MINI SEARCH ENGINE

PROJECT REPORT

BY:
KUMAR PRASAD DAHAL

Table of Contents

1. Introduction

Information retrieval is the process of getting relevant information from a big collection of data set, usually web pages to address the desire of users to search for the information using query. The collection can be anything that contains useful information like books, research papers, journals and web pages. We have implemented search engine by implementing the required component in an incremental way. The major components and process involve: web crawler, preprocessing the documents, building an inverted index and implementing retrieval logic. At first, input documents are collected by building a web crawler which will visit web pages starting from seed URL and continuously browse page and store page content in a text file. For now, the number of pages to crawl is set to 10,000 and only pages from Memphis.edu domain are crawled. After crawling, each document is preprocessed to remove stop words, punctuation and special character. Words are stemmed in their root form to handle morphological variation.

Next, using preprocessed documents inverted index is created. Finally, using the inverted index cosine similarity score is calculated between query and documents and relevancy is set to each document based on similarity score. Finally, relevant pages based on similarity score are presented to the user through the graphical user interface.

2. Approach

There are three types of information retrieval models- Boolean Model, Vector Space Model and Probabilistic Model. The Boolean model lies under a set-theoretic model which is mainly based on either query term are in documents or not. It does not consider frequency. Vector Space model is an algebraic model in which query and document both are represented in term of vector and relevance ranking is performed on the basis of cosine similarity between document and query vector. In the probabilistic model, a similarity score is calculated based on the probability of a document relevance to the query. In this project, I have implemented vector space model.

3. Design

This section discusses design of web crawler, preprocessor, indexer and retrieval and ranking engine. Mainly, there are three layers in search engine- document collection, search engine logic and user interface. Below is the high level architecture of search engine.

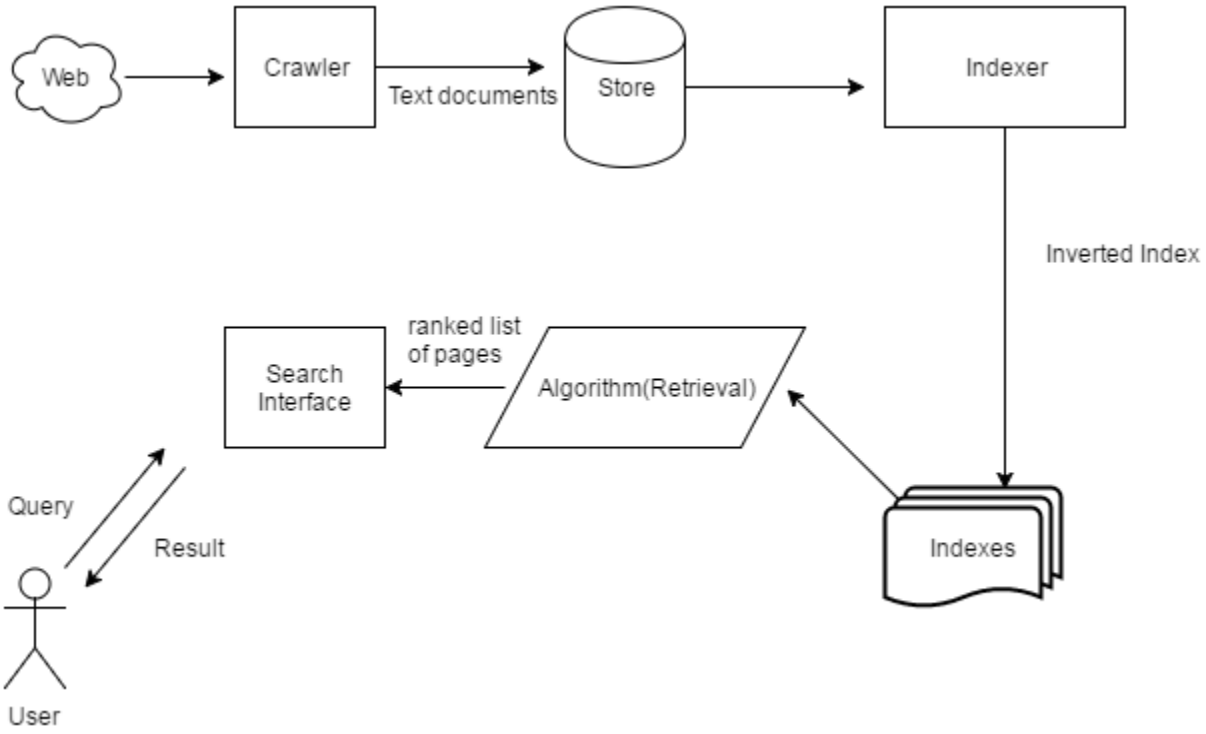


Fig: Search engine architecture

3.1 Crawler

The web crawler will continuously browse the web pages and collect data for the purpose of indexing. It will start visiting the page from seed URL, download the content of the page and save it in a local directory and collect all the links contained in the page and store them in an URL-collector list. URL collector list is the global URL collector where URLs will be added on each page visit by the downloader. Before a URL feed to the downloader, the crawler will check if it is already visited or not to avoid duplicate visit of the page. The user can set the limit on the number of pages to visit. Time pause of 3 seconds after each page visit has been set to avoid denial of service from the host. Below is the flow chart of the crawler.

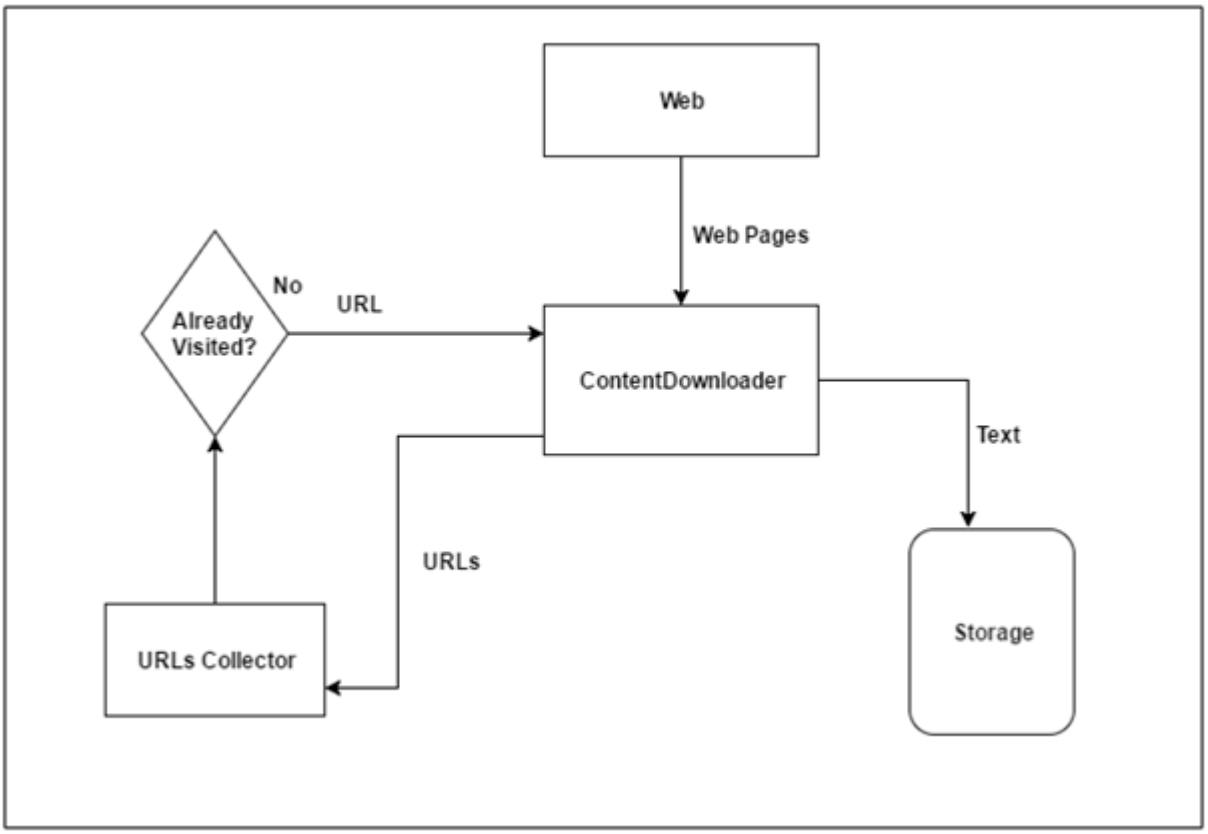


Fig: Web crawler architecture

3.2 Preprocessor

As we know, web pages do not contain clean texts; rather they contain a lot of punctuations, special characters, URLs and email address. In preprocessing, we, basically, remove all of this unnecessary content at first. After this, we will get the clean text. We will then convert them into lower case and stem them to root form. We will also remove stop words. Stop words are those words that appear in more than 80% of total documents.

3.2 Indexing

Basically, indexing will create an inverted index. The inverted index is an index which will map each word to the documents containing that word along with the frequency of the word in that document.

Indexing is done in order to quickly identify the documents containing query words so that we can calculate score only for few documents, not the whole collection. It is the matrix of word versus documents which shows the count of each word in each document. We will also store the term-document frequency in the hash. It will be used in calculating IDF.

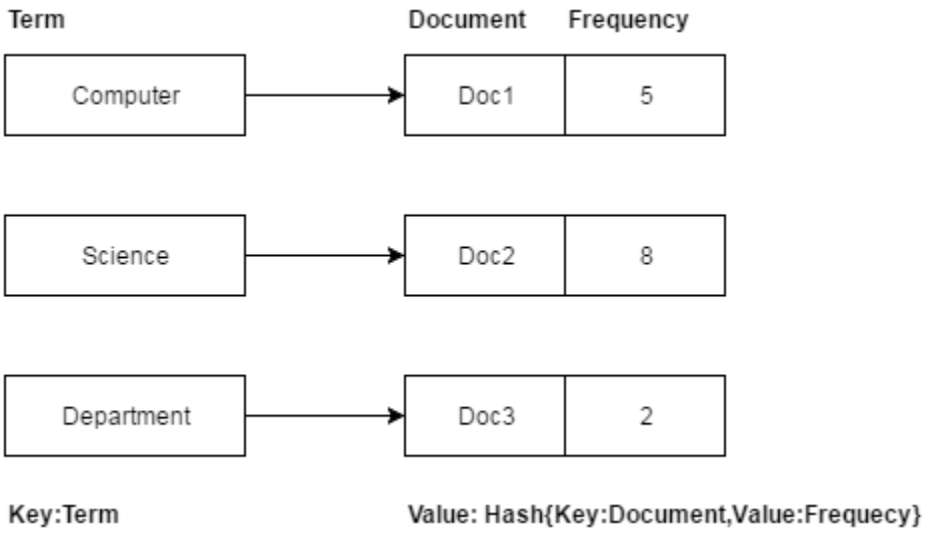


Fig: Sample Inverted Index

The weight of each term is calculated as the product of its term frequency (TF) and inverse document frequency (IDF). IDF of a term is calculated as the log ratio of the total number of documents over the number of documents containing that term.

3.3 Retrieval Engine

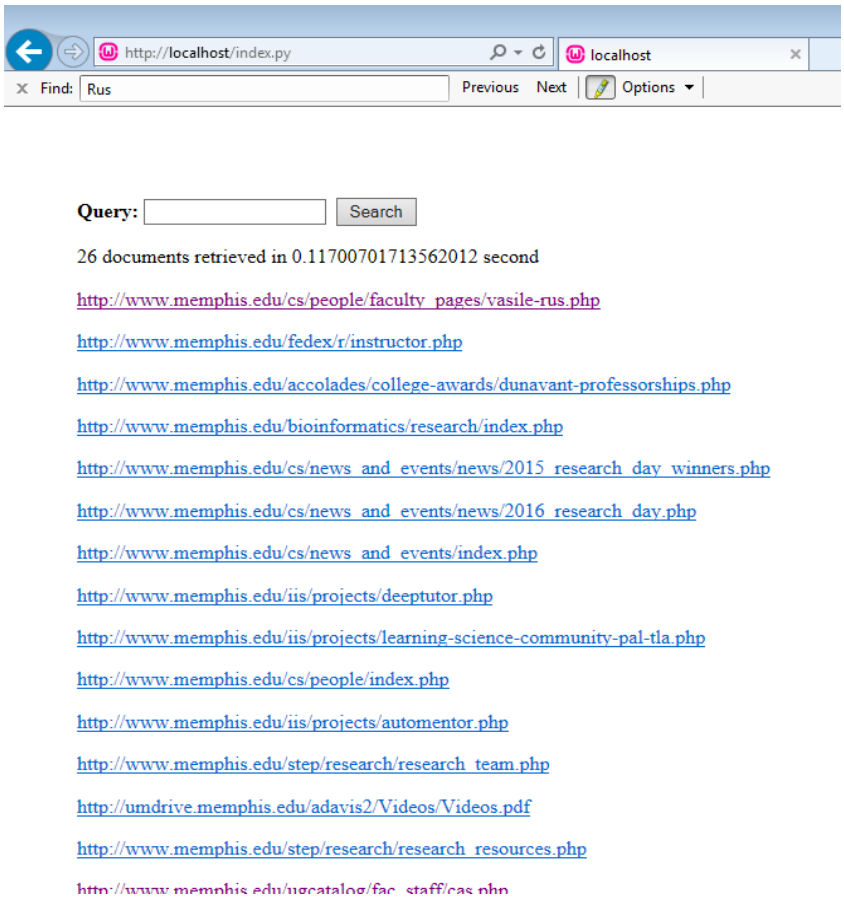
In retrieval, relevance documents are identified and scored for the ranking purpose. Unlike Naïve approach, we will only score those documents contained in the inverted index. First query words will be searched in inverted index and documents that are pointed by the word in inverted index are taken. Now we will compute cosine similarity between query and documents and assign the score to each document.

$$\cos sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{\left|\vec{d}_j\right|\left|\vec{d}_k\right|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2}\sqrt{\sum_{i=1}^{n} w_{i,k}^2}}$$

To apply this formula we will compute the weight of query, the weight of document, length of query and length of the document and finally compute the score for each document. Document with a high score will be at higher rank during retrieval.

3.4 Search Interface

Since we need to pass query from the browser, we need to design a page with the text box to submit the query to retrieval engine. The interface has been created in python. Below is the snapshot of User Interface. The results displayed are of query "Vasile Rus".



3.5 Retrieval engine

In order to communicate from client side to server side, I have hosted this project on wamp server. Common Gateway Interface (CGI) is used to communicate messages between the web server and python script.

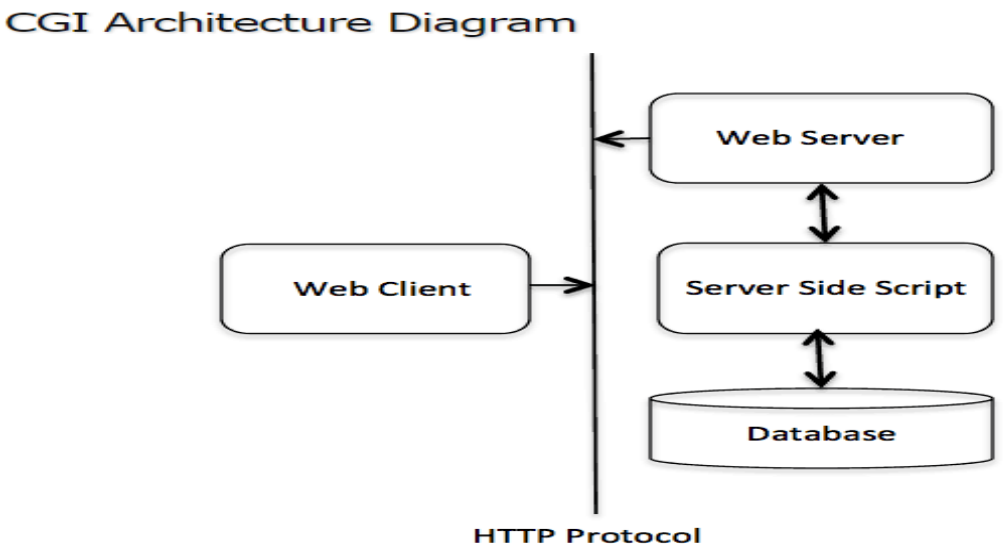


Fig: CGI architecture
Source:https://www.tutorialspoint.com/python/python_cgi_programming.htm

Here, we will send the query from browser to the server. The server passes that query to the python script. Python script does computation and sends the result to the server. The server passes that result to the client.

4. Implementation

This project is implemented in python for server-side scripts and client-side design as well. CGI was used to pass the message from script to server. The project is carried out using the approach of incremental component development. Each module will do a specific job required for information retrieval.

Firstly, module crawlweb.py crawls pages for the purpose of indexing. Starting from seed URL Memphis.edu, it will continuously visit pages of Memphis domain and store text data in a directory and also collect links to each page and store in a list. URLs will be continuously fed to the downloader until the number of pages visited reach user-defined pages visit limit 10,000. Only HTML pages are downloaded by the crawler. Also, to prevent from the duplicate visit of the page already crawled list will save all visited URLs and before new URL is feed to downloader it will be checked against URLs in the crawled list. Three second of delay after each page visit has been applied in order to prevent application from denial of service from host.

Second, another module preprocessdocuments.py will iteratively take each crawled document at a time and preprocesses the content. Preprocessing includes removing punctuation, special character, email address and URLs. Also, it will convert text into lower case and stem each word. Stemming is the process of converting a word to its root form. For example, stemmed form of word computer is compute. Porter stemmer included in NLTK library is used for stemming. After that, all stop words are removed. Stop words are those words that occur more than 80 % of the time in the collection. After applying this preprocessing logic, words in each document are

saved in another text file with source URL on the first line. Thus preprocessed files contain URL and preprocessed words.

Third, using preprocessed words, the module in invertedindex.py creates an inverted index. Inverted index, basically, is a matrix that holds each word and each document with the frequency of the word in the document as value. Thus inverted index helps us to identify the documents containing the query terms very quickly. Also, we will create the term-document matrix to calculate TF-IDF weight of each term.

Fourth, module retrivedocuments.py module calculates cosine similarity between document and query vector and ranks the documents based on similarity score. It will calculate TF-IDF for document terms and query terms, length of document and length of the query to compute similarity score.

Finally, module index.py creates the user interface to take user query and to pass that query to retrivedocuments.py module. Query terms are also preprocessed at first to match the inverted index key. Common Gateway Interface implemented in this module helps to communicate query and result with the server.

5. Results

Analysis of results is done by calculating precision, recall, and F-measure. For simplicity, I have chosen only top 50 ranked pages as the reference. I annotate the 50 ranked pages manually. Based on the query I assigned the relevancy to the documents. Then, I pass the query to retrieval engine and calculate precision and recall for the result by pointing out the rank of retrieved documents. Precision, Recall and F measure are calculated using the following relations:

$$Precision = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ documents\ retrieved}$$

$$Recall = \frac{Number\ of\ relevant\ documents\ retrieved}{Total\ number\ of\ relevant\ documents}$$

$$F = \frac{2PR}{P+R} = \frac{2}{\frac{1}{R}+\frac{1}{P}}$$

The following table shows the precision, recall and F measures for 10 queries.

SN	Query	Precision	Recall	F-Measures
Q1	Career Service Memphis	0.408163	0.666667	0.506329
Q2	software engineering research	0.3125	0.428571	0.361445783
Q3	Cookie	0.666667	0.4	0.5

Q4	president of the university	0.3125	0.5	0.384615385
Q5	computer science research awards	0.354167	0.485714	0.409638554
Q6	semantic similarity	1	0.6	0.75
Q7	tiger bike's current offer	0.416667	0.571429	0.481927711
Q8	tiger bioinformatics internship	0.361702	0.425	0.390804598
Q9	How to graduate with honors?	0.319149	0.5	0.38961039
Q10	Who teaches web search?	0.021739	0.2	0.039215686
	Average	0.417325	0.477738	0.421358722

6. Future Works

Now, it will take a long time to crawl 10,000 pages since we are not crawling multiple pages in parallel. In future, we can implement distributed crawler in order to crawl documents with high scalability utilizing full resources and crawling pages in parallel. Also, we have limited the pages to 10,000. We can increase this to many pages to visualize the real-time search, its limitation, and challenges.

References:

1. https://www.tutorialspoint.com/python/python_cgi_programming.htm

2. https://en.wikipedia.org/wiki/Information_retrieval