CSC326 Lab 4 Final Report

Group: g326-1-017
Sai Kiran Varikooty, Student Number: 998440307
Shafaaf Khaled Hossain, Student Number: 998891515

2. Search Engine Design

The Quest search engine is composed of a front-end user interface and parsing component using the Bottle framework. The front end connects to a Redis in-memory database server that has been preloaded with website data for a specified set of websites crawled by a crawler in the backend component.

Front End Functionalities

Spell Correction

If a word or several words are spelled incorrectly in the search, the search engine returns a spell corrected version of the word or multiple words to the user. Only individual words are corrected and only if there is a small error in spelling in like in 1 word. E.g toront is spelt as “toronto” as suggestion. This was chosen because a more robust spell checker would take more processing time and so this functionality was relaxed a bit.

Search Suggestion

When a query is searched, the search engine returns a corrected clickable link for the suggested search that will search the corrected version instead. The new link is clickable instead of user typing it again fully.

Query Phrase Interpretation

For simple math formulas (2+4, 7*(3+4), etc) that deals with addition, subtraction, multiplication, division can be entered into the query, and the search engine will compute the result and return a page with the result instead of searching for this directly as a text search.

Minimize Number of Clicks

The Quest search is user friendly and shows a query box on each page of the results, making it easier to make a new search. The suggested searches are also clickable links to the results of the suggested search.

Customized Mobile Devices

The results adjusts to the screen size for mobile devices. As the screen size is narrower, the search button drops below the box and the links. The link text sizes are also an abbreviated form which allows for easier viewing on a mobile device. This works for all pages and with smaller links. This is not done for queries which return big links which take up 2 or more lines.

Animated Logo

The animated logo is a glowing logo highlighting the name of our search engine.

Visually Appealing Profile

If the user clicks on their profile button in the top right corner of the screen, the user gets information about their profile such as first name, last name, email, and profile picture along with history. If no profile picture is not available, a default profile picture is provided. Having a popup makes it more satisfying for the user to see.

Input error testing
Input which include equations which make no sense are handles accordingly without the server crashing. E,g. 5 * h

Get prev, next results without loading page
The previous and next link buttons get the other urls using AJAX calls instead of reloading the page. Thus, reduces processing time.

Backend Functionalities

Optimized Search Engine Data Structure
As data is crawled and discovered by the crawler, it is normalized and added to a Redis in-memory database store using various data types for the different data structures used in processing (inverted index, lexicon, document index, pageranks). Redis is an in memory storage and so much faster than SQL databases like MySQL, PostGreSQL. Various data structures use the Redis API for different information such as hashes, strings, and sorted sets.

The search engine data structures are optimized for fast search of the information using the Redis in memory data store. The Redis API function used “zrevrange” specifies that the search has a time complexity of O(log(N)) (since in our case only a constant number of elements are being returned).

The search implementation is implemented in a separate module that uses keywords provided by the input processing module to build queries to the Redis server for fast query searches. Minimal processing of these results lead to a fast search response since it is done in memory.

Complex Ranking System
Additionally, an enhanced ranking system is used for multi word searches that weighs the pageranks of pages for the word set in the search query. This takes the total weight of all the pages for all the words in the query as opposed to a single word. By taking the weight of the pages appearing for all the words as a set, a better result match for the whole group of words is returned.

Multi Word searching
Results are based on all the words entered in the query by the user instead of the first one.

3. Differences in Proposed and Actual Design

The proposed design included the Sqlite relational database server. However, this was not chosen as the disk access required for SQL queries increases the response time of the search engine. When benchmarked on earlier versions that included Sqlite, the disk access was shown to be significant. With the Redis in-memory database server, the memory access

time is significantly less than disk access and thus the search engine response time can be reduced for an enhanced user experience.

4. Testing Strategy

The testing strategy included unit testing using the Python unit testing module. There were many cases considered, by examining incorrect or improper input to various parts of the search engine. For example, query phrase interpretation with incorrect operators, or invalid urls in the url file, accessing pages not in the server.

5. Lessons Learned

The key lessons learned were to use modular design and to plan an architecture before implementation. By identifying key components and the interfaces between components, the design can easily be broken down and separate pieces can be easily implemented. This approach was not taken initially and the lack of an architecture design caused repeated work in some cases and unneeded project development dependencies. Some practical lessons learned included the use of common formatting and style rules within the team for easier integration.

6. Project Reflection

If we were to complete the project again, we would spend more time on the software architecture and consider more design decisions to weigh tradeoffs. If we had more time, we would include more metrics in the ranking of the results and consider different types of data and categorize the results. For example, different types of input searches such as images, maps, videos or last modified information or sorting by size, etc.
Some parts that took longer than we thought were the front-end design which included considering different types of user input and many cases and also the database queries to Sqlite. This took longer since we did not predict that the design of the database table relationships and schema would be important for the search results needed.

7. Course Material Application

The course material was useful in many ways for the implementation of the project. Most useful was the understanding of different paradigms which led to some solutions that were easier for different problems in the project. For example, the functional programming paradigm was used along with list comprehensions in many cases where filtering or reducing of the results was needed. The course also introduced some technologies that we used such as databases and redis. These were used in different components of the search engine and the concepts and examples of these topics were helpful in development.

8. Design Time

The time for the project included approximately 30 hours of development time outside of the lab section. This time has different components however. The initial learning curve for some of the concepts and technologies expends time at the beginning to get used to the concepts and technologies while later hours are spent in actual development of features.

9. Importance of Components

The most important components of the project are the design of the database store of crawled data results and the front-end parsing of components for searching. These are the most important for two reasons. First, the design of the database store influences the ease of search provided by different inputs and the sorting of these inputs. Second, the front-end parsing can differentiate between different types of inputs to provide outputs that are relevant to the user. For example, an arithmetic expression entered can be evaluated if provided with the correct inputs and multiple words can be searched as text. Thus, the parsing and the structure of the database together are the most significant components the affect the ease of search and the usefulness of the results to the user.

10. Less Significant Components

The deployment of the search engine to AWS can sometimes be a complex process for certain errors that can occur. If there was an easier deployment process, this might save some time. Although this is an important part of web applications, the material is not directly related to the course and it might save some time if there was some easier deployment process through a script for example.

11. Recommendations

Overall the course was excellent and the project was interesting and fun to implement. Since the concept is something we are familiar with, the building of the search engine is also an interesting project. Some possible recommendations for the project in the future would be to include the option of other popular Python frameworks like Django and Flask and include an option for full stack Python frameworks. The course material was thorough and covered an excellent range of topics. Some possible recommendations for additional content would be to include parsing topics in Python (possibly some topics for Compilers), cython and including Python front ends for C/C++ code.

12. Work Tasks

Sai Kiran – Back end implementation of crawler modifications, implementation of interaction with relational database and Redis in-memory store, design of data storage format, enhanced ranking system for multi word search
Shafaaf – Front end implementation, Javascript and AJAX, input processing (query phrase interpretation, search suggestions, spelling corrections)