COMP 4321 - Search Engines for Web and Enterprise Data

Project – Final report

InstaSearch

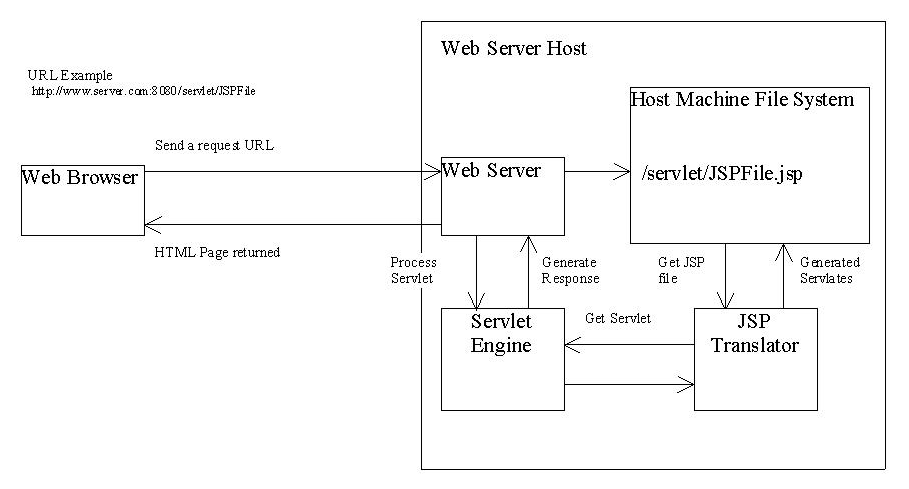
Group member:

MOK Chi Wing 20213055

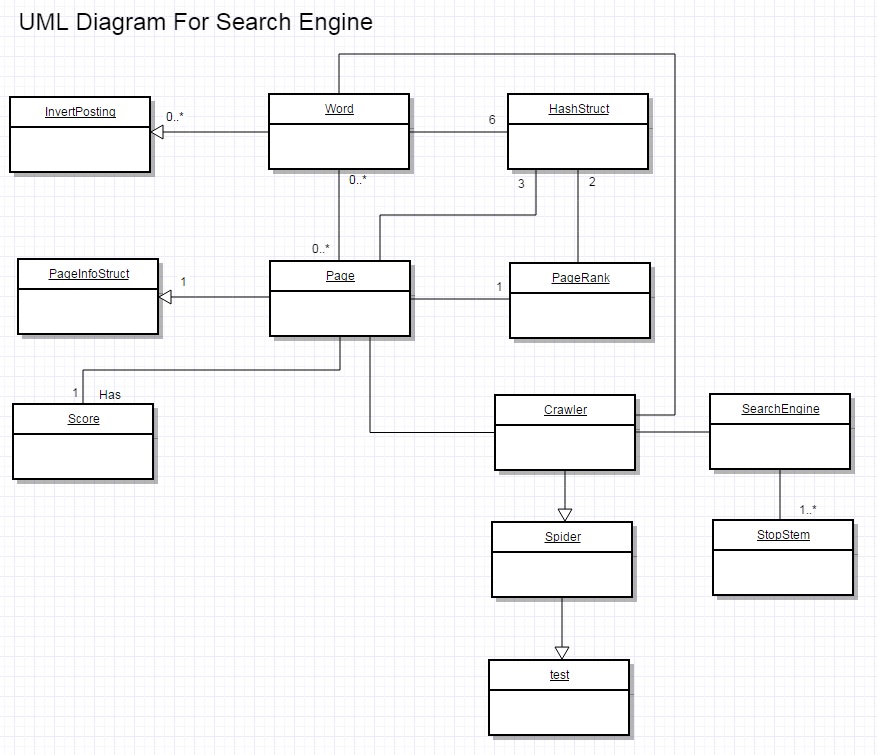
CHOY Ting Wai 20211980

CHU Kwok Ning 20212001

# 1 Overall design of the system



In this project, we make use of JSP and HTML to handle user query and display result, jdbm as database and JAVA as our spider’s programming language.



The system can be break down into two parts, namely, spider and search engine interface. For spider, it will index 300 pages from the target website using a breadth-first strategy and store it into jdbm. For search engine interface, it accept user query and calculate the similarity, using Cosine similarity, and ranking, using both cosine similarity and Google PageRank. Finally, it will display the top 50 results in descending order of their scores.

# 2 The file structures used in the index database

We use hash table in our jdbm database.

The format of each table is as Table\_Name (Key, Value)

1. **pageId (URL, PageId)**

This is a mapping table for URL to reduce the overhead for other index tables. Each unique URL will be assigned a unique ID, e.g.: 0000000, 00000001 etc.

1. **page (PageId, URL)**

This is a mapping table for retrieve page URL. Our system can make use of this table to retrieve the specific page URL by giving the specific page ID.

1. **wordId (Word, WordId)**

This is a mapping table for word to reduce the overhead for other index tables.

Each unique word will be assigned a unique ID, e.g.: 0000000, 00000001 etc.

1. **word (WordId, word)**

This is a mapping table for retrieve word. Our system can make use of this table to retrieve the specific word by giving the specific word ID.

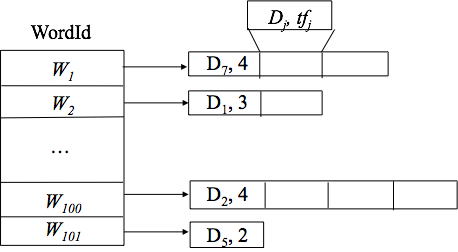
1. **pageInfo (pageId, (title, last modification, size))**

This is a table that stores page information, including title, last modification time and size, for each page by page ID.

1. **BodyWordPosting (WordId, List<(PageId, term frequency, List<word position>)>)**
2. **TitleWordPosting (WordId, List<(PageId, term frequency, List<word position>)>)**

BodyWordPosting and TitleWordPosting store the posting list for each word by word ID. BodyWordPosting will handle all the word which extracted from the pages, while TitleWordPosting only handle the title’s word extracted from the pages. Our search engine will derive and implement a mechanism to favor matches in title. For example, a match in the title would significantly boost the rank of a page. The posting list will store the Page ID, term frequency in the document and the word position. This will be used to calculate cosine similarity.

The word position will be used to support phase searching in our search engine.



1. **InvertedBodyWordPosting (PageId, List<WordId, term frequency>)**
2. **InvertedTitleWordPosting (PageId, List<WordId, term frequency>)**



InvertedBodyWordPosting and InvertedTitleWordPosting are the inverted posting list which is used to support deletion of a webpage entry. It will store the list of word ID and term frequency in the page by page ID. Upon deletion, we will retrieve the forwarding list for this particular webpage and update the inverted posting list accordingly.

1. **ParentLink (PageId, List<Parent link>)**
2. **ChildLink (PageId, List<Child link> )**

ParentLink and ChildLink store the parent link and child link for each page by page ID. These tables are used to implement Google PageRank in final phase.

**12. PageRankScore(PageId, Score)**

PageRankScore stores the score of Google PageRank. Pages will have their own pre-calculated score. It is equivalent to the authority weight for each page.

# 3 Algorithms used

**3.1 Term weighting**

wt = tf x idf/max(tf), where

tf = frequency of term

idf = inverse document frequency of term = log2(N/df)

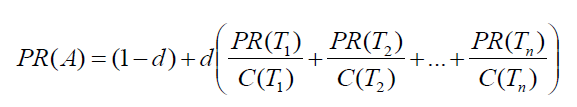
**3.2 Cosine Similarity Measures**

Cosine similarity measures the cosine of the angle between two vectors

**CosSim(D, Q) = ,**

The weight of each query term is 1.

**3.3 Google PageRank**



C(x) is the number of links going out of page x

d is a damping factor which set to be 0.85

The total iteration of calculation is 20.

Google PageRank is used to rank the top-50 result. Page that has large PageRank will receive a bonus score.

**3.4 Mechanism for favoring title matches**

The match in Title will receive a larger bonus score than match in body word. The ratio of scores Title:Body word is set to be 1.5 times of wt (= tf x idf/max(tf))

**3.5 Insertion sort for Ranking**

Insertion sort is used to sort the top-5 frequency of term in each page and the top-50 scores page in result list.

**3.6 Phrase search**

It is done by checking whether the words in the phrase are in consecutive positions inside documents. Multiple phrases search is supported.

# 4 Installation procedure

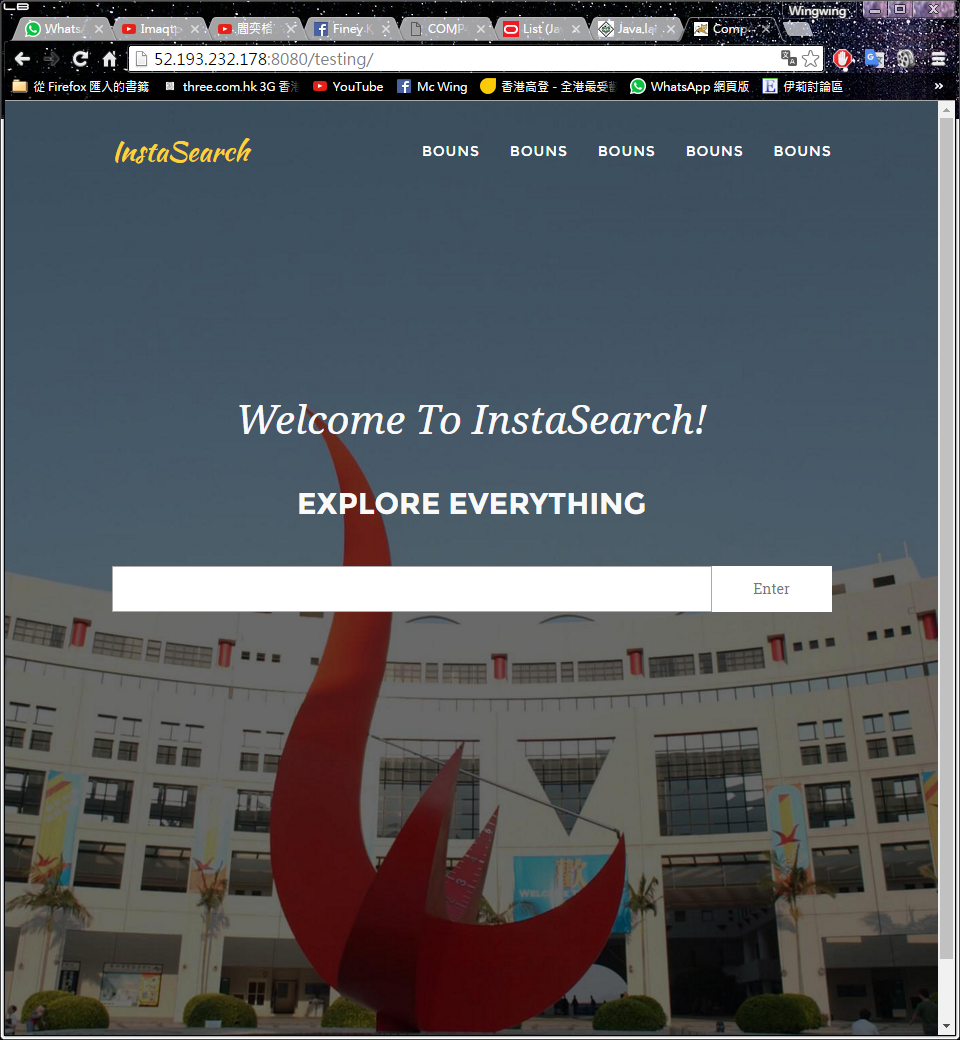
The followings have shown the installation process of our system:

1. Import all the .java file into eclipse.
2. Run Spider.java in eclipse to generate searchEngine.db and searchEngine.lg, it may take a few minutes depend on computation power of the computer
3. Set up a server and install tomcat.
4. Setup the path of the application as “/opt/tomcat/webapps/intrasearch/”
5. Place searchEngine.db and searchEngine.lg to /opt/tomcat/webapps/intrasearch/
6. Place all \*.jsp and file inside /opt/tomcat/webapps/intrasearch/ folder in the server
7. Place all \*.class file to WEB-INF/classes/ in the server
8. Place all \*.jar to WEB-INF/lib/ in the server
9. Change directory to /opt/tomcat/webapps/intrasearch/
10. Type chmod g+r WEB-INF –R
11. Type chmod g+rx \*.jsp

# 5 Highlight of features beyond the required specification

**5.1 User-friendly interface**

Our search engine interface makes use of HTML5, CSS, JavaScript to achieve a attractive, dynamic layout. The following screenshot has shown our search engine interface.

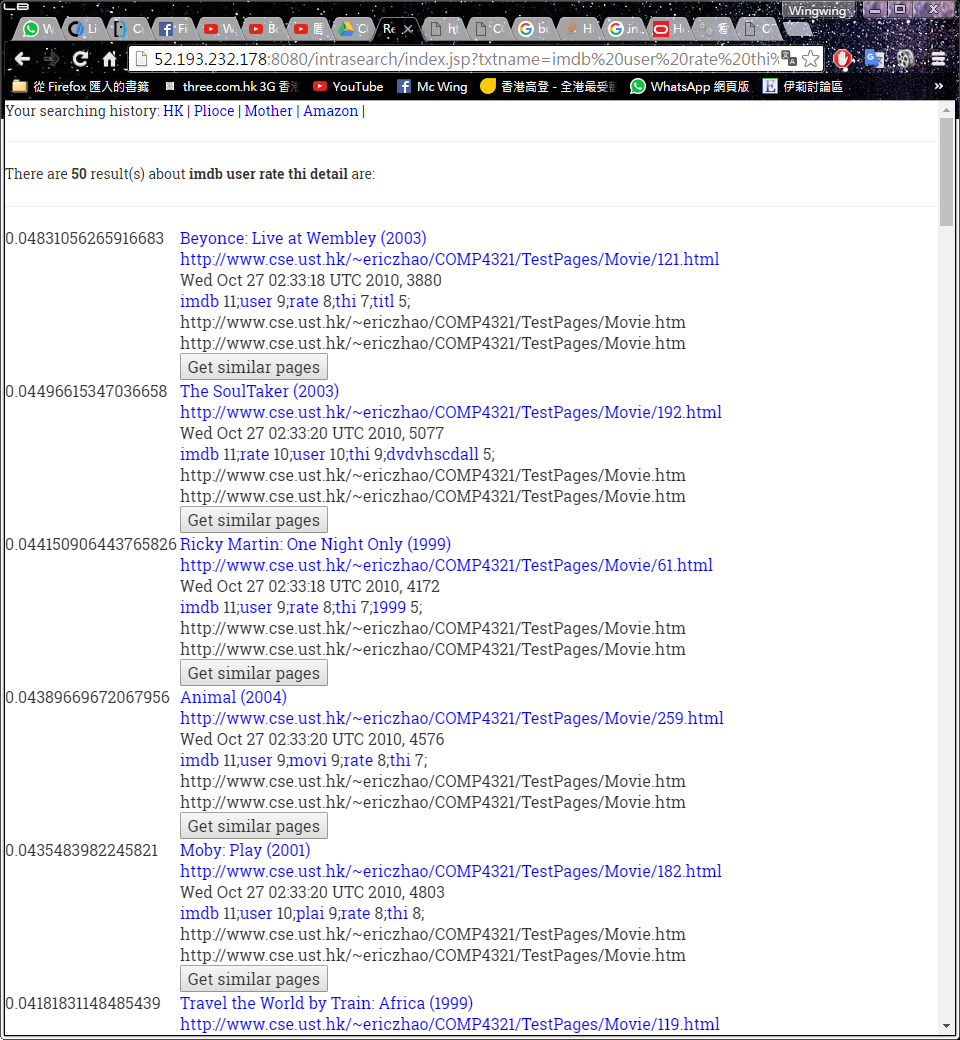


**5.2 Consider links in result ranking (Page Rank)**

In our search engine, links of the page will be consider in result ranking. We make use of Google PageRank to calculate the PR value for each page and store it in a table in JDBM. The process can be done offline and hence would not affect the indexing speed of our search engine. Then, the score of the page equal to Cosine similarity \* PR value of the page. And the result is sorted in descending order of their score.

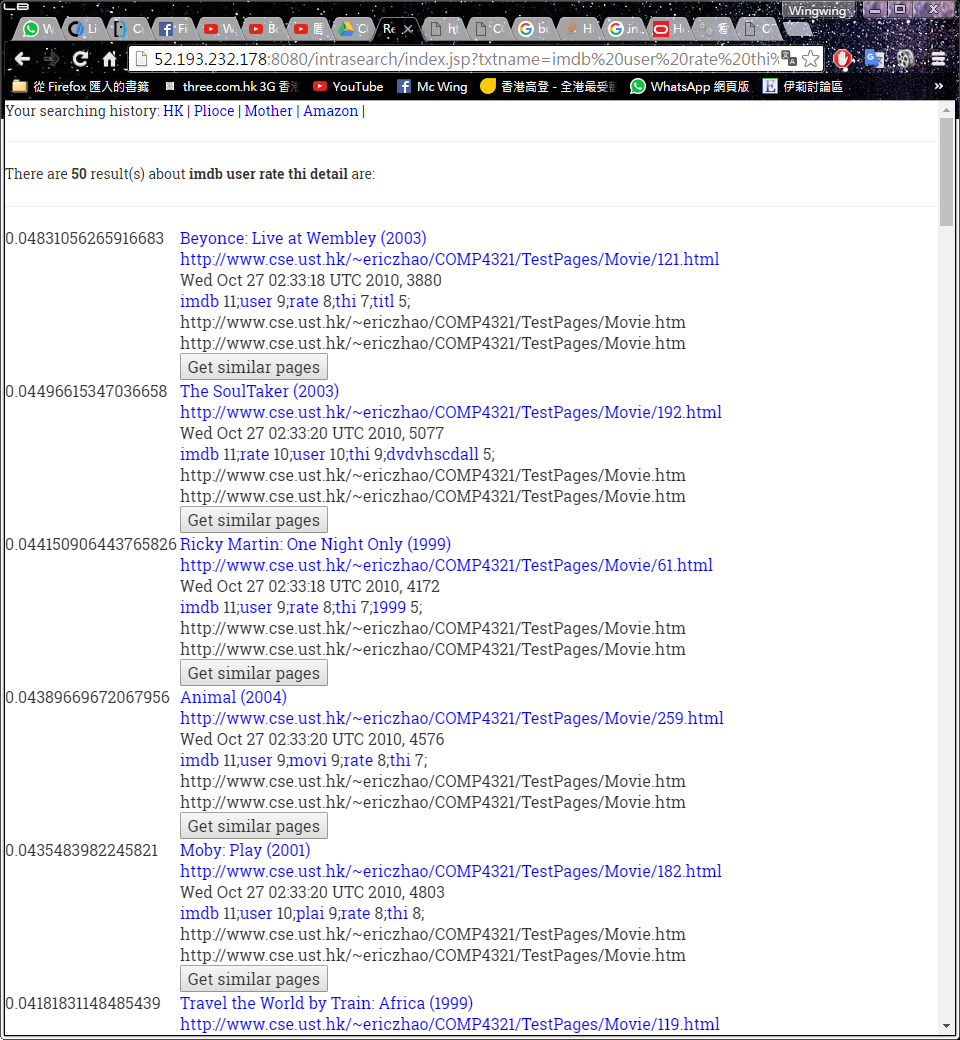
**5.3 Get similar pages**

Clicking the “Get similar pages” button will extract the top 5 most frequent keywords (excluding stop words) from the page; rewrite the original query and submit the revised query for a new search.



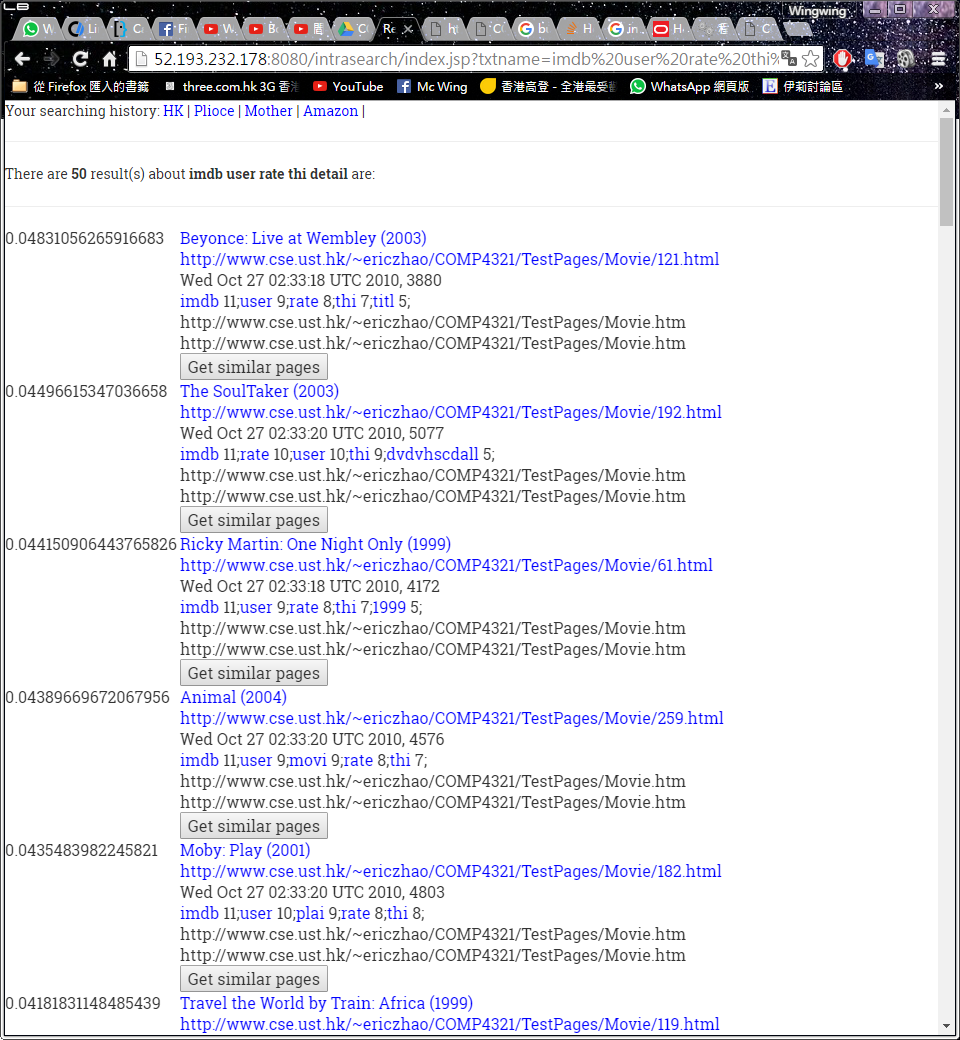
**5.4 See list of stem word and re-submit them as a vector-space query**

Allow the user to see a top 5 freq of all (stemmed) keywords indexed in the page, browse through them, and select the keywords he/she is interested in, and then submit them as a vector-space query to your search engine.



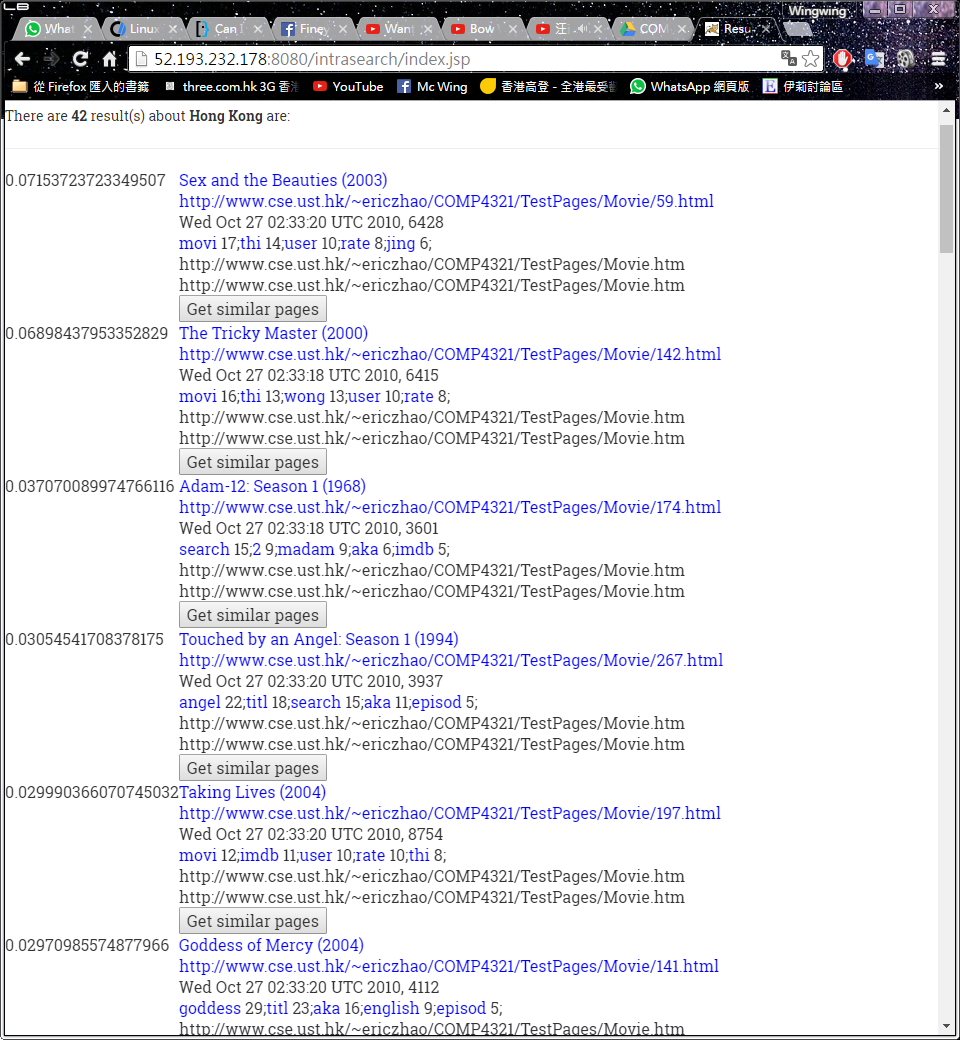
**5.5 Keep track of query history using cookies**

Use cookies to store the query history and allow users to submit new query of a previous query by clicking the cookies.

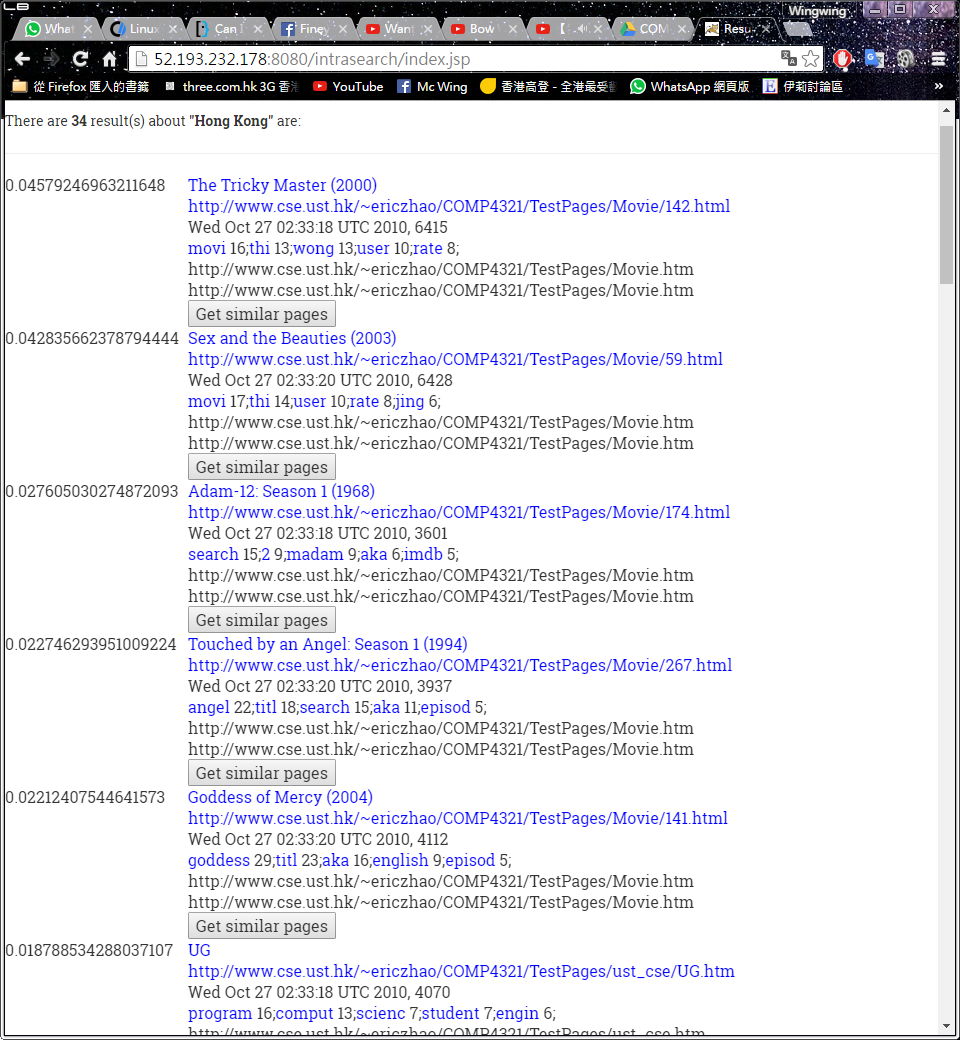


# 6 Testing of the functions implemented

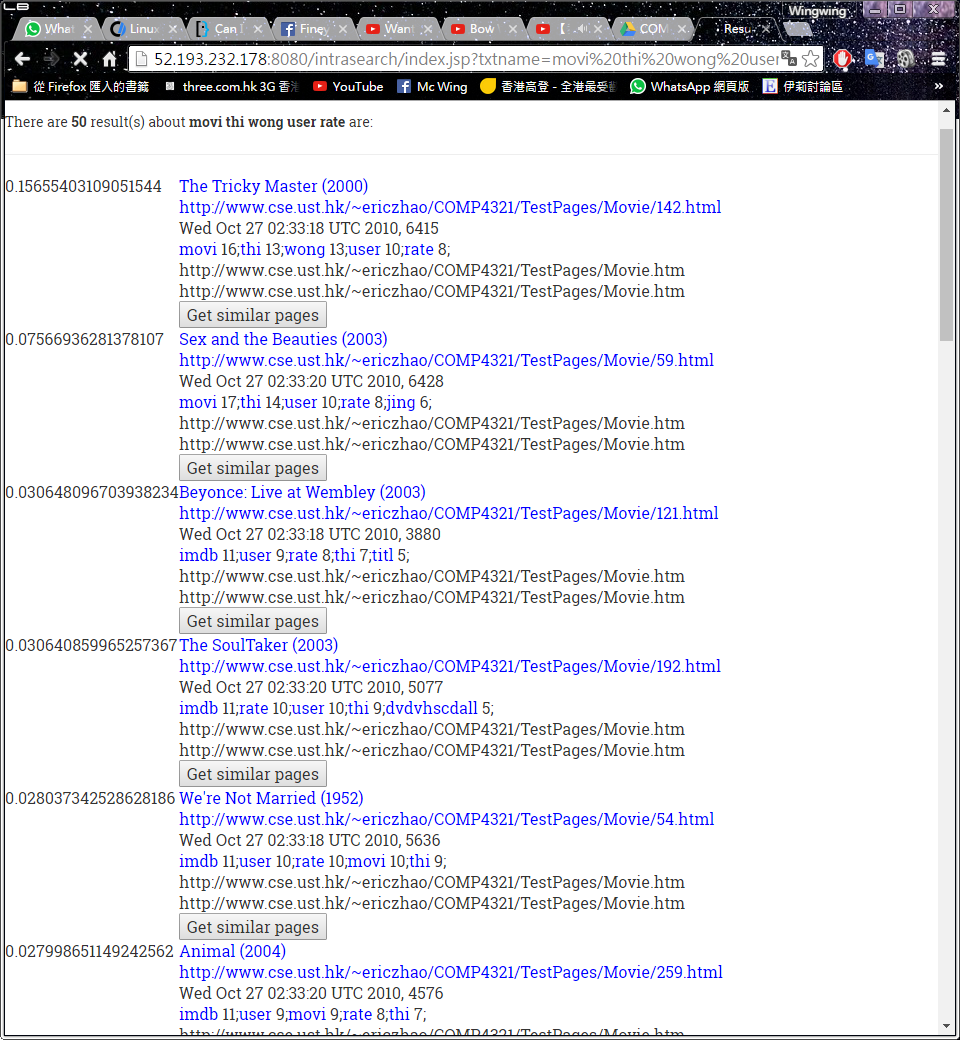
**6.1 Searching for the word “Hong Kong”**



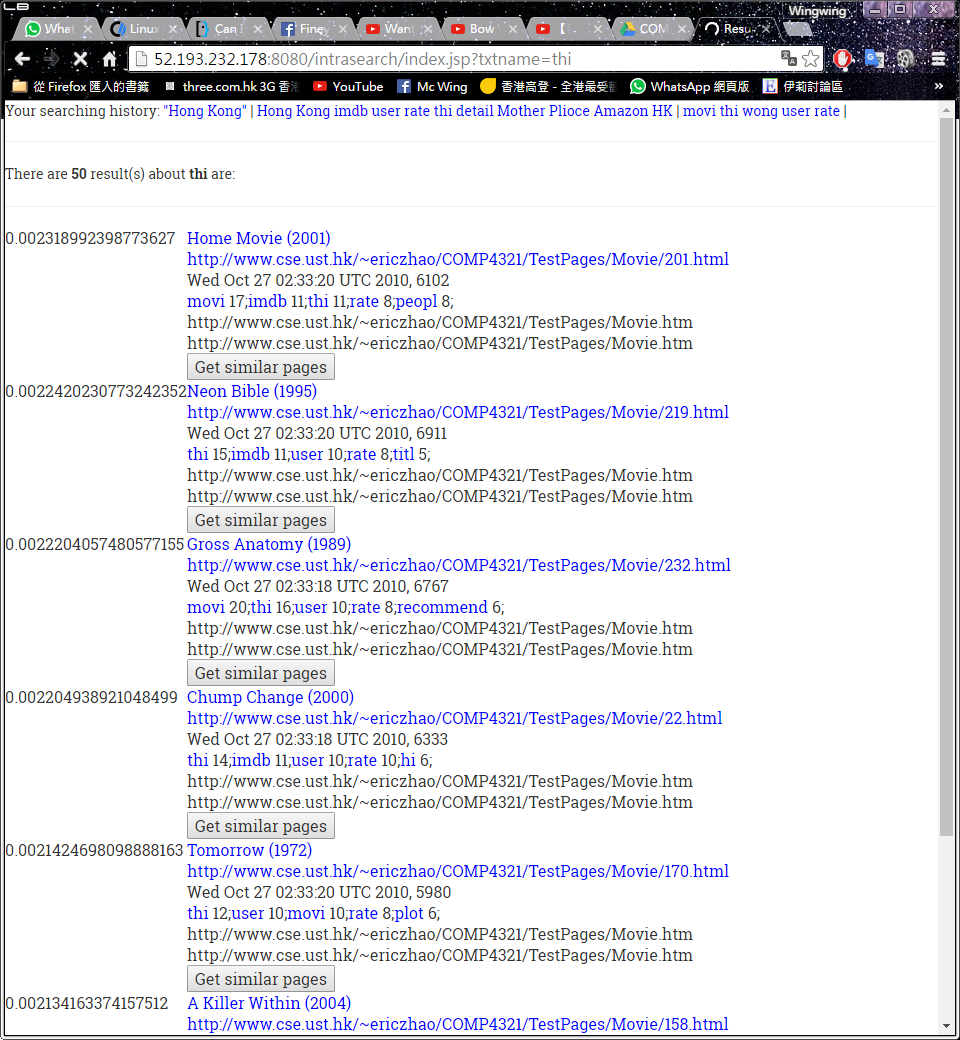
**6.2 Searching for the phrase “Hong Kong”**



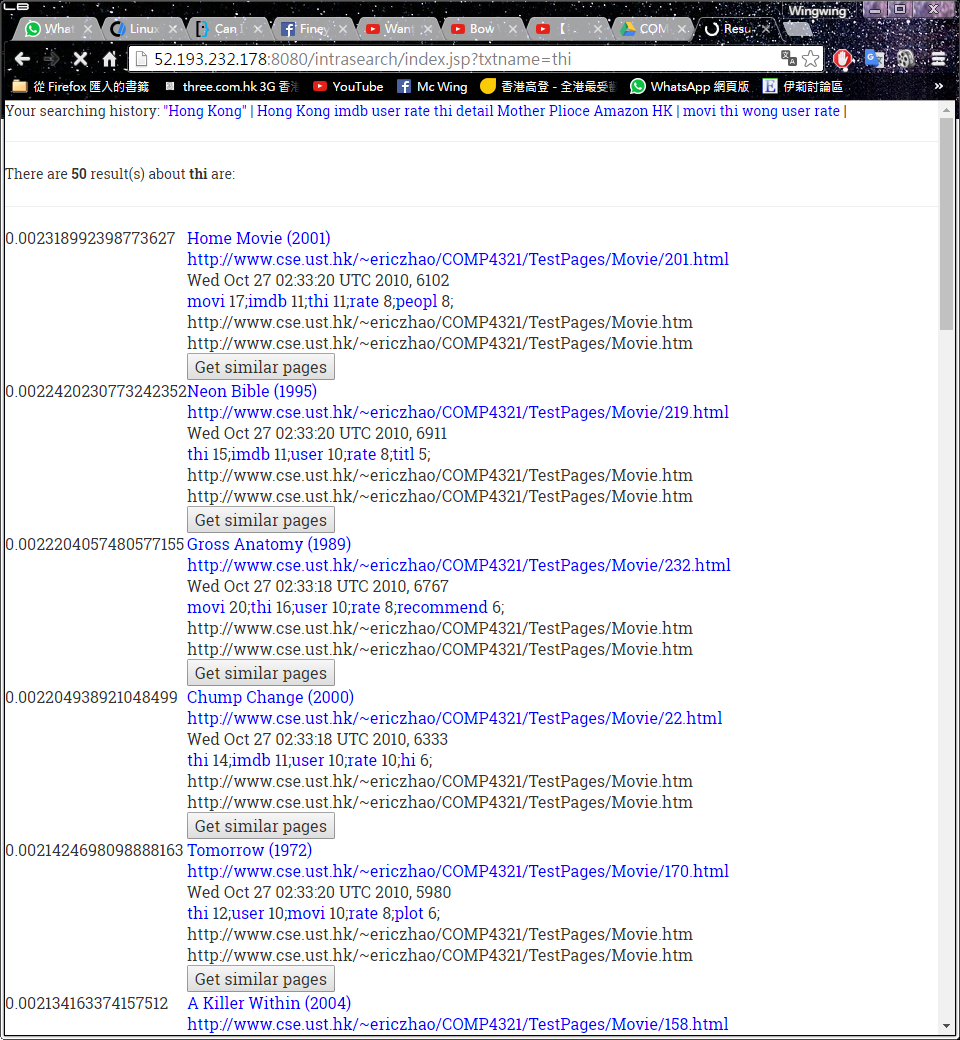
**6.3 Get similar pages**



**6.4 See list of stem word and re-submit them as a vector-space query**



**6.5 Keep track of query history using cookies**



# 7 Conclusion

**7.1 Strengths of our search engine**

Our search engine provides an attractive layout which may attract user to try it out. Also, our system combine Cosine similarity and PankRank when consider ranking the result. Furthermore, our search engine support phrase search like “Hong Kong”, “Harry Potter” etc.

**7.2 Weaknesses of our search engine**

The indexing speed of our search engine is slow. The stem and stop word removal algorithm is not robust. Also, our search engine doesn’t support operator like “+”, “\*” etc.

**7.3 Things would have done differently if I could re-implement the whole system**

We would consider to use another algorithm to calculate the similarity between query and indexes instead of using cosine similarity. Also, we would try to use another stemming method as Porter’s algorithm (1980) is a little bit too old and we believe there is a better stemming method on internet.

**7.4 Things would have added if I could re-implement the whole system**

There are a lots of things we would like to add:

1. Support different operator in searching including “+”, “\*” etc

2. Highlighting the keyword find in result page

3. Spell checking for input query

4. Keep track of query history

5. Add advertisement in our result page

# Work contribution and tasks performed by each project member:

|  |  |  |  |
| --- | --- | --- | --- |
| Work contribution % | Mok Chi Wing | CHOY Ting Wai | CHU Kwok Ning |
| Overall | 33.3% | 33.3% | 33.3% |

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Mok Chi Wing | CHOY Ting Wai | CHU Kwok Ning |
| Database design | ✔ | ✔ | ✔ |
| Spider |  | ✔ |  |
| Parent and child link table |  |  | ✔ |
| Word and Page table | ✔ |  |  |
| Test program | ✔ | ✔ | ✔ |
| Debug | ✔ | ✔ | ✔ |
| PageRank |  |  | ✔ |
| JSP | ✔ |  |  |
| Search engine interface | ✔ |  |  |
| Server setup | ✔ |  |  |
| Cosine Similarity |  | ✔ |  |
| Ranking result |  | ✔ |  |
| Deletion in index |  |  | ✔ |