

COS 226: Introduction to Data Structures

Sudarshan S. Chawathe

University of Maine

Fall 2007

Data structures are schemes that organize data to permit efficient access in certain modes. The desired modes of access (different kinds of look-ups and modifications), and their relative importance in an application, typically guide the choice of existing data structures and the design of new ones. A judicious choice of data structures often results in very significant improvements in the running time of a program. In order to make such decisions, as well as to design new data structures, we need to understand existing data structures, their access modes, and performance characteristics. In this course, we study data structures from several perspectives, including design, analysis, and application.

News and Reminders:

- Please read the newsgroup for timely announcements.
- Class newsgroup: Local group `umaine.cos226` on NNTP server `news.cs.umaine.edu`. Web interface to get started: <http://cs.umaine.edu/~chaw/news/>.
- Please use the PDF version of this document for printing and reference: `cos226.pdf`

Goals

- Understand several popular data structures and their properties.
- Learn how to use data structures and other tools to solve problems in various application areas.
- Practice the appropriate and ethical use of existing material of different kinds, such as source code, services, and documentation.
- Gain experience in contributing to the body of knowledge.
- Learn how to analyze of the running times of programs using simple mathematical methods.

- Gain experience in conducting and documenting experimental studies of programs.
- Improve our programming skills, with attention to software engineering principles.
- Improve our communication skills, with particular emphasis on written communication and, further, well-written programs.
- Learn how to manage a self-directed project.

Contact Information

Class meetings:

Time: Tuesdays & Thursdays, 2:00–3:15 p.m.

Location: Neville Hall, Room 227.

Instructor: Sudarshan S. Chawathe

Office: Neville Hall, Room 224.

Office hours: (Please check for changes.)

Tuesdays & Thursdays: 10:30–11:00 a.m.,
1:45–2:00 p.m., 3:15–4:00 p.m.

Phone: +1-207-581-3930.

Email: chaw@cs.umaine.edu

Use email only for messages unsuitable for the newsgroup. (See below.) Please put the string *COS226* near the beginning of the Subject header of your messages to me.

Web: <http://cs.umaine.edu/~chaw/>.

Teaching Assistant: Mark Royer

Office: East Annex, Room 229.

Office hours: (Please check for changes.)

Mondays & Wednesdays: 1:00–4:00 p.m.

Phone: +1-207-581-2005.

Email: mroyer@cs.umaine.edu

Online Resources

Class Web site:

<http://cs.umaine.edu/~chaw/cos226/>

We will use the class Web site for posting homework assignments, hints, solutions, etc. Please monitor it.

Class Newsgroup: We will use the local USENET newsgroup `umaine.cos226` on the NNTP server `news.cs.umaine.edu` for electronic discussions. If you are unfamiliar with USENET, you may find the Web interface at `http://cs.umaine.edu/~chaw/news/` useful as a quick way to get started. You may find further information on USENET at `http://en.wikipedia.org/wiki/Usenet`. The newsgroup is the primary forum for electronic announcements and discussions, so please monitor it regularly, and post messages there as well. Unless there is a reason for not sharing your question or comment, please *use the newsgroup, not email*, for questions and comments related to this course.

Class mailing list: *Please make sure you are on the class mailing list.* A sign-up sheet is circulated at the first class meeting. If you miss it, please contact me to get on the list. We will use this mailing list only for urgent messages because all other messages will go on the class newsgroup. I anticipate fewer than a dozen messages on this list over the semester.

Grading Scheme

Grades: Grades are based on class participation (5%) homeworks (25%), two mid-term exams (15% each), a final exam (20%), and an independent project (20%).

Class participation: Students are expected to contribute to learning by asking questions and making relevant comments in class and on the class newsgroup. Quality is more important than quantity. Disruptive activity contributes negatively. Please make sure all disruptive devices are disabled while in class. If you have a good reason for wanting to be disturbed in class, please contact me to make the appropriate arrangements.

Homeworks: Homeworks include programming and non-programming ones, often mixed. No collaboration is permitted. You are allowed, and encouraged, to discuss the problems and solution strategies *at a high level*, but the final solution

and details must be your individual work. If you are unclear on the boundary between permissible and non-permissible interactions in this regard, please ask me.

Exams: All exams are *open book, open notes*. You are free to bring with you any resources that you find useful. However, no communications are permitted other than between students and me. The use of computers during exams is strongly discouraged, but brief use is permitted *provided it does not cause a disturbance*. You may use the Internet, but only as a library to look up material you may find useful. As above, check with me if you are unclear on what is permitted. The exams are designed to require no equipment other than a pen and paper.

Project: In addition to the programming and other homeworks, this course features a semester-long independent project. You may work either individually or in groups, although I encourage the latter. The details of the project are fairly flexible, and you are encouraged to propose a project that excites you. I will also propose a few projects that you could use, perhaps with some of your own modifications. The main requirement for the project is that it demonstrate the ability to work independently and apply the concepts studied in the course to an application. Projects will be graded based on a written project report, the submitted source code, a demonstration, and a question-and-answer session following the demonstration. Further details will follow.

Policies

Special needs: If you have special needs of any kind, including, but not limited to, disabilities, absences due to participation in sports or other activities, etc., please contact me *as soon as the need is known to you* and I will try to accommodate them as much as possible.

Attendance: Although I expect students to attend all class meetings, I will not be taking attendance. If you miss a class meeting, you are responsible for making up the lost material. If you have a valid reason for missing a class, let me know early and I will try to help you make up the class.

Make-up classes: I may have to reschedule a few classes due to my other professional commitments. I will make every attempt to minimize the number of such occurrences and to reschedule for a time that works for most students. Further, I will make sure no student is penalized by such occurrences.

Due dates: All due dates are strict, as announced in class. If you believe your work was delayed by truly exceptional circumstances, let me know as soon as those circumstances are known to you and I will try to make a fair allowance. However, *the default is that you get a zero if you don't turn in the work on time.*

Academic honesty: I expect you to hold yourselves to the highest standards of academic honesty. Please take this point very seriously. If you are not sure if something is permitted, check with me. *All help you receive, even if permitted, must be prominently noted in all work you submit.* Erring on the side of giving too much credit is far better than the alternative. Plagiarism and other forms of cheating will result in very stiff penalties (including, but not limited to, an F grade in the course and further disciplinary action from the university).

Programming

The focus of this course is on data structures, algorithms, algorithm analysis, and problem solving techniques in Computer Science, and not on programming, much less programming in a particular language. Programming is, however, a valuable part of the course as it helps us solidify the abstract concepts we study. We will use Java as the primary programming language. Submissions will be in the form of packaged Java *source* files.

Programming Environment and Tools: You are free to choose details such as operating system, development environment, and editor based on your preferences. However, no matter what you use, the result should be a source-file package that works on any platform supported by Java. Further details on the packaging, submission, and testing procedure appear below, on page 5.

Other Languages: If you prefer to use a language other than Java, please contact me. I am quite open

to the idea, and encourage interested students to explore it further. However, please check with me very early in the course so that we can go over the specifics to make sure your submissions can be tested and graded fairly. You should avail of this option only if you are confident enough of your programming skills to not require any programming help.

Literate Programming: All submitted work must use a *literate programming style*: Your programs must be designed with a human as the intended reader, although they must also compile and run correctly. *Programs that do not meet this requirement are likely to receive a zero score with no further consideration.* Details will be discussed in class. The use of any specific literate-programming or documentation tool is neither necessary nor sufficient for this requirement.

Class Accounts: Although the use of official class accounts (on department computers) is not required (see above), it is a good idea for everyone to have accounts on both our main Unix host (gandalf) and the cluster of PCs. Class accounts will be generated based on the forms distributed at the first class meeting. If you miss them, please get in touch with me. You should be able to access your gandalf account from anywhere on the Internet (including the labs in Neville Hall and elsewhere on campus) by using *ssh* to connect to `cs.umaine.edu`. On most Unix hosts, the command `ssh -l username cs.umaine.edu` should suffice. For Windows hosts, the freely available *PuTTY* program works well: <http://www.chiark.greenend.org.uk/~sgtatham/putty/>. *Do not use unencrypted telnet sessions to connect to your account!*

Schedule

At the beginning and end of each class, I typically announce sections of the textbook covered in each class and those due at the next class. An approximate schedule appears in Figure 1. Please use it only as a rough guide to plan your studies. *Do not* use it to schedule travel or other events. If you need a definite answer on when something will or will not occur, you should check with me.

#	Date	Material
1	<i>09-04</i>	Introduction 1; trees; 18.0–18.3.
2	-06	Introduction 2; traversals; binary search trees; order statistics; 18.4–18.end., 19.0–19.2.
3	-11	Analysis of algorithms; maximum contiguous subsequence; 5.0–5.3.
4	-13	Static searching; further analysis; 5.4–5.end.
5	-18	BST analysis, AVL trees; 19.3–19.4.
6	-20	Red-black trees; AA-trees; 19.5–19.6.
7	-25	B-trees; disk data structures; catchup; 19.7–19.end.
8	-27	catchup; review.
9	<i>10-02</i>	Midterm Exam 1.
10	-04	Special tutorial.
	-09	No class (Fall break Oct. 6th–9th).
11	-11	Midterm discussion; AA-trees; 19.6.
12	-16	AA-trees; B-trees; 19.6, 19.8.
13	-18	B-tree; binary heaps; 19.8, 21.1–21.3.
14	-23	Splay trees; 22.1–22.2.
15	-25	Splay trees; 22.3–22.4.
16	-30	catch-up; review.
17	<i>11-01</i>	Midterm Exam 2.
18	-06	Midterm discussion; skew heaps 23.1.
19	-08	Pairing heap; 23.2.
20	-13	Hashing; 20.1–20.4.
21	-15	Hashing; 20.5–20.7.
22	-20	Graphs; shortest paths; 14.1–14.3.
	-22	No class (Thanksgiving break Nov. 21st–25th).
23	-27	Graphs; shortest paths...; 14.4–14.5.
24	-29	Sorting; 8.1–8.4.
25	<i>12-04</i>	Sorting; selection; 8.5–8.8.
26	-06	Special topic.
27	-11	Tutorial.
28	-13	Review.
29	12-20	Final Exam; 12:15 p.m.–2:15 p.m. ; location TBA.

Figure 1: **Approximate Schedule.** The numbers refer to sections in the textbook. Tuesday dates are in italics.

Textbook and Readings

Textbook: Mark Allen Weiss. *Data Structures and Problem Solving Using Java*. Addison-Wesley, 3rd edition, 2006. The university bookstore carries this book, which is a “required textbook” for this course. The edition is important.

The core topics for this course are found mainly in Chapters 18 and beyond; a few earlier chapters, such as 5, 8, and 14 are also relevant. Detailed coverage information will be announced as we progress in the semester. Most chapters in roughly the first third of the textbook, as well as some later chapters, discuss topics that are covered in the prerequisite course, COS 225. We will not be covering these topics in this course.

Other Readings: All the following are recommended, but not all are required. Further details and additional readings will appear here.

1. Gilad Bracha. Generics in the Java programming language. Tutorial. <http://java.sun.com/>, July 2004.

This tutorial is optional reading but I strongly suggest that everyone read it. The concepts explained here are essential for making good use of generics in Java and it is very painful to learn them the hard way (e.g., while debugging your code).

2. Sudarshan S. Chawathe. Segment-based map matching. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, Istanbul, Turkey, June 2007.

The main purpose of this paper, for this course, is providing a concrete example how data structures and related concepts find use in current research and applications. Sections I, II, and III are required reading. The rest of the paper is optional reading.

3. Derrick Coetzee. An efficient implementation of Blum, Floyd, Pratt, Rivest, and Tarjan’s worst-case linear selection algorithm. <http://moonflare.com/>, January 2004.

4. Jon Bentley and Don Knuth. Programming pearls: Literate programming. *Communications of the ACM*, 29(5):364–369, May 1986.

5. Paul E. Black. Dictionary of algorithms and data structures. <http://www.nist.gov/dads/>, September 1998.
6. Lloyd Allison. Suffix trees. <http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Tree/Suffix/>, 2000.
7. Samuel W. Reynolds. A generalized polyphase merge algorithm. *Communications of the ACM*, 4(8):347–349, 1961.

This paper provides a succinct and readable description of polyphase merging. It is a very useful supplement to the description in the textbook, which is missing many important details.

Homeworks, Tests, and Notes

Homework assignments, exams, and solutions will appear here as we move along the semester.

- Homework 1: [hwq/hw01.pdf](#).
- Homework 2: [hwq/hw02.pdf](#).
- Midterm 1: [hwq/mt01.pdf](#); sample solutions: [p/mt01s.pdf](#).
- Homework 3: [hwq/hw03.pdf](#).
- Homework 4: [hwq/hw04.pdf](#).
- Midterm 2: [hwq/mt02.pdf](#); sample solutions: [p/mt02s.pdf](#).
- Homework 5: [hwq/hw05.pdf](#).
- Homework 6: [hwq/hw06.pdf](#).

A few ideas for term projects: [notes/projideas.pdf](#)

Homework Submission

Handwritten answers to on non-programming problems should be submitted in class on the due date, at the beginning of class, unless prior alternate arrangements are made. If you prefer to type your answers, please make sure the result uses the proper symbolic notation for relational algebra operators and other constructs. Illegible or hard to read submissions, whether handwritten or typed, are likely to be returned without grading, for zero credit. Answers to programming problems should be submitted electronically, using the packaging and submission procedure outlined below.

Packaging Each package you submit should include a **README** file and a **Makefile** (at the top level of the subdirectory described later). The **README** file is a *plain text*, *ASCII* file (not a word-processing file) that includes a brief description of your submission, compilation instructions, and any special instructions or other information relevant to your submission. At the very least, it should include information identifying the submission and its author. The **Makefile** is used by the program **make** to determine how to build object files and other derived files from source files. (For more information, type **man make** at a shell prompt.) A very simple **Makefile**, suitable for typical Java submissions, appears in Figure 2. You may need to edit it to suit your purposes. You may find copies of this and other files in the Sample Code section on page 7.

For most of the programming assignments, we will use the **jar** archiving tool that is part of most Java environments (such as Sun’s JDK). The following directions for creating a jar package assume a Unix-based operating system but you may use them on Windows systems by replacing forward slashes (/) with backslashes (\) in the paths.

Each submission should consist of a single jar archive file. You should generate a jar archive called **M-hwX-N.jar**, replacing **M** with your last-name, **X** with a two-digit homework number, and **N** with an arbitrary 4-digit integer (e.g., **Doe-hw01-4242.jar**).

The jar file should contain a *single subdirectory* whose name is obtained by excluding the **.jar** suffix from the name of the jar file (e.g., **Doe-hw01-4242**). In turn, this directory should contain the source files, **README**, and **Makefile** of your submission. For example, the **Doe-hw01-4242** directory may contain the following.

```
Makefile
README
Test.java
Driver.java
```

The jar file should not contain any non-source files, such as object code, .class files, and binary executables.

While various IDEs, such as *Dr. Java*, have built-in jar creation functionality, please use the command line **jar** tool to create the jar files, since it is simpler and more consistent across platforms. In particular, many IDEs produce, by default, jar files that contain either only **.class** files or both **.class** and **.java** files, without a **Makefile** or **README** file, and are thus unsuitable.

```

CLASSPATH=.

default: all

all:
    @javac *.java

clean:
    @find ./ \( -name "*~" -o -name "*.class" \) -exec rm '{}' \;

```

Figure 2: A simple Makefile.

Assuming that you have created a directory `Doe-hw01-4242` as a subdirectory of the working directory, and that this directory contains the necessary submission files (and no others), you may create a suitable jar file using:

```
jar -cvf Doe-hw01-4242.jar Doe-hw01-4242
```

The resulting jar file, `Doe-hw01-4242.jar`, will include a Manifest file called `MANIFEST.MF`. A manifest file describes the contents of a jar file. The default manifest is fine for most purposes, but in the future you may wish to specify an alternate manifest file. More information about creating manifests and jar files can be found at <http://java.sun.com/docs/books/tutorial/deployment/jar/build.html>.

Testing procedure: We will use a procedure similar to the following to set up your submission for grading using the jar file you submit. *Please test carefully that the following procedure works with the jar file you plan to submit.* You should test your work in a temporary directory, not the directory containing your code or other material. Begin by creating a temporary directory for testing as a subdirectory of your home directory on `gandalf`, and making it the working directory:

```
cd
mkdir testdir
cd testdir
```

Copy your jar file to the newly created directory. For example, if your jar file resides in a subdirectory `hw01` of your home directory, type

```
cp ~/hw01/Doe-hw01-4242.jar .
```

Now extract the jar file by typing

```
jar -xvf Doe-hw01-4242.jar
```

The above command should result in the creation of a directory `Doe-hw01-4242` as a subdirectory of the current working directory, which is `/tmp/testdir` in our case. Now you should be able to change to the newly created directory by typing

```
cd Doe-hw01-4242
```

Typing `ls` should display the contents of the newly created directory. Typically, these contents include Java source files (`*.java`), a `Makefile`, and a `README` file. *Please make sure there are no class files, object files, or other compilation products in the directory at this point.* (Your submission should contain only source files.) Finally, typing

```
make
```

should result in the complete compilation of your program. Using the sample `Makefile` of Figure 2, typing

```
make clean
```

will remove any class files that the compilation creates.

Submission Once you have created a jar file as described above, you should upload it using anonymous FTP (`anonymous` as the user name and your email address as the password) to the FTP server `cs.maine.edu` in directory `/incoming/cs/cos226/`. (Detailed FTP instructions appear below.) If you need to upload an updated version of your submission for any reason, you may repeat the above using a different four-digit integer in the file name. (If you try using the same file name as your earlier submission, the upload will likely fail.) You will not be able to list the FTP upload directory (standard security setup), so pay attention to the diagnostic messages from your FTP program. If the messages indicate success, your

file will have been uploaded. *Be sure to use the binary transfer mode when transferring binary files (including jar files) using ftp. Otherwise your files will be corrupted.*

You must upload the file before you submit your hard-copy homework. *Write down the name and size (in bytes) of the file you submitted near your name on your hardcopy submission for the non-programming problems.*

FTP on Unix-based systems: On your local machine (laptop, home computer, etc.), set the working directory to directory containing your jar file, which we will call `Doe-hw01-4242.jar` below. Then type

```
ftp cs.umaine.edu
```

at the shell prompt. The rest of the commands below are to be issued from within the `ftp` program, at various prompts. You should first be prompted for a user name, when you should type

```
anonymous
```

Next, you should be prompted for a password, when you should type your email address, for example,

```
foo@bar.baz.net
```

You should ensure that the file transfer is in binary and passive modes by typing

```
binary
passive
```

Set the working directory on the server side to the submission directory by typing

```
cd /incoming/cs/cos226
```

Now upload your file by typing

```
put Doe-hw01-4242.jar
```

substituting your own file name for `Doe-hw01-4242.jar`. Finally, exit the `ftp` program using

```
quit
```

which should return you to the shell prompt. As with most Unix commands, you can find out more about `ftp` by typing `man ftp` at a shell prompt. If you wish to transfer files to and from your user account on `gandalf`, you should transfer them over a secure connection. Use the `scp` program for this purpose. For details, type `man scp` at a shell prompt. The graphical tool `gftp` may also be useful.

FTP on Windows systems: You may use the ftp client *FileZilla*, which is available at <http://sourceforge.net/projects/filezilla/>. In *FileZilla*, create a connection to the FTP server using the menu `File->Site Manager`. Click on `New Site` and type `cs.umaine.edu` as the host name. For `Servertype`, select `SFTP using SSH2`. Specify your `gandalf` username and click `connect`. You will be prompted for your password and, once entered, your home directory on `gandalf` will be displayed. You can now copy files back and forth between your local machine and `gandalf`. Since you are likely to be transferring binary files, you should select `Transfer->Transfer Type->Binary`, else your transferred files will be corrupted. For more information on *FileZilla*, use the `Help` option in its file menu, or see <http://filezilla.sourceforge.net/documentation/>.

Sample Code

Sample code and other files will appear here.

- `code/Tree.java`: Tree interface for Homework 1.
- `code/TreeNode.java`: `TreeNode` interface for Homework 1.
- `code/TestMyTree.java`: A simple driver and test program for Homework 1.
- `code/Makefile`: A simple makefile, suitable for Java.

Additional material, such as recently added files as announced in class or on the newsgroup, may be found in the code subdirectory:

```
http://cs.umaine.edu/~chaw/cos226/code/
```