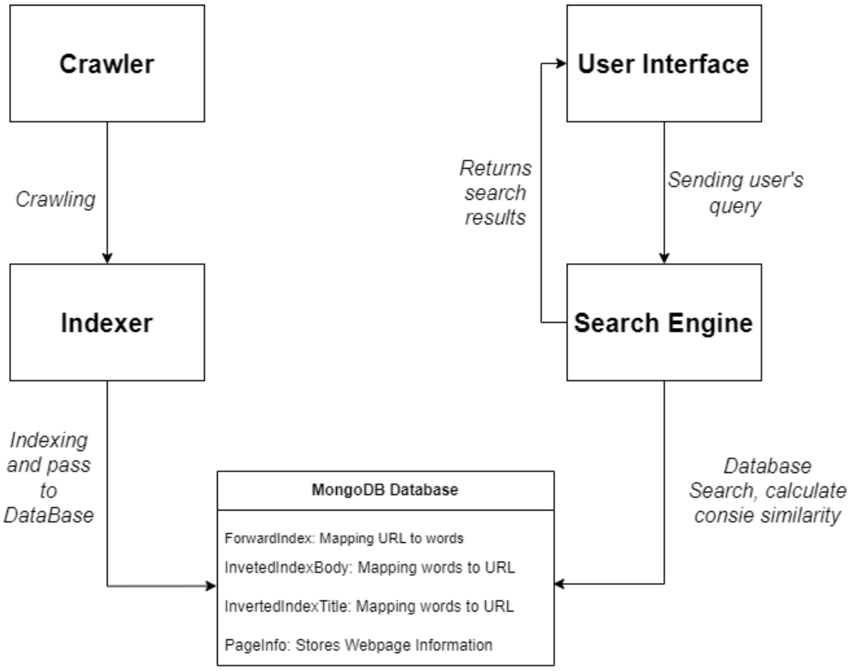
COMP4321 Group 35 Project Final Report

By Chia Tik Hin, Hui Cheung Hin, MORSI Mohamed Sobhy

1. Overall Design of the System


Crawler
User Interface
Crawling
Returns search results
Sending user's query
Indexer
Search Engine
Indexing and pass to DataBase
Database Search, calculate consie similarity
MongoDB Database
ForwardIndex: Mapping URL to words
InvetedIndexBody: Mapping words to URL
InvertedIndexTitle: Mapping words to URL
PageInfo: Stores Webpage Information

The design of our system is based on the above structure.

The backend includes the crawler and indexer which are responsible for crawling and indexing the web pages. The crawler will add the webpage information into the database collection PageInfo while the indexer will index the page and place the result into the database collection “InvertedIndexBody”, “InvertedIndexTitle” and “ForwardIndex”. If any web page is modified, after running the crawler and indexer, the database will be updated.

The frontend is built with React, Typescript and Next.js. It starts from the user interface, where the client can search their query in the input tab. The query is then sent to the search engine, search engine will retrieve relevant data from the MongoDB database. Then, the search engine will carry out calculations on TF/IDF and Cosine similarity to find the most relevant document with the corresponding query. At last, the user interface will receive relevant documents and display them on the browser.

2. File Structure of the Database

We had four collections of documents to support the system, namely “forwardTable”, “invertedIndexBody”, “invertedIndexTitle”, and “pageInfo”.

2.1 Forward Table

Field	*Type*	*Description*
_id	URL in String	URL as document id for look up
postingList	Object: {wordId, termFrequency}	An array of object that contains the terms in the webpage and their corresponding frequency

The forward table is a collection that stores the words present in a webpage along with their respective frequencies. By providing the URL, we can retrieve the posting list, which contains the terms and their frequencies within the webpage.

2.2 Inverted Index Body

Field	*Type*	*Description*
_id	Word in String	Word itself as word id for look up
postingList	Object: {documentID, termFrequency}	An array of object that contains the documentID(in url) and the page body term frequency of the corresponding webpage

The Inverted Index Body is a collection that stores the inverted index of words found in the body section of the webpage. To search for the occurrence of words within the webpage, we can use the word itself as an identifier to query this collection. The posting lists returned by this query will contain the web pages that contain the words, along with their corresponding frequencies in the page body.

2.3 Inverted Index Title

Field	*Type*	*Description*
_id	Word in String	Word itself as word id for look up
postingList	Object: {documentID, termFrequency}	An array of object that contains the documentID(in url) and the page title term frequency of the corresponding webpage

Similar to the Inverted Index Body, the Inverted Index Title is a collection that stores the inverted index of words found in the title section of the webpage.

2.4 PageInfo

Field	*Type*	*Description*
_id	URL in String	URL as document id for look up

Title	String	Title of the webpage
LastModifiedDate	String	Last modified date of the website
PageSize	Int	The page size of the webpage
LastCrawledDate	String	The date and the time that the page is fetched into/updated in the database
ChildLink	Array of String	Child links of the webpage
ParentLink	Array of String	Parent links of the webpage

PageInfo stored the URLs of the crawled web pages as its identifier which simplify the lookup of the web page in the database. By utilising the last modified data, the crawler are able to know if the webpage has been updated or not and update the database correspondingly. Page size (the size of webpage) is also stored here. Last but not least, the child link array stores the accessible outgoing links of the webpage, while the parent link array stores the accessible incoming links of the webpages.

3. Algorithm Used

3.1 Breadth First Search Algorithm(BFS):

Breadth First Search Algorithm(BFS) was implemented in crawler.java. When we crawl the website, we will extract the terms and child links. From the child links, it will redirect us to another website and keep crawling. Therefore, we implemented BFS to start crawling from the first website, and store a queue of immediate child links to extract layer by layer. After crawling a website, the data will be pushed to pageInfo, and let indexer.java to perform forward indexing, inverted index of title and body of this website.

However, as the child links might point back to the parent(or grand grand parent), it may extract the website again and again, which forms an infinite loop. Therefore, we can check if the extracting website in PageInfo in MongoDB exists. If it exists in the database, we will skip extracting this website and continue with the next website on the queue. The time complexity for checking if the extracted website exists in the database is O(1) because it is a key-value database.

The time complexity for BFS is O(V+E), where V is the number of websites, and E is the number of child nodes of each website.

3.2 Stemming and Stopword removal

Stemming is crucial for reducing the number of distinct terms and identifying words with similar meanings. To effectively increase the recall, stemming and stopword removal are necessary for both in the query and operations in forward index, inverted index for both title and body for the website.

For stemming, we implement Porter's algorithm from the lab. For the java part of the project, we referenced the original java class provided from the lab and used it in our project. Moreover, as our frontend is implemented in Next.js, we have made another version of Porter's algorithm for the query part, as it is incompatible with java class. The stemmed words are then passed to stop word removal.

For Stopword removal, we used the stopword text file provided in the lab to check if the stemmed word exists in the stopword text file. If it is not a stopword, we will then push to the forward index, inverted index for both title and body. For the queries in the browser, it will go through the same procedure of removing the stopword word then searching for the database, as this will increase the matching result.

3.3 TF/IDF

TF/IDF is for determining the important words in a document. This is useful when we want to search for a word or a phrase from the query. It is implemented in searchAction.ts.

3.4 Cosine Similarity

Cosine similarity measure is the similarity measure that we implemented in the vector space model to measure the similarity between the query and the document. It is also implemented in searchAction.ts.

3.5 Vector Space model

In information retrieval systems, documents and queries are represented as vectors, where each dimension corresponds to a keyword or term, and the value associated with each dimension represents the weight or importance of that term in the document or query. This vector representation allows for efficient computation of various statistics and measures, TF/IDF and cosine similarity which we have mentioned above, are used for ranking and retrieving the most relevant documents for a given query. The use of hash maps facilitates efficient storage and retrieval of term weights, enabling fast vector operations.

4. Installation Procedure

1. Install the current version (7.0.9) of MongoDB Community Edition through https://www.mongodb.com/try/download/community
    1.1. In the MongoDB installation set first click next
    1.2. Accepts the licence agreement and click next
    1.3. Click on complete
    1.4. In the next page, make sure that install MongoDB as a service is ticked, click next
    1.5. Make sure to install MongoDB Compass
    1.6. MongoDB is then installed, you can click of MongoDB compass, connect to local host and have a graphical interface of the database
2. Download the latest version of IntelliJ IDEA Community Edition
    2.1. In the welcome page of IntelliJ, open the directory “COMP4321ProjectG35”
    2.2. If the Java SDK is not setted, go to “project structure” and chooses SDK to be version 17 and language level to be SDK default
    2.3. Reload all maven projects if necessary
    2.4. Go to src/main/java and run testProgram’s main function to do the crawling
    2.5. You can view the crawling result in the Mongodb Compass
3. Use VSCode to open the “COMP4321ProjectG35” and change directory to “\frontend”
    3.1. In terminal enter “npm i” then “npm run dev”
    3.2. Open http://localhost:3000 with your browser to see the result.
    3.3. You can start editing the page by modifying app/page.tsx. The page auto-updates as you edit the file.
    3.4. This project uses next/font to automatically optimise and load Inter, a custom Google Font.

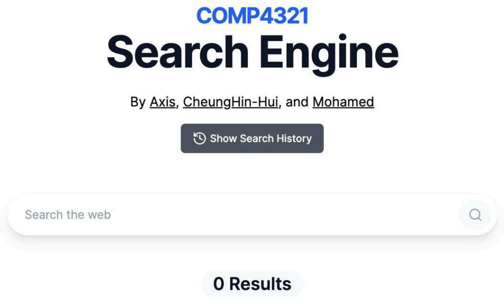
5. Highlight of Features

5.1 Search similar pages

In our search engine, we have a “Find Similar Page” button. When we click on this, it will fire a function and append the top 5 most frequent query directly into the search bar continuing from the original query. This feature allows users to conveniently search for similar pages with ease with our search engine with just a click of the button.

5.2 Persistent Search History

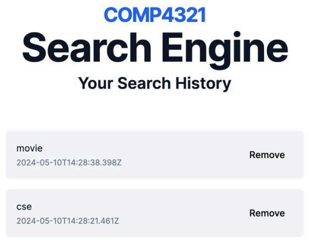
In our search engine, we have implemented “persisting search history” which records user’s search history and user are able to remove it as well.



COMP4321

Search Engine

By Axis, CheungHin-Hui, and Mohamed



COMP4321

Search Engine

Your Search History

movie
2024-05-10T14:28:38.398Z

cse
2024-05-10T14:28:21.461Z

5.3 User-friendly Interface

Furthermore, our system features a user-friendly interface that displays the page title, word count, prominent keywords, query similarity score, child links, etc. The UI also allows users to navigate to the resulting web page easily. For the database, the maintainer of the system can use MongoDB compass to manage the database consisting of the crawled result to see if the crawler and indexer are working as expected.

5.4 Navigation to Parent or Child Links

When the related documents are retrieved, we can press on the parent or child hyperlinks to directly navigate to the target website. This allow user to navigate to parent and child links if they are interested

5.5 Auto phrase detection

Our search engine will auto detect phrases and search for them, providing users with advanced search experience like what they have done in Google Search.

5.6 Separation of concerns

Separating the implementation of the crawler and the web interface allows for independent modification of each component without affecting the other. This separation of concerns enables greater flexibility, as changes to the crawler's functionality or the web interface's behaviour can be made independently, promoting modular and maintainable code.

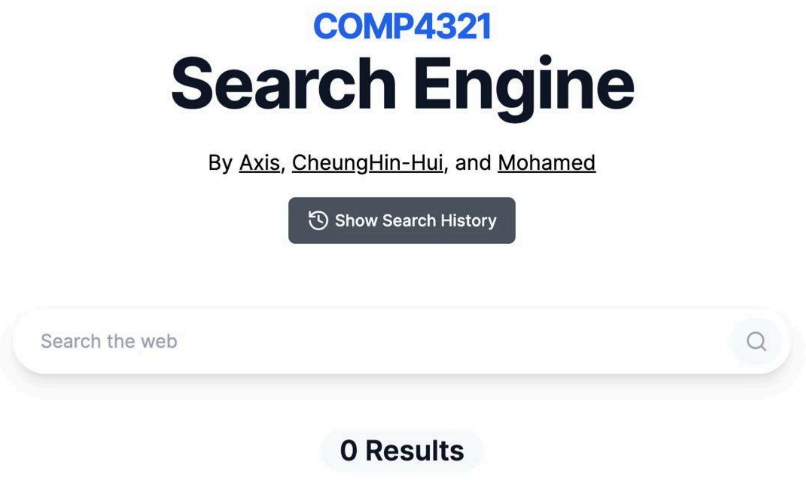
6. Testing of Functions Implemented

6.1 Crawler and indexer

To test the functionality of web crawler and indexer class, simply go to the ./testProgram.java file, assign the website that you want to crawl, for example, "https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm" to testURL variable, then run the main program.

After the main program ends, we can check the result of crawling in MongoDB. There would be 4 collections, which are pageInfo, forwardTable, invertedTableTitle, and invertedTableBody. The corresponding wordId, DocId and term frequency should be recorded into the collection.

6.2 Search in query

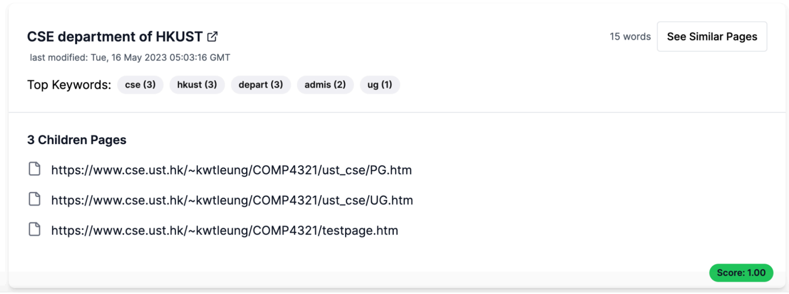


COMP4321
Search Engine

By Axis, CheungHin-Hui, and Mohamed

In the website, there is a “Search the web” that we can input our query to search for the relevant document. Simply type the words that we want to search on the input tab, and press enter or the search button on the right, and it will show the relevant documents below.

6.3 Similar page



CSE department of HKUST
last modified: Tue, 16 May 2023 05:03:16 GMT
15 words
Top Keywords: cse (3) hkust (3) depart (3) admis (2) ug (1)

3 Children Pages
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/PG.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/ust_cse/UG.htm
https://www.cse.ust.hk/~kwtleung/COMP4321/testpage.htm

Score: 1.00

This is one of the example documents that we have retrieved with our query. On the top right corner, there is a “See Similar Pages” button. When the button is being pressed, it will feed the top 5 keywords to the query tab and render the related document again.

7. Conclusion

7.1 Strength of the System

With the utilisation of the MongoDB database, our system has the capability to be deployed as a fully operational website. By using cloud services such as MongoDB Atlas Database and Vercel, the search engine can be deployed on the web for other people's usage with the pre-crawled data stored in the database.

Additionally, the "Find Similar Page" function directly inserts the suggested query directly into the search bar next to the original query. This feature allows users to conveniently search for interested pages using the precise query provided by our search engine. Our system can also process the queries in swift with an expected response time of less than 1 seconds.

7.2 Weakness of the System

The first weakness is that certain web pages have numerous incoming and outgoing links. When such a web page appears in the search result, its children link and parent link section take out a large portion of the webpage. This may negatively impact the user experience as users cannot view a lot of results without scrolling. If I could could reimplement the whole system, I would add a "show more' function, hiding the majority of parents and child links at first, only show them if the user press the "show more" function

Another weakness is that the tf/idf calculation is not performed during the crawling and indexing process, but rather on the client side. This can potentially decrease the performance of the search engine, as it needs to calculate the tf/idf each time a user submits a query to the search engine. If I could re-implement the whole system, this will be the first task to tackle.

7.3 Interesting Features to Add

1. Allow the web-based search engine to store user's previous input by setting up a log-in system and record their previous 10 search
2. Add a dropdown list user the search bar such that when user type in a portion of the words, the full words will be shown below and user can select these words and add them into the query
3. Showing a brief description of the returned page in the result
4. Implement PageRank algorithm

8. Contribution

8.1 Contribution of Chia Tik Hin(33.3%):

Chia Tik Hin's contributions include:

- Development of the indexer

- Formalization of the schema for "forwardIndex," "invertedIndexBody," and "invertedIndexTitle"
- Implementation of the Porter's algorithm and stopword removal program in Java
- Translation of the Porter's algorithm and stopword removal program into TypeScript
- Writing the final reports

8.2 Contribution of Hui Cheung Hin(33.3%):

Hui Cheung Hin's contributions include:

- Development of the crawler using BFS (Breadth-First Search)
- Modification of the crawler to support page updates
- Setting up a cloud MongoDB database for team usage
- Formalization of the scheme for "pageInfo"
- Writing the phase 1 report and final report

8.3 Contribution of Mohamed Sobhy(33.3%):

Mohamed Sobhy's contributions include:

- Development of the website and its user interface (UI): frontend sub-directory
- Development of the search engines that match queries with documents (implementing vector space model), tfidf normalisation
- Development of the "Find similar page" function
- Development of the "Prioritise matches in the title" function
- Writing the final report