CSC326 Project - Snow Search

7/12/12

Erwin Magnaye

997664601

magnayee

Divya Malik

9977278075

malikdiv

Zhaoyi Ding

998088386

dingzha1

**1. Summary**

The challenge of the CSC326 project is to create a website that mimic's Google search engine. Given a keyword query from the web user, the frontend is able to search through an indexed database and to return a list of related URLs. These URLs are sorted in order according to the pagerank of the website and its relation to the keyword. The backend is responsible for building the indexed database for the front end to use. It recursively searches through URLs, parses and returns a word count for site it visits. In addition, the backend will calculate a pagerank for each of these sites.

**2. Design**

**2.1 Front End**

The frontend uses the Bottle web framework to generate the search engine website. It uses an html format to be compatible with web browsers. The web browser used to test the website is Google Chrome.

The biggest design decision made in the frontend is that it is responsible for retrieving the URLs. That is, the frontend searches through the database of indexed URLs and returns related websites. It will then calculate and sort these websites in order by its score. Each score is calculated by pagerank \* word rank, where we define the word rank as number of times it appears in the document with keeping into consideration its font size. By making this design decision, the time it takes to search a keyword will take much longer as the database grows. The benefit of this, however, is a much more simple design and a smaller database. The database is smaller since each keyword found in the lexicon is not directly indexed to URLs.

Other design decisions made on the frontend include the engine's special features. One feature is that it includes an intelligent spell check. If a user misspells a word, our spell check will catch this and suggest a different spelling for the search. It will automatically search this word instead, only if the original keyword does not return results. Another simple design feature is that the site is visually appealing with a winter theme. Background scenery with the effect of falling snow has been added to our snow search theme. Furthermore, the snow direction will follow the cursor.

**2.2 Back End**

The backend is built on the crawler template. Crawler is designed to recursively visit URLs of depth one and parsing through each site. The crawler will build a database with URL lists, word lists, and a list of web links. It will also calculate the page rank using the page rank function.

The biggest design decision in the backend is developing the database. The persistent database of choice, SQLite, is used with the Python interface to store data. The organization of the database consists of the following tables: **lexicon**, **document**, **link**, **url\_list**, and **doc\_id\_#**. The lexicon table indexes all words found from parsed websites. The document table indexes URLs parsed. The link table establishes the links of URLs parsed. The url\_list indexes ONLY possible URLs to be returned. Finally, the doc\_id\_# consists of multiple tables, one for every document parsed in the document table. This makes it much easier to find and search through a parsed document. Each doc\_id\_# has a list of word ids and its word rank. The downside to this approach is a less organized database with possibly hundreds or thousands of tables.

One feature included in the backend is the small tweak of calculating the word rank. The word rank is based on the number of occurrences of the word in the document as well as including the font factor.

**3. Test Results**

Refer to TestResults.docx included in zip file.

**4. Installation Instructions**

Refer to install.txt

**5. Team Member Roles**

The three members that designed and implemented this project are Divya, Erwin, and Ryan. Each team member was given certain roles. However, we discovered that working closely together and understanding each other's code helped a lot in writing their own code.

Divya's role was to implement the query function in the frontend, which calculated and returned the URLs related to the search. Erwin's role was to build the database using SQLite and crawler in the backend. Ryan's role was to design the website as well as the many snowy features in the website. Ryan and Divya also implemented the test suite and Erwin wrote the project report.

**6. Lessons Learned**

Several lessons were learned in this project. One lesson is teamwork. Each person has different strengths and weaknesses. Discovering this is essential in assigning different team member certain portions of the project.

The material learned in the course helped substantially. The HTML and bottle framework tutorials as well as the several techniques learned in Python were very helpful in the project. One example includes the extensive use of lists, tuples, and dictionaries. Organizing data and calculating with these sequences were proved to be helpful in the simplicity of the code.

The hardest and longest part was to understand the code given to us, as well as the instructions on what to do. After this problem was overcome, the remaining was straightforward and was completed smoothly. If we were to do the project again, we would begin by doing more research and understanding the code prior to starting to code. If we had more time we would add more features in the frontend and backend. This would include a more appealing website. Also a more efficient algorithm to search keywords would be implemented.

One testing strategy during development was testing each section alone and after it was working correctly, we would add these sections together in increments. Some sections tested separately are the SQLite database, website in HTML, pagerank, and query.