

Teaching Key Topics in Computer Science and Information Systems through a Web Search Engine Project

MICHAEL CHAU

The University of Hong Kong

ZAN HUANG and HSINCHUN CHEN

The University of Arizona

Advances in computer and Internet technologies have made it more and more important for information technology professionals to acquire experience in a variety of aspects, including new technologies, system integration, database administration, and project management. To provide students with a chance to acquire such skills, we designed a project called “Build Your Search Engine in 90 Days,” in which students were required to build a domain-specific Web search engine in a semester. In this paper we review the tools and resources available to students and report our experiences in having students to work on this project in a course at the University of Arizona. We also review two tools, called AI Spider and AI Indexer, we developed for students in this project. We highlight a few search engines that were created by the students and suggest some future directions in improving the tools and expanding the project.

Categories and Subject Descriptors: D.2 [**Software**]: Software Engineering; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.3.5 [**Information Storage and Retrieval**]: Online Information Services; H.3.7 [**Information Storage and Retrieval**]: Digital Libraries; H.5.4 [**Information Interfaces and Presentation**]:

Hypertext/Hypermedia; K.3.2 [**Computers and Education**]: Computer and Information Science Education
General Terms: Design

Additional Key Words and Phrases: Web search engine, Web computing, education, Web spiders, indexing

1 INTRODUCTION

System development projects often require that information technology (IT) professionals such as programmers and systems analysts be proficient in more than one area. Real-world information system projects often require knowledge of databases, backend servers, and Web-based interface, among others. It has become increasingly important for IT professionals to know not only the individual technologies but also how to choose the best ones and integrate them into system development. This requires developers to possess skills in system design, project management, database administration, and personal communication.

This research was supported in part by the NSF Digital Library Initiative-2, “High-performance Digital Library Systems: From Information Retrieval to Knowledge Management,” IIS-9817473, April 1999 – March 2002; and by the NSF National SMETE Digital Library, “Intelligent Collection Services for and about Educators and Students: Logging, Spidering, Analysis and Visualization,” DUE-0121741, September 2001 - August 2003.

Authors' addresses: M. Chau is with the University of Hong Kong, Hong Kong; Z. Huang and H. Chen are with the Department of MIS, University of Arizona, Tucson, AZ 85721. Emails: zhuang@eller.arizona.edu

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee. Permission may be requested from the Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036, USA, fax:+1(212) 869-0481, permissions@acm.org

© 2004 ACM 1531-4278/04/0600-ART2 \$5.00

Due to the growth of e-business, the World Wide Web has become a critical part of many real-world systems. Thus, it is increasingly important that IT professionals be proficient and knowledgeable in various Internet technologies that link front-end and back-end components like Web request processing, script languages, and Internet-related protocols, which are also evolving at a rapid rate, making it critical to keep up-to-date with them.

To address these evolving needs, we decided to develop a class project to give students a chance to acquire the complex skill-set discussed above, resulting in a project called “Build Your Search Engine in 90 Days,” in which groups of four to six students were required to develop a domain-specific Web search engine in three months’ time. The project was designed with the following objectives:

1. to allow students to acquire system integration, database administration, and project management skills;
2. to provide students with hands-on experience with Internet technologies and Web computing; and
3. to allow students to learn more about data structures and algorithms and their applications in systems.

There are several reasons for choosing the Web search engine as our topic. First, building a Web search engine requires knowledge in a diverse set of areas (e.g., databases, Web servers, information retrieval, data structures, Web request processing, user interface design, algorithms), providing students with opportunities to explore various areas. Second, a search engine’s different components allow students to acquire experience in system integration, project planning, and system analysis and management, which are critical skills for IT professionals [Noll and Wilkins 2001]. Third, search engines, both generic and domain-specific, are considered one of the most useful services on the Internet [Brewer 2002]. Most users begin their Web activities by querying a search engine, so it is reasonable to believe that all students have used a Web search engine, and most are familiar with the Web and such search systems. Hence, we believe that this makes the project more interesting, as the students can build something that they might consider useful.

The project was first carried out in Fall 1999 in a graduate course at the Department of Management Information Systems (MIS) at the University of Arizona. It was a data structures and algorithms course, a core course for graduate students in the department. In this article we report on our experience with this project and the related computing resources available to educators and students. The article is structured as follows: in Section 2 we review the components of a search engine and available resources for each component; Section 3 describes the structure of our project and shows an example of how students can use our tools to develop their search engines; in Section 4 we give some examples of the Web search engines developed by our students over the past three years. In the last section, we discuss some lessons learned from the project and suggest some possible uses of our project and tools in different areas.

2 BACKGROUND

2.1 Teaching Software Development, Programming, and Information Retrieval

Programming and software development courses are taught in most computer science and information systems curricula as lectures or lectures with lab discussions, using

individual programming or written assignments for assessment. In the last decade, group projects or group programming exercises have become more popular. It is suggested that such group activities promote cooperative learning and a positive experience among students [Dutt 1994; McConnell1996]. It has also been shown that group programming exercises used in an introductory programming course improve problem-solving abilities, and increase knowledge and interpersonal skills [Granger and Lipper 1999; Poindexter 2003]. Similarly, Harris [1995] suggests that group projects may provide opportunities for students to improve their written and oral skills. Several researchers also stress the importance of allowing students to work on projects that can be applied to real-world problems, and thus gain valuable hands-on experience [Fox 2002].

In recent years, a number of information retrieval courses have required students to work on projects that involve the creation of a search engine. For example, Davison's course on search engines required students to implement a search engine for the Lehigh University Website [Davison 2003]. Students were asked to work in groups to implement the crawler, parser, indexer, interface, retriever, and link analyzer for their search engines. Search engine projects at other universities are similar to this one, but are more focused on programming and implementation. These courses allow students to understand in detail the programming techniques involved in search engine development. However, given the vast amount of Web-based components available, few of these projects focused on teaching students how to integrate different existing tools to build their search engines. The aim of this study is to focus on application development using existing tools, so that students can spend more time and utilize their creativity on developing more novel functionalities for the search engine, rather than devote effort to implementing the core engine of the system.

2.2 The Web Search Engine

Before describing the project in detail, it is useful to review the architecture and components of typical Web search engines such as Google (www.google.com), AltaVista (www.altavista.com), and Lycos (www.lycos.com). A simple search engine architecture is shown in Figure 1. A search engine usually contains spiders, a Web page repository, an indexer, search indexes, a query engine, and a user interface. These components are described in the following. (See Brin and Page [1998] and Arasu et al. [2001] for more detailed discussions on search engine architecture.)

1. *Spiders*: also referred to as Web robots, crawlers, worms, or wanderers, are the programs behind a search engine that retrieve Web pages by recursively following URL links (Uniform Resource Locator) in pages using standard HyperText Transfer Protocols (HTTP) [Cheong, 1996]. First, the spiders read from a list of starting-seed URLs and download the documents at these URLs. Each downloaded page is processed, and the URLs contained within it are extracted and added to the queue. Each spider then selects the next URL from the queue and continues the process until a satisfactory number of documents is downloaded or local computing resources are exhausted. To improve speed, spiders usually connect simultaneously to multiple Web servers in order to download documents in parallel, either using multiple threads of execution [Heydon and Najork 1999] or asynchronous input/output [Brin and Page 1998].
2. *Web pages*: Documents retrieved by the spiders are stored in a repository of Web pages. To reduce the needed storage space, the pages are often compressed before

- being stored. The repository is usually in the form of a database, but it is also common for small-scale search engines to simply store the documents as files.
3. *Indexers*: An indexer processes the pages in the repository and builds an underlying index of the search engine. The indexer tokenizes each page into words and records the occurrence of each word in the page. The indexer is also used to calculate scores such as the term and document frequencies of each word, which can be used for search result rankings or further processing.
 4. *Inverted index*: The indexing results from the indexer are then converted into an “inverted index.” While the original indexing results map a document to a list of words contained within it, an inverted index maps a word to a list of documents containing the word. This allows fast retrieval of documents when a search query is passed to the search engine. The resulting searchable *indexes* are usually stored in a database.
 5. *Query engines*: A query engine accepts search queries from users and performs searches on the indexes. After retrieving search results from the indexes, the query engine is also responsible for ranking the search results according to content analysis and link analysis scores. It is also responsible for generating a summary for each search result, often based on the Web page repository. The query engine in some search engines is also responsible for caching the results of popular search queries. After all the processing, the query engine generates and renders a search result HTML page and sends it to the user interface.
 6. *A Web user interface*: allows users to submit their search queries and view the search results. When a user performs a search through the Web interface, the query is passed to a query engine, which retrieves the search results from the index database and passes them back to the user.

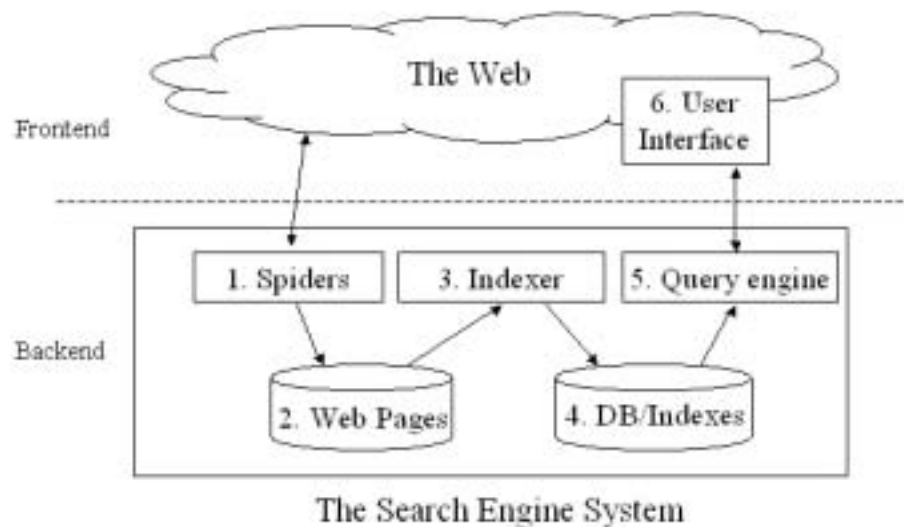


Fig. 1. The architecture of a simple search engine.

Due to the enormous size of the Web, general-purpose search engines cannot (usually) satisfy the needs of most users searching for specific information on a given topic. Hence, many domain-specific search engines have been built to facilitate more efficient search in specific domains, which usually provide more precise results and more customizable functions. For example, *LawCrawler* (www.lawcrawler.com) allows users to search for legal information; *BuildingOnline* (www.buildingonline.com), *SciSeek* (www.sciseek.com), and *BioView* (www.bioview.com) are a few other examples.

2.3 Developing Tools for Search Engines

There are many existing tools, mostly freeware or advertisement-based, that provide one or all of the components of a search engine, i.e., spiders, indexers, query engines, and index storage. The Web site SearchTools.com (www.searchtools.com) provides a comprehensive list of these tools, some consisting of all the components needed to create a search engine and enable users to build their own search engines; some popular examples are WebGlimpse [Manber et al. 1997]; ht://dig [ht://dig Group]; GreenStone [Witten et al. 2001]; and Alkaline [Vestris Inc. 2001]. These tools take a list of Web sites from a user as seed URLs, collect Web pages based on these seeds, index the pages, and set-up a user interface for querying and browsing; some other current and past examples include Harvest [Bowman et al. 1994], URL Spider Pro, and FusionBot.

Although these toolkits provide integrated environments to enable users to build their own domain-specific search engines, they are usually not appropriate for educational purposes; some do not provide enough technical details and the components and building steps are tightly coupled. As a result, users cannot look into the architectural and implementation details for deeper technical understanding. For example, in Alkaline, all the intermediate results of the spidering and indexing processes are hidden from general users. Although other tools provide system manuals with system architecture and other implementation details, they are often written for advanced-search engine developers who want to customize or maintain the software. A strong technical background and experience with the specific technologies used in different search engine toolkits are often required. Most importantly, integrated search engine toolkits do not allow students to acquire experience in system integration and the development of Web computing technologies.

Hence, we decided not to use integrated search engine toolkits as the foundation for the course project. Instead, students were allowed to select and integrate existing individual search engine components like spidering and indexing tools into their domain-specific search engines (there are also many individual search engine components available on the Web). Spidering tools, for instance, have been available since the early days of the Web; *tueMosaic* is an early example of a personal Web spider [DeBra and Post 1994]. Using *tueMosaic*, users can enter keywords, specify the depth and width of search for links in the current homepage, and request the spider to fetch homepages connected to the current homepage. WebRipper, WebMiner, Teleport, and crawl-it are some examples of software that allows users to download massively from Web site files of particular types or with particular attributes. Some spiders are designed to provide additional functionalities; for example, the Competitive Intelligence Spider performs breadth-first search on given URLs and applies linguistic analysis and clustering to the search results [Chen et al. 2002].

SMART (System for the Mechanical Analysis and Retrieval of Text), developed by Salton, is probably the earliest indexing tool [Salton and McGill 1983]. *SMART* creates an index for a set of documents and assigns a weight to each term-document association, based on the TFIDF formula [Salton 1989]. TFIDF, calculated by term frequency multiplied by inverse document frequency, is widely used in both traditional and Web-based information retrieval systems. Term frequency represents how frequently a term appears in a document, while inverse document frequency represents the specificity of a term (e.g., a term that appears in only a few documents is considered more specific, and thus more important). The combined TFIDF score is often used to represent the importance of a term in a document and for ranking search results. *Swish* and *Glimpse* [Manber et al. 1997] are another two tools that create indexes over a set of documents, especially Web documents. These tools scan through the documents and build an inverted, searchable index between each term and each document, based on their occurrence. Both tools were later enhanced to become *Swish-e* [Swish-e Development Team 2002] and *WebGlimpse*, respectively, to include Web page-fetching capabilities (i.e., spiders). *Lucene* is another tool that has become increasingly popular for document indexing due to its ease-of-use and fast indexing speed.

Individual search engine components are sometimes still difficult to use for the specific purposes of this project. Spidering and indexing tools on the Web are usually designed as stand-alone software, rather than being embedded in a search engine system as a component. For example, some spidering tools are designed to fetch an entire Web site, thereby creating a directory structure that replicates the original website structure to store the collected Web pages. This feature, which is nice for browsing the collected pages, imposes great difficulty for later processing of the collected Web pages in a search engine project. Similarly, the indexing tools discussed usually aim to produce full-text searching capability on a specified set of documents, resulting in a tightly integrated indexer and query engine. So customizing the query engine, for example to improve the ranking of search engine results, is typically difficult and sometimes impossible. Because the existing tools are not highly appropriate for this project, we decided to provide students with two simple tools, called the *AI Spider* and the *AI Indexer*¹, specifically designed as individual components of the search engine architecture described here. The spider tool collects Web pages and stores them in a way that favors later processing. The indexer tool separates indexing processes from the query engine by generating the indexing results as plain text files which students can import into databases and implement their own query engines.

The *AI Spider* is a simple breadth-first search spider extracted from the Competitive Intelligence Spider, discussed earlier. While not very sophisticated compared with other spidering tools available nowadays, *AI Spider* suffices for users to collect tens of thousands of Web pages to create a simple search engine. Written entirely in Java, it performs breadth-first search based on a set of URLs supplied by the user. A number of threads are then created to simultaneously fetch pages from the Web. The default number of threads is 10, which can be adjusted by the user according to the processing power of the user's computer. Because the response time of Web servers varies greatly, the use of

¹ Source codes of the *AI Spider* and *AI Indexer* can be found in the our course Web site: <http://ai.bpa.arizona.edu/hchen/mis531a/F2001/>.

multiple threads is one of the popular ways employed in Web spiders to increase performance [Heydon and Najork 1999]. Each thread connects to the target Web server to download the Web page using the `URLConnection` and the `HttpURLConnection` classes defined in the Java SDK (Software Development Kit) libraries. After a page is fetched, it is analyzed and the URL links contained in the page are extracted (parsed) and put into a first-in-first-out queue. The corresponding thread then picks up a new URL from the queue and continues the fetching process. The system continues to run until a user-defined number of pages have been collected from the Web. The Web pages are stored in their original formats in the local hard drive. An index file is also created, in plain text format, to match the URL of each collected Web page to its corresponding location on the local drive. This ensures that other applications can easily reuse the output from the AI Spider.

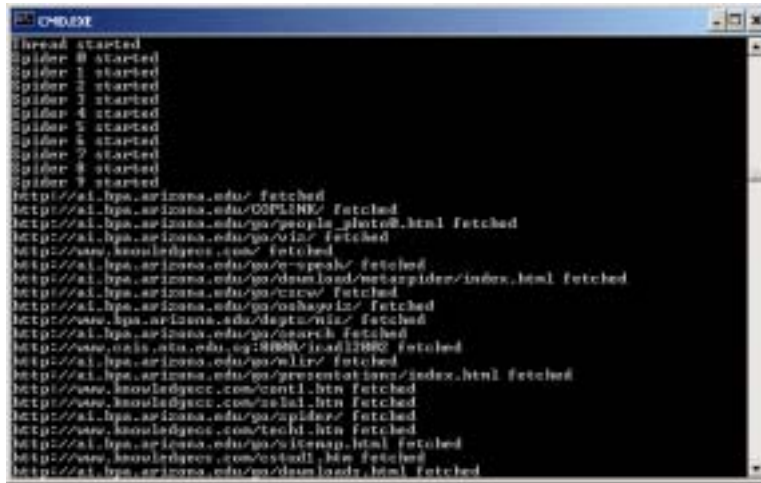
The AI Indexer is a tool based on automatic indexing [Salton 1989; Chen and Lynch 1992] and is written in C. It performs four steps on each document, including word identification, stop-word removal, phrase formation, and indexing. First, the tool identifies individual words in the document. Second, all the “stop words” in a document are removed based on a pre-defined list of non-semantic-bearing, functional and high-frequency words like “a,” “the,” and “of.” The tool then forms phrases by combining adjacent words (1, 2, 3, or 4 words) that appear between the stop words. Finally, the indexer records the information about the relationships between these phrases and the documents to support searching. This information includes the id of the document in which a term appears and the term frequency. Since we would like the index created by the AI Indexer to be easily read by users for debugging purposes or re-used in other applications, we stored the index in plain text format. Although a binary file format will allow faster retrieval and more efficient use of storage space, it is difficult for humans to read or for it to be re-used in other applications. As a result, the index file is in a more readily re-usable format than other similar applications that store their indexes in binary formats.

3 CREATING SEARCH ENGINES WITH AI SPIDER AND INDEXER

In the class project we allowed students to use the AI Spider and the AI Indexer as well as any individual components freely available on the Web, such as various spidering and indexing tools. However, because one of the main purposes of the project is to let students learn system integration and database management skills, we do not allow students to use integrated, “all-in-one” tools like GreenStone or Alkaline.² In this section, we present a sample user scenario to describe how users can use the individual components to build a simple domain-specific search engine.

The users first need to choose a particular domain for their search engine and identify a set of URLs that they believe contain useful content as well as outgoing links for the domain. These URLs can then be used as seeds for the AI Spider. Users can also specify other parameters, such as the number of requested Web pages and whether to restrict the spider to particular Web sites. Since Java source codes are also made available, users can modify the spiders according to their needs (see Figure 2).

² Starting with the third year of this project, we require each group of students to build two search engines – one from individual components and one from an integrated tool. The students are then asked to compare the results of the two search engines.

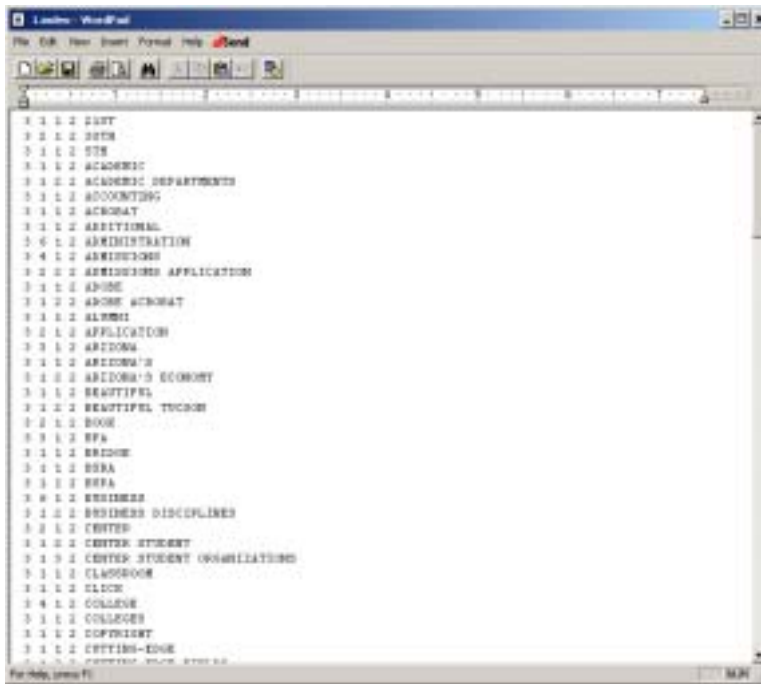


```

CMD:
Thread started
Spider 0 started
Spider 1 started
Spider 2 started
Spider 3 started
Spider 4 started
Spider 5 started
Spider 6 started
Spider 7 started
Spider 8 started
Spider 9 started
http://ai.bpa.arizona.edu/ fetched
http://ai.bpa.arizona.edu/COPYLINE/ fetched
http://ai.bpa.arizona.edu/ga/people_photos.html fetched
http://ai.bpa.arizona.edu/ga/via/ fetched
http://www.knowledge.com/ fetched
http://ai.bpa.arizona.edu/ga/c-speak/ fetched
http://ai.bpa.arizona.edu/ga/danload/metaspider/index.html fetched
http://ai.bpa.arizona.edu/ga/cricw/ fetched
http://ai.bpa.arizona.edu/ga/osh/giz/ fetched
http://www.bpa.arizona.edu/dgta/via/ fetched
http://ai.bpa.arizona.edu/ga/search fetched
http://www.oasis.ata.edu/cgi/DBMS/load/DBMS fetched
http://ai.bpa.arizona.edu/ga/oltr/ fetched
http://ai.bpa.arizona.edu/ga/presentations/index.html fetched
http://www.knowledge.com/control.htm fetched
http://www.knowledge.com/asal.htm fetched
http://ai.bpa.arizona.edu/ga/spider/ fetched
http://www.knowledge.com/teach.htm fetched
http://ai.bpa.arizona.edu/ga/itemap.html fetched
http://www.knowledge.com/studi.htm fetched
http://ai.bpa.arizona.edu/ga/danloads.html fetched

```

Fig. 2. A screenshot of the AI Spider output message search from <http://ai.bpa.arizona.edu/>



```

1 1 1 1 DIST
2 1 1 1 DIST
3 1 1 1 DIST
3 1 1 2 ACADEMIC
3 1 1 2 ACADEMIC DEPARTMENTS
3 1 1 2 ACCOUNTING
3 1 1 2 AERONAUT
3 1 1 2 ADDITIONAL
3 6 1 2 ADMINISTRATION
3 4 1 2 ADMINISTRATIONS
3 2 1 2 ADMINISTRATIONS APPLICATION
3 1 1 2 ANIME
3 1 1 2 ANIME AERONAUT
3 1 1 2 ALUMNI
5 1 1 2 APPLICATION
3 1 1 2 ARIZONA
3 1 1 2 ARIZONA'S
3 1 1 2 ARIZONA'S ECONOMY
3 1 1 2 BEAUTIFUL
3 1 1 2 BEAUTIFUL TUCSON
3 1 1 2 BOOK
3 1 1 2 BPA
3 1 1 2 BRIDGE
3 1 1 2 BSA
3 1 1 2 BSA
3 0 1 2 BUSINESS
3 1 1 2 BUSINESS DISCIPLINES
3 1 1 2 CENTER
3 1 1 2 CENTER STREET
3 1 1 2 CENTER STUDENT ORGANIZATIONS
3 1 1 2 CLASSROOM
3 1 1 2 CLDCE
3 1 1 2 COLLEGE
3 1 1 2 COLLEGES
3 1 1 2 COPYRIGHT
3 1 1 2 COTTAGE-EDGE

```

Fig. 3. Sample output file of the AI Indexer.

After the AI Spider has finished fetching Web pages, the users can run the AI Indexer on these pages. The AI Indexer converts the HTML pages to a simple SGML format and then extracts words and phrases from the document via automatic indexing. Figure 3 shows a sample output of the AI Indexer. Each term is preceded by four numbers, which represent, respectively, the document id, term frequency, number of words in the term, and the part of the document in which the term appears (e.g., title or body). The indexer will also generate a file that contains all the document ids and their corresponding URLs.

The users can then load these indexes into any relational database of their choice, such as Oracle, Microsoft SQL Server, or MySQL. They can then build a Web-based user interface that can query the database via middleware connections (such as CGI, PHP, ASP, JSP, or Java Servlet), which in turn connects to the database through standard database connectivity like ODBC and JDBC. The basic structure of a simple search engine is essentially complete at this stage.

4 EXAMPLES OF ALREADY BUILT SEARCH ENGINES

Over the past three years, students taking the graduate class on data structures and algorithms at the University of Arizona developed a total of 42 search engines as part of the course requirement. These search engines covered various domains, ranging from country music to Pokémon, and from American history to nanoscale science. In this section we present a few examples, illustrating what students achieved in this project.

NanoSearch is one of the search engines created by the students for the nanotechnology domain — the study of science, engineering, and technology at the scale of nanometer (a billionth of a meter). The group used AI Spider and AI Indexer to build the search engine and integrated the components using a Java Servlet and JSP framework. The design of the system is shown in Figure 4.

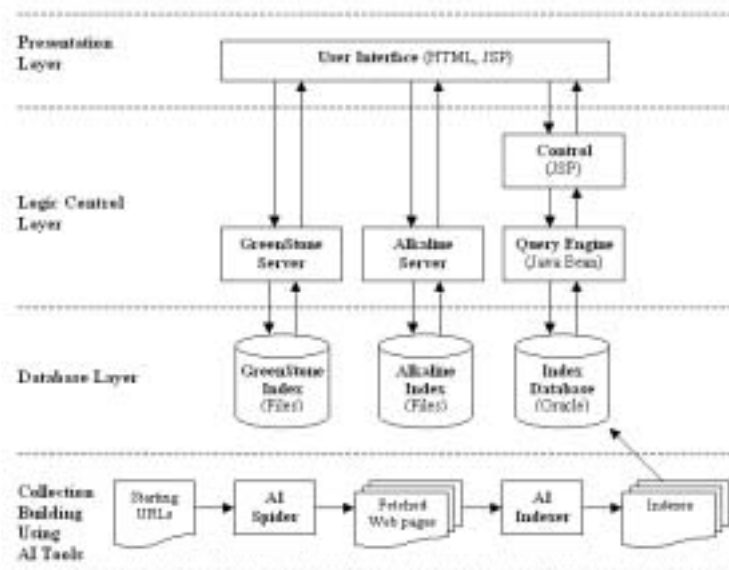


Fig. 4. Architectural design of NanoSearch.

The main programming languages and technologies used in NanoSearch include HTML, Java, Java Server Page (JSP), Java Bean, and Java Database Connectivity (JDBC). First, the search collection and index were built during a batch process. AI Spider was used to collect Web pages in the nanotechnology domain. It was also modified to calculate the number of incoming links for each Web page. The pages collected by the spider were then processed by the AI Indexer to generate index files, which were automatically loaded into an Oracle database by a tool called the SQL Loader. The database thus contained a searchable index for the collected Web pages. In addition to the collection built by the AI Spider and the AI Indexer, the students also created two search services in the same domain using, respectively, the GreenStone and the Alkaline software.

In the logic-control layer, Java beans were used to implement the query engine responsible for database access and search result-rankings. Search queries were forwarded to the database through JDBC. The Java bean then performed ranking on the retrieved results based on well-known metrics like the TFIDF score and the number of incoming hyperlinks. Java server pages (JSP) were used to control the logic between the user interface and the query engine. At the client side, the HTML and JSP components in the presentation layer determined the outlook of the search engine's Web-based user interface, which was connected to the query engine created by the students, as well as to



Fig. 5. A sample search session with NanoSearch.

the GreenStone and Alkaline search servers. A sample search session is shown in Figure 5. First, a user inputs the search query through the Web interface (step 1); “nanomedicine” is used in the example. If the user desires, he or she may also choose to submit the query to the GreenStone or Alkaline search services in the interface, as shown in the figure. The search query is then sent to the back-end search engine and the results displayed to the user (step 2). Much like a typical search engine, the user can click on any result and see the corresponding Web page (step 3).

In addition to creating their own search engines using our tools, the group also used the Alkaline and Greenstone toolkits to develop search services. The two search services are also accessible to users, as shown in Figure 5.

The *CookGuru* system is another search engine created by the students (see Figure 6). It is a recipe search engine that allows users to search for cooking recipes on the Web. We observed many interesting functionalities implemented in the system because this project let students demonstrate their creativity and innovation. **We observed how this project let students demonstrate their creativity and innovation as many interesting functionalities were implemented in the system.** For example, a simple multilingual indexer was developed to index the Chinese recipe Web pages. The system also applied both content analysis and link analysis techniques in ranking search results, as do to most commercial search engines. In addition, the system allowed users to create their profiles and use the system to store and organize recipes.

At the end of the class project, most groups developed a Web portal that encompasses a suite of online functionalities, in addition to Web page searches like chatting, message boards, and Web directories. The Tee2Green project is one such example. It is a golf search engine that allows users to not only search golf-related Web pages, but also chat



Fig. 6. The CookGuru search engine allows users to search for recipes as well as organize them.

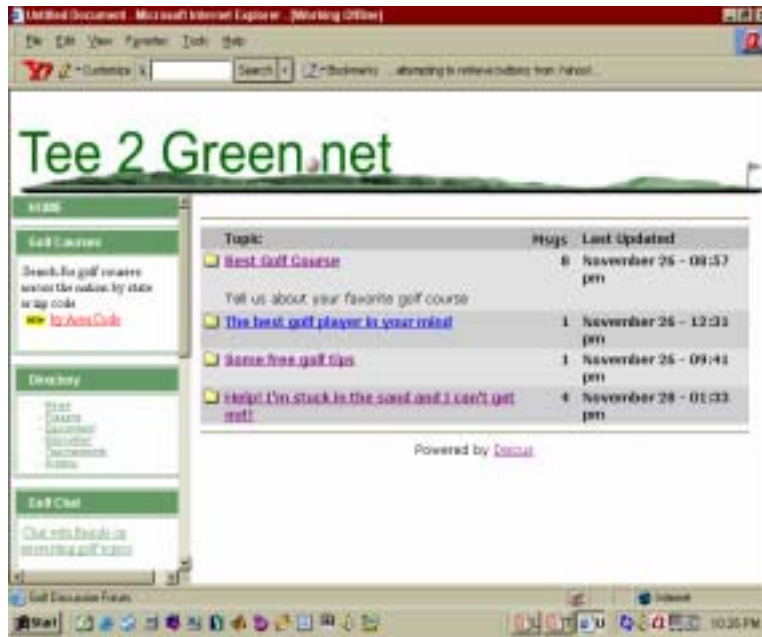


Fig. 7. Tee2Green golf search engine supports various functionalities like message board and online chat.

with other golfers, search for golf courses by area code, and post messages in a discussion forum (see Figure 7). Some of these functionalities were implemented by the students from scratch, while others were developed by integrating freely available tools with the system, such as *Discus* (www.discusware.com), a discussion board software package. It was observed that incorporating these functionalities allows students to understand system integration better and to learn more about Web system development.

It is also interesting to observe that many students applied what they learned in the class to their search engine projects. For example, an artificial neural network [Lippmann 1987] was one of the topics covered in our class; a few groups applied a feedforward/backpropagation neural network or Kohonen's self-organizing map in categorizing or clustering search results in their search engines. Various sorting algorithms and text analysis techniques covered in our class were also used by students in ranking search results in their search engines.

5 CONCLUSION AND FUTURE DIRECTIONS

Most students said that this class project was particularly rewarding; as a software development project, it offered hands-on experience in building a Web-based system that integrates various back-end and front-end components. Many technical components had to be integrated in this project, including Web computing technologies, databases, user interfaces, and so on. The three-month project gave a small-scale but complete version of real-life software projects. All major stages of system development were involved, including requirement analysis, architecture design, detailed component design,

implementation, and testing. During the development process, students also had a chance to improve their project management and teamwork skills by working with other group members on such a time-critical project. The search engine project was also formulated as a business plan, in which students were required to identify their own application domains after a complete analysis of the current market situation and potential competitors. Technology choices had to satisfy specific application-domain characteristics and, ideally, provide competitive advantages over competitors. At the end of the semester, a formal business presentation was also required. The project as a whole gave students extensive experience with the various challenges faced by today's information technology professionals.

We believe the project is useful for helping students understand some key computer science and information system concepts, acquire sufficient background in Web computing technologies, and obtain experience with various types of real-life challenges in system development projects. The project could also be used in many other types of computer science or information system classes; individual spidering and indexing tools are generic in nature and could easily be applied to other projects that involve Web computing.

Currently, we are working to develop a better set of tools for building search engines. The new software will be a system that integrates spidering, indexing, searching, and storage. Ideally, users should be able to use the individual components together as a whole system as well as separately. We are also working to improve the speed and reliability of the software to make it more useful for building search engines and other Web systems in real applications. For example, the Web pages collected by AI Spider could be used for other Web applications such as creating domain-specific lexicons, digital archiving, or developing a testbed for other IR research [Chau and Chen 2003]. Similarly, the AI indexer can be used to index documents other than Web pages (e.g., news articles or emails) so that these documents can be processed by other information retrieval techniques such as classification or clustering.

REFERENCES

- ARASU, A., CHO, J., GARCIA-MOLINA, H., PAEPCKE, A., AND RAGHAVAN, S. 2001. Searching the Web. *ACM Trans. on Internet Technology* 1, 1 (Feb. 2001), 2-43.
- BOWMAN, C. M., DANZIG, P. B., HARDY, D. R., MANBER, U., SCHWARTZ, M. F. 1994. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference* (Chicago, IL, Oct. 1994). 763-771.
- BREWER, E. A. 2002. The consumer side of search. *Communications of the ACM* 45, 9 (Sept. 2002), 40-41.
- BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International WWW Conference* (Brisbane, Australia, April 1998).
- CHEN, H., CHAU, M., AND ZENG, D. 2002. CI spider: A tool for competitive intelligence on the web. *Decision Support Systems* 34, 1 (2002), 1-17.
- CHEN, H. AND LYNCH, K. J. 1992. Automatic construction of networks of concepts characterizing document databases. *IEEE Trans. on Systems, Man and Cybernetics* 22, 5 (1992), 885-902.
- CHEONG, F. C. 1996. *Internet Agents: Spiders, Wanderers, Brokers, and Bots*. New Riders Publishing, Indianapolis, IN, 1996.
- DAVISON, B. D. 2003. Course web page of CSE 498: WWW search engines: Algorithms, architectures, and implementations. Available at <http://www.cse.lehigh.edu/~brian/course/searchengines/>
- DEBRA, P. AND POST, R. 1994. Information retrieval in the World-Wide Web: Making client-based searching feasible. In *Proceedings of the First International World Wide Web Conference* (Geneva, Switzerland, 1994).

- DUTT, J. 1994. A cooperative learning approach to teaching an introductory programming course. In *Proceedings of the Ninth International Academy for Information Management* (Las Vegas, NV, Dec.1994).
- FOX, T. L. 2002. A case analysis of real-world systems development experiences of CIS students. *J. Information Systems Education* 13, 4 (2002), 343-350.
- GRANGER, M. AND LIPPERT, S. 1999. Peer learning across the undergraduate information systems curriculum. *J. Computers in Mathematics and Science Teaching* 18, 3 (1999), 267-285.
- HARRIS, A. L. 1995. Developing the systems project course. *J. Information Systems Education* 6, 4 (1995), 192-197.
- HEYDON, A. AND NAJORK, M. 1999. Mercator: A scalable, extensible Web crawler. *World Wide Web* 2, 4 (1999), 219-229.
- HT://DIG GROUP. 2004. htdig reference. Available at <http://www.htdig.org/htdig.html>
- LIPPMANN, R. P. 1987. An introduction to computing with neural networks. *IEEE Acoustics Speech and Signal Processing Mag.* 4 (1987), 4-22.
- MANBER, U., SMITH, M., AND GOPAL, B. 1997. WebGlimpse: Combining browsing and searching. In *Proceedings of the USENIX 1997 Annual Technical Conference* (Anaheim, CA, Jan. 1997).
- MCCONNELL, J. 1996. Active learning and its use in computer science. In *Proceedings of the SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education* (Barcelona, Spain, June 1996).
- NOLL, C. L. AND WILKINS, M. 2001. Critical skills of IS professionals: A model for curriculum development. *J. Information Technology Education* 1, 3 (2001), 143-154.
- POINDEXTER, S. 2003. Assessing active alternatives for teaching programming. *J. Information Technology Education* 2 (2003), 257-266.
- SALTON, G. 1989. *Automatic Text Processing*. Addison-Wesley, Reading, MA.
- SALTON, G. AND MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.
- SWISH-E DEVELOPMENT TEAM. 2002. The Swish-e documentation. Available at <http://swish-e.org/current/docs/index.html>
- VESTRIS INC. 2001. Alkaline: A UNIX/NT search engine. Available at <http://alkaline.vestris.com/docs/pdf/alkaline.pdf>
- WITTEN, I. H., BAINBRIDGE, D., AND BODDIE, S. J. 2001. Greenstone: Open-source DL software. *Communications of the ACM* 44, 5 (2001), 47.

Received February 2003; revised July 2004; accepted September 2004