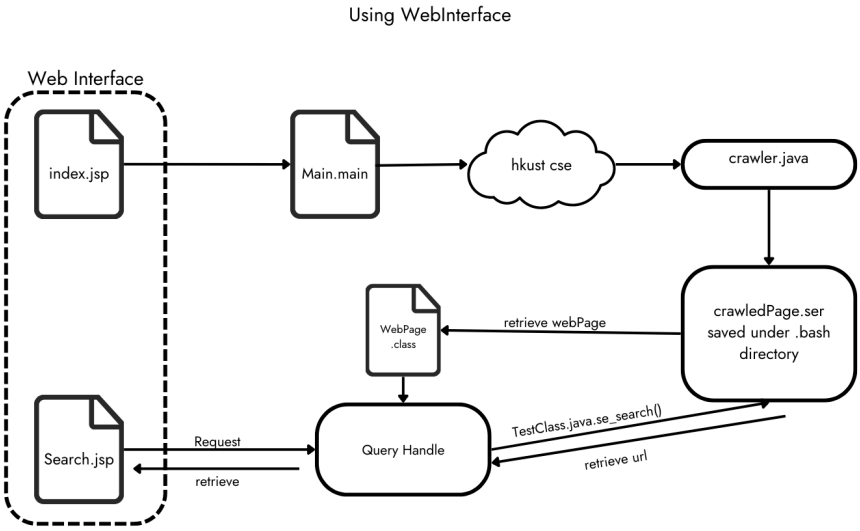
1. Introduction

This course is intended to have the whole picture of a search engine that has been used around the world. This project is one of the practical ways to perform the outcome of the courses. We establish the structure of the search by initiating the Crawler with Main.java as the main search that retrieves the crawled information, processes the index, connects the query input, etc. The programming language that we used is Java, and this project is built and run with IntelliJ IDEA community with smart-tomcat configuration.

2. Design Overview



a. Crawler

The crawler is an essential part of the project. The crawler is used for crawling the website to get the content of a website. It will record all the information under *https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm* and spread through the child links inside the web page. crawledPage.ser is the object of generated *HashMap<String,WebPage>* saved outside which retrieves all information regarding the webpage. The String URL is the unique identifier of the web page.

b. Indexer

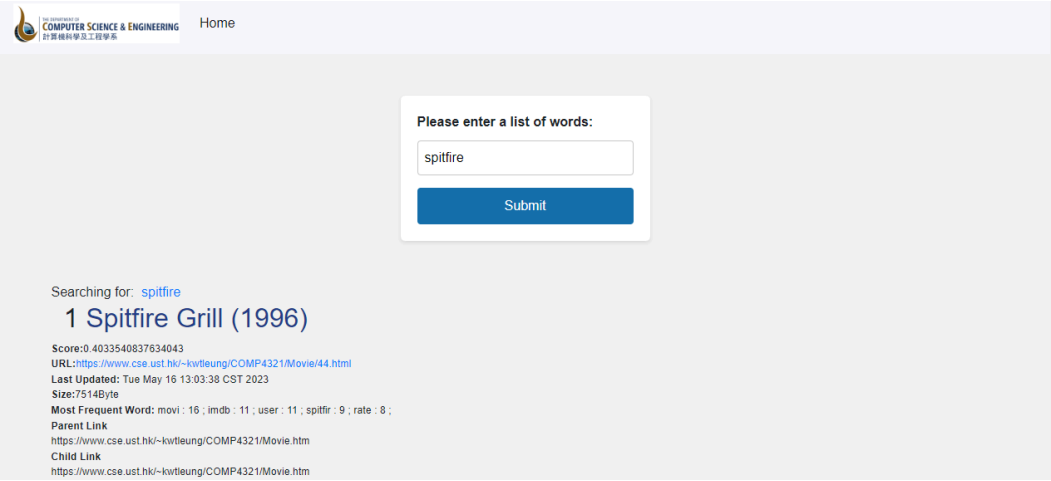
The indexer is a critical component of the document processing pipeline, responsible for performing tasks such as stemming and word processing. By reducing words to their base form through stemming, the indexer enables efficient search and retrieval of information. Users can leverage the stemmed words to conduct targeted searches, gaining access to a broader range of relevant

documents. After extracting the keyword and stemming, the forward index is stored at ForwardIndexTree.db

c. Query Handler

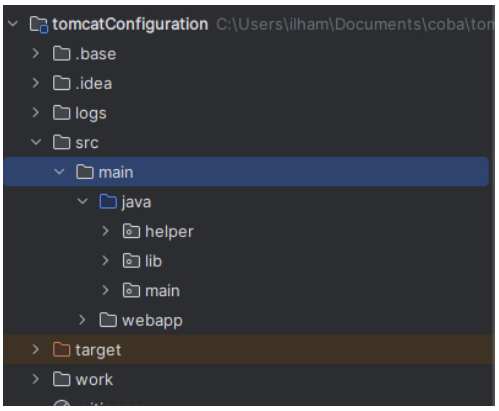
The query handler supports input with or without double quote words, where double quote words indicate the phrase search. When there is no double quote, the query will find all the documents that contain at least one word from the query. When there is a double quote, the query will find all the documents that contain the exact phrase in the double quote. Then, the function will determine the cosine similarity between the page and the query vector. And return a score. The score will be modified by the PageRank score, and whether the term appears on the title of the page.

d. Web Interface



The user's query is transmitted through the web interface, which initiates an HTTP request to the JSP engine running on the server. The JSP engine processes and compiles the JSP (JavaServer Pages) code into a Servlet class. This generated Servlet class is then executed by the Servlet engine. As the Servlet class runs, it dynamically generates HTML-formatted output. The resulting HTML content is sent back to the web server as an HTTP response. Finally, the web browser receives the HTTP response and renders the content on the web interface using the received HTML.

3. File Structures



tomcatConfiguration

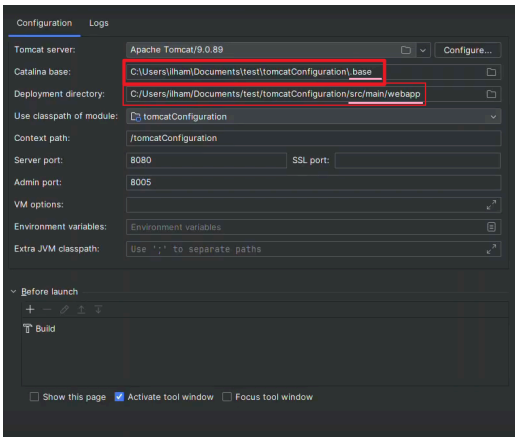
- **.base** — The base configuration for "smart-tomcat"
- **.idea** — The intellij configuration for IntelliJ IDEA
- **logs** — This folder stores log files generated by your Tomcat server during runtime.
- **src** — The directory of the program sources
  - **main** — for easiness of use
    - **java** — The directory of crawler, indexer, search engine
    - **webapps** — The directory of web-interface
- **target** — to store compiled output (e.g., compiled Java classes or JAR files)
- **works** — for temporary working files related to JSP compilation

The file directory is divided into two different parts. The first one is to */src/main/java* where it stores all the backend of the crawler, indexer, and search engine. The second is */src/main/webapp* where it stores all the User interfaces for the input queries. Since there are 2 ways to initiate the crawling, .bash is utilized of the purpose of running the backend on the jsp file (search.jsp and index.jsp) while */src/main/java* will store the output from compiling and running directly the main.java under */src/main/java* directory,

4. **Algorithms Used**
    a. **Crawler**
        i. **Breadth-First-Search**
        Breadth-First-Search (BFS) is a graph traversal algorithm that explores all the vertices of a graph in breadth-first orderThis property of BFS makes it suitable for implementing web crawlers because it ensures that the crawler systematically explores webpages in a breadth-first manner, starting from a given seed URL.
    b. **Retrieval**
    We utilized many prevalence algorithms in order to retrieve relevant webpage to query input. It combined into one which resulted in PageRank.
        i. **TF-IDF**
        The TF-IDF score is used while calculating the weight of the parameters, It is helpful for getting a correct vector for calculation in the later stage and correctly weight the terms in the document.
        ii. **Cosine Similarity**
        Cosine similarity was used after calculating the vector of both input query and the tf-idf score. It is helpful for deciding which page should be returned, to make our search engine
        iii. **Page Rank**
        After calculating the cosine similarity, a weighted score calculated by PageRank is added to the score, this allow our search engine consider more
        iv. **Title Favour-Weighted**
        After calculating the above score, if a page **title have the query term**, a bonus of (75% * Number of distinct words in title appear on the query). This helps highly prioritize the page with title matches.

    c. **StopStem Techniques**
        i. **Stopword Removal**
        Stopwords are commonly used words in a language that do not carry significant meaning and are often removed from text during preprocessing. We utilized this technology in order to remove all unnecessary words that are not related to the information.
        ii. **Stemming**
        The purpose of stemming is to group together words that have the same meaning but different inflections or variations. For example, stemming would reduce the words "running," "runs," and "ran" to their common stem "run”. In this case, it helps us to refer to which webpage is more significant for the query input.

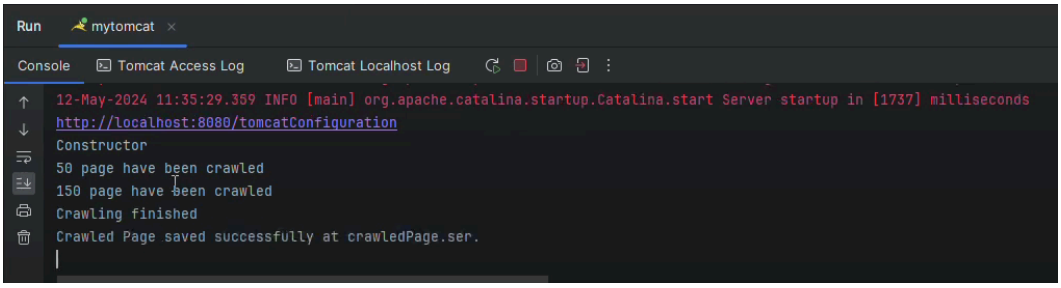
5. Installation Procedures



- New version of java 20 and JDK20 is used.
- Put stopwords.txt at root directories (*./stopwords.txt)* and helper folder *(./src/main/java/helper/stopwords.txt)*

If you want to use the User Interface:

1. Open the tomcatConfiguration directory
2. Use Smart Tomcat configuration
3. Make sure that the catalina base directory and deployment directory is the same.
4. Run the smart tomcat.
5. To crawl for the use of user interface, please open http://localhost:8080/tomcatConfiguration/ after smart tomcat is working Do not terminate program after crawling.
6. To search a phrase, go to http://localhost:8080/tomcatConfiguration/search.jsp
7. Please make sure that the crawling page is done before starting to search. You can see the progress under mytomcat terminal



8. If page modification is done, restart 5 to 7 again, to do the re-crawling..

If you want to crawl and search directly

1. Open the tomcatConfiguration directory
2. Run *src/main/java/main/Main.java*
3. Modify testClass or create testClass object and call se_search(String[]) where an array of string is the parameter of the queries. It will return the unique identifier of URL which can be used by calling a WebPage object.
   Please see the index.jsp under

6. Highlight of features (Bonus Points)

1. Similar Page (Bonus 1)

To find a similar page based on the five most frequent words within a webpage, we analyze the content and identify the words that occur most frequently. These frequent words serve as a reference for new input queries. By comparing the word frequencies of different pages, we can identify those that share similar characteristics. This approach helps in finding pages that are likely to have related or relevant content based on the commonality of their most frequently used words.

2. User Interface (Bonus 3)

It is important to create a user-friendly interface for the search engine. A well-designed interface helps users easily understand how to utilize the search engine without any inconvenience. By providing clear instructions and intuitive features, users can quickly grasp how to conduct searches effectively.

3. **PageRank Scoring** (**Bonus 6**)

PageRank is also considered in our implementation. The formula is the same as the Lecture Notes, "(1-d)+d(sum of(PR(Ti)/C(Ti)))". 2 iterations of the PageRank have been done As there exists pages which have many parent nodes (Movie Index Page), to prevent getting dominated by those pages, a low damping factor (0.015) is used. A weight small have been multiplied, due to the fact that cosine similarity will not be greater than 1, and PageRank score can go up infinitely, especially when a huge amount of parent page, a heavy penalty(multiplied by 1.5%) is applied to prevent dominate by the PageRank score.

4. Optimizing (Bonus 7)

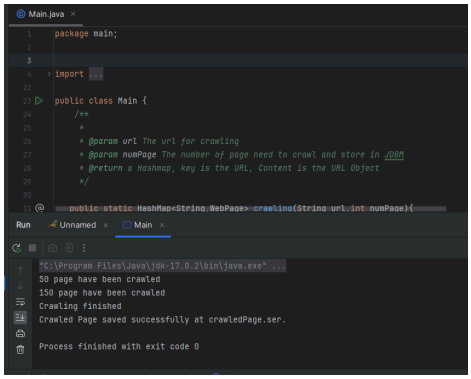
We attempted to reduce the access time and total search time **bonus feature 7** by optimizing our code. This reduces the time needed for restraining and storing data from the database. And this increases the time behavior of our search engine.

5. Other features

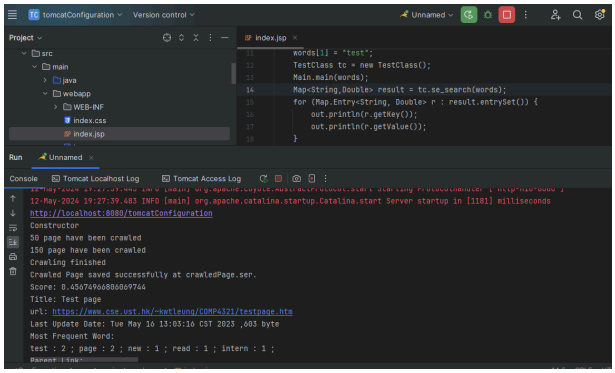
We also support clicking the link of the page, the parent page, and the child page, like the online search engine, it can help users to access the link easier.

7. Functions Implemented Testing

Main.main() is used to start crawling procedures.

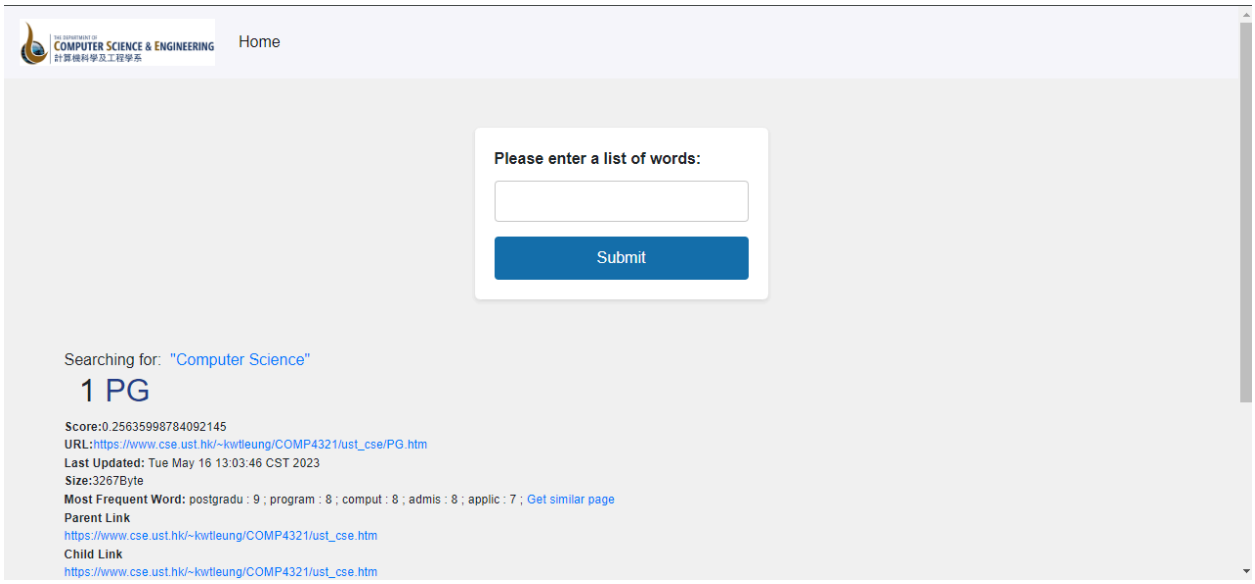


Index.jsp : is used to start crawling procedures for the web interface.



Crawled!

search.jsp(): The search.jsp is the interface for query input.



8. Summary

The project involves the development of a search engine for web and enterprise data. It consists of components such as the crawler, indexer, query handler, and web interface. The crawler utilizes the Breadth-First Search (BFS) algorithm to systematically explore web pages starting from a seed URL, recording information and spreading to child links. The indexer processes the crawled data, performs tasks like stemming and word processing, and builds an index for efficient search and retrieval. The query handler supports different types of searches, including phrase searches, using algorithms like TF-IDF and cosine similarity. The web interface allows users to input queries, initiates HTTP requests, and displays HTML-formatted search results. The project also incorporates algorithms like PageRank for retrieval and scoring, as well as techniques like stopword removal and stemming for text preprocessing. Bonus features include finding similar pages based on frequent words, a user-friendly interface, and optimization techniques for enhanced performance.

One of the weaknesses in our search engine lies in the implementation of the linear search algorithm. While this algorithm allows users to perform phrase searches by specifying multiple words in a specific order, it relies on a sequential scan of indexed documents. This approach can result in slower search performance, especially when dealing with a large number of documents. To change this, our solution if we re-implement is to have a better vector space which contains a large amount of correlation variable between one page and another. For example, implementing GloVe is one way.

Overall, the project aims to create an effective search engine that provides relevant results by leveraging crawling, indexing, and various algorithms.