# A Comprehensive Project for CS2: Combining Key Data Structures and Algorithms into an Integrated Web Browser and Search Engine

**Tia Newhall and Lisa Meeden**
**Computer Science Program**
**Swarthmore College**
**Swarthmore, PA 10981**
**{newhall, meeden}@cs.swarthmore.edu**

## Abstract

We present our experience using a large, real-world application as a course project for the second half of the semester of a CS2 course. Our primary goal for the project was to create an engaging application that incorporated most of the key data structures and algorithms introduced in the course. Specifically, the project uses binary search trees, priority queues, hash tables, and graphs. The project consisted of four parts combined to build an integrated web browser and search engine in Java. A key benefit of an incremental, long-term project of this type is that students quickly learn that their initial design and implementation decisions have a significant impact on the eventual extensibility and performance of their software. This provides numerous opportunities for students to recognize the importance of software engineering techniques and complexity analysis in the development of a successful application. We present students' responses to the project which show that they overwhelmingly enjoyed the project and felt that it helped them to see how the data structures and algorithms discussed in the course are used in real software.

## 1 Introduction

At Swarthmore College, the computer science curriculum offers a broad exposure to the discipline through three introductory courses, which could be termed CS1, CS1.5, and CS2. CS1 takes an imperative approach using C, CS1.5 takes a functional approach using Scheme with Abelson and Sussman's text [1], while CS2 takes an object-oriented approach using Java with Goodrich and Tamassia's text [3]. Students who have already had significant exposure to computer science in high school are encouraged to skip CS1 and begin with either CS1.5 or CS2. All majors are required to take CS1.5, but many non-majors take only the CS1-CS2 sequence.

Our version of CS2 is called *Algorithms and Object-Oriented Computing*. Topics for this course include the philosophy of object-oriented programming, the basics of algorithmic analysis, and the classic data structures and their associated algorithms.

According to the Steelman draft of the proposed 2001 computer science curriculum [4], the "introductory computer science experience should certainly expose students to the design, construction, and application of computer systems" [4, p. 25]. This statement argues for the inclusion of some significant programming projects into the introductory curriculum.

Many institutions have already taken this project-based approach in their CS2 courses [6, 7, 2]. A survey of these project proposals reveals that there are many issues to consider when developing a course project. For instance: should the project span the entire term or a portion of the term; should the instructors develop the overall system architecture or provide the students with only an open-ended problem description; and should all of the students work on the same problem or work on unique pieces of a larger problem?

In developing our CS2 project, we chose to have our project span half the term, we provided the overall system design to the students, and we had each student team work on the same problem. In planning for the CS2 course, we followed the schedule of topics shown in Table 1. We felt that the early topics, especially Java programming and complexity analysis, were best served by short-term homework assignments where students could get immediate feedback on how they were doing. In addition, students need to develop some comfort with the object-oriented paradigm before they can begin to see how a larger system might be constructed. The project began in week 7 and continued through the end of the term. In order to meet our goal of incorpo-

| Week | Topic |
|---|---|
| 1 | Introduction to Java and OOP |
| 2 | Java and OOP continued |
| 3 | Complexity analysis |
| 4 | Stacks and queues |
| 5 | GUI programming in Java |
| 6 | Lists |
| 7 | Trees |
| 8 | Trees continued |
| 9 | Priority queues and heaps |
| 10 | Dictionaries and hashing |
| 11 | Dictionaries and hashing continued |
| 12 | Graphs |
| 13 | Graphs continued |
| 14 | Searching and sorting |

Table 1: CS2 Schedule of Topics

rating all of the major data structures presented in the course into the project, we needed to define the overall system architecture in advance. Finally, we wanted each student to have hands-on experience with every key data structure, so we chose to have all of the students work on the same problem.

Although we found the Goodrich and Tamassia text [3] to be particularly good in its presentation of new data structures and algorithms (we really liked its use of pseudo code, its illustrative examples, and its focus on complexity analysis), we did not use the authors' Java class library for implementing the data structures in our course. Instead, we tried to conform to Sun's Java class library as much as possible. In particular, we modeled many of our abstract data types after the Java Collections Package.

Our approach to teaching this material was to first present each new data structure as an abstract data type. We then considered a variety of possible implementations for that abstract data type. For example, students were given a Java interface called `BinarySearchTree` and had to experiment with two different implementations of this interface called `LinkedBinarySearchTree` and `ArrayBinarySearchTree`. Different implementations of the same abstract data type were then compared in terms of the efficiency of their operations.

## 2 Project Goals

The primary goal of our CS2 project was to reinforce the data structures and algorithms presented in class through the creation of a challenging and fun application. We also hoped to accomplish a number of secondary goals: to demonstrate the importance of complexity analysis in determining performance efficiency when considering design alternatives; to illustrate the benefits of software engineering techniques; and to provide an opportunity for students to work in teams.

We found that the Java-based web browser and search engine project lends itself well to meeting all of these goals. To meet the primary goal of reinforcing the lecture material, each part of the project focused on one key data structure from the second half of the semester: binary search trees, priority queues, hash tables, and graphs. To address the importance of complexity considerations, successive parts of the project enhanced the efficiency of previous parts and required students to provide written analyses of these improvements. It proved to be quite simple to illustrate the benefits of software engineering. As students had to reuse their own code over the course of two months, they came to recognize the importance of good design and clear commenting. The large scope of the project demonstrated the benefits of teamwork as students found that having a partner provided help with design, coding, and debugging.

An additional benefit of this type of project is that students get excited about implementing the course programming assignments. The reward of seeing their large final application work keeps them interested and motivated to work on the individual parts. Breaking up the large project into smaller individual assignments makes what seems like a daunting design and coding task manageable for this level of student.

Such a project can also be used to challenge some of the more ambitious students by providing extra credit parts that focus on adding functionality or improving performance in ways that may go beyond the scope of the course. This helps to keep all students challenged and excited about the project.

The project proved to be an excellent vehicle for demonstrating how data structures and algorithms are used in practice, as well as for developing programming, communication and problem solving skills in students. The fact that the students really enjoyed the project led them to think critically about the design issues even more than we had hoped. Many students came up with ways in which the search engine and browser could be improved and extended beyond the course assignment.

## 3 The Project

The proposed 2001 curriculum notes that "technological advancement over the past decade has increased the importance of many curricular topics, such as ... the world wide web and its applications" [4, p. 8]. Most first year college students are already very adept, perhaps more adept than their instructors, at using the web and its applications. Creating their own versions of tools which they use on a nearly daily basis is an

exciting proposition for the students.

Upon completion of the project, the web tool that the students construct has the following capabilities for a limited portion of the web situated on our local server:

- Display a web page given a URL.
- Display connectivity information of web pages.
- Answer questions about connectivity of web pages.
- Find matching web pages given a query and display the resulting URLs in order of best match to worst.
- Automatically display the best matching URL result of a search.

The project was divided into four parts, which are described in the following subsections. The entire project, as provided to the students, is available on the web [5].

### 3.1 Analyzing the content of web pages

The first step to building a search engine is to analyze web pages based on their content. This will allow us to rate how relevant a particular page is for a given user request. The content of a page is obviously correlated with the words it contains. For this portion of the project, students used a binary search tree to store words and calculate their frequencies in a given web page. In class, students were asked to consider why a binary search tree is a better data structure for this task than a list. They were also asked to develop a list of words that should be ignored during this analysis, such as common short words and `html` tags.

Given a file name of a web page as input, this initial piece of the project simply outputs all of the words (which were not on the ignore list) that occurred at least a minimum number of times.

Students were given a `Scanner` class to parse text files, which included a `getNextToken()` method. Students were also given the binary search tree abstract data type, but had to complete aspects of the linked implementation such as the insertion method.

### 3.2 Processing user queries to locate relevant web pages

Given the word frequency counts for a web page, the search engine must determine how to rate a page's relevance to a query. Initially, we took a rather simplistic approach to this task—a web page's relevance was defined as the sum of the word frequency counts of each word that appears in the query. For example, if the query is *artificial intelligence* and a web page contains the word *artificial* 3 times and the word *intelligence* 5 times, than its relevance is 8, where higher scores indicate more relevance.

Given a file containing a list of URLs as input, this portion of the project initiates a text-based interaction loop with the user that requests a query, responds to the query, and repeats. Initially, each URL contained in the given file is converted to a file name, the file is analyzed for its content, and the resulting word frequency tree is saved. To process a query, the relevance of each URL is calculated using the saved word frequency trees and a priority queue of the URLs is created which is ordered by their relevance values. Using this priority queue, the program outputs the relevant URLs in order of highest to lowest relevance.

Students were given the priority queue abstract data type, but had to complete aspects of the heap implementation. A number of students were dissatisfied with the simplistic relevance measure and experimented with other ways of calculating relevance, such as adding a bonus if all the query words appeared in a web page.

### 3.3 Adding a cache of query results and creating a GUI front-end

Because they were familiar with search engines such as Yahoo, Lycos and Google, students quickly realized that our unsophisticated search technique would not scale to the entire world wide web. However, by caching previous results, the search engine would be able to significantly improve its average response time by reusing previously calculated relevancies.

The cache was implemented as a hash table where the key was a string containing an entire query or an individual word in a query. Associated with each key was a `SavedResult`. Each `SavedResult` contained another hash table where the key was a URL with an associated word frequency count.

The cache was used to completely or partially compute query results. For example, if a user entered the query *artificial intelligence*, the search engine would first check its cache to see if it had seen this same query before. If no similar query is found, then the search engine would see if it had any saved results for the individual words in the query: *artificial* or *intelligence*. If a user had previously searched for *artificial christmas trees*, the search engine would have cached the results for the entire query as well as the results for each individual word. The engine would use the cached relevancy scores for the word *artificial* in every URL and would calculate the relevancy scores for the word *intelligent* in every URL and then merge the contents of the two secondary hash tables to produce the final search result for *artificial intelligence*.

In addition to creating a cache for the search engine, students replaced the previous text-based interface with a graphical user interface. The graphical user interface helps to keep students excited about the project because they enjoy adding their own personal touches to the look and feel of the application. More importantly though, the GUI component forces them to think about problems of good user interface design, and it allows us to introduce event-driven and threaded programming models.

Upon the completion of this portion of the the project, the students have a functional search engine and web browser for our local domain. Most students added support for displaying any URL on the entire web, which is trivial to do using Java. Obviously, other improvements would be necessary before this could be a viable tool for the entire web.

### 3.4 Adding a hyperlink graph for examining web connectivity

For the final part of the project, students added a graph of URL links to their web browsers. Given a starting URL as input, the graph is created by parsing the associated web page and finding `href` links to other local web pages, parsing them, and so on, until a given link depth is reached. The graph contains a vertex for each URL and an edge between URLs if there is a hyperlink between them.

Once this connectivity graph was built, the students created an additional GUI to allow a user to examine features of the local web's structure. Given a URL, a user can find all other URLs reachable from that point and the shortest paths from that URL to any other reachable URL. A user can also print a textual representation of the entire graph.

More importantly, this connectivity information was integrated back into the search engine. If a web page that matches a query is linked-to by many other web pages, then its relevancy should be increased. It was left up to the students to decide how best to incorporate linked-to counts into the final relevancy score for a URL.

We had students submit their projects as if they were releasing their web browser and search engine as software. Students created a web page that described how to use their web tool, including details on individual features and answers to questions that we asked about ways in which their search engine could be made better. In addition, students made their `.class` file solutions available for us to run and test rather than submitting their `.java` code. This way, students were forced to document how to use all of the features implemented in their program to ensure that we would test them.

### 3.5 Extra credit

There were several extra credit options for the project that add functionality to web browser and search engine. The extra credit extensions include:

- Adding Home, Back, and Forward buttons to the web browser.
- Enabling links in the displayed web page and in the list of URL results displayed by the search engine; clicking on a link results in its page being displayed by the web browser.
- Adding support for building the link-graph from any starting URL (not just from our local domain). This involves adding support for loading any web page from the world wide web and parsing the loaded web page to find links that are then used to load and parse subsequent web pages.

To date, several students have completed the first two extra credit extensions, but no one has implemented the third extension. Additional extra credit parts that make the search engine more efficient or implement better criteria for ordering good matching URLs could be added.

## 4 Student Response to the Project

One of the most satisfying results of using this project was that, besides meeting its pedagogical goals, students really enjoyed it. It is so much easier for them to learn the material if they are excited and motivated to do the work. One student commented that "building the Web Browser was cool and fun! My friends at other schools were very impressed."

Students' written responses to end of the semester evaluations about the project were overwhelmingly positive. Most felt challenged, but not over burdened, by the project. Many said they were initially a bit concerned about the scope of the project, but once they started working on it they were surprised to find that they could do it, and they felt a great deal of satisfaction upon completing it. Almost all students were able to complete the full project. Students who did not complete a part had access to our compiled solutions so that they could continue on to the next part.

Students felt that the project helped to reinforce the data structures and algorithms presented in class. All "of [the project parts] required understanding the [data] structures, both at theoretical and practical levels." Another student said, "in general, the homework assignments were great in helping to reinforce the material covered in lecture." Students felt that assignments that

included written analysis of the algorithms were particularly helpful in reinforcing their understanding.

Students said that they liked that individual assignments incrementally built one large piece of real world software: "[I liked that] each was in some sense its own assignment, yet still cumulative, and had a distinct functionality that was fun to get working." In addition, students mentioned that they learned good software design in the process of implementing this project: "building a very large piece of software gave a better look at OO design and forced us to use good coding style [and] techniques."

The vast majority of students enjoyed working with a partner and felt that it was beneficial to them. A few students mentioned some of the difficulties of working with a partner, such as being dependent on someone else's schedule and having to agree on an implementation. Overall, they felt it was a good experience despite a few difficulties. Students mentioned the benefits of seeing different coding styles, reducing the amount of debugging time, dividing the work, and the chance to discuss design decisions and come to a consensus. One student commented that working with a partner "allowed me a very clear view of what we were doing at any point."

## 5 Conclusions

We found that using a single, large, real-world application as a second half of the semester project worked extremely well in our CS2 course. Giving students practice learning object-oriented programming with smaller assignments in the beginning half of the semester, resulted in students feeling more confident in their ability to complete the larger project assignment during the second half of the semester. Our approach of giving students starting points to each piece of the project allowed students to implement a complete web browser and search engine in only seven weeks. In addition, it allowed students to focus on implementing the parts of the assignment that involved the more difficult and important algorithms without overwhelming them with a large volume of code to write.

We have used this project in our CS2 course for the past two semesters. Responses from students in the first semester led us to re-design a few of the parts, and the second semester students had few, if any, problems with understanding or completing the project assignments. Based on the overwhelmingly enthusiastic responses we have received from students, we plan to continue to use this project in our CS2 course.

## References


[1] Abelson, H., and Sussman, G. J. *Structure and Interpretation of Computer Programs, Second Edition.* McGraw Hill, 2001.

[2] Godfrey, M., and Grossman, D. JDuck: Building a software engineering tool in Java as a CS2 project. *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (1999).

[3] Goodrich, M. T., and Tamassia, R. *Data Structures and Algorithms in Java, Second Edition.* John Wiley and Sons, Inc., 2001.

[4] Joint Task Force on Computing Curriculum. Computing curricula 2001: Steelman draft, August 2001. www.acm.org/sigcse/cc2001/steelman/.

[5] Newhall, T., and Meeden, L. Building a web browser and search engine in Java: A half semester project for a CS2-type course. www.cs.swarthmore.edu/~newhall/sigcse02/cs2project.html.

[6] Rebelsky, S. A., and Flynt, C. Real-world program design in CS2: The roles of a large-scale, multi-group class project. *Proceedings of the Thirty-First SIGCSE Technical Symposium on Computer Science Education* (2000).

[7] Turner, J. A., and Zachary, J. L. Using course-long programming projects in CS2. *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (1999).