CSCI 345 Data Structures and Algorithms

Spring 2020 MFW 12:55–2:05pm Meyer 131

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

Thomas VanDrunen

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

Office: MEY 163 Office hours: MWF 3:30–4:30pm; Tu 1:30-3:00pm; Th 9:00–10:30am.

Contents

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.

Textbook.
Thomas VanDrunen, *Algorithmic Commonplaces,* 2019. Under contract with Franklin, Beedle and Associates. Preprint available at the Wheaton College bookstore. [Updates to certain sections may be made available on Schoology.]

Purpose of the course. This course is a central part of the computer science major. This is the place where students solidify their understanding of algorithmic problem-solving, data abstraction, and the trade-offs of data structuring strategies. 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.

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

Course Outline. (For a schedule, see the course website.)

I. Prolegomena

Studying data structures and algorithms requires clear expression of the basic principles. When studying algorithms, we have two dimensions. For a given algorithm, we want to determine whether it is correct or not and how efficient it is; both of these things can be studied both formally and empirically. For that end we consider formal techniques for reasoning about what an algorithm does and categorizing its running time. We also identify basic algorithms that are used as elements in more complicated algorithms, which builds our intuition for determining an algorithm's complexity.

Moreover, we need a clear distinction between *abstract data types* (ADTs), which specify how a container or other composite type presents itself to the rest of the system, and *data structures*, which are strategies for implementing ADTs. We define a set of standard ("canonical") ADTs, as well as identify the basic data structures that are used as elements in more complicated data structures.

A. Algorithms and correctness

B. Algorithms and efficiency

C. Abstract data types

D. Data structures

II. Case studies

Before hitting the meat of the course, we explore a few extended examples. Each of these serves to to reinforce and illuminate the principles laid out in the previous unit. Moreover, each is used as elements in certain later topics.

A. Linear sorting

B. Disjoint sets and union/find

C. Heaps and priority queues

D. Bit vectors

III. Graphs

The graph, as a mathematical abstraction, is one of the most important tools for representing the structure of information and relationships within information, used in one form or another by many fields. Accordingly, efficient ways to store and process graphs are a crucial piece of the study of data structures and algorithms.

This exploration of standard graph algorithms is both an exercise in applying the principles of the course and a primer in algorithmic elements whose variations can be applied to many specialized applications.

A. Graph concepts and implementation

B. Basic traversal algorithms: Depth-first and breath-first

C. Minimum spanning trees

D. Shortest paths

IV. Trees

In this chapter we embark on a task that stays with us for the rest of the course: Developing efficient data structures for implementing unordered collections, such as the map ADT. The naïve implementations have linear-time operations. This chapter considers one family of solutions: the binary search tree abstraction and various tree data structures that afford map implementations with logarithmic-time operations.

Specifically we consider several strategies for *self-balancing binary search trees* and compare their performance and other practical aspects.

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

V. Dynamic programming

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

Our exploration is organized around an introduction of several problems suited to dynamic programming solutions, the explanation of the principles of dynamic programming, the application of those principles to the given problems, and, finally, the working out of a larger dynamic programming solution to a more complicated problem.

That more complicated problem is a continuation of our quest for improved implementation of the map ADT: constructing a binary search tree not to achieve balance but optimality based on the look-up probabilities of the keys.

A. Three problems
B. Principles of dynamic programming
C. Three dynamic programming solutions
D. Optimal binary search trees

VI. Hashing

Having achieved logarithmic-time operations for a map using a binary search tree, we pursue the quest further with hashtables, a family of data structures that support constant-time map operations. We consider several specific approaches and compare their real performance.

A. Hashing general concepts
B. Separate chaining
C. Open addressing with linear probing
D. Hash functions
E. Perfect hashing

VII. 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. This includes considering a data structure for efficient map operations assuming string keys.

A. Sorting strings
B. Tries
C. Regular expressions

Learning outcomes. Corresponding to the six big ideas of the course listed earlier, this course's aim is that at the end of the semester, students are able to

1. State and demonstrate a loop invariant for an algorithm.
2. Determine the performance of an algorithm and identify a big-oh or big-theta category.
3. Differentiate between ADTs and data structures and articulate the trade-offs among data structures studied in class.
4. Articulate the factors in implementing a binary search tree and how they affect the performance of a BST implementation of a map.
5. Articulate the factors in implementing a hashtable and how they affect the performance of a hashtable implementation of a map.
6. Articulate the attributes of a problem that is suited for a dynamic programming solution and implement a given dynamic programming algorithm.

In addition to these, together we have the general objective of seeing algorithm design and implementation as a creative expression for God's glory and observing algorithmic solutions as part of God's creation.

Course procedures

How we do this course. The rhythm of this course consists in students reading about new ideas in the text; the ideas being discussed and practiced in class; and students working out the details of the ideas in projects. Most of students' work for this course is in projects in which they will implement the things we discuss in class. Occasionally there may be in-class lab activities, quizzes, and smaller assignments. On tests, students apply the things we have learned to new problems.

Readings. Readings corresponding to class days can be found on the schedule on the course website. I have separated the readings into two categories.

Readings coded magenta are the "full version" of the material presented in class. Although I will lecture on the material, class time does not permit as much detail as the book does. You are responsible for all the material in the book. Although you should at least skim these readings before class, you can determine what works before for you for a careful read. Some students may find they learn better skimming the relevant before class and then reading carefully after class; others may prefer to read carefully before class and then review the material after class.

Readings coded green must be read carefully before class. Rather than lecturing on this material, the class meeting will consist in working on exercises to apply the material, with the assumptions that students have read it.

Projects. As stated above, projects are where students will do most of their work and, presumably, most of their learning. To a certain extent, 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 the last day of classes, May 1. (Official deadline is midnight between May 1 and May 2. Enforced deadline is when I wake up in the morning on May 2.) Projects are graded for correctness, verified by unit tests (and inspection for general conformity to the intent of the project). Students are 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. Other information about projects, including practical suggestions, can be found on the course website.

Tests. There are four tests, currently scheduled as Test 1 on Fri, Feb 28, Test 2 on Wed, Apr 8; and Tests 3 and 4 on Wed, May 6 (at 8:00 am, this course's final exam block). Note that Tests 3 and 4 together constitute the final exam for this course, but are treated separately for the purposes of this syllabus. Specifically,

- Test 1 is on paper and has *conceptual problems* from the first third of the course (prolegomena, case studies, and graphs).
- Test 2 is at a computer and has *programming problems* from the first half of the course (prolegomena, case studies, graphs, and trees).
- Test 3 is on paper and has *conceptual problems* from the last two-thirds of the course (trees, dynamic programming, hash tables, and strings).
- Test 4 is at a computer and has *programming problems* from the second half of the course (dynamic programming, hash tables, and strings).

See the *Policies etc* section of this syllabus for details about the tests at a computer.

Grading. In order to pass the course (that is, to receive a D grade or better), a student must (a) achieve a minimum score of 75% on the projects and (b) achieve an average score of at least 50% on the tests.

(Note that students are given all the test cases used in grading and may resubmit projects throughout the semester. There is no reason for a student who takes the course seriously and acquires basic competency not to get 100% on projects.)

For students who have met the minimum requirements, their semester score is calculated as the weighted average of the following scores (each as a percentage):

instrument	*weight*
Projects	30
Tests	60 (4 @ 15 each)
Other	10

. . . where *Other* comprises short assignments, in-class activities, readings, quizzes, code style, etc. Some of these may be graded for correctness, others may be marked as done or not. Also, students may get extra credit points applied to the "other" category for each mistake in the book they are the first to discover, at the instructor's discretion.

Policies etc

ACADEMIC INTEGRITY. Collaboration among students in the class is permitted on projects and most assignments. Using *code* for projects from any outside resources (print, electronic, or human) is not permitted. Using *ideas* from outside resources in projects is not recommended, but if a student does get use any outside resources for concepts used in projects, they 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.

As much as possible, the projects are designed to minimize the opportunity for students to fool the automated grading, that is, to submit solutions that happen to pass the tests but fail to implement the intended approach to the problem. Students, however, are expected to abide by the intent of the projects and shall not submit solutions that "cheat" the unit tests if they discover a way to do so.

A project or assignment on which a student violates these policies will be rejected without option to resubmit. 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 credit will be given for late work.

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 during grading may be different from that provided to students, to prevent students from simply writing code that apes the responses the specific unit tests check for. **Extra credit** shall 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** shall be awarded if the instructor's own solution fails the new test. Moreover, **and this is important**, that new test case shall 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.

CODE STYLE. To encourage students to practice good programming style, the instructor reserves the right to assess submitted project code for style and efficiency, to be incorporated into the *Other* score.

TESTS USING COMPUTERS. For Tests 2 and 4, the computer science lab (Meyer 154) shall be reserved for our use. The lab machines shall be logged into special accounts for the purpose of test-taking. Internet access shall be turned off. Students shall use only Eclipse and a web browser; the browser shall be used only to view a local copy of the Java API. Students shall 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 block, when Tests 3 and 4 are held, is Tuesday, May 7, at 10:30 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.*

Confidentiality and mandatory reporting. I'm committed to help maintain a safe learning environment on campus. As a faculty member I am required to share with College authorities any information about sexual misconduct that may have occurred on Wheaton College's campus. Confidential resources available to students include Confidential Advisors, the Counseling Center, Student Health Services, and the Chaplain's Office. More information on these resources and the college's policies is available at `www.wheaton.edu/sexualassaltresponse`.

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: If you have a documented need for accommodations, I will have received a letter on your behalf from the Disability Services Office. But *please talk to me* about what accommodations are most useful to you. In particular, if you desire accommodations for test-taking, talk to me a reasonable amount time in advance (say, at least two class periods) so arrangements can be made.

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. My intent is for class to be an electronic-device-free zone. Please keep all laptops, tablets, phones, etc, silenced and put away. If you absolutely need to check your phone for something, please discreetly step out in to the hall. **NO TEXTING IN CLASS.**

All this, the Lord willing.