**COMP4321 - Search Engine for Web data**

**Final Report**

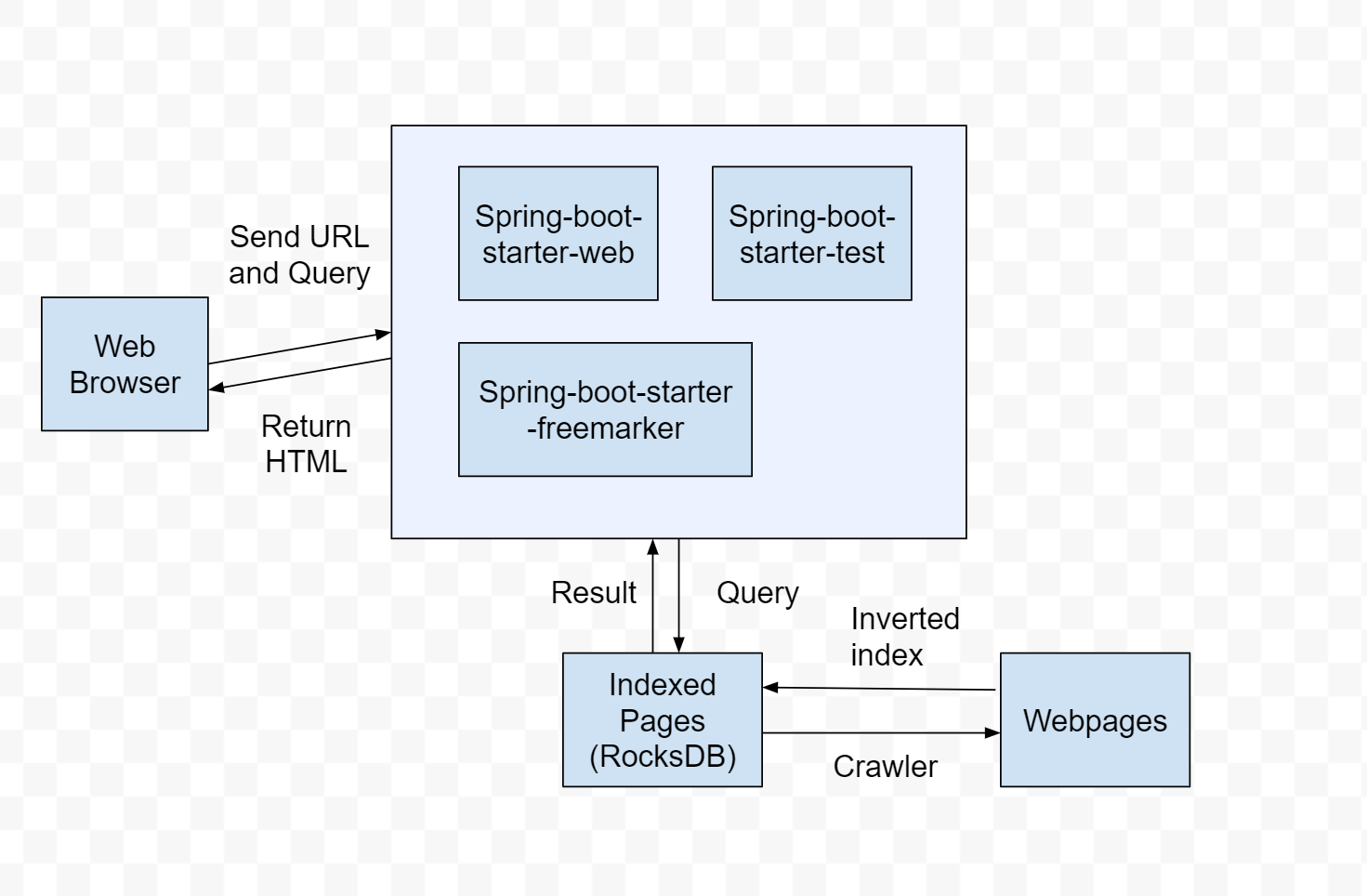
Spring 2019 - Group 25

AU, Ho Leong ([hlauac@connect.ust.hk](mailto:hlauac@connect.ust.hk)) 20260060

KIM, Hyun Gyu ([hgkim@connect.ust.hk](mailto:hgkim@connect.ust.hk)) 20375138

CHEUNG, Ka Ho ([khcheungap@connect.ust.hk](mailto:khcheungap@connect.ust.hk)) 20465294

## Overall design



### CRWALER and indexer

The crawler fetches the pages in [www.cse.ust.hk](http://www.cse.ust.hk) recursively. 11 RocksDBs are created when the program is running. Stop-word Removal and Stemming are processing towards the words in a webpage.

### search engine

After getting the indexed pages from the database and the queries from web interface, the page scores of queries are calculated according to tfidf and cosine similarity. The sorting results will be sent to the web interface.

### Web Interface

There are two webpages written in html which serves as a user interface, which is ‘index’ and ‘result’. Index page allows user to input their queries. The result page shows the search result to the user.

## File structure

The source code of the whole environment are;

|  |  |  |  |
| --- | --- | --- | --- |
| src/main/ | kotlin/ | main/ | SpiderMain.kt  TfidfMain.kt |
|  |  | Spring/ | Application.kt  Web.kt  WebController.kt |
|  |  | util/ | CSVParser.kt  HTMLParser.kt  Porter.kt  Ranker.kt  RocksDB.kt |
|  | resources/ | templates/ | index.html  result.html |
|  |  | application.yml |  |
|  |  | favicon.ico |  |
|  |  | stopwords.txt |  |

The database includes 11 different indexes in different RocksDB.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| DB | Key | datatype | Content | datatype |
| URL\_DB | Web url | String | Web id | Int |
| URL\_INFO\_DB | Web id | Int | Triple(Title, Date-Modified,  Size) | (String, Long, Int) |
| URL\_CHILD\_DB | Web id | Int | Child Web id | List(Int) |
| URL\_PARENT\_DB | Web id | Int | Parent Web id | List(Int) |
| WORD\_DB | Word | String | Word id | Int |
| SPIDER\_DB | Word id | Int | List(Web id, Word Location) | List(Int, Int) |
| URL\_WORDS\_DB | Web id | Int | List(WordID) | List(Int) |
| PAGE\_RANK\_DB | Web id | Int | PageRank | Long |
| URL\_WORD\_COUNT\_DB | Web id | Int | List(wordID, count) | List(Int, Int) |
| TF\_IDF\_DB | Web id | Int | TF-IDF | Long |
| URL\_LENGTH\_DB | Web id | Int | Length | Long |

## Algorithms used

### Links

To get links as much as possible, the algorithm is recursively crawling the child links. However, the increase of time required is exponential after more than 2,000 links are crawled. We decide to stop the crawler after finishing 2,000 crawling.

### Keywords

To perform stemming and stop-word removal, the Porter’s algorithm is used.

### Queries

After user submits their queries, the queries are being processed by stop-word removal and stemming. The queries are transformed to the search engine and it applies tfidf and cosine similarity with pages.

### Web application

We use Spring Boot and WebJars. Sprint Boot provides a good platform for Java/Kotlin developers to develop a stand-alone and production-grade spring application that you can just run. Many of the steps found on the [Spring Guides](https://spring.io/guides) for creating a RESTful service can be followed verbatim for Kotlin.

WebJars are client-side web libraries packaged into JAR files. It can explicitly and easily manage the client-side dependencies in JVM-based web applications

### Phrase search

In order to achieve phrase search, query terms are first be translated into word IDs and then parsed as a List<List<String>>. If the query does not contain any phrases, the resulting list will simply be a list of List<String> of all size 1. Phrases are denoted by enclosing quotation marks, e.g. “Hong Kong”, in the query. If a phrase is detected, it will be parsed as a List<String> containing the phrase, with each word as a String within the list.

For example, let’s say the word IDs of Hong, Kong and Computer are 1, 2 and 3 respectively. If the query is *Hong Kong Computer* without any phrases (i.e. no quotation marks), the query will be parsed as [[1], [2], [3]]. However, if the query is *“Hong Kong” Computer* with Hong Kong as a phrase, the query will instead be parsed as [[1, 2], [3]].

Once the query is parsed to List<List<String>>, we go through this list and look for phrases, i.e. the inner list has size > 1. Once a phrase is detected, we check whether this phrase appears in the document, if so, how many times does it appear in the document. This can be achieved since we can get the sequence of word IDs in a document from urlWordsDB given a document ID, and we can simply do a string compare to see whether the sequence of wordIds contains the sequence of query term IDs, using the .contains() function in the String class. To count how many times the phrase appears in the document, we used the .countMatches() function in StringUtils class in apache.

To get the tfidf score for the phrase, we simply sum up the two words’ individual tfidf scores and multiply it by the number of times the phrase appear in the document. In other words, we are essentially treating the phrase as a single entity and checking whether the entire entity appears in the document.

For example, let’s say a document is made up of the wordIds [1, 2, 3, 4, 5, 2, 3, 7, 2, 9] and the query is [[1], [2, 3]]. Our algorithm will parse the query terms one by one, check whether the wordID(s) appear in the query, and get its tfidf score. Since 1 appears in the doc, its tfidf will be added to the doc for calculation of cosine similarity later. Then for [2, 3], since it spears in the document twice, 2 x the sum of the tfidf scores of 2 and 3 will be added. Although 2 appears 3 times in the document, since we are using .contains() and .countMatches() functions, and that “2, 3” only appears twice in the wordIds, it will correctly calculate the score

## Installation procedure

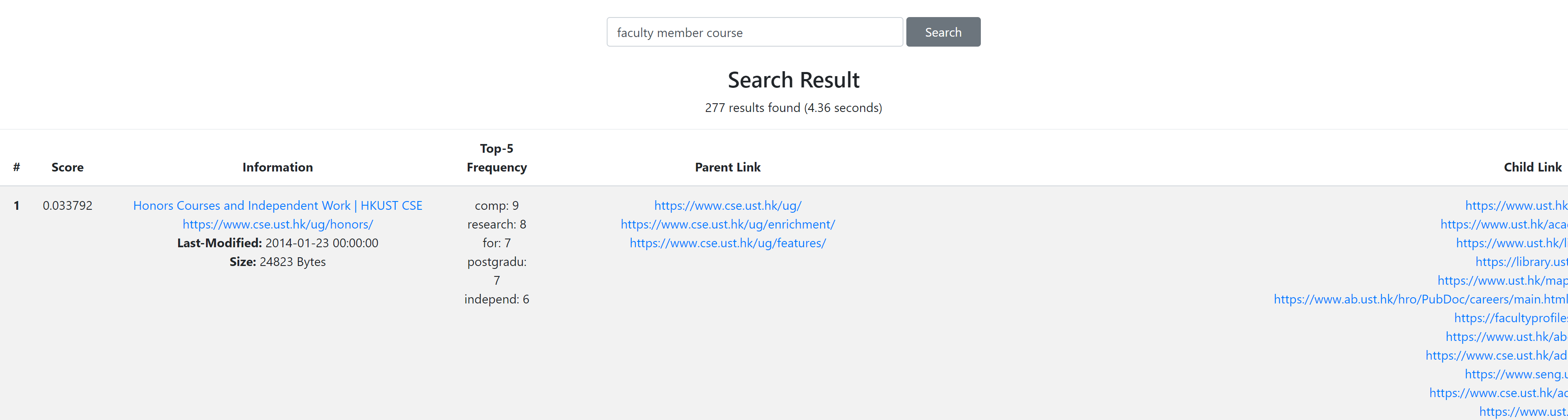
(it could be as simple as “Type make in the project directory”)

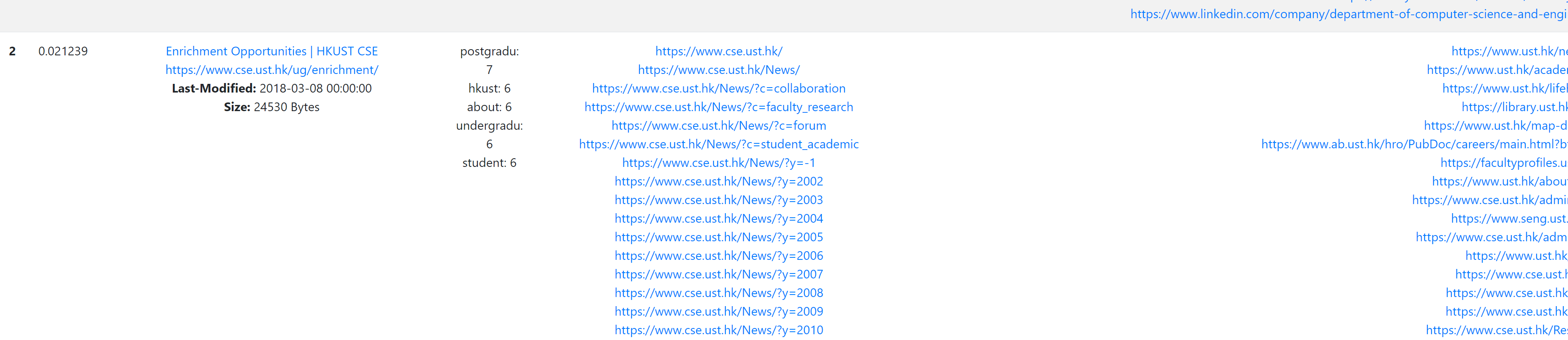
## Fatures beyond the required specification

## Testing

Testing 1:

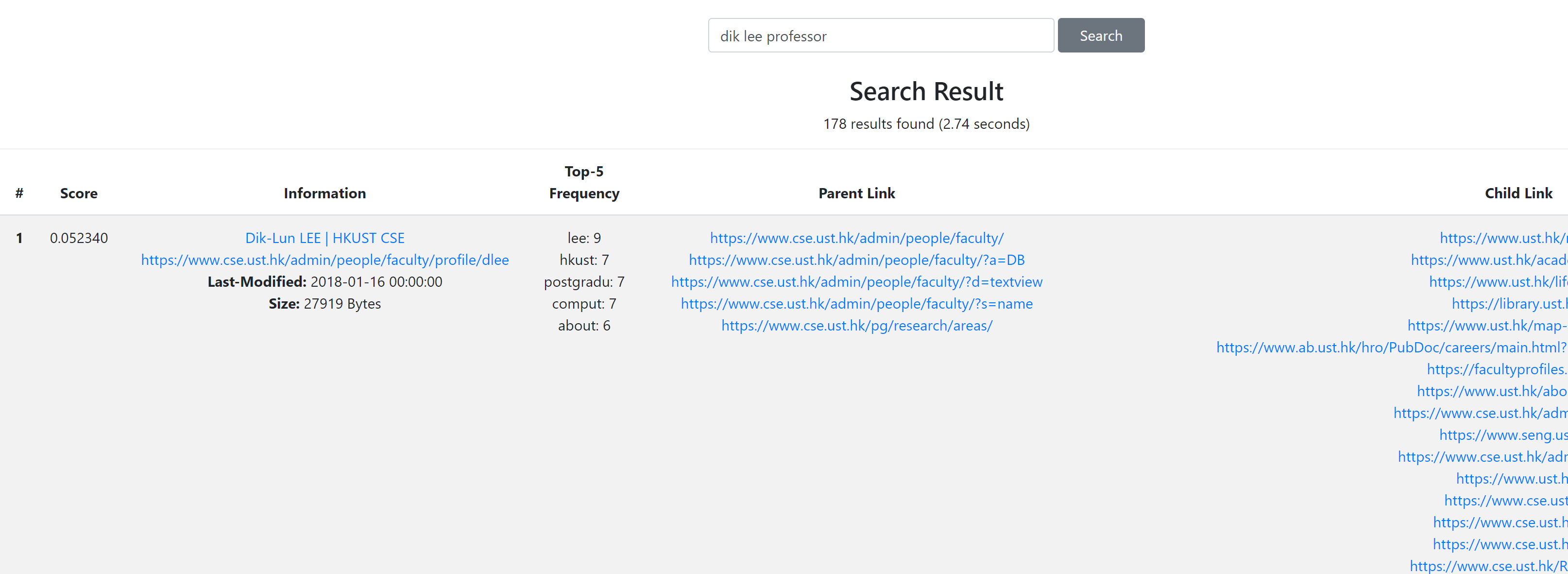
|  |  |  |
| --- | --- | --- |
| Query | | faculty member course |
| #1 | Score: 0.033792 | Honors Courses and Independent Work | HKUST CSE  <https://www.cse.ust.hk/ug/honors/>  Last-Modified: 2014-01-23 00:00:00  Size: 24823 Bytes |
| #2 | Score: 0.021239 | Enrichment Opportunities | HKUST CSE  <https://www.cse.ust.hk/ug/enrichment/>  Last-Modified: 2018-03-08 00:00:00  Size: 24530 Bytes |





Testing 2:

|  |  |  |
| --- | --- | --- |
| Query | | dik lee professor |
| #1 | Score: 0.052340 | Dik-Lun LEE | HKUST CSE  <https://www.cse.ust.hk/admin/people/faculty/profile/dlee>  Last-Modified: 2018-01-16 00:00:00  Size: 27919 Bytes |
| #2 | Score: 0.037488 | Faculty | HKUST CSE  <https://www.cse.ust.hk/admin/people/faculty/>  Last-Modified: 2019-03-27 00:00:00  Size: 190581 Bytes |





## Conclusion

### Strengths and weaknesses

At the time we crawled [www.cse.ust.hk](http://www.cse.ust.hk) , the memory gets full and swapping occurs. The increase of time required of crawling is exponential. We decided to crawl the first 2000 pages in cse department first.

Some links that we have crawled are restricted to faculty member, they are encrypted so we cannot access in those webpages. By default, those webpages’ title, last-modified date and size are ‘unauthorized’, ‘1990-01-01 00:00:00’ and 0 bytes respectively.

### Improvement

To improve the search engine, we can provide phrase search with specified in double quotes.

### Interesting features to add

For ranking function, tfidf and cosine similarity weighting are not enough. We hope we can use more advanced algorithm along with other personalizations.

There are many Chinese words in the webpages, we hope we can handle the Chinese word processing if we had more time.