init command's count is always correct, etc).
      a. "I <count> <hwy #1> <hwy #2> …". Perform the init() function on the list of highways. For example,
         I 3 90 520 405
         I 1 405
         The first initializes three highways, 90 520 and 405. The second initializes a single highway, 405. You can assume there will be exactly one init command, and that it will be the first line of the file.
      b. "A <ht> <hwy>". Perform add(ht, hwy).
      c. "R <hwy>". Perform remove_bottleneck_tunnel(hwy).
      d. "B". Print the system bottleneck. You should print this to standard output (e.g. System.out.println(), or printf(), depending if you are using Java or C++).

   A sample test file is provided on the assignments page,
   http://www.cs.washington.edu/education/courses/cse326/04wi/assignments/wet2-files/
   You should also make your own files for testing.

   Do not print out anything except in response to the B command. If you want to output anything for debugging, make sure it is turned off in the version you hand in. **Printing out anything extra may result in point deduction.**

Development environment.
As an implementation language you are allowed to use either Java or C++. Be sure to avoid using features that are specific to a particular version of a compiler, since if you do so we likely will be unable to compile your code and you will lose points.

Submission.
The submission is electronic and part of it is also paper-based, as specified below.
We need *only one* electronic submission and *only one* printed submission per team.

What to submit. All of the following (1-6) need to be submitted *electronically* by class time on the due date:

1) Your code – the class `HighwaySystem.java` or `HighwaySystem.cpp` (depending on which language you use) with the code for all of the implemented operations. Also include any supporting headers, packages, and the test module(s) you wrote.
2) A description of all modules your code consists of (including module names and a one-paragraph description of what each module does), as well as *clear* instructions on how to compile and run your code. Put this in a plain text file, called `description.txt`.
3) A description of the data structures you chose to use and a *clear* explanation of the algorithms for each of the four required operations (above). Put this in a plain text file, called `algorithms.txt`.
4) Your brief argument why the time and space complexities of your algorithms are what the specification asked for (and not worse). Put this in a plain text file, called `complexities.txt`.
5) The answers to the questions from the *Analysis* section above. Put this in a plain text file, called `analysis.txt`.
6) A brief description of who (of the two partners in the team) did what part of the assignment: the different parts of the code, the complexity arguments, the analysis questions, and anything else you did. Be sure to include your names and emails here.
   You are encouraged to work on all parts together, and if so, indicate that this is indeed the case. If, however, you decided to split the work, we also need to know that, as well as how you did it. (Note that both partners are responsible for all parts of the assignments regardless of whether and how you may decide to split the work among yourselves.) Put this description in a plain text file, called `contributions.txt`.

In addition, *print* and bring to class on the due date:

1) The code of the `HighwaySystem` class *only*. (Please, *do not* print header files, packages, or test modules/classes!)
2) The description text files 2-6 (described above). To save paper and trees, please merge them together into one file and then print.

Do not forget to write the names of both partners on the printed sheet you submit.

Printing guidelines. Double-sided printing and condensed code printing (i.e., double-sided at 2 pages of code per side) is highly encouraged! (This shows that you care to present yourselves as professionals.) Modern printers, including many of those available at UW facilities, can do that. Ask the lab person for assistance if you are unsure how to do it.

Where to submit. Instructions on where to electronically submit your assignment will appear on the course web page and will be announced on the mailing list.

Quality criteria.
For full credit, you need to meet the following quality criteria:

1) Your submission needs to contain all parts described in the previous section.
2) Your code needs to:
   - Compile and run without errors or warnings. (We will follow the instructions you will have provided us with, so those need to be accurate.)
   - Contain comments (this is really important!), making it easier for a human to understand what you have done.
3) Your argumentation about the time and space complexities needs to be sound. Brevity is encouraged but not at the expense of clarity or soundness.
4) Your printed materials need to look professional (i.e., be printed and stapled together rather than hand-written and loose).

Our advice.
By now you know that programming assignments take time, so be sure to *start early*!

Good luck!