COMP 4321 Search Engine

FINAL REPORT

Lam Hon Wa 20348745

LI, Junze 20413186

XU, Feiting 20329359

1. OVERALL DESIGN

The search engine consists of three major components: database, retrieval engine, and the interface webpage. The relationship between these three components is shown in the Figure 1.

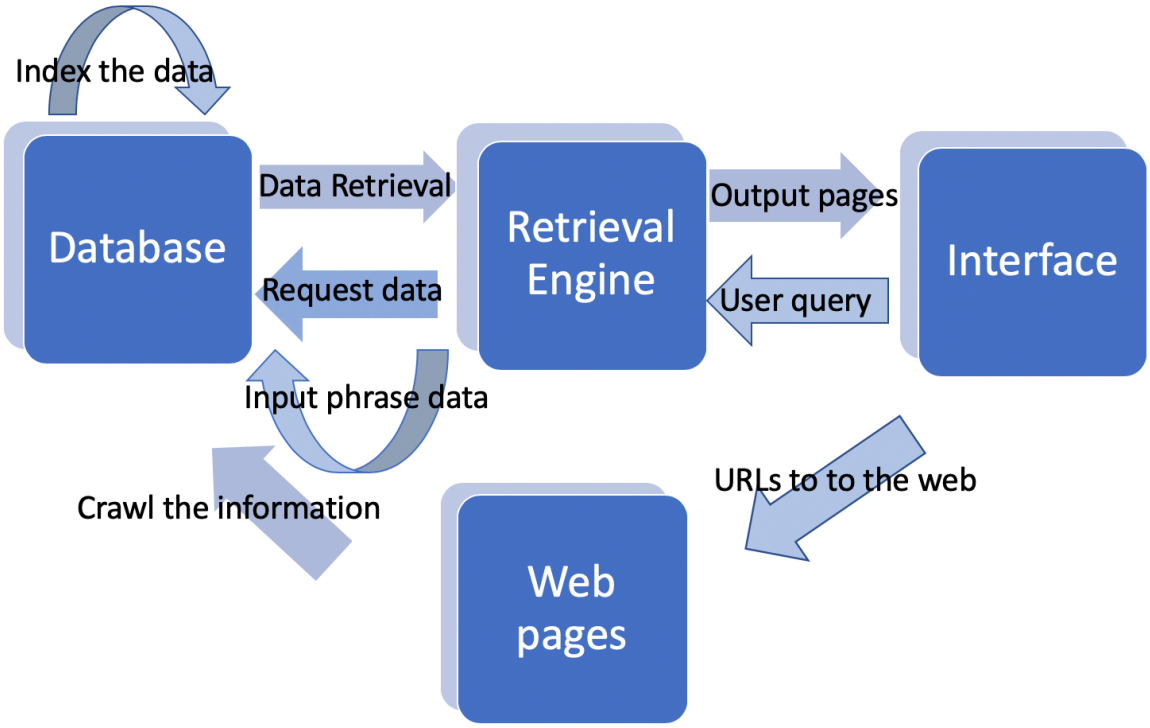


Figure 1

Database

The database is used to store the crucial information like keywords crawled from the web pages. The java document “project_database.java” is written to build different databases and include spider and indexer functions. ROCKSDB is used to build the databases with the format of key-value pairs structure.

SPIDER

The spider application is built by using HTTP, HTML, and JSOUP parsers. HTTP is part of the standard JAVA library. HTML and JSOUP are imported from the “htmlparser.jar” and “jsoup-1.8.1.jar” respectively.

Breath first approach is used when exploring the web by maintaining a single queue of URLs to be searched. Only new URLS which doesn’t exist in the database will be crawled and then index it. For the information crawled, stop word removal and stemming process are done before the index and insert process.

INDEXER

The indexer is in charge of indexing and inserting extracted documents into different databases. URL and words extracted are indexed for the forward and inverted files. Words are transformed into stem terms using the Porter’s algorithm. Stem terms extracted from

the page body and title are inserted into different inverted files. There are different databases to store varied content by the index, more detailed could be found on the FILE STRUCTRURES part.

Retrieval Engine

The retrieval engine handles the query request from the interface page and returns up to 50 documents from the database. The documents and ranked order are based on the vector space model. Cosine-similarity($\frac{\boldsymbol{D} \cdot \boldsymbol{Q}}{||\boldsymbol{D}||\,||\boldsymbol{Q}||}$) is used to calculate the similarity and the term weight is based on the formula ($\frac{\mathrm{tf} * \mathrm{idf}}{\max(\mathrm{tf})}$). Besides, the query undergoes preprocessing, such as stop word removal and stemming, identical to the process done in the crawling stage. Phase search and title favor mechanism is supported by this retrieval engine. More detailed could be seen on the algorithm part.

Interface

The web interface is responsible for dealing with the user search query and sending it to the retrieval engine. The web interface uses the Java Server Page supported by the Tomcat Server. Through the http request and response, the results of the query are sent to the interface and the interface will output the pages with relevant information.

2. File Structure

Design of the RocksDB database

Database Name	Key	Content	Special Key
url	URL	page_id	max_id (store max page_id)
word	word	term_id	max_id (store max term_id)
forward	page_id	term_id frequency,	
term_weight	page_id	term_id term_weight,	
inv	term_id	page_id position,	
title_inv	term_id	page_id position,	
parent_child_relation	parent_page_id	child_page_id,	
child_parent	child_page_id	parent_page_id,	

page_info	page_id	title; date; size; |D|	
idf	term_id	idf(=log2(N/df))	
inv_url	page_id	URL	
inv_word	term_id	word df	
keyword_list	alphabet	word	
query_history	query_id	query_string	
query_result	query_id	page_id score,	max_id (store max query_id)
query_index	query_string	query_id	

3. Explanations on the database design:

Key and value in databases

url contains unique page_id for each URL crawled
word contains unique term_id for each word encountered
forward contains every word (represented by term_id) in a page (represented by page_id) and its corresponding term frequency (represented by an integer following term_id)
term_weight contains the term weight (=tf x idf/max(tf)) of each word in a page
inv contains in which pages the word in content is in and its corresponding position
title_inv contains in which pages the word in tile is in and its corresponding position
parent_child_relation contains parent page id and its corresponding child page id
chile_parent contains the child page id and its corresponding parent page id
page_info contains title, date and size of a page and the length of the document vector (|D|)
idf contains idf value for each word (for the purpose of saving computation time during search)
inv_url is the reverse of url
inv_word is the reverse of word with extra document frequency of each word
keyword_list contains the keywords with document frequency higher than 10 sorted by first character
query_history contains unique query_id for each query history
query_result contains query_id and the corresponding search result, containing page_id and the score
query_index is the inverse of query_history

Usage of databases

url, word, inv_url, inv_word, query_history, query_index is for mapping

idf, history, inv_url, inv_word, term_weight is for pre-computing and speeding up searching time.
parent_child_relation, child_parent is for crawling and retrieving
forward, page_info is for finding and displaying page content
inv, title_inv is for indexing and matching the query and getting position of word
query_result is for storing the past query results

Remarks:

If there are multiple values in each entry of the content, the values would be separated by space. (expect for the page_info database, it would be separated by ';')
If there are multiple entries in the content of the key, the entry would be separated by comma.

Example:

If the term_id 2 keyword appears in page_id 2 with position 45 and in page_id 3 with position 88, then the content of inv database with key 2 would be 2 45,3 88.

4. Algorithm

Vector Space Model

Cosine Similarity is used to determine document-query similarity as mentioned. During the process of calculating the inner product, the weight of query term is assumed to be 1 if the term presents in the query and 0 otherwise.

Phrase Search:

Phrase search requires the inversion database to store value in page_id ascending order. This special requirement could speed up the phrase search process although it may not support crawl database update later.

The process of the phrase search is to split the query into phrase part and non-phrase part first. The phrase part should be put into " ". (eg. "Hong Kong" University) For the non-phrase part, the algorithm just process it by Stop Stem function and then store it in database named "query_tid_array" if it's not empty. For the phrase part, do stop stem as the non-phrase part first. If there it reduces to single words or empty, then treat it as the case of single word. Otherwise, start processing the phrase in a recursive manner. If current phrase doesn't exist on the database, process the phrase as a new term and store its corresponding term_id, inverted file, count its document frequency and calculate its term weight. Then the term_id for the phrase is also put into the query term id array for search.

Users are recommended not to enter phrase with stop words in it. Otherwise, undesirable results may be returned. The search engine would stem the phrase first before searching it in the index file. Users may fool the search engine by including a phrase "hong a a kong", where "a" will be removed by the stop stem algorithm and phrase would be stemmed to

“hong kong” and those documents with “hong kong” would be considered as containing the terms.

Title favor mechanism:

The database title_inv is built to store the content inside the title of the pages. During the search process, the algorithm will search on the title_inv. If there exist term match the query, 0.10 score will be added directly for title term match each time.

Stopwords Removal and Stemming

The stopwords removal and stemming algorithm is written in the “StopStem.jar”. Before processing the crawled data and the query term, it will help to remove the stopwords according to the lists in the “stopword.txt”. Stemming is also performed at the preprocessing stage. The provided Porter’s algorithm is used to perform the stemming inside the algorithm.

5. Installation Procedure

As mentioned in the overall design, the search engine consists of three major components. Therefore, we need to run those three components separately.

Database	1. Put below files in ROOT: • -project_database.java • -stopwords.txt • -StopStem.class 2. Put below files in ROOT/lib: • -htmlpraser.jar • -rocksdbini-6.0.0-linux64.jar • -jsoup-1.8.1.jar 3. Create directory db in the root folder 4. Compile the file with following comment in the root file: • javac -cp .:lib/* project_database.java 5. Run the file with following comment: • java -cp lib/*:. project_database
Retrieval Engine	1. Put below files in ROOT/db/WEB-INF/classes/Database: • -Search.java • -rocksdbjni-6.0.0-linux64.jar • -StopStem.class 2. Put below files in ROOT/db/WEB-INF/classes/Database/lib: • -htmlpraser.jar • -rocksdbini-6.0.0-linux64.jar • -jsoup-1.8.1.jar 3. Compile the file with following command in Database file: • javac -cp .:lib/* Search.java
Web interface	1. Put below files in ROOT/db/WEB-INF/lib: • -htmlpraser.jar

	• -rocksdbini-6.0.0-linux64.jar • -jsoup-1.8.1.jar 2. Put below files in ROOT: • form.html • afterSubmit.jsp • keywordList.jsp • queryHistory.jsp • similarPages.jsp 3. visit VM_location:8080/form.html to use the search engine

6. Highlight of the Features

Get similar page

After each query search, the search engine will count the term frequency of the terms in the returned pages. And then the similar page function is given by automatically searching the query containing the top 5 frequency terms.

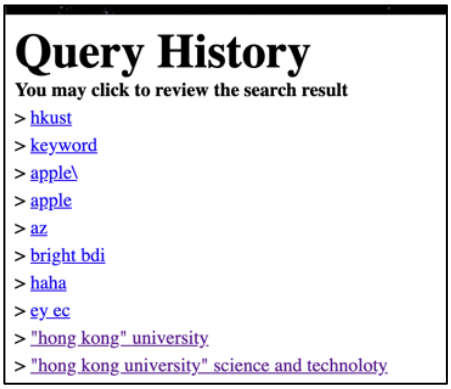
List of keywords, select and combine

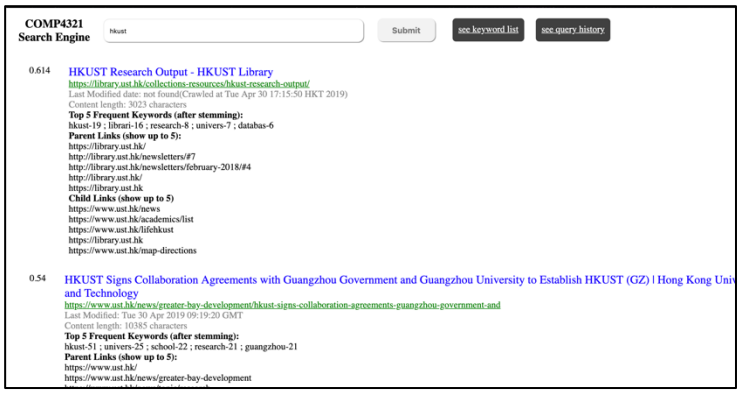
Our database stores the terms that appear in at least 5 documents by alphabet. The keyword list is then provided in the web interface and users could select or deselect the terms to form a query. Note that since we index the document after stop & stem, the keyword here are all stemmed terms.

Query History

When users enter the query using our search engine, the search engine will store the query into database and later on users could check the previous queries and click on it to view the results.

User-friendly Interface





The interface is clear and easy to use. The history included the keyword you type will listed as figure 2 shown. After finishing the query, user can click on the buttons rightward to use the features mentioned above.

Figure 2

Figure 3

Figure 4. see query history

Figure 5. result of “get similar page”

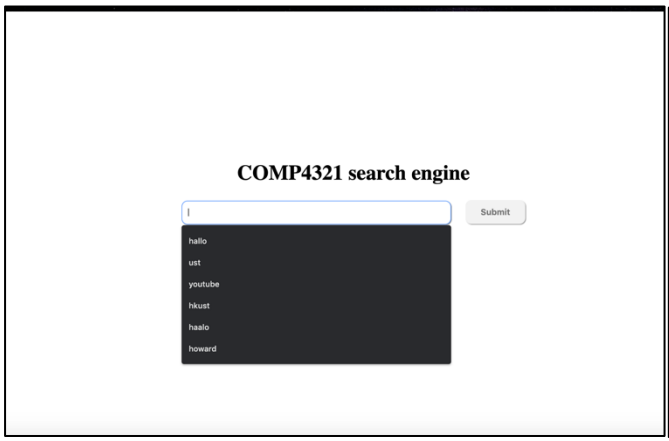
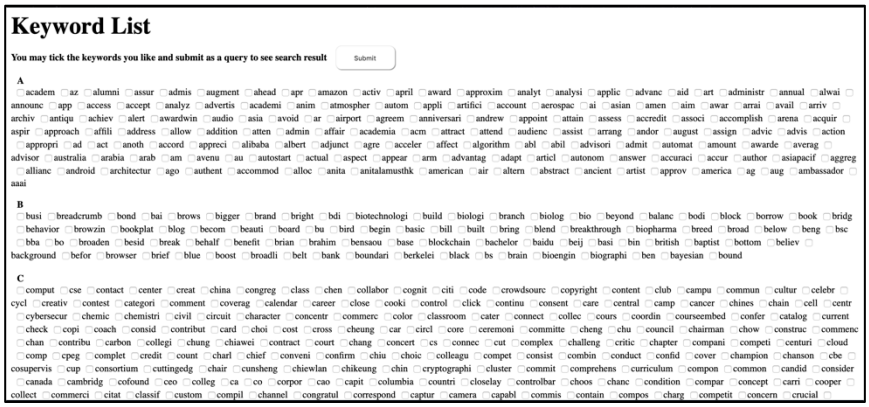


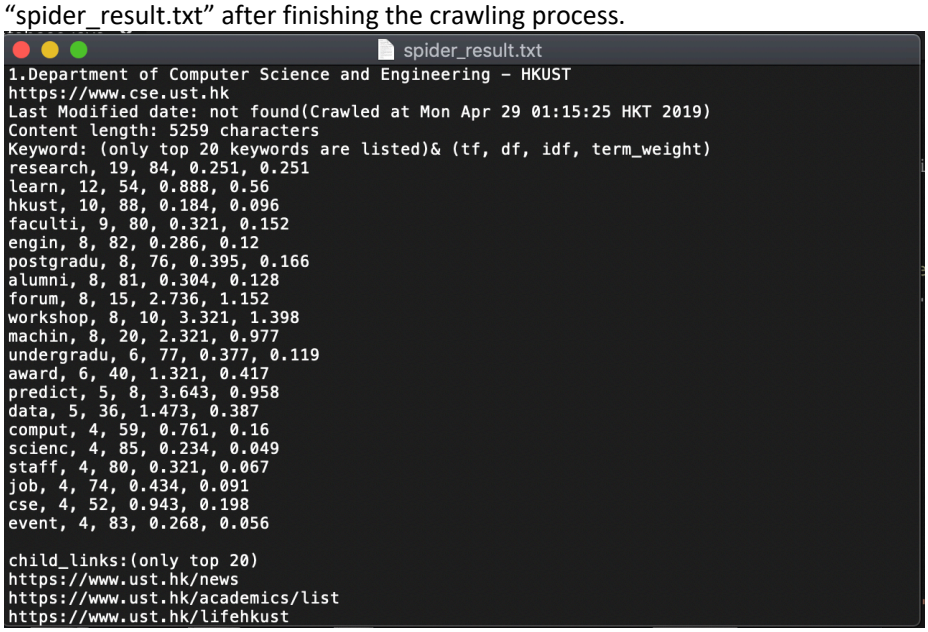
Figure 6. see keyword list



7. Testing

Database

All the database on the data store in will be printed "spider_result.txt" after finishing the crawling process.



As the graph shown, you could find the information like title, keyword, and child links. The numbers output after the keyword is following the order shown on the bracket (tf, df, idf, term weight). Since all the data are extracted from the databases, it could help to check if all the data is input correctly.

Retrieval Engine

When implementing the file "Search.java", 3 options could be chosen as the graph shown below.

```
[hwlamad@vml1wk111 project]$ java -cp lib/*:. Search
Select the option:
1: List of keywords with document frequency>5
2: Enter a query(double quotes to quote the phrase)
3: Print past query history
```

Option 1 could check the words stored with df>10 in alphabetical order. This is used to checked if the word lists are successfully extracted and it will be used on the keyword list search function.

```
w: welcom wechat women wai workshop world wider websit wide wellround
 water wei web wang worldwid wong
x: xiaoi
y: youtub ying yang yeung
z: zhang
Select the option:
1: List of keywords with document frequency>5
2: Enter a query(double quotes to quote the phrase)
3: Print past query history
```

```
Select the option:
1: List of keywords with document frequency>5
2: Enter a query(double quotes to quote the phrase)
3: Print past query history
2
Enter the query please:
["Hong Kong" Univeristy
begin phrase process: hong kong
finish phrase process: hong kong
Query Term 1 Hong Kong
Query Term 2  Univeristy
not phrase part: univeristi
27 0.081
12 0.08
20 0.079
31 0.07
70 0.059
====query is "Hong Kong" Univeristy
==============printing result0
0 0.081
1 Contact Us | HKUST CSE
2 https://www.cse.ust.hk/admin/contact/
3 Last Modified date: not found(Crawled at Mon Apr 29 01:15:37 HKT 2019)
4 Content length: 1679 characters
```

Option 2 is the simulation of the query search performed on the web interface. It's used to check the result output before outputting in the web interface. For the option 3, it's used to check if we could successfully collect the search history.

8.Conclusion

Strength & Weakness

Strength	Weakness
-High reaction speed to handle user query -Easy to maintain and extend functionalities -User friendly web interface -Allow similar page search -Allow phrase search -Title favor mechanism -Could list keyword stored in the database -Memorize the query history to fasten search speed for repeated query	-All terms are assumed to be independent in the vector space model -Could not handle the Chinese or other language that not uses the English alphabet -Not involve algorithm related to authority and hub, thereby the result may be not perfect -Website end with "/", "#" will be treated as different website

Future Improvement

Page Rank

Page rank algorithm could improve the precision and recall significantly, therefore it's planning to increase this feature by calculating the authority and hub of the pages.

Personalization

Since one search cannot fit all, adjusting the results based on the user preference will be necessary in order to improve the search engine. One possible way to collect the user feedback is to monitor the click through rate and use the preference mining strategies to analysis the user preference.

Contribution

Name	Responsible part
Lam Hon Wa	Project database
XU, Feiting	Retrieval algorithm
LI, Junze	Web interface