BI-VWM
Semestral Project
Documentation

Boolean Model

Galymzhan Dosmagambet
2020

Project description

The goal of the project is to implement and represent Boolean Model and Inverted Index search on different collections of documents. Representation includes comparison to Sequential (Linear) search over documents.

Output of the project is web application which uses Boolean model and Inverted index and Sequential search against 3 different sets of documents. Also application shows how much Boolean model is beneficial and prints out results of a query (list of documents).

Solution

Main approach is built around Boolean model and Inverted index.

Before querying it is required to preprocess data:

- From each document in database terms are extracted and lemmatized.
- To every lemma a document it appears in is added (Inverted index).
- All lemmas stored in sorted LemmaStorage.

General algorithm of querying over text file database is:

- Receiving query.
- Extracting tokens for parsing.
- Parsing tokenized query using expression/term/factor (ETF) grammar and providing Abstract Search Tree (ATS).
- Going through ATS. In case of Sequential search tokens(lemmas) in leafs of ATS will be searched linearly through the whole database. When in the Inverted Index search list of documents where the lemma appears will be given right away. Then internal nodes of ATS (operators AND, OR, NOT) will process results of leafs. Time used for each search is recorded and saved as well as the result.
- Returning respond with needed operational time and query result.

Implementation

The web application is written in Java 8. It runs on Apache Tomcat 9 web server (https://tomcat.apache.org/download-90.cgi). And uses one third party library StanfordNLP for lemmatization (https://stanfordnlp.github.io/CoreNLP/).

Application has 3 logically different parts: Web Interface, Core and Database.

Web Interface is using Apache Tomcat to handle http communication. Java Servlets are used to serve http requests and responses among with JSP files to support dynamic web pages. Java Servlet uses Core (or main Service) to process queries.

Core of application is a Service which does all main logic and implements algorithms described above, but for 3 different databases (100, 500 and 1000 documents). It is made for representational and experimental purposes, so user can see efficiency growth with only one query.

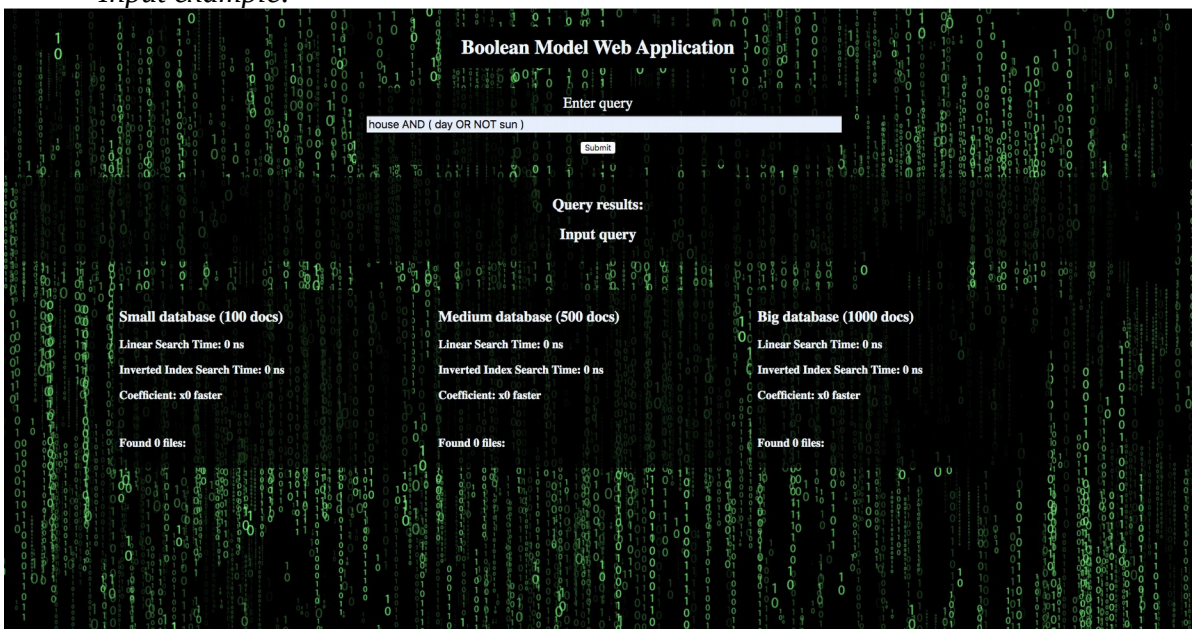
Database as mentioned consists of 3 sets of text documents. It is not relational database, just text files.

Running requirements are Java 8 and Apache Tomcat 9 web server (but I believe 7+ should work too).

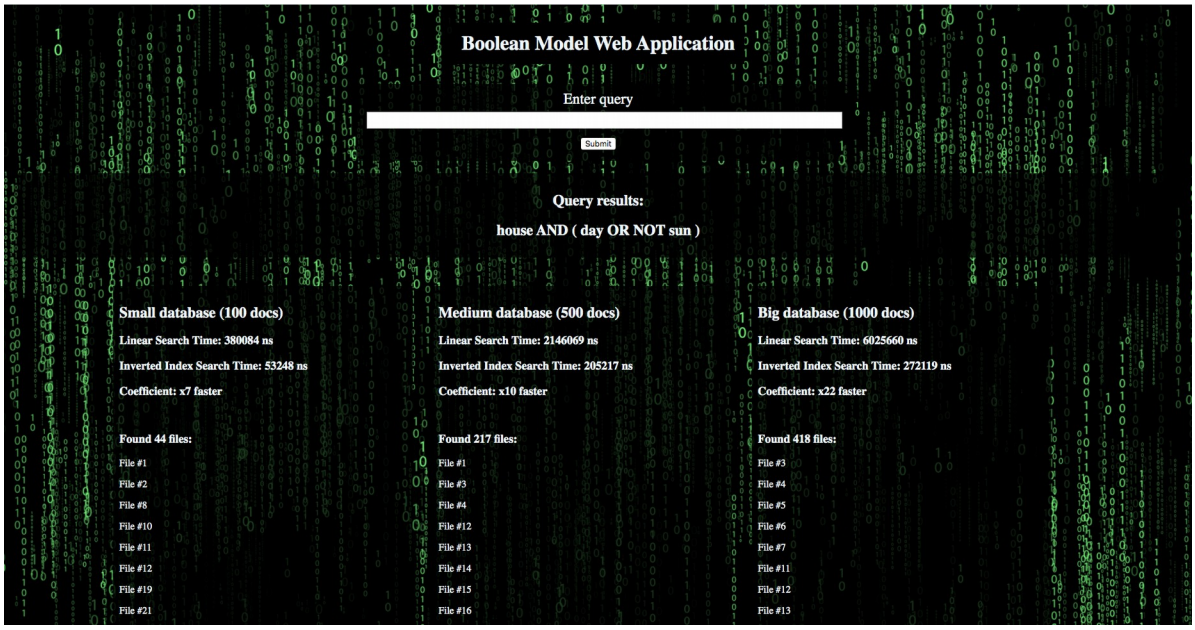
Examples of Input and Output

Input should be valid boolean query (only AND, OR, NOT, “(“, and “)” allowed).

Input example:



Output example:



Experiments

Experiments were conducted at 3 different bases with different amount of documents to show how efficiency coefficient grows. On next 3 tables there are results for databases where queries are same.

Small Database (100 docs)				
Query	**Linear time (ns)**	**Inv. Index time (ns)**	**Coefficient**	**Files**
money	72318	5866	12	17
house	82250	6802	12	44
office	77469	6525	11	36
NOT money	162753	21489	7	83
NOT house	109010	15822	6	56
NOT office	107650	14646	7	64
house AND day	288668	28179	10	25
house AND NOT day	176005	24059	7	19
house AND day AND sun	187032	14770	12	1
office OR flower	109415 96436	6928	13	36
office OR NOT flower	133827	20538	6	100
office OR flower OR gun	157107	9958	15	39

Medium Database (500 docs)				
Query	**Linear time (ns)**	**Inv. Index time (ns)**	**Coefficient**	**Files**
money	879815	11565	86	98
house	955504	12590	75	220
office	1206609	15771	76	173
NOT money	759061	39015	19	402
NOT house	760360	54498	13	280
NOT office	810064	62304	13	327
house AND day	1566876	60244	26	154
house AND NOT day	1554633	98215	15	66
house AND day AND sun	1561487	51068	30	4
office OR flower	1079863	13236	81	181
office OR NOT flower	1509403	115611	13	492
office OR flower OR gun	1638884	25340	64	199

Big Database (500 docs)				
Query	**Linear time (ns)**	**Inv. Index time (ns)**	**Coefficient**	**Files**
money	2421929	18192	133	217
house	2853011	21611	132	423
office	3917387	23190	168	348
NOT money	2586434	70870	36	782
NOT house	2780836	106737	26	576
NOT office	2597284	89702	28	651
house AND day	4293803	88954	48	286
house AND NOT day	6195255	200898	30	137
house AND day AND sun	7945598	114984	69	9
office OR flower	4030245	21435	188	362
office OR NOT flower	5784310	165580	34	985
office OR flower OR gun	6505269	39833	163	394

On the next table shown comparison of coefficients between databases. It is clear that with bigger database speed of search increases dramatically.

Coefficients			
Query	**Small DB**	**Medium DB**	**Big DB**
money	12	86	133
house	12	75	132
office	11	76	168
NOT money	7	19	36
NOT house	6	13	26
NOT office	7	13	28
house AND day	10	26	48
house AND NOT day	7	15	30
house AND day AND sun	12	30	69
office OR flower	13	81	188
office OR NOT flower	6	13	34
office OR flower OR gun	15	64	163

Discussion

Boolean Model has some huge advantages like simplicity and speed in particular cases, but doesn't make it any worse. Thus it is important tool. Disadvantages present as well and partially resolved in Extended Boolean and Model Vector Model, but it is a situation depended.

Since Boolean Model is pretty a simple concept, an implementation of it in the application almost repeats its idea. Benefits of using it are more visible with increasing amount of data. Although not every query gives a good example of efficiency. With growing complexity of queries more optimization is required, especially for an operator NOT. Another interesting moment is hardware. I was using my PC for testing and because I had to restart a web server quite a lot of times while testing and some results were ridiculous because of decreasing performance of my PC. Also it really depends on amount of data. Most of queries (even in linear search) take less than second to process on 1000 files. Difference between 1 thousand and 1 billion ns looks good, but actually does not feel a lot in real life. I would use bigger database, but my laptop takes lots of time during every restart of the application. One of the solutions I see is to do a preprocessing in separate application or independent (micro) service and connect to it with main one. Overall each step in algorithm could be improved and optimized (for example lemmatization or indexing). Here only basic and general approach is implemented and for small simple queries it works great.

Conclusion

Boolean Model and Inverted Index is a great example of simple but powerful search over huge pool of text files even though it requires preprocessing. Boolean Model does not work efficiently with small amount of data or complex queries and does not give you much information about context of document. But provides you with another type of instrument which fast and simple in implementation.

It is a good start for exploring other similar and more complex models with more parameters and slightly different goals. Apart from other advantages presented in lecture I find this project very educational. I have learned new interesting concepts and understand them deeply. So to prove that concept really works I had to make sure that every step from web page to actual execution of query works correctly. And exactly this verification helped me to understand better not only Boolean Model and Inverted Index but also ETF grammar, architecture of this type of application (which in my case can be improved) and other small things.