COMP4321 – Project Report

Ivan Fung (20115291), Alan Shum Ka Yi (20110916)

Last Updated: 2016/5/4

I. Overall Design:

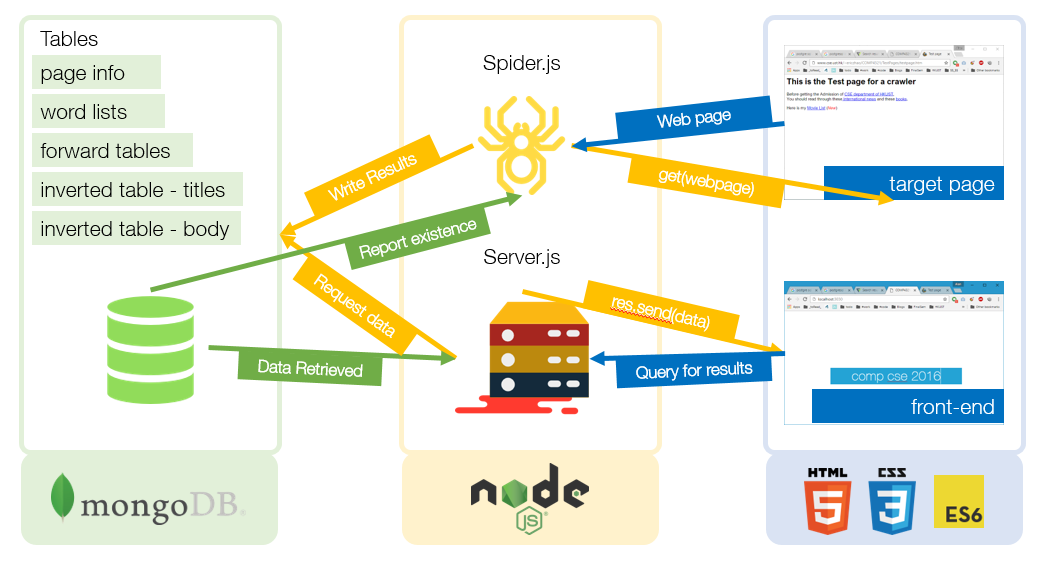
Different from most other teams, we are using "nodejs" (a javascript framework) as our backend language.

We also used a different database called "mongodb", which is document storage purpose when comparing to most of other types of databases like relational databases or grpah databases.

We use "mongoose" as the adapter from nodejs to mongodb to reduce the pain when doing transactions, and for better organizing of codes.

Most of the important configuration parameters like “maxRankPages”, “maxPages”, “titleWeight”, “rootURL” are separately maintained in the file called “config.js” to ease the work to change these important parameters.

The program can be run separately to trigger spider or work as a server.



# II. File Structure

We use five tables and the detailed and concerning schema and structures are described in the below:

|  |  |  |
| --- | --- | --- |
| **page info** |  |  |
| **word list** |  |  |
| **forward table** |  |  |
| **inverted table body** |  |  |
| **inverted tables title** |  |

# III. Algorithm

A. Favoring Title matches

We adopt a “favoring title matches” mechanism by first separately calculating the similarity ranking of documents with title and body respectively. Afterwards, we will multiply a “titleWeight” which is located in “config.js” as mentioned before, by the following formula:



Then, we will sort the finally similarity, slice the low ranking pages and return to the user.

Please note that the current titleWeight is default to be 0.75 to favor much more on title matches.

B. Phrase matching

Within our expectation, there are two methods to do the phrase matching algorithm. The first one is our method, which is to just process the ranking list of finalSimilarity by filtering out those documents without the exact phrase in it.

This method will produce results that MUST contain the phrase, which is more reasonable as we think that most likely when users use double quotes to do the phrase query. He/she most likely wants to see the exact phrase in the results, rather than seeing results WITHOUT the phrase he intended to find exactly.

The second method, is to see the phrase as a “special word” when calculating the similarity measure. This method may produce results that do not have the phrase appeared in the pages, which is undesirable to us with the argument above. So we do not adopt this method in our project.

# IV. Installation Procedure

As said in the overall design, we can separately run our program to do crawling and server. Please note that it is easy to build with the internet as nodejs has its own package management system to resolve all dependency automatically.

|  |  |
| --- | --- |
| **Repository** | 1. Download the source files 2. Install node.JS v6.0.0 (or 5.9.x) and npm (v3.7.x)    1. Node.JS (Current release, instead of LTS, to support ES6): <https://nodejs.org/en/download/current/>    2. Npm: Should be included in Node.JS installation 3. “cd <repository> && npm install” |
| **Database** | 1. Install MongoDB Server    1. [https://www.mongodb.org/downloads#production](https://www.mongodb.org/downloads" \l "production) 2. Start MongoDB Server    1. “mongod –dbpath <path to database folder>”: If mongod is not available yet after mongoDB installation, try restarting Windows Explorer (Windows user) or rebooting computer. 3. Create a database to use for this project    1. “mongo” (start mongo client), then “use <database name>”    2. In the project, I use “comp4321”. Then , “use comp4321” 4. Configure “<project root>/config.json”    1. Modify the link to database: {“mongoDB”: “mongodb://localhost:27017/<database\_name>”} |
| **Start Crawling** | “node spider” |
| **Start Server** | “node server” |

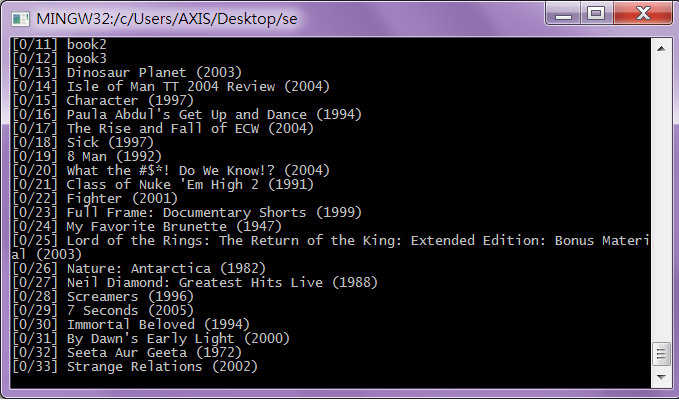
# V. Bonus Features

We have attempted a lot of bonus apart from the main program, as listed below:

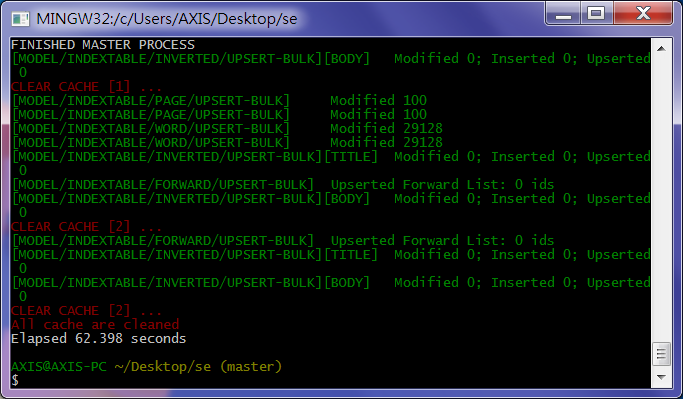
|  |  |
| --- | --- |
| **Front-end** | 1. AJAX implementation:    1. Pagination without reloading the page    2. Searching without redirecting   2. Auto pop-up indexed word list for user to choose |
| **Crawling** | 1. Bulk operation: Boosted crawling performance (for crawling 300 pages)    1. Can be tested against phase-1 version, without bulk operations: ~20minutes  * 1. Optimised Version: 2~3minutes  1. Fav-Icons are also crawled from website, for users to recognise websites more easily when searching |
| **Searching** | 1. Multi-cluster implementation: Avoid server from crashing.    1. Number of cluster = Number of CPU cores.    2. Each process cluster handle request separately, utilising the performance of the CPU    3. If one cluster fails, it will automatically restart |

# VI. Testing

Crawling:



Bulk insert:



Search:



# Conclusion

## Strengths & Weakness

|  |  |
| --- | --- |
| **Strength** | **wEAKNESS** |
| 1. Fast to execute 2. Built with latest modern web technologies for both the back-end and front-end, it is easier to maintain in the future, as well as extending its functionalities    1. Backend (Node.JS + MongoDB): Node.JS communities comprised a lot of open-source modules to use; MongoDB, being one of the most famous noSQL database solution, its performance is of no doubt, for simple query operations.    2. Front-end (HTML5 + ES6 + CSS3): Future-proof | 1. May be allow users to have phrase search to have results that do not have the phrase (method2) is better |

## Self-reflection

|  |  |
| --- | --- |
| **Ivan** | We have done a great job but as time is scarce, we have tried a lot of hard work in making the correctness the first priority. But nodejs sometimes is very difficult to debug, and we encountered many unexpected and strange bugs that we have to be stuck for a long long time.  The algorithms part are all designed and implemented by me and sometimes it is hard to make sure it runs correctly, especially using a script engine.  If we have time and a second chance , we may just find out more about debugging and testing in programming in nodejs so to relive our pain in the future. |
| **Alan** | Regarding the decisions on implementing the project using newer technologies (i.e. implement with Node.JS + MongoDB instead of Java), I am satisfied with the decision, with tears and joys involved.  For the tears, I cannot find any previous work of this project for reference, meaning our group have to implement everything from scratch, turning algorithms into codes. Besides, there is not reference on the performance, issues, and anything else on developing a search engine with Node.JS. This sometimes causes us troubles.  Regarding the crawling, the performance was acceptable in the first phase (several pages only), but unexpectedly slow when crawling 300 pages (7 minutes, originally). Then I found out that the original method used in phase 1 is not inadequate to handle that amount of database operations. Therefore, I have optimised the database operations, by storing several pages in memory and write them all (in the same cache) at a time, so as to reduce legacies in database connection and writing, as well as utilising the “bulk operation” natively supported by Mongodb. This may sound easy, but handling memory use is quite troublesome. For instance, after each writing, the cache is cleared, but the Node.JS programme is still using it. In that case, I have to turn from asynchronous procedures into synchronous version, for the database operation part.  Regarding the searching part, |

## Future Development

|  |  |
| --- | --- |
| **Front-end** | 1. Provide more information on query results (e.g. time of querying) |
| **Crawling** | 1. Implement word stemming with more sophisticated methods (e.g. Entropy method) if processing capacity allows 2. Speedup crawling speed with more simple database (e.g. use Cassandra or redis) instead of mongodb, which is too comprehensive, causing potential performance issue when crawling many pages. |
| **Searching** | 1. Match words by its linguistic meaning instead of stemmed words 2. Support other languages (e.g. Chinese, Japanese, …) |

## Distribution of work (50%: 50%)

|  |  |  |
| --- | --- | --- |
|  | **Ivan** | **Alan** |
| **Front-end** | * Provides API for easy usage | * User interface |
| **Crawling** | * Implemented inverted tables with title matches favouring + forward tables | * Optimised database manipulations with bulk operations * Implemented infrastructure for extending crawling function (e.g. pageinfos, wordlists) |
| **Searching** | * Implemented algorithms | * Implemented infrastructure for searching function * API (server routing) implementation |