

Overall Design of the System

In this project, a **web-based search engine** is built.

On the web interface, user can submit request of fetching pages given a starting url and number of pages to fetch. User can also check the existing database, and clear the database. On the other hand, user can search by submitting a search query, even with quotation marks on phrases to boost ranking of pages containing the phrases. Pages will be ranked by descending scores, and at most 50 pages will be displayed. Page Title, url, last modified date, size of page, top 5 keywords and frequency, and parent and child links will be displayed.

Class Spider, Indexer, Database, StopStem, Porter, SearchEngine are used to develop the search engine with several jsp and html files for supporting the web interface.

Class Spider will fetch pages recursively given a starting url and number of pages to fetch. When fetching a page, Spider will call class Indexer to extract words, titles, bigrams, and trigrams, and insert them to the database through class Database. Dealing with the words, stop words will be removed and then stemmed by class StopStem and Porter. After crawling the pages, the $\text{tfidf}/\text{max}(\text{tf})$ matrix will be calculated for each Page and Word, ready for the search process. Class SearchEngine given a query will then calculate scores for each page with favors in matches in title, and matches of phrases, and get the top 50 scores pages.

JDBM Database Scheme of the Indexer

1. PageID – URL Mapping Tables:

PageIDtoURL(PageID, URL), URLtoPageID(URL, PageID)

For simplicity of storing the URL in other databases, an ID is given to the page and recorded on this mapping table. Both directions are recorded, for efficiency of retrieval.

2. Table for title of each web page:

PageIDtoTitle(PageID, Title)

The Table is for recording the title of the fetched page.

3. Table for LastModifiedTime of each web page:

PageIDtoTime(PageID, LastModifiedTime)

The LastModifiedTime is for recording the last modified time of the pages fetched. When the Spider has fetched that page before, and the page is not modified after that, the last modified time from the page and from the table would be the same. Then the Spider will skip this page this time, as it is not modified. This can help handle the cyclic links.

4. Table for Length of each web page:

PageIDtoLength(PageID, Length)

The length is for recording the length of the pages fetched. All three Content-Length, HTML Length, and Number of Words will be recorded as some may be missing fetching a page.

5. WordID – Word Mapping Tables:

WordIDtoWord(WordID, Word), WordtoWordID(Word, WordID)

For simplicity of storing the word in other databases, an ID is given to the word and recorded on this mapping table. Both directions are recorded, for efficiency of retrieval.

6. Forward and Inverted Indexes for storing Parent – Child relationship:

ParenttoChild(ParentPageID, ChildPageID), ChildtoParent(ChildPageID, ParentPageID)

These tables store the Parent – Child relationships. Both directions of retrieval are recorded, for efficiency.

7. Forward and Inverted Indexes for page and title words on page:

PageIDtoTitleWordID(PageID, WordID), TitleWordIDtoPageID(WordID, PageID)

These tables store the title words crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

8. Forward and Inverted Indexes for page and word on page:

PageIDtoWordID(PageID, WordID), WordIDtoPageID(WordID, PageID)

These tables store the words crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

9. Forward Index for page and top five word on page:

PageIDtoTopFiveWordID(PageID, WordID)

The table store the top five words crawled from the web pages.

10. Forward and Inverted Indexes for page and title word bigrams on page:

PageIDtoBiTitleWordID(PageID, WordID), BiTitleWordIDtoPageID(WordID, PageID)

These tables store the title word bigrams crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

11. Forward and Inverted Indexes for page and title word trigrams on page:

PageIDtoTriTitleWordID(PageID, WordID), TriTitleWordIDtoPageID(WordID, PageID)

These tables store the title word trigrams crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

12. Forward and Inverted Indexes for page and word bigrams on page:

PageIDtoBiWordID(PageID, WordID), BiWordIDtoPageID(WordID, PageID)

These tables store the word bigrams crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

13. Forward and Inverted Indexes for page and word trigrams on page:

PageIDtoTriWordID(PageID, WordID), TriWordIDtoPageID(WordID, PageID)

These tables store the word Trigrams crawled from the web pages. Both directions of retrieval are recorded, for efficiency.

14. Table for TFxIDF:

PageIDtoTFxIDF(PageID, TFxIDF)

The table stores the TFxIDF weights of web pages and words.

15. Table for SearchHistory:

SearchHistory(Index, SearchHistory)

The table stores the past search history inputted by the user.

Algorithms

1. Crawl

If a page is not fetched before, it will not be in the database URLtoPageID, then it will be fetched. If a page is fetched before, it will be in the database URLtoPageID and last modified time will be extracted from the database PageIDtoTime. If the new last modified time is earlier than or same as the one in database, then it will be ignored, and next page in pages_queue will be tried to be fetched. If new last modified time is later, then it will be fetched.

Fetching a page, it will be added to visited_pages (for this run) to avoid circular loop. Titles, words, bigrams, trigrams will be extracted and stored after stop word removal and stemming. At the same time, the Indexer will also keep track of the top five frequent words. Links will be stored and added to pages_queue, if not visited.

After fetching the required number of pages, TfxIDF/max(TF) matrix will be calculated and stored to the database.

2. Search

After extracting phrases from the query, and performing stop word removal and stemming, scores of each page will be calculated using the formula below.

For each matching of query word and title word,

$$AddScore = 1 * \log_2 \left(\frac{N}{df_i} \right)$$

N is the number of documents

df_i is the number of documents with title containing word i

For each matching of query word and word in page,

$$AddScore = \frac{tf_{ij}}{tfmax_i} * \log_2 \left(\frac{N}{df_j} \right)$$

tf_{ij} is the frequency of term j in document i

$tfmax_i$ is the max term frequency in document i

df_j is the number of documents with title containing word j

As $tf/\max(tf)$ is upper bounded by 1, the score added from matching in title would usually be larger (if similar df). Also the scores from matching in title is considered as a bonus besides of the score from $tfidf/\max(tf)$, which gives favors in matching in title.

For each matching of query phrase and the title word/bigram/trigram,

$$AddScore = 2 * 1 * \log_2 \left(\frac{N}{df_i} \right)$$

N is the number of documents

df_i is the number of documents with title containing phrase i

For each matching of query phrase and word/bigram/trigram in page,

$$AddScore = 2 * \frac{tf_{ij}}{tfmax_i} * \log_2 \left(\frac{N}{df_j} \right)$$

tf_{ij} is the frequency of phrase j in document i

$tfmax_i$ is the max phrase frequency in document i

df_j is the number of documents with title containing phrase j

Similar to the score from matching in title, the scores from matching phrases in title/page is considered as a bonus besides of the score from $tfidf/\max(tf)$, which gives favors in matching in phrases. The coefficient 2 is to weight more on phrase match than normal title/page match.

Finally, after calculating the scores for each page, pages will be sorted in descending order of scores. Top 50 Pages with non-zero scores will be normalized to 0 to 100 and printed on the web interface.

Installation Procedure

Download the files on Github and unzip it

Add environment var:

`CATALINA_HOME = "your tomcat path (e.g. /Users/ABC/apache-tomcat-9.0.74)"`

`JAVA_HOME = "your jdk path (e.g. /Users/ABC/jdk-20.0.1.jdk/Contents/Home)"`

Go to `$CATALINA_HOME/webapps` and create a folder "Search"
Put the files under `$CATALINA_HOME/webapps/Search`

Then build the program by:

`> javac -cp combined.jar WEB-INF/classes/project/*.java`

Then start it by:

`> $CATALINA_HOME/bin/startup.sh`

Open in your browser:

<http://localhost:8080/Search/Search.html>

to test the program

Shut it down by:

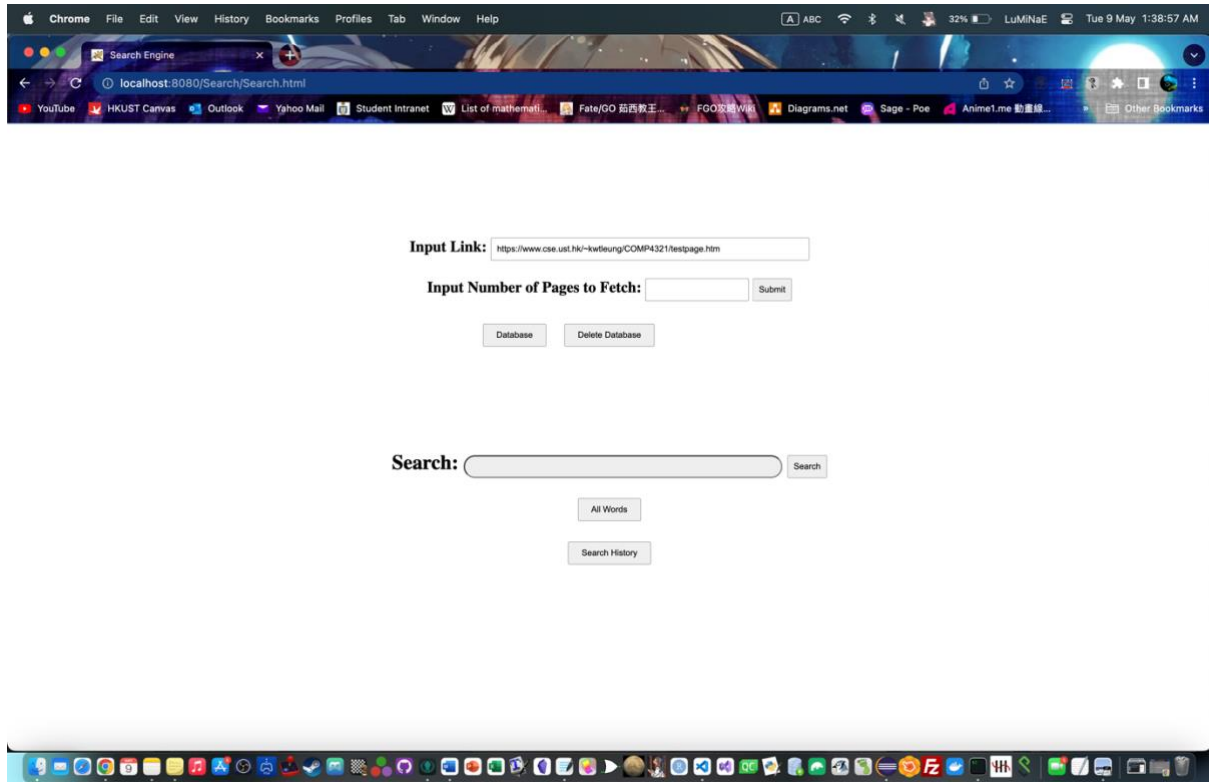
`> $CATALINA_HOME/bin/shutdown.sh`

Highlight of Features

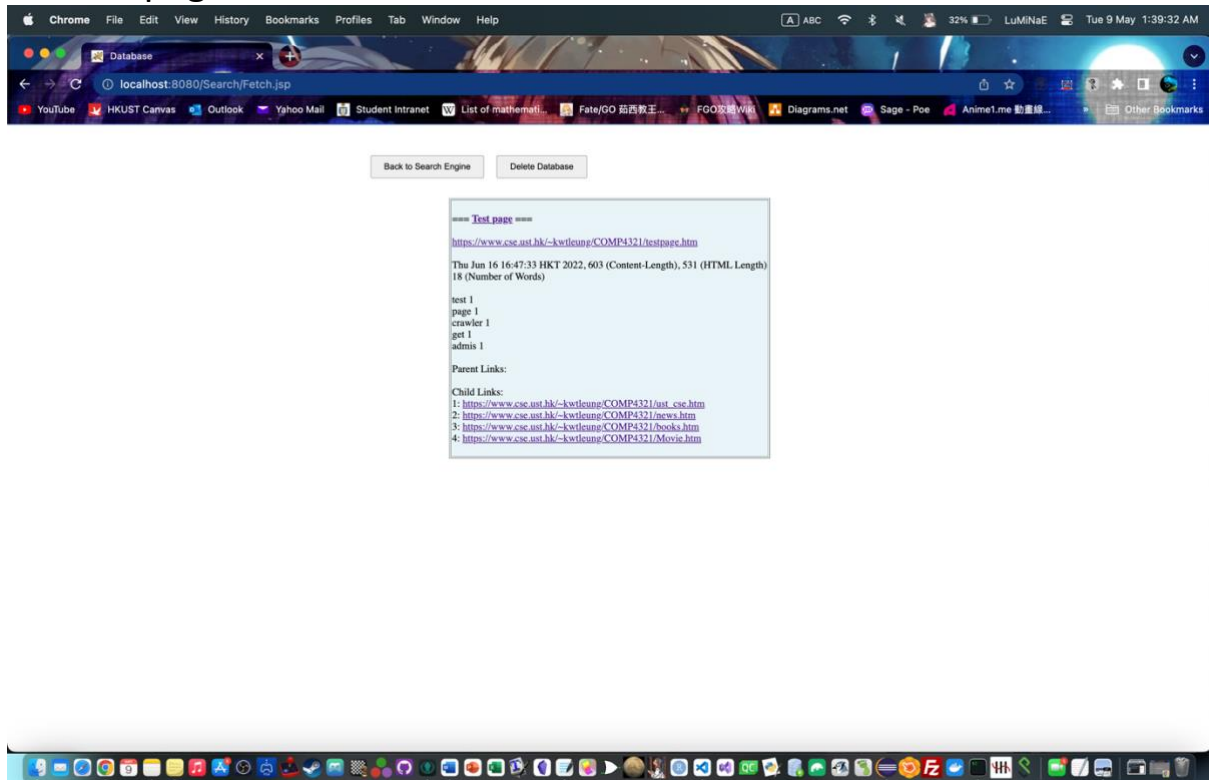
1. **Get Similar Pages:** equivalent to search pages with the top 5 most frequent keywords from that page.
2. **List of Stemmed Keywords (Dictionary):** you can browse through and select keywords to search.
3. **Query Search History:** you can browse through and select queries to search.
4. **Delete Database and Delete History Button:** you can delete database and past search history by one click.
5. **Sensitive Words:** if you add words in sensitivewords.txt, and if your query submitted contains any of the sensitive word, you will get nothing, but a reminder of words that violates the regulation.

Testing of Functions

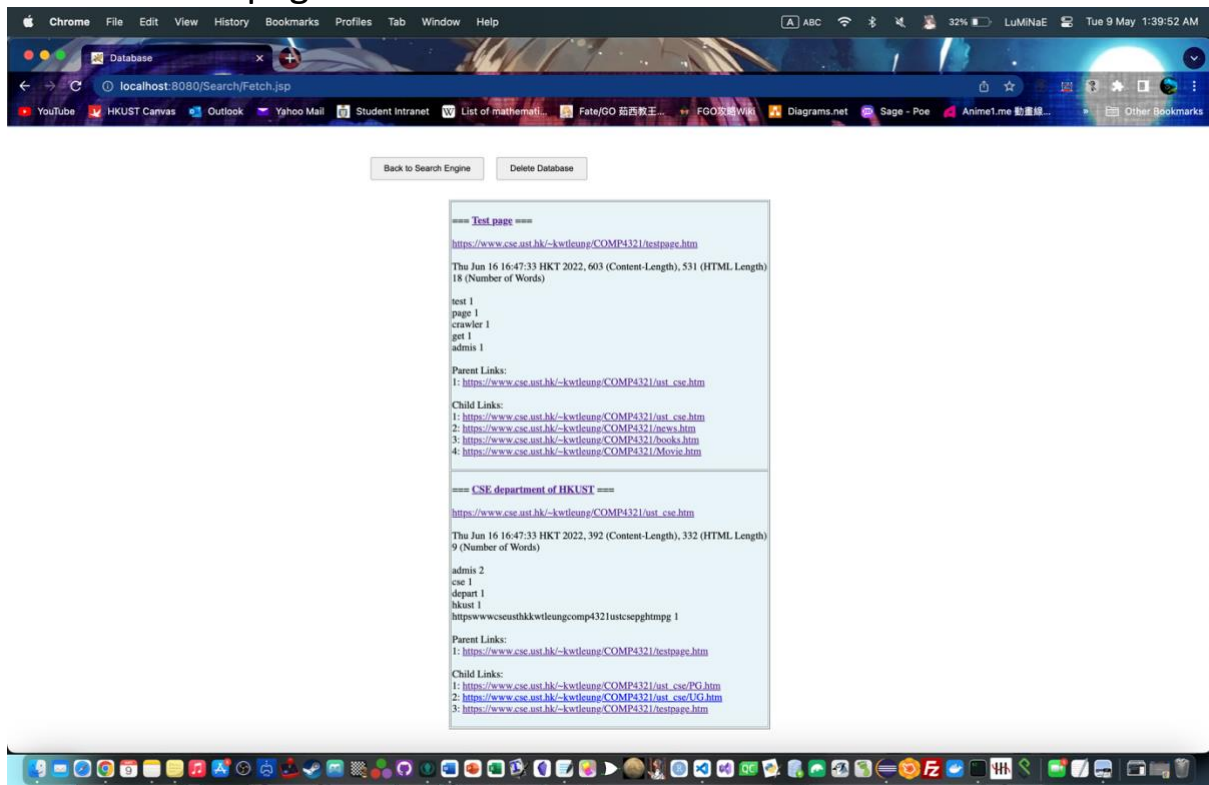
Web Interface



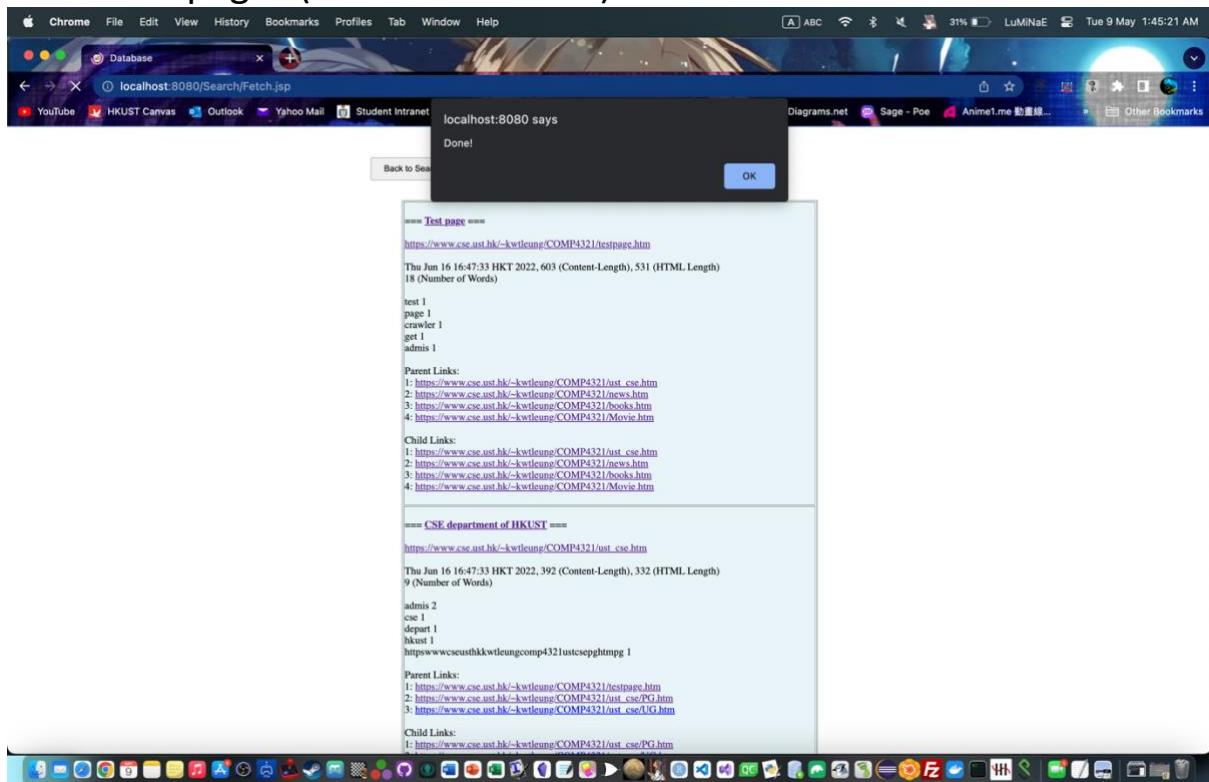
Fetch 1 page



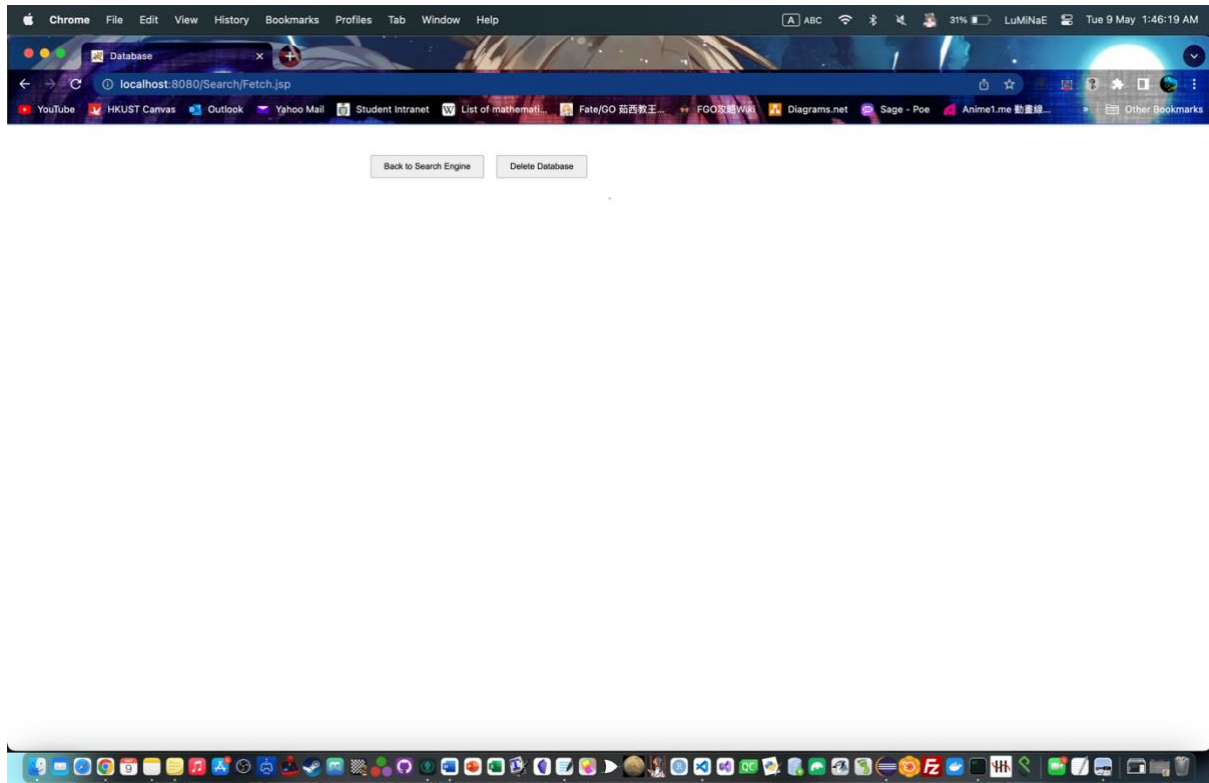
Fetch 1 more page



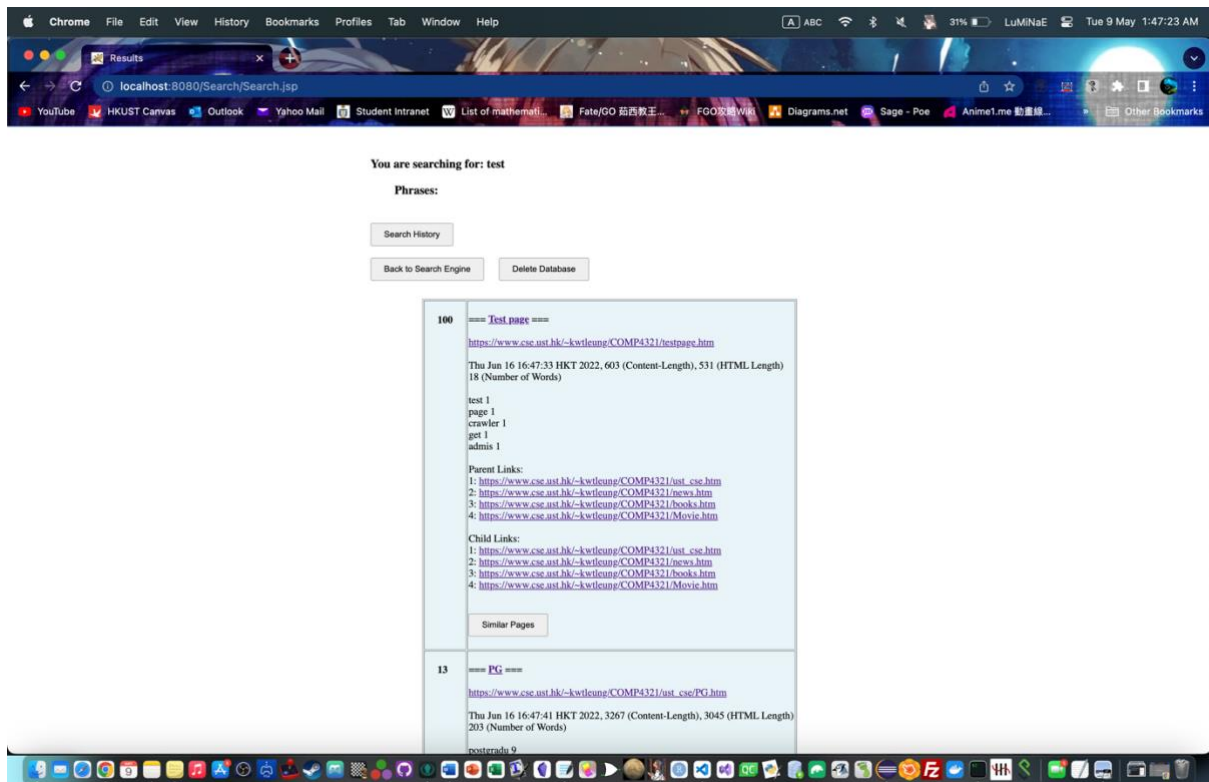
Fetch 500 pages (Around 2-3 mins)



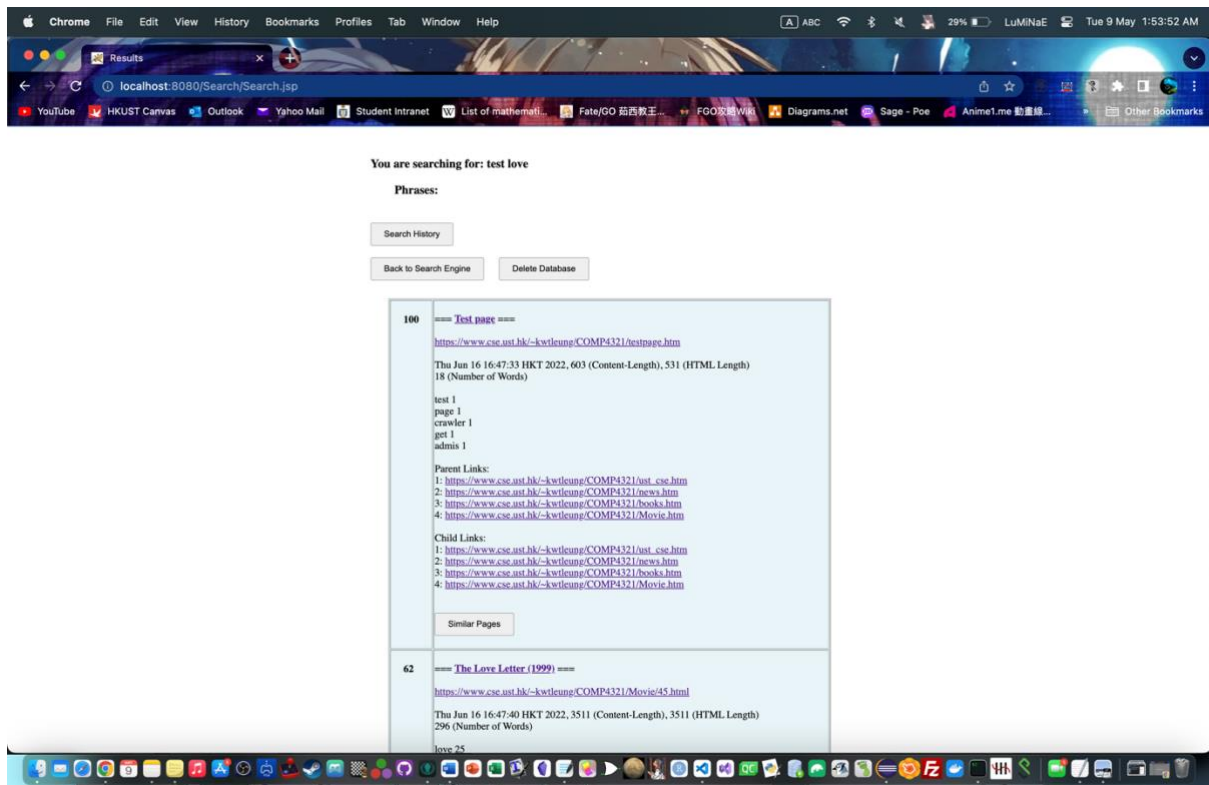
Delete Database



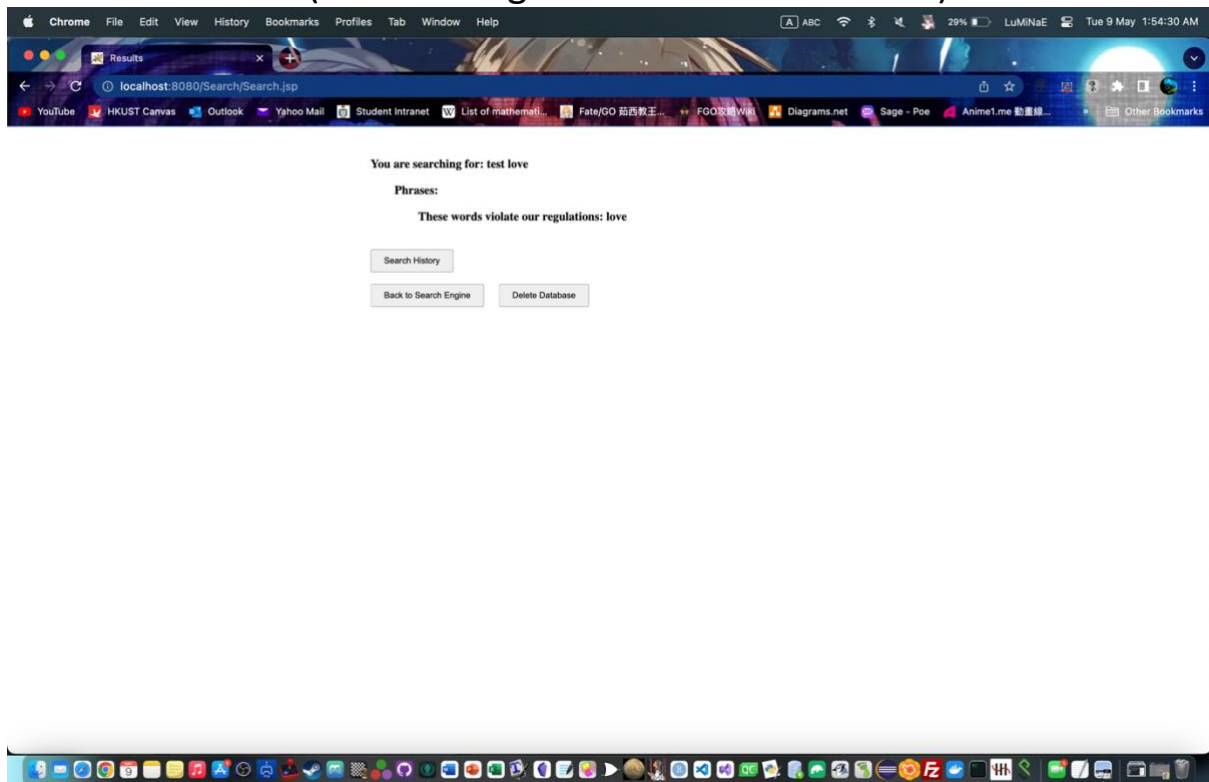
Search Test



Search Test Love



Search Test Love (after having Love as Sensitive Word)



Search movie Amazing Grace Hymn

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Search/Search.jsp'. The search results page has a header that says 'You are searching for: movi amaz grace hymn' and 'Phrases:'. Below this are buttons for 'Search History', 'Back to Search Engine', and 'Delete Database'. The search results are displayed in a list format. The first result is for 'Jonah: A Year's Tales Movie: Bonus Material (2002)' with a URL 'https://www.cse.ust.hk/~kwleung/COMP4321/Movie/51.html'. It includes metadata such as 'Thu Jun 16 16:47:40 HKT 2022, 2781 (Content-Length), 2781 (HTML Length) 203 (Number of Words)' and a list of tags: 'search 15', 'movi 15', 'titi 10', 'vegetal 8', 'jonah 7'. It also provides 'Parent Links' and 'Child Links' to the same URL. The second result is for 'The Powerpuff Girls Movie (2002)' with a URL 'https://www.cse.ust.hk/~kwleung/COMP4321/Movie/84.html'. It includes metadata such as 'Thu Jun 16 16:47:41 HKT 2022, 8532 (Content-Length), 8532 (HTML Length) 690 (Number of Words)' and a list of tags: 'movi 18', 'powerpuff 11', 'rate 11', 'imdb 11', 'user 10'. It also provides 'Parent Links'.

You are searching for: movi amaz grace hymn

Phrases:

Search History

Back to Search Engine

Delete Database

100

==== **Jonah: A Year's Tales Movie: Bonus Material (2002)** ====

<https://www.cse.ust.hk/~kwleung/COMP4321/Movie/51.html>

Thu Jun 16 16:47:40 HKT 2022, 2781 (Content-Length), 2781 (HTML Length) 203 (Number of Words)

search 15
movi 15
titi 10
vegetal 8
jonah 7

Parent Links:
1: <https://www.cse.ust.hk/~kwleung/COMP4321/Movie.htm>

Child Links:
1: <https://www.cse.ust.hk/~kwleung/COMP4321/Movie.htm>

Similar Pages

100

==== **The Powerpuff Girls Movie (2002)** ====

<https://www.cse.ust.hk/~kwleung/COMP4321/Movie/84.html>

Thu Jun 16 16:47:41 HKT 2022, 8532 (Content-Length), 8532 (HTML Length) 690 (Number of Words)

movi 18
powerpuff 11
rate 11
imdb 11
user 10

Parent Links:

Search movie "Amazing Grace" Hymn

The screenshot shows a web browser window with the address bar displaying 'localhost:8080/Search/Search.jsp'. The search results page has a header that says 'You are searching for: movi amaz grace hymn' and 'Phrases: amaz grace,'. Below this are buttons for 'Search History', 'Back to Search Engine', and 'Delete Database'. The search results are displayed in a list format. The first result is for 'Silkwood (1983)' with a URL 'https://www.cse.ust.hk/~kwleung/COMP4321/Movie/83.html'. It includes metadata such as 'Thu Jun 16 16:47:41 HKT 2022, 7241 (Content-Length), 7241 (HTML Length) 601 (Number of Words)' and a list of tags: 'imdb 11', 'movi 11', 'silkwood 10', 'user 10', 'rate 8'. It also provides 'Parent Links' and 'Child Links' to the same URL. The second result is for 'Unconstitutional: The War on Our Civil Liberties (2004)' with a URL 'https://www.cse.ust.hk/~kwleung/COMP4321/Movie/205.html'. It includes metadata such as 'Thu Jun 16 16:47:40 HKT 2022, 7114 (Content-Length), 7114 (HTML Length) 562 (Number of Words)' and a list of tags: 'archiv 16', 'imdb 11', 'user 10', 'movi 9', 'rate 8'. It also provides 'Parent Links'.

You are searching for: movi amaz grace hymn

Phrases: amaz grace,

Search History

Back to Search Engine

Delete Database

100

==== **Silkwood (1983)** ====

<https://www.cse.ust.hk/~kwleung/COMP4321/Movie/83.html>

Thu Jun 16 16:47:41 HKT 2022, 7241 (Content-Length), 7241 (HTML Length) 601 (Number of Words)

imdb 11
movi 11
silkwood 10
user 10
rate 8

Parent Links:
1: <https://www.cse.ust.hk/~kwleung/COMP4321/Movie.htm>

Child Links:
1: <https://www.cse.ust.hk/~kwleung/COMP4321/Movie.htm>

Similar Pages

93

==== **Unconstitutional: The War on Our Civil Liberties (2004)** ====

<https://www.cse.ust.hk/~kwleung/COMP4321/Movie/205.html>

Thu Jun 16 16:47:40 HKT 2022, 7114 (Content-Length), 7114 (HTML Length) 562 (Number of Words)

archiv 16
imdb 11
user 10
movi 9
rate 8

Parent Links:

Results

localhost:8080/Search/Search.jsp

YouTube HKUST Canvas Outlook Yahoo Mail Student Intranet List of mathemat... Fate/GO 茆西教王... FOO 及 其 We Diagrams.net Sage - Poe Anime1.me 動畫線... Other Bookmarks

You are searching for: movi amaz grace hymn

Phrases: amaz grace hymn,

Search History

Back to Search Engine Delete Database

100 == Silkwood (1983) ==

<https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/83.html>

Thu Jun 16 16:47:41 HKT 2022, 7241 (Content-Length), 7241 (HTML Length)
601 (Number of Words)

imdb 11
movi 11
silkwood 10
user 10
rate 8

Parent Links:
1: <https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm>

Child Links:
1: <https://www.cse.ust.hk/~kwtleung/COMP4321/Movie.htm>

Similar Pages

34 == Jonah: A VeggieTales Movie: Bonus Material (2002) ==

<https://www.cse.ust.hk/~kwtleung/COMP4321/Movie/51.html>

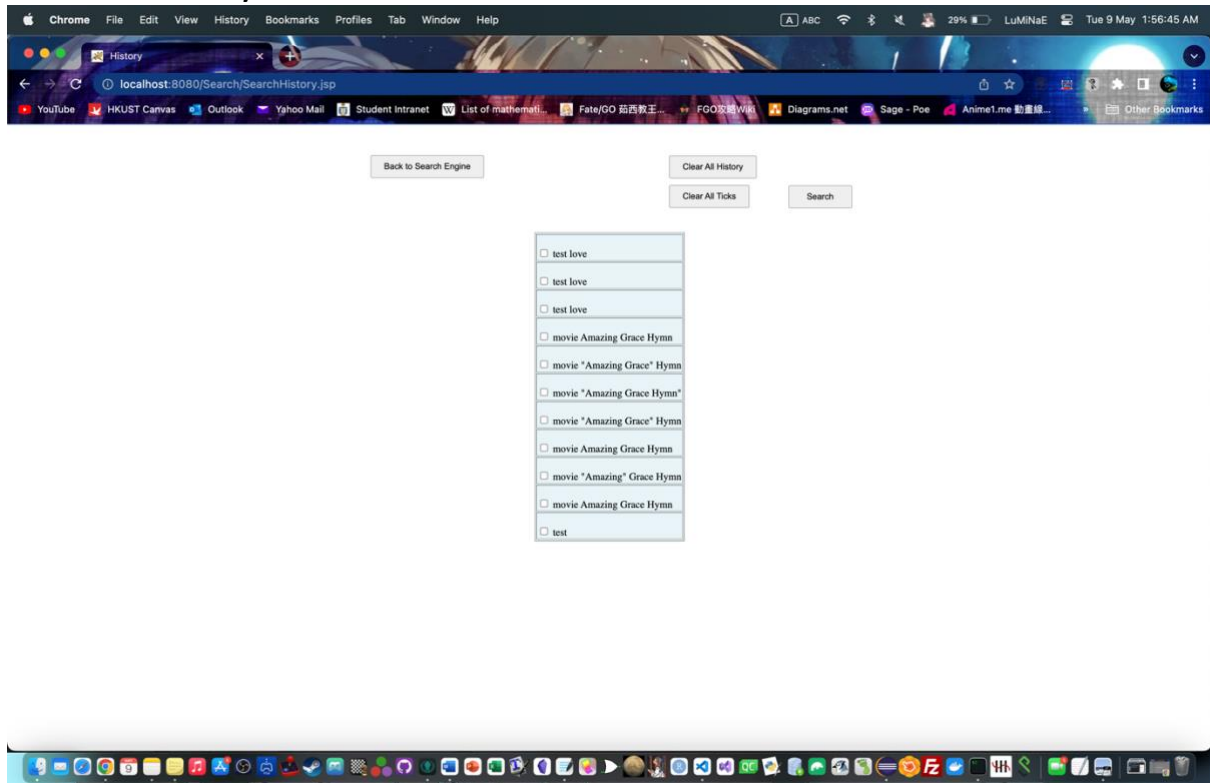
Thu Jun 16 16:47:40 HKT 2022, 2781 (Content-Length), 2781 (HTML Length)
203 (Number of Words)

search 15
movi 15
titi 10
veggietal 8
jonah 7

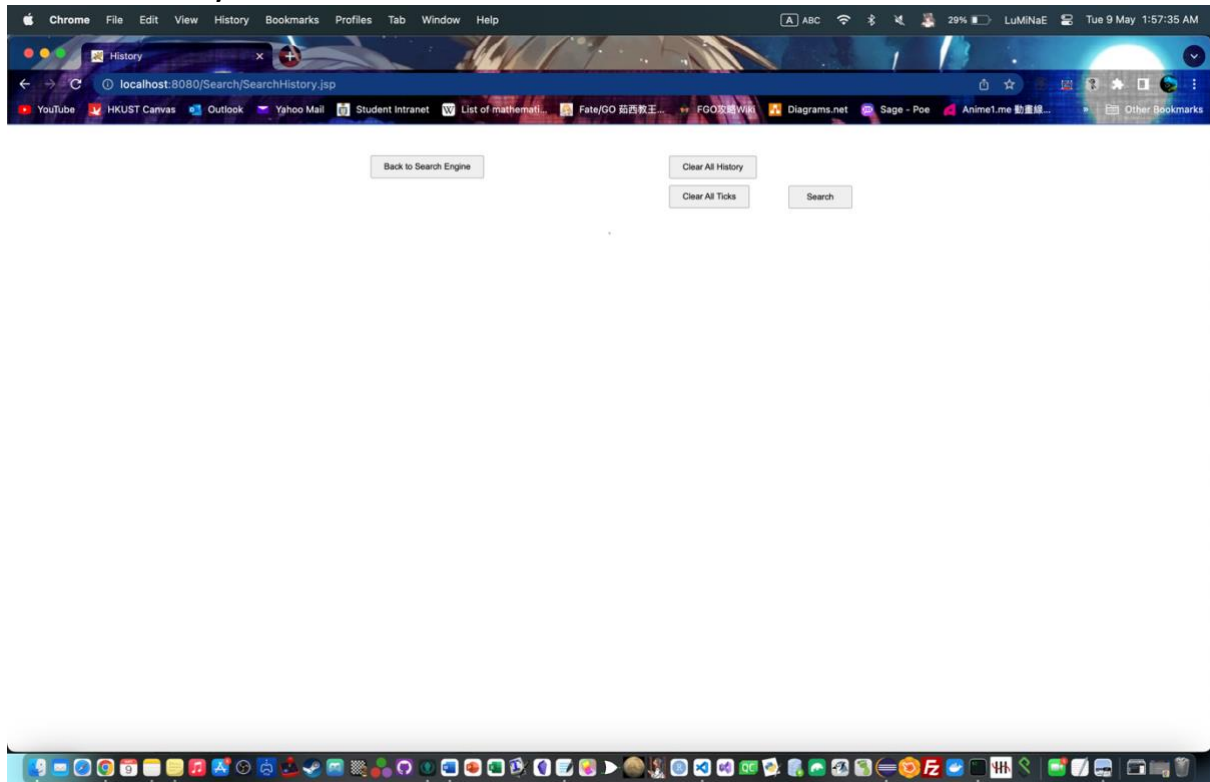
Parent Links:

Dictionary

Search History



Clear History and Reload



Conclusion

The strengths of my system are the number of extra features, the algorithm of flavoring title phrase matching, and a reasonable time to fetch 300+ pages.

The weaknesses of my system are improvable outlook, code can be simplified.

If I could re-implement the whole system, I would have put more effort on visualization (outlook of the web) and factoring the code.

Interesting features I would like to add to my feature are pagerank, but it would require much more effort.