

Query Processor

By

Vivek Desai (Poly Id: 0507166)
Pratik Patel (Poly Id: 0508504)

Web Search Engines (CS 6913), Spring 2014

Guided by

Prof. Torsten Suel

New York University

Index

Table of Contents	Page Numbers
1. Overview	3
2. Architectural Diagram	4
3. Query Processor	6
4. Design Adopted	7
4.1 Document at a Time(DAAT) v/s Term at a Time (TAAT)	7
4.2 Conjunctive vs Disjunctive	7
4.3 Score/Ranking	8
5. Implementation	8
5.1 Inverted Index Building Phase: Assignment 2	8
5.1.1 Building the Inverted Index	9
5.1.2 Compression	9
5.2 Query Processing	10
5.2.1 Page Finder Logic	11
5.2.2 BM25 Score Calculation	12
5.2.3 AND Semantics logic	12
5.3 Executing the Project on Command Line	13
5.3.1 Building the Inverted Index using Command Line	13
5.3.2 Executing Query Processor using Command Line	13
6. Major Functions Implemented	13
6.1 Add on features	13
7. Performance Parameters	14
8. Limitations	15
9. Output	16
10. References	20

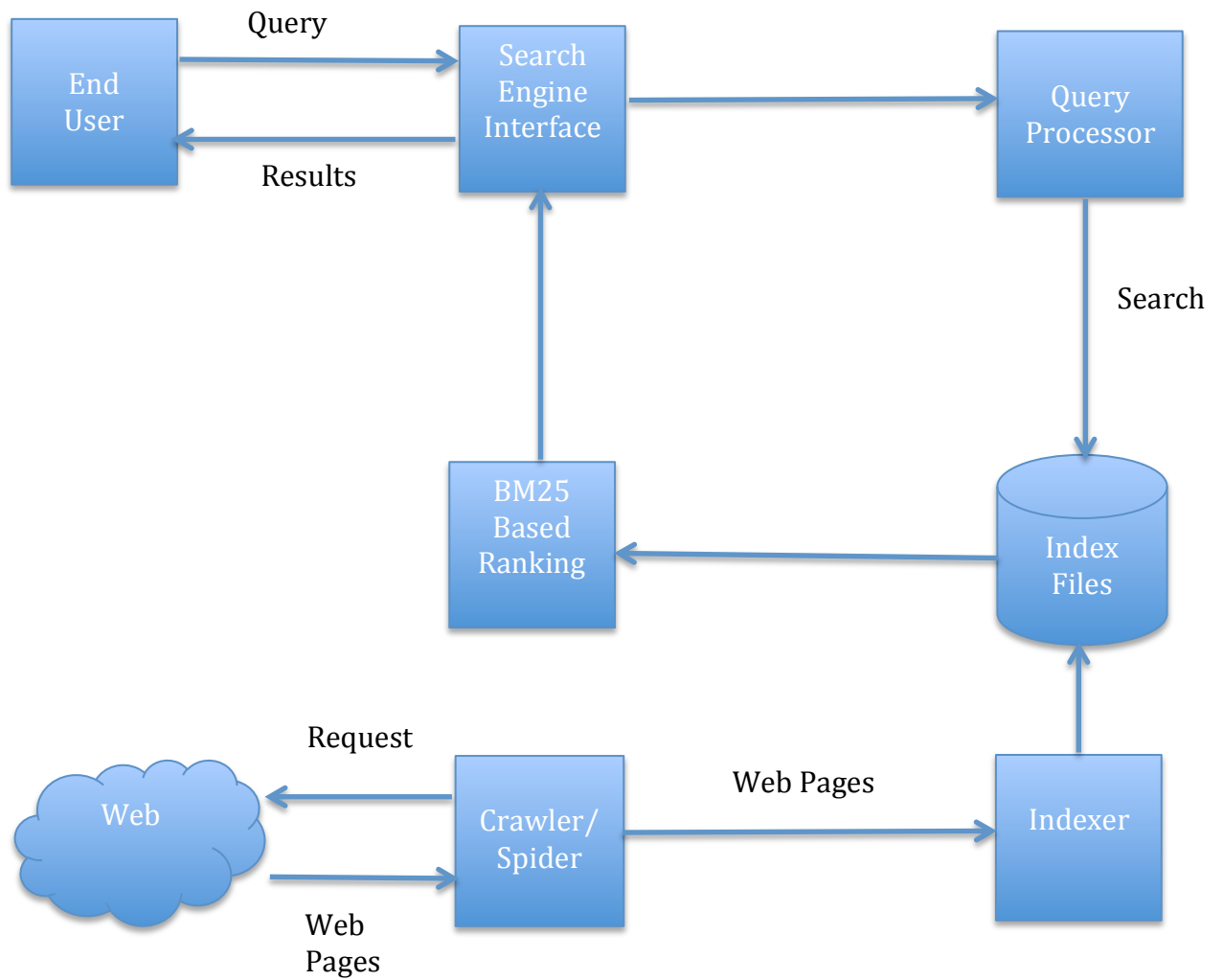
1. Overview

Main aim of this assignment is to implement the third phase of a Web Search Engine, which is Query Processor. Query Processor processes the query entered by the user and searches through the index via lexicon structure to find that particular query. A web search engine is a software system that is designed to search for information on the World Wide Web. The search results are generally presented in a line of results often referred to as search engine results pages. The information may be a specialist in web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike web directories, which are maintained only by human editors, search engines also maintain real-time information by running an algorithm on a web crawler[1].

We have developed our search engine in following three different phases:

- a. **Web Crawling:** Where we fetch all pages from internet and dump the content of it in a particular file
- b. **Indexing:** In this stage we parse each page and create the inverted index for all the pages.
- c. **Query Processor:** In this stage user enters the search query and query processor after processing the search query returns the result.

2. Architectural Diagram



Components Explanation:

Crawler: Crawler also known as spider, which a software code is used to fetch the pages from the web automatically and systematically, which are then used for the indexing purpose.

Indexer: All the web pages(html) that have been downloaded are parsed and tokenized to create following three structures:

- a. **Lexicon Structure:** It contains list of all possible distinct words in existing all html web pages. Also it contains the line number of the inverted index file on which that particular word occurs. Along with this it also stores the name of the index file in which that word occurs.
- b. **Inverted Index:** In this structure we are storing the doc id that is the page number in which a particular word occurs and the count of that word.
- c. **URL Table:** In this structure we are storing the doc id and its corresponding URL. Along with this we are also storing the length of pages in bytes.

Query Processor: Query Processor takes the search query from web search engine interface. It then searches for the query into the inverted index via lexicon structure. If results are found it returns the top 10 results based on BM25 ranking technique

Indexer: In this stage we parse the document and tokenize each and every downloaded page with assigned doc id and get the count of the tokenized word in that particular page. Also during this stage we build Lexicon Structure to provide mapping with inverted index list.

Index Files: Index files saves the doc id that consists a particular call and the count of number of times that word occurs.

BM25 Ranking: BM25 Ranking module is used to compute the ranking of that particular page. It returns the top 10 urls based on the BM25 score for each page. A page with high score has high priority.

Search Engine Interface: User uses this search engine interface to enter the search query.

3. Query Processor

Query processor performs the following operations in order to return accurate results to the end user:

- a. Tokenizing: As soon as user enters the search query from search engine interface we tokenize the search query to break down it into words.
- b. Remove Stop words: After tokenizing the words we remove all the stop words from it.
- c. Search Lexicon and Index: After tokenizing for each word in query we search for that word in lexicon and via mapping we search it index file. After getting results for each word we perform conjunction operation on the results of each word so that we get results that consist of search query.
- d. BM25: After performing conjunction operation on the results we compute BM25 score for each page and return top 20 results with highest BM25 score.

4. Design Adopted

The Following designs strategies were adopted for Query Processing.

4.1 Document at a Time(DAAT) v/s Term at a Time (TAAT)

The DAAT algorithms simultaneously traverse the postings lists for all terms in the query. The naive implementation of DAAT simply merges the involved postings lists (which are already sorted by doc id in the index) and examines all the documents in the union. A min-heap is normally used to store the top-k documents during evaluation. Whenever a new candidate document is identified it must be scored. The computed score is then compared to the minimum score in the heap, and if it is higher, the candidate document is added to the heap. At the end of processing the top-k documents are guaranteed to be in the heap. DAAT is document at a time query processing. Find the first document that has all the terms in the query and calculate its score and then the next document and so on[2][3].

TAAT is term at a time query processing. Process the inverted list of the first term (or the term with shortest inverted list) in the query and move to second and so on. Create hash buckets for each term in the shortest list with doc id as key and cosine measure as value and when move to the next inverted list we add values to these buckets.

We have used TAAT for query processing.

4.2 Conjunctive vs Disjunctive

Disjunctive means that in order for a document to be ranked it need not contain all the terms in the query. Even a document having only one term of the entire query will be ranked.

Conjunctive means only those documents, which contain all the terms in the query will be ranked.

We have used conjunctive query processing because implementing it much simpler and returns most of the times relevant and accurate results.

4.3 Score/Ranking

The ranking function that we used in our project is **BM25**, whose details are discussed in implementation section.

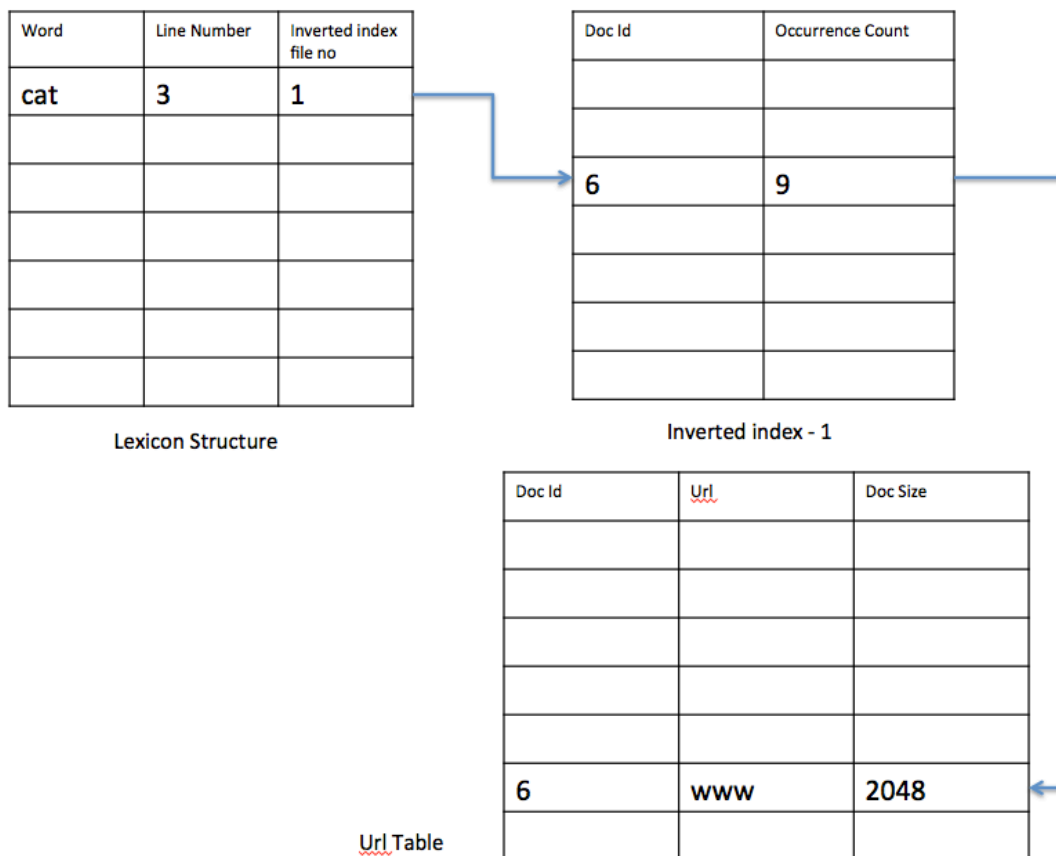
In information retrieval, BM25 is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document[4].

5. Implementation

Query Processor was implemented in Python. We built it in two phases:

5.1 Inverted Index Building Phase: Assignment 2

Following is the architecture that was adopted to build the inverted index from “NZ” data set.



In above diagram we can see the 3 structures that we have used for building inverted index:

- a. **Lexicon Structure:** Lexicon structure consists of Word, Line Number and Inverted Index File Number. Word is being used as word id to identify it self as a unique word. Line number save the line number on which that particular word is saved in the index file. Inverted Index File Number saves the number of inverted index in which that word occurs.
- b. **Inverted Index:** Inverted index consist of doc id and the frequency that is the number of times that particular word occur in its corresponding HTML page.
- c. **Url Table:** Url table consists of doc id, which is assigned to each html page while processing it during indexing stage. It also consists of URL and also the length in bytes of that html page.

To search “cat” we first look into lexicon structure from there we get the line number and the inverted index number. Then we look into inverted index, we jump to the line number, which was found in lexicon structure. From there we get the doc id that is the html page in which that word is stored, Also we get the occurrence that is the number of times that word occurs. Then we look into Url table to get the Url of the html page and to get the length that html page to later computer the BM25 score.

5.1.1 Building the Inverted Index:

Before building inverted index we read the names of all the data and index files which are compressed, into list structure in python. Then one by one we open the file and process each of them in sorted manner.

For each HTML page in data file we parse it using NLTK library to tokenize the page into words and to get position of each word. Before tokenizing we check for its status code “200”, if and only if the status code is “200” we parse the document. After tokenizing the words we dump all the words into intermediate files and then from all intermediate files we create Inverted index using M-way merge sort. Before doing M-way merge sort we perform unix sort on each and every intermediate files. Upon creation of the inverted index we create lexicon structure from the inverted index files, which provides mapping between the words and the page in which that word occurs.

5.1.2 Compression

Variable byte (VB) encoding uses an integral number of bytes to encode a gap. The last 7 bits of a byte are “payload” and encode part of the gap. The first bit of the byte is a continuation bit . It is set to 1 for the last byte of the encoded gap and to 0 otherwise. To decode a variable byte code, we read a sequence of bytes with

continuation bit 0 terminated by a byte with continuation bit 1. We have used following encoding and decoding algorithm for Variable byte compression [5]:

Encoding algorithm:

```
VBENCODE(numbers)
1  bytestream  $\leftarrow \langle \rangle$ 
2  for each n  $\in$  numbers
3  do bytes  $\leftarrow$  VBENCODENUMBER(n)
4    bytestream  $\leftarrow$  EXTEND(bytestream, bytes)
5  return bytestream
```

Decoding Algorithm:

```
VBDECODE(bytestream)
1  numbers  $\leftarrow \langle \rangle$ 
2  n  $\leftarrow 0$ 
3  for i  $\leftarrow 1$  to LENGTH(bytestream)
4  do if bytestream[i] < 128
5    then n  $\leftarrow 128 \times n + \text{bytestream}[i]$ 
6    else n  $\leftarrow 128 \times n + (\text{bytestream}[i] - 128)$ 
7    APPEND(numbers, n)
8    n  $\leftarrow 0$ 
9  return numbers
```

Changes that were introduced to Assignment 2 while doing Assignment 3

1. Divided index file into multiple indexes.
2. Implemented Variable Byte Compression.
3. Removed Delta Compression.
4. We are now writing files in binary format.
5. Storing only frequency of the word and not position.
6. Reduced Inverted index creation from 11 minutes to 4 minutes on NZ2 data set.

5.2 Query Processing Phase – Assignment 3

5.2.1 Page Finder Logic:

We perform following steps to find results for a particular search query:

- a. Load Lexicon file and URL table into Main memory.
- b. Get the search query from user.
- c. Remove stop words from search query.
- d. Start searching for the keyword in Cache.
- e. If found in cache then increment number of times (LRU counter) used by 1 and do not look into index file.
- f. If not found in cache then get the line number and index file number of the word from lexicon which is available in Main memory.
- g. If word not found in lexicon then none of the pages contain that keyword.
- h. If word found in lexicon file then open the particular index file and do sequential scan in the index file until we reach to particular line number.
- i. Get the document id and frequency of the word from index file.
- j. Cache word, document id, frequency and number of times used (Initially 1) in our cache dictionary.
- k. Before caching current word, check if the length of cache dictionary has reached to 10. If so then remove least frequent used word from cache dictionary and cache the current word.
- l. Decode document id and frequency, which were encoded using variable byte.
- m. Repeat step d to l for all the words in keyword.
- n. Start finding common document id for all the words in keyword.
- o. If no common document id found then none of the pages contain whole keyword (Conjunctive).
- p. Calculate BM25 score for the entire common document id set.
- q. Get top 20 results (document id) from the result and get their URL from URL table, which is already available in Main memory.
- r. Display top 20 results (URL & BM25 Score) to the user.
- s. Go to step no. b

5.2.2 BM25 Score Calculation

The BM25 score is calculated using the following formula[4]:

$$BM25(q, d) = \sum_{t \in q} \log\left(\frac{N - f_t + 0.5}{f_t + 0.5}\right) \times \frac{(k_1 + 1)f_{d,t}}{K + f_{d,t}}$$

- N : total number of documents in the collection;
 - f_t : number of documents that contain term t ;
 - $f_{d,t}$: frequency of term t in document d ;
 - $|d|$: length of document d ;
 - $|d|_{avg}$: the average length of documents in the collection;
 - k_1 and b : constants, usually $k_1 = 1.2$ and $b = 0.75$
- $$K = k_1 \times ((1 - b) + b \times \frac{|d|}{|d|_{avg}})$$

We use above formula to calculate the BM25 score for the final shortlisted results. According to BM25 score we display the results with page having Higher BM25 score on top.

5.2.3 AND Semantics Logic

For getting all the relevant pages that consists all the search query terms, we implemented following AND semantic logic in python under “getResult.py” file[6].

```
for (i = 0; i < num; i++) lp[i] = openList(q[i]);

did = 0;
while (did <= maxdocID)
{
    /* get next post from shortest list */
    did = nextGEQ(lp[0], did);

    /* see if you find entries with same docID in other lists */
    for (i=1; i<num) && ((d=nextGEQ(lp[i], did)) == did); i++);

    if (d > did) did = d;          /* not in intersection */
    else
    {
        /* docID is in intersection; now get all frequencies */
        for (i=0; i<num; i++) f[i] = getFreq(lp[i], did);

        /* compute BM25 score from frequencies and other data */
        <details omitted>
        did++;          /* and increase did to search for next post */
    }
}

for (i = 0; i < num; i++) closeList(lp[i]);
```

5.3 Executing the Project on Command Line

To execute the project one can simply run main file of both project respectively. In following section can seen on how to execute each of the project:

5.3.1 Building the Inverted Index using Command Line

Following are the instructions to run the query processor:

```
python Main.py
```

NOTE: Paths are needed to be configured for reading all Gzip files.

5.3.2 Executing Query Processor using Command Line

Following are the instructions to create the Inverted Index:

```
python getResults.py
```

NOTE: Paths are needed to be configured for the index files, lexicon file and url table.

6. Major Functions Implemented

Following are the major functions that we have implemented for this project:

- a. Performed Document at a time processing of query
- b. Implemented BM25 scoring function
- c. Loaded info from secondary memory into main memory
- d. Loaded Lexicon from secondary memory into main memory
- e. Loaded Url table from secondary memory into main memory
- f. Implemented Var Byte compression
- g. Inserted page into Heap Queue on basis of BM25 score
- h. Poped pages from Heap Queue
- i. K-way IO Efficient merge sort

6.1 Add on Features:

- a. Implemented caching of words, encoded doc id and frequency from inverted index. Least frequently used algorithm is used for caching.

7. Performance Parameters.

Following are the parameters that are calculated for this project which is performed on **NZ10** Data Set :

a. How long it takes it takes to search on the provided data set?

It depends on the search query. For some query it takes less than 0.08 second, for some it takes 0.10-0.70 seconds. Usually the search result is obtained with in a second. We can observe the screenshots in the output section

b. How large the resulting index files are?

Index size varies from 180-200 mb, it depends on the data being that is saved in the inverted index

c. How long the lexicon and Url Table Structure are?

Lexicon Structure: 3.9mb

Url Table: 17.1mb

8. Limitations

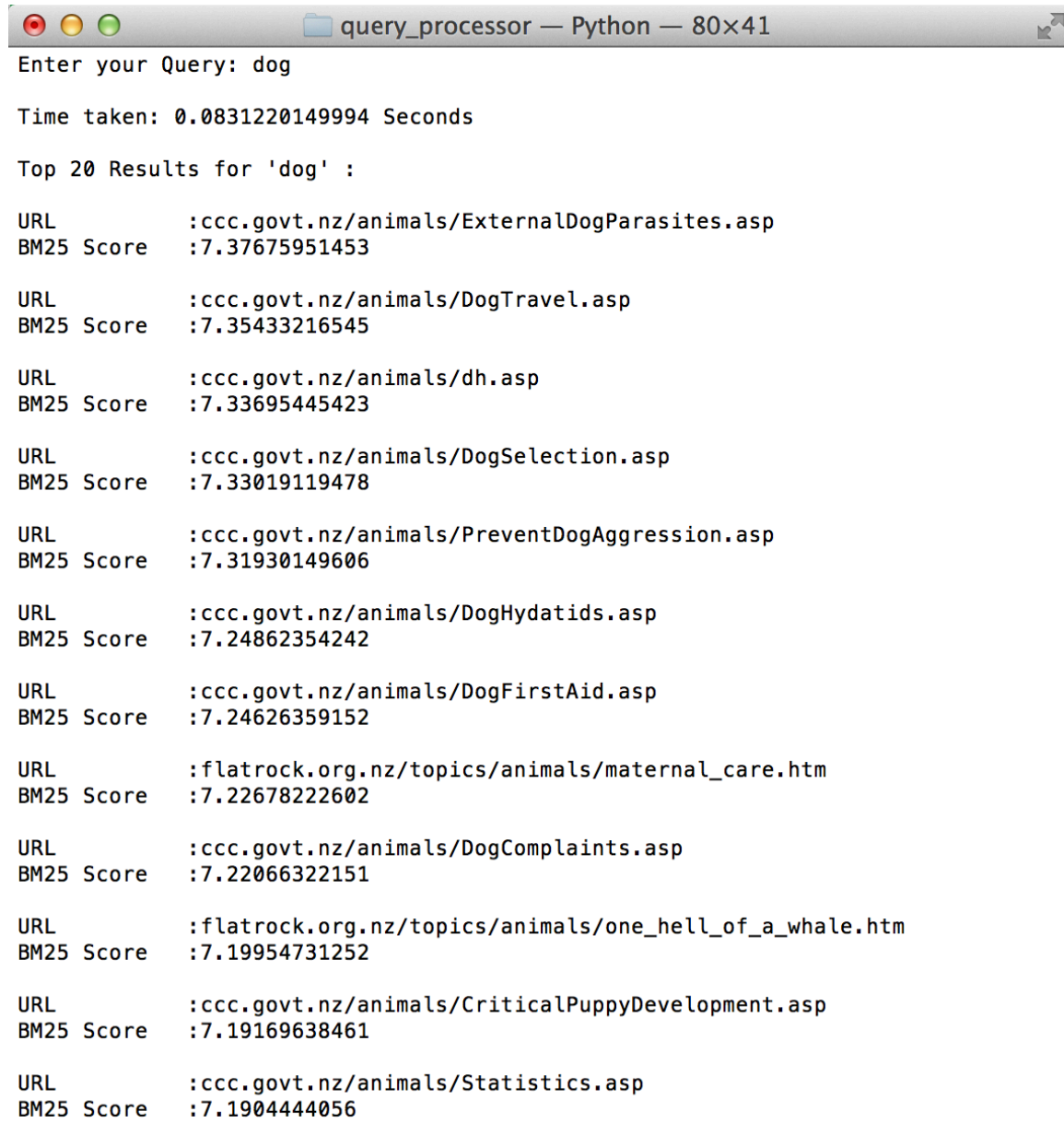
If the query will be disjunctive it won't show any results. It would simply return results not found. We would like to integrate this as well in future. Also we wanted to encode and compress our lexicon structure and url table too.

We are storing words in lexicon as well as index files, we would want to remove this redundant data from index files, which would reduce the file size of index files. We are also not saving the context and the position for each word, we would want to save that information also, which is important for any search engine. The parser that we used to parse this document is NLTK, which is very slow hence increasing the time for inverted index creation.

If the search query consists of only stop words it will display "No matched page found". We would like to generate results for stop words too in future.

9. Output.

Following are some screen shots with different Search Terms.



```
query_processor — Python — 80x41
Enter your Query: dog

Time taken: 0.0831220149994 Seconds

Top 20 Results for 'dog' :

URL          :ccc.govt.nz/animals/ExternalDogParasites.asp
BM25 Score   :7.37675951453

URL          :ccc.govt.nz/animals/DogTravel.asp
BM25 Score   :7.35433216545

URL          :ccc.govt.nz/animals/dh.asp
BM25 Score   :7.33695445423

URL          :ccc.govt.nz/animals/DogSelection.asp
BM25 Score   :7.33019119478

URL          :ccc.govt.nz/animals/PreventDogAggression.asp
BM25 Score   :7.31930149606

URL          :ccc.govt.nz/animals/DogHydatids.asp
BM25 Score   :7.24862354242

URL          :ccc.govt.nz/animals/DogFirstAid.asp
BM25 Score   :7.24626359152

URL          :flatrock.org.nz/topics/animals/maternal_care.htm
BM25 Score   :7.22678222602

URL          :ccc.govt.nz/animals/DogComplaints.asp
BM25 Score   :7.22066322151

URL          :flatrock.org.nz/topics/animals/one_hell_of_a_whale.htm
BM25 Score   :7.19954731252

URL          :ccc.govt.nz/animals/CriticalPuppyDevelopment.asp
BM25 Score   :7.19169638461

URL          :ccc.govt.nz/animals/Statistics.asp
BM25 Score   :7.1904444056
```



```
query_processor — Python — 80x42
Enter your Query: cat dog
Time taken: 0.0754010677338 Seconds
Top 20 Results for 'cat dog' :

URL          :flatrock.org.nz/topics/animals/maternal_care.htm
BM25 Score   :13.9492587132

URL          :flatrock.org.nz/topics/animals/one_hell_of_a_whale.htm
BM25 Score   :13.9023338742

URL          :curliwinks.co.nz/Pages/Abt%20CW%20Photo%20Belle.htm
BM25 Score   :13.8932603634

URL          :ccc.govt.nz/animals/Statistics.asp
BM25 Score   :13.8766925703

URL          :flatrock.org.nz/topics/animals/that_wasnt_chicken.htm
BM25 Score   :13.8159961509

URL          :flatrock.org.nz/topics/animals/love_that_dare_not_squeak.htm
BM25 Score   :13.8075777188

URL          :blondini.orcon.net.nz/nes/nes68.html
BM25 Score   :13.4891411819

URL          :ash.co.nz/sites/touchwood/therbalharvest.html
BM25 Score   :13.3950373788

URL          :amazingtours.co.nz/
BM25 Score   :13.3694390419

URL          :desinz.co.nz/links3-1.html
BM25 Score   :13.3432949016

URL          :doc.govt.nz/Conservation/International/Convention-on-International
-Trade-in-Endangered-Species/002%7ECITES-in-New-Zealand/index.asp
BM25 Score   :13.3116002312

URL          :flatrock.org.nz/topics/animals/losung.htm
BM25 Score   :13.1857878337
```

```
query_processor — Python — 80x43
Enter your Query: cat dog animal

Time taken: 0.101301908493 Seconds

Top 20 Results for 'cat dog animal' :

URL          :anzccart.rsnz.govt.nz/text/Glossary/10.html
BM25 Score   :18.8498773794

URL          :flatrock.org.nz/topics/animals/one_hell_of_a_whale.htm
BM25 Score   :18.6094867519

URL          :flatrock.org.nz/topics/animals/love_that_dare_not_squeak.htm
BM25 Score   :18.5772563076

URL          :flatrock.org.nz/topics/animals/that_wasnt_chicken.htm
BM25 Score   :18.5687930157

URL          :flatrock.org.nz/topics/animals/maternal_care.htm
BM25 Score   :18.5514247011

URL          :ccc.govt.nz/animals/Statistics.asp
BM25 Score   :18.4397591224

URL          :ash.co.nz/sites/touchwood/tanimalhealth.html
BM25 Score   :18.4175251725

URL          :flatrock.org.nz/topics/animals/art_nouveau_snakes.htm
BM25 Score   :18.214444251

URL          :flatrock.org.nz/topics/animals/losung.htm
BM25 Score   :18.0565786338

URL          :doc.govt.nz/Regional-Info/006~East-Coast-Hawkes-Bay/005~Publicatio
ns/Annual-Report/Organisational-capability.asp
BM25 Score   :17.9090918722

URL          :desinz.co.nz/links3-1.html
BM25 Score   :17.8468849681

URL          :doc.govt.nz/Conservation/002~Animal-Pests/Policy-Statement-on-Deer
-Control/002~Introduction.asp
BM25 Score   :17.7930460008
```

```
query_processor — Python — 80x42

Enter your Query: cat dog animal food

Time taken: 0.209339857101 Seconds

Top 20 Results for 'cat dog animal food' :

URL      :flatrock.org.nz/topics/animals/losung.htm
BM25 Score :19.4439961354

URL      :flatrock.org.nz/topics/animals/one_hell_of_a_whale.htm
BM25 Score :19.4107228978

URL      :flatrock.org.nz/topics/animals/that_wasnt_chicken.htm
BM25 Score :19.3520213183

URL      :flatrock.org.nz/topics/animals/maternal_care.htm
BM25 Score :19.3512000234

URL      :ccc.govt.nz/animals/Statistics.asp
BM25 Score :19.3395278894

URL      :flatrock.org.nz/topics/animals/art_nouveau_snakes.htm
BM25 Score :19.0754846752

URL      :flatrock.org.nz/topics/animals/dalmatian_kittens.htm
BM25 Score :19.0321598589

URL      :desinz.co.nz/links3-1.html
BM25 Score :18.8020336463

URL      :doc.govt.nz/Regional-Info/006~East-Coast-Hawkes-Bay/005~Publicatio
ns/Annual-Report/Organisational-capability.asp
BM25 Score :18.734985534

URL      :doc.govt.nz/Conservation/002~Animal-Pests/Policy-Statement-on-Deer
-Control/002~Introduction.asp
BM25 Score :18.616031593

URL      :doc.govt.nz/Conservation/002~Animal-Pests/001~Control-Methods/010~
The-Use-of-1080-for-Pest-Control/060~Information-about-1080/002~The-life-cycle-o
f-1080.asp
BM25 Score :18.466121775
```

10. References.

1. http://en.wikipedia.org/wiki/Web_search_engine
2. <https://www.stanford.edu/class/msande239/taatDaatProject.pdf>
3. <http://fontoura.org/papers/vldb2011.pdf>
4. http://en.wikipedia.org/wiki/Okapi_BM25
5. <http://nlp.stanford.edu/IR-book/html/htmledition/variable-byte-codes-1.html>
6. <http://cse.poly.edu/cs6913/>