CS310: Advanced Data Structures and Algorithmns

Spring 2014 Assignment 4

Due: Tuesday, March 25, in class

Goals

Sorting, Sets and Priority Queues

Questions

1. Weiss, exercise 6.4. Show T(N) analysis for all (two) cases that can sort at all by simple inserts, then removals.
2. JDK Sorting. Give the calls to the java Arrays.sort() needed to sort the following. Read the JDK documentation for all the possible sort prototypes.
   (a) String[] names = {"Joe", "Annie", "Bruce", "Tanya"};
   (b) int[] numbers = {3,2,10};
   (c) names again, but in reverse order (use Collections.ReverseComparator)
   (d) SimpleStudent students = {new SimpleStudent("Bob",0), new SimpleStudent("Joe", 1), new SimpleStudent("Bob",2)} using their natural order.
   (e) students again, but using their ids. This is the only part of this problem where you need to write more than one line of code.
3. A Set application – dating/roommate service. You are running a dating/roommate service. Each of your clients fills out a form selecting phrases that they agree with from a list like this:

   1. Programming is fun
   2. Cats are nice pets
   3. Chinese food is the best

   ...

   100. Watching TV all evening is my favorite past-time

   Thus each client provides a Set of ints corresponding to the statements with which he or she agrees. Clients are identified by their social security numbers, which are conveniently just ints. The client questionnaires are all filled out on line and automatically converted to a Set<Integer> providing method getPrefs(...). Given a class WebInterface, an object WebIF of that class will give the following:

   - Set<Integer> clients = webIF.getClients();
   - Set<Integer> prefs = webIF.getPrefs(ssn);
   - String name = webIF.getName(ssn);

   Where prefs is the set of user preferences for one client, name the name of the client, and clients the set of SSNs for all the clients. In the rest of this question assume that this functionality is already implemented. The method that will make your business a success is

   ... findBestMatch(client);

This function returns the person (name and SSN) whose preferences best match those of the input client.

(a) Give a class design for your application (formatted like Fig. 20.9, pg. 789 – declaration of classes and function prototypes for each class). Don't make your design too complex: two or three classes should be enough. You can use any Collections classes of Java for data structures. Do not try to implement the entire thing, just give prototypes.

(b) Write method findBestMatch, assuming that by "best match" we mean "largest number of matching phrase numbers between clients".

(c) Critique the definition of "best match" above. What if some client turned in an empty questionnaire? What if someone checked every box? Is it true that if FindBestMatch says A is the best match for B then it will also say B is the best match for A? Suggest a better way to deal with the problems raised in this section.

4. For the modem simulation of Section 13.2.2, pp. 514–522 and discussed in class, show the contents of the priority queue for the first 12 lines of the simulation depicted in Figure 13.4, except as follows. Expand the modem pool to 5 modems, and for any users who previously heard the busy signal but now get a modem, have user 4 dial for 10 minutes, user 5 dial for 3 minutes and user 6 dial for 7 minutes. Show the contents of the priority queue like Figure 13.10, but for ease of typing please simplify the elements in the queue to (time, what, who), and enclose the whole queue in square brackets, so for example, the 4th line should look like this: $[(6, HANG\_UP, 1), (2, DIAL\_IN, 2)]$

5. Show how a TreeMap can be used to implement the priority queue methods in use in the modem simulator. Be sure to handle the case of multiple values of one key – you can use the values in the TreeMap for the multiplicity information as well as other information (as hinted in class). Write a wrapper class PriorityQ (avoiding the JDK name PriorityQueue) that HASA TreeMap and can do the job, i.e., has the needed methods supported by a priority queue.