CSCI 345

Data Structures and Algorithms

Spring 2016 MFW 2:00–3:05 pm Meyer 131

http://cs.wheaton.edu/~tvandrun/cs345

Thomas VanDrunen

☎630-752-5692 630-639-2255 ✉Thomas.VanDrunen@wheaton.edu

Office: Meyer 163 Office hours: MTuWThF 9:15–10:15am; Tu 1:30–3:30pm.

Contents

Catalog description. Stacks, queues, lists, trees, hashes, basic manipulation algorithms, sorting and searching, information hiding, abstract data types, memory management. Prerequisites: CSCI 243 and CSCI 245. [This is catalog description is a bit dated. I'll probably replace it with the following:]

Re-written catalog description. Formal and experimental approaches to verifying algorithms' correctness and analyzing their efficiency. Abstract data types and their implementations. Efficient implementations of maps using balanced binary search trees and hash tables. Graph algorithms. Dynamic programming. Prerequisites: CSCI 243 and CSCI 245.

Textbooks.
Robert Sedgewick and Kevin Wayne, *Algorithms*. Upper Saddle River, NJ, Addison-Wesley, Fourth Edition, 2011.

Purpose of the course. This course is a central part of the computer science major. This is the place where students' understanding of algorithmic problem-solving, data abstraction, and the trade-offs of data structuring strategies are capped off and mastered. Moreover, students add knowledge of crucial algorithmic techniques and data structures to their toolkit. Specifically there are six big ideas (the first three are pursued throughout the course, the last three refer to specific units):

- The correctness of an algorithm can be verified formally (using loop invariants and other proof techniques) and empirically (with unit tests).
- The efficiency of an algorithm can be measured formally (using algorithmic analysis, big-oh categories, etc) and empirically (by running experiments).
- Abstract data types (especially list, stack, queue, set, bag, and map) are specified by how they are used; data structures (arrays, linked lists, binary trees, hash tables, etc) are implementation strategies, each with trade-offs.
- Searching in an unordered data structure (such as a map) can be done in logarithmic time using a balanced binary search tree (in principle).
- Searching in an unordered data structure can be done in constant time using a hash table (in principle).
- Problems with overlapping subproblems and optimal substructure can be solved efficiently using dynamic programming.

Curricular objectives. This course supports students' development towards the computer science major's learning outcomes #1 (General—problem-solving), #2 (General—abstraction), #4 (Formal—algorithmic analysis), and # 5(Practical—programming).

COURSE OUTLINE. (For a schedule, see the course website. In some cases the chronological order of topics differs from the conceptual ordering found here.)

I. Prolegomena

Studying data structures and algorithms requires clear statement of the basic principles and our assumptions and model. Much of this is review from Programming II, but done more carefully and sometimes in a different style.

A. Abstraction and implementation
B. Abstract data types
C. Algorithms and correctness
D. Algorithms and efficiency
E. Case studies
    1. Linear sorting
    2. Disjoint sets and union/find
    3. Heaps and priority queues
    4. Bit vectors

II. Graphs

Many problems can be conceptualized as graph problems, and thus many useful algorithms can be built on variations of standard graph algorithms.

A. Graph concepts
B. Basic traversal algorithms: Depth-first and breath-first
C. Minimum spanning trees
D. Shortest paths

III. Trees

Our main interest is the several strategies for *self-balancing binary search trees*; to get there we first look generally at trees, binary trees, and binary search trees. Self-balancing binary search trees are a way to address the searching problem.

A. Binary trees
B. Binary search trees
C. Balanced binary search trees
    1. AVL trees
    2. Red-black trees
    3. Left-leaning red-black trees
    4. 2-3 trees
    5. B-trees

IV. Dynamic programming

Dynamic programming is a technique that uses tables to store intermediate results of recursive algorithms for certain divide-and-conquer problems with overlapping subproblems.

V. Hashing

Hash tables and other structures that use hashing allow for fast (near constant time) look-ups while still allowing the structure to grow and shrink easily. They represent a second way to address the searching problem. You have seen one approach to hash tables in Programming II (specifically what's called "separate chaining"); we here consider other approaches.

A. Hashing general concepts
B. Separate chaining
C. Hash functions
D. Open addressing with linear probe
E. Alternate strategies
F. Perfect hashing

VI. String processing

A lot of data used in modern applications is textual data. String processing plays a crucial role in many of the common uses of computation today.

A. Sorting strings
B. Tries
C. Compression
D. Substring search

Course procedures

Important general note. In the course this semester there will be several disruptions (guest lecturers in January, my family having a new baby in March) and I will be experimenting with some new course procedures. Accordingly circmstances may turn out that some specifics listed here, including the dates, nature, and coverage of the tests, will need to be adjusted. My pledge is to keep with the spirit of the procedures described here, even if the letter of the procedures changes.

How we do this course. Algorithms and data structures will be presented in class for discussion about their correctness and efficiency and to examine aspects of their implementation. In some cases the students will read about the concepts ahead of time to prepare for class discussion; at other times ideas will be presented first in class. Most of students' work is in projects in which they will implement the things we discuss in class. Occasionally there will be in-class lab activities, quizzes, and smaller assignments. On tests, students will apply the things we have learned to new problems.

Projects. As stated above, projects are where students will do most of their work and, presumably, most of their learning. Students may work on projects at their own pace; there are no due dates for projects except that all projects must be turned in by midnight between Apr 29 (the last day of classes) and Apr 30. Projects will be graded for correctness to be verified by unit tests (and inspection for general conformity to the intent of the project). Students will be supplied all unit tests used in grading (except that the right is reserved to modify the data used in the unit tests to make sure the code hasn't been hardwired to pass unit tests with specific data). See the *Policies etc* section of this syllabus for other details.

Tests. There will be three tests, currently scheduled for Feb 19, Apr 1, and Thursday, May 5 (at 1:30 pm, this course's final exam block). Specifically,

- Test 1 will be on paper and will have *conceptual problems* from the first third of the course (prolegomena and graphs).
- Test 2 will be at a computer and will have *programming problems* from the first half of the course (prolegomena, graphs, and trees).
- Test 3 will have both a paper portion with *conceptual problems* from the last two-thirds of the course (trees, dynamic programming, hash tables, and strings) and a portion at a computer with *programming problems* from the second half of the course (dynamic programming, hash tables, and strings).

The test held during the final exam block (hereafter "test 3", not "the final exam") is longer and covers a larger portion of the course than the other tests, but it will not be explicitly cumulative over earlier parts of the course and it will not be worth more than the other tests. See the *Policies etc* section of this syllabus for details about the tests at a computer.

Readings. There will be occasional readings from the textbook. Unless otherwise noted, students should send a summary of the reading to the instructor by email before class (preferably the night before, so that the instructor can gauge students' reaction to the reading). See the course website for the format of the summaries.

Grading.

instrument	*weight*
Projects	40
Tests	45 (3 @ 15 each)
Other	15

. . . where *Other* comprises short assignments, in-class activities, readings, quizzes, etc. Some of these may be graded for correctness, others may be marked as done or not.

Policies etc

Academic integrity. Collaboration among students in the class is permitted on projects and assignments. Using *code* for projects from any outside resources (print, electronic, or human) is not permitted. Any *ideas* from outside resources used in projects must be cited using the same standards as would be used in a research paper. If you relied on an outside resource in any way for any project, submit a file with your project named `CITATIONS` giving citations for the resources and an explanation of how and why you used them. On any assignment given from the textbook, no resources that specifically serve as solutions to exercises from the textbook may be used.

A project or assignment on which a student violates these policies will be rejected. Repeated offenses will be handled through the college's official disciplinary procedures.

Late assignments. Projects have no due date except that all projects must be turned in by the last day of classes. No projects will be accepted after midnight between Apr 29 and Apr 30. No credit will be given for late work on other kinds of assignments.

Project grading. My intention is to grade projects solely on correctness as demonstrated by unit tests, but I reserve the right to inspect code for conformance to specifications that are not easily verified automatically. You will be given all the unit tests used in grading, except that the raw data used by the unit tests may be different, to prevent students from simply writing code that apes the responses the specific unit tests check for. **Extra credit** will be given to students who discover cases not covered by the given unit tests; in that case, the student must write a JUnit test case and provide a would-be solution to the project that passes the current set of unit tests but fails the new one. **Bonus extra credit** will be awarded if the instructor's own solution fails the new test. Moreover, **and this is important**, that new test case will be distributed to the class and added to the test cases used in grading. Any student who has already submitted a solution that does not pass the new test case should resubmit a corrected solution.

Tests using computers. For tests 2 and 3, the computer science lab (Meyer 154) will be reserved for our use. The lab machines will be logged into special accounts for the purpose of test-taking. Internet access will be turned off. Students will use only Eclipse and a web browser; the browser will be used only to view a local copy of the Java API. Students will be given a few JUnit tests to clarify the problem but **not** the JUnit tests to be used in grading.

Attendance. Students are expected to attend all class periods. It is courtesy to inform the instructor when a class must be missed.

Examinations. Students are expected to take all tests, quizzes, and exams as scheduled. In the case where a test must be missed because of legitimate travel or other activities, a student should notify the instructor no later than one week ahead of time and request an alternate time

to take the test. In the case of illness or other emergencies preventing a student from taking a test as scheduled, the student should notify the instructor as soon as possible, and the instructor will make a reasonable accommodation for the student. The instructor is under no obligation to give any credit to students for tests to which they fail to show up without prior arrangement or notification in non-emergency situations. The final exam is Thursday, May 5, 1:30 am. I do not allow students to take finals early (which is also the college's policy), so make appropriate travel arrangements.

Gender-inclusive language. The college requires the following statement to be included on all syllabi: *For academic discourse, spoken and written, the faculty expects students to use gender inclusive language for human beings.*

Special needs. *Institutional statement:* Wheaton College is committed to providing reasonable accommodations for students with disabilities. Any student with a documented disability needing academic adjustments is requested to contact the Academic and Disability Services Office as early in the semester as possible. Please call 630.752.5941 or send an e-mail to `jennifer.nicodem@wheaton.edu` for further information.

My own statement: Whenever possible, classroom activities and testing procedures will be adjusted to respond to requests for accommodation by students with disabilities who have documented their situation with the registrar and who have arranged to have the documentation forwarded to the course instructor. Computer Science students who need special adjustments made to computer hardware or software in order to facilitate their participation must also document their needs with the registrar in advance before any accommodation will be attempted.

Office hours. I try to keep a balance: Stop by anytime, but prefer my scheduled office hours. Any time my door is closed, it means I'm doing something uninterruptible, such as making an important phone call. Do not bother knocking; instead, come back in a few minutes or send me an email.

Dress and deportment. Please dress in a way that shows you take class seriously—more like a job than a slumber party. (If you need to wear athletic clothes because of activities before or after class, that's ok, but try to make yourself as professional-looking as possible.) If you must eat during class (for schedule or health reasons), please let the instructor know ahead of time; we will talk about how to minimize the distraction.

Electronic devices. Please keep laptops and all electronic devices put away and silenced during class. That's right, this is a computer science course, but you're not allowed to use a computer during class. Trying out programming concepts on your own during class time (for example) is not productive because it takes you away from class discussion. You cannot multitask as well as you think you can. Moreover, ***text in class and DIE.***