Computer Science 345
Test 2. Ch 1-5. Abstract data types + case studies; graphs; balanced trees

As stated in the syllabus, the dates, nature, and coverage of the four tests in this course are

Test 1	Feb 8	on paper	conceptual problems on correctness and efficiency, ADTs, the "case studies"~~, and graphs~~
Test 2	~~Apr 4~~ Apr 6	at a computer	programming problems on ADTs/case studies, graphs, and trees
Test 3	May 3	on paper	conceptual problems on *graphs*, trees, dynamic programming, hash tables, and strings
Test 4	May 3	at a computer	programming problems on dynamic programming, hash tables, and strings

Test 2 consists in three programming problems, one each from the categories of ADTs (ch 1–3), graphs (ch 4), and BSTs (ch 5). Each will be similar to the projects but smaller, intended to be doable in about 15 minutes each. Similar to programming problems on tests in courses like Programming II, they will ask you to write a method or finish a class or the like. The practice problems for this test should give you a good feel for what the test problems will be like.

Administration of the test.

If you are taking the test in the CSCI lab,

Before the test, there is nothing to be done.

At the beginning of the test period, find a workstation that is logged in with a special test account. Do not log in to your own account on any workstation. Find the open Eclipse window. In the file `NAME`, write your name (and save). Find starter code in the other files. There is a single project with three packages, one for each problem.

At the end of the test period, make sure that the files you have edited (including the file called `NAME`) are saved. Close Eclipse and log out. Nothing needs to be turned in.

If you are using your own laptop,

Before the test, make a new Eclipse workspace with no projects in it. Close all windows except Eclipse (in the new workspace) and a terminal (for getting starter code and turning in your solutions).

At the beginning of the test period, pull from the `CSCI345_DSA_S22` repository, and find a folder `tests/test2`. Make a new Eclipse project in that `test2` folder (which contains packages, one for each problem).

At the end of the test period, turn in the files you edited to a turn-in folder like the ones you used to turn in projects (`/cslab/class/cs345/[your userid]/test2`) using scp.

Grading

When grading the test I will use JUnit tests as an aide to understanding your code, but your score will **not** be based on the number of JUnit tests your code passes. Rather, your submission will be scored (and partial credit will be assessed) based on conceptual pieces I find when reading your code. You may include comments, which I will read. But since running your code against test cases will be part of the grading process, you don't want to submit code that doesn't compile.

Unlike projects *you will not have the JUnit tests I use for grading.* I will give you one or two simple JUnit tests per problem, but these will only be to clarify the problem, analogous to a clarification like "For example, if your method is given x, it should return y." **Whether your code passes the one given JUnit test is *not* a good indicator of whether your solution is correct.** Of course you may write your own JUnit tests, though time constraints may make that difficult.

Kinds of problems to expect.

There will be one problem (possibly with several smaller parts) for each of the following areas:

Basic abstract data types, their implementations, and other algorithms we've seen, including case studies. Expect a problem that asks you to write a class that implements a standard ADT (or variation on one) with one difference from things we've seen so far. Make sure the intent behind the standard ADTs is fresh in your mind. Be familiar with algorithms like binary search, the various sorting algorithms, and the algorithms supporting ADT implementations. Be familiar with arrays, linked structures, bit vectors, and heaps as implementation strategies. Iterators make great problems or parts of problems.

Graphs. Expect a problem that asks you to write a method that operates on a graph (searches for something in a graph, tests an attribute of a graph, builds a related graph, etc). Be familiar with the `Graph` interface as we had in class and the projects, as well as the concepts behind adjacency-list and adjacency-matrix implementations. Concepts from BFT, DFT, MST, and SSSP will also be worth reviewing.

Balanced trees. Expect a problem that asks you to write a method that operates on some kind of binary search tree: unbalanced, AVL, red-black (traditional and/or left-leaning), and/or 2-3. The method will either be an instance method of a class for one of these kinds of trees or will be a stand-alone (static) method that operates externally on one of these trees. The tree class will be a reduced form of the classes we've seen in class and used in the project. Make sure you are familiar with the constraints/properties of the various kinds of balanced trees.