CSC326 FINAL REPORT

Group 23: Chih Jung Hsu (1000135766)

Dingjie Wang (999876353)

Suyuan Yu (1000573694)

2. Search Engine Design

The “Dinosaurch” search engine is composed of a front-end user interface and parsing component built on the Bottle framework. The front end connects to a SQLite database server that is prepopulated with PageRank data for a list of websites crawled by the backend crawler.

2.1. Feature Enhancements

Multi-word Searching

The search engine implements multi-word searching. Each word is queried in the database, with least recently used caching (see 2.2. performance enhancements). Webpages that contain multiple search words are given higher PageRank. In this case we simply maintain a “PageRank total score” for each URL and increment it by the URL’s PageRank for each additional search word that the URL contains.

Spell Correction

If a search returns no search results, the search engine attempts to autocorrect words in the search string and runs a new search on the autocorrected words. Autocorrection is based on the python-autocorrect library.

Search suggestion

If a search returns no search results, and it cannot be autocorrected to a version that generates non-empty search results, search suggestions are given based on similar past searches that generated non-empty search results. Past searches are regex matched based on the first and last letter of each word in the search string.

Minimize Number of Clicks

The Dinosaurch search engine shows a query box on the results page, making it easy to make new searches. No button needs to be clicked to return to the query page.

Recommended Testing Flow

The following short search flow demonstrates feature enhancements added in lab 4.

1. Search “Torontos Corporatoin”. This gets autocorrected to “Toronto Coporations”, and corresponding results are listed.
2. Search “Torontos”. This gets autocorrected to “Toronto”, and corresponding results are listed.
3. Search “Corporatoin”. This gets autocorrected to “Corporation”, and corresponding results are listed.
4. Search “Totoro”. This has no search results and cannot be autocorrected to a version that generates non-empty search results. Search suggestions are given based on similar past searches.

2.2. Performance Enhancements

Search string caching with custom LRU Cache:

A Least Recently Used Cache was implemented using a dictionary and a doubly linked list. Search strings are cached by the search engine. When the search cache is at full capacity, the least recently used search result is evicted to make room for the new search result. This improves search engine performance by avoiding repeated database queries while staying within memory constraints.

2.3. Project Contents

The project has the following directory structure:

- aws_setup.py: Search engine AWS instance deployment script
- aws_terminate.py: AWS instance termination script
- frontend.py: Frontend implementation built using bottle framework.
- client_secrets.json: stores Google API OATH2.0 parameters.
- views: folder containing frontend bottle templates.
    - search.tpl: search page template.
- logo.jpg: **DinoSaurch** search engine logo.
- requirements.txt: Python package dependencies.
- crawler.py: Crawler implementation built on top of starter code.
- dbFile.db: Database file containing results from crawler.
- urls.txt: Test URLs for manually verifying crawler functionality.
- tests: folder containing unit tests and fixtures.
    - crawler_test.py: Automated unit tests for crawler.
    - test_server.py: Test server with custom HTML to facilitate unit testing.
    - test_urls.txt: URLs for crawler to crawl in unit tests.

3. Project Code Overview

3.1 Backend Overview

- As required in lab 3, page rank and insert to database functions were implemented in get_page_rank(), and insertdatabase().

- The caching feature is implemented by changing the original data structure for doc_id_cache and word_id_cache to a LRU cache implementation, which is a queue that enforces least recently visited replacement policy. The reference LRU implementation is in cache.py.
- When capacity constraints are met in document_id or word_id functions, the LRU cache will evict an entry, and the crawler will store that entry in the database if it's not in there (this is enforced by the unique constraint of the tables). The pageRanks and get_resolved_inverted_index will then get the entire table from database instead of

using data stored in memory (since it may not be complete), and carry on calculation as before.

- A lite mode was introduced in which invertedIndex is stored in memory. If this mode is not selected, invertedIndex will always be stored in the database unless get_resolved_inverted_index() is called. Then it would load the data from the database and continue operation.
- A Least Recently Used cache is implemented in cache.py.

3.2 Frontend Overview

- Main search engine views were implemented as templates in the "views" folder.
- Webserver logic was implemented in frontend.py.
- The bjoern backend server was used as an asynchronous server written in C to improve web server performace. The bjoern server was chosen over other servers like rocket and cherrypy following performance tests.
- The autocorrect library is used to facilitate spell correction.

Differences in Proposed and Actual Design

Apart from enhancements, there were no significant deviations between the proposed and actual design.

Testing Strategy

Unit tests in earlier labs were implemented using the Python unittest module (see tests/ directory). A test server was implemented as ground truth for the crawler to crawl through. The crawler_test.py script automates all steps of testing, including starting and stopping the test server. Due to time constraints, unit tests were not added for recently implemented crawler functionality.

In labs 3 and 4, our testing strategy has centered around automated testing. Group members who worked on frontend components tested the backend components and vice versa. We examined different paths in the main flow and edge flow, designing user scenarios to test features such as multi-word search, spell correction, and search suggestion.

Lessons Learned

This lab taught us about web server implementation, front end design, crawling and ranking pages, as well as the deployment and management of web servers. If we had more time for this lab, we would implement the features that we did not have enough time to finish such as the autocompletion of searches. Finding and learning third party libraries was unexpectedly time-consuming. Significant amounts of time were required to fully understand third party APIs.

Application of Course Material

The materials taught in the course was essential to the development of this project. Since the project was coded in Python, many of the python coding taught during lectures helped us develop the search engine gradually. Although the professor did not specifically indicate what will be useful to us for the

lab, we found that many of the concepts taught in class were used in the coding of our frontend and backend.

Time Required

It took us approximately 5 hours to complete labs 1 and 2. Lab 3 took about 8 hours to complete. The final lab required additional planning of the overall design with the addition of a report; therefore, it took the group at more than 15 hours to completely the entire lab 4.

What was useful

For the frontend, we learned how to use HTML and CSS with python. HTML and CSS is in high demand in industry, but we only spent one tutorial covering it. Adding additional optional tutorials to teach frontend design could be beneficial for many students. In addition, it was interesting to learn parts of the Google API for our frontend login system. For the backend, it was exciting to learn about the basic inner workings of a search engine.

What was useless

Many of the requirements for lab 1 and lab 2 were not included in the final project. The top 20 words search history implemented in lab 1 was not used in lab 2. The recently searched words list implemented in lab 2 was not part of the requirements for lab 3. Overall, we feel that the labs could have been better connected to form a cohesive final project.

Feedback and recommendations

The concepts taught in this course are useful in both academia and industry. However, course organization could be improved. Lab instructions were vague and times, with changes made via announcements sometimes days before the lab due date. This made it difficult to make changes to completed code.

Team Member Responsibilities

The project workload was evenly distributed among group members. Every member was assigned individual tasks that could be done separately. Once individual components were complete, they were integrated and polished by all group members.

David's responsibilities included the development of the backend for labs 1 and 2, and the development of the frontend for labs 3 and 4. Jennifer's responsibilities included the development of the frontend for lab 1,2,4. Suyuan's responsibilities included the development of the backend for lab 1,2,3,4, and the launch and termination scripts for lab 4.