CS 6200 Final Report

Rujun Yao

Repository: https://github.com/pandalv9999/disease-

Abstract

Disease- is a simple search engine consists of a web crawler, and indexer and a ranking system that is implemented intended to solve the problem that people are reluctant to go to hospital during the pandemic of COVID-19 and some existing medical websites' searching inefficiency. The presented code does not have a web user interface, but an android user interface is attached to it. The crawler, indexer and ranking system does not utilize any existing open source tools. An evaluation is attached at the latter part of his report.

Introduction

A web search engine is a software system that is designed to carry out web search, which means to search the world wide web in a systematic way for particular information specified in a textual web search query. The search results are generally presented in a line of results, often referred to as search engine results pages. The information may be a mix of links to web pages, images, videos, infographics, articles, research papers, 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. Internet content that is not capable of being searched by a web search engine is generally described as the deep web.

Currently as the development of the Internet, most people relied on using web search engine to search for information they need. *Google, Bing, Baidu, Yandex and Yahoo!* are the top five most famous search engine in the world. Although they are used by people speaking different languages (*Google, Yahoo!* and *Bing* are for English speakers, *Yandex* are for Russian speakers and *Baidu* are for Chinese speakers), they share some similarities and those similarities are the reason for their success. First, they have a simple and direct user interface. Most user could understand the usage of the search engine once they see the interface and within few minutes, they could examine through the returned result and possibly got their desired information. Second, they have a robust and precise ranking system. Relevant data are most likely to be retrieved and present to user. Those search engine support various kind of search, such as image search and news search and usually the best choice for a random user for information retrieval. Besides that, there are many other search engines that are used among professionals and are powerful for retrieving field related data. Those search engines might be difficult for non-professionals to use, but they are able to retrieve relevant documents and satisfy user's need.

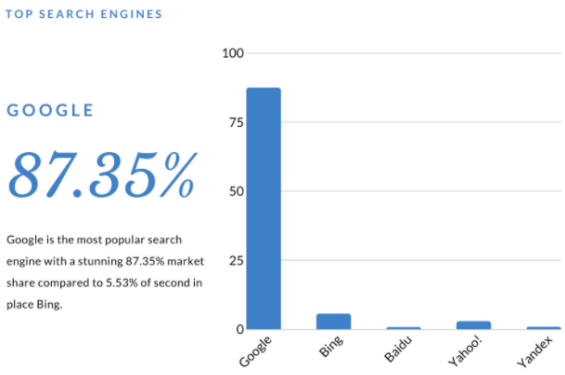

Figure 1: The top five search engine in the word by popularity.

Although the functionality of search engine is powerful, its basic structure and algorithm are not difficult. Based on what I have learnt in the course of information retrieval, I have implemented Disease-, a simple search engine consists of a web crawler, and indexer and a ranking system. Disease- is focused on the keyword of disease symptoms, disease name and possible treatment of a disease. More information would explain in detail in the following sessions.

Problems

Currently we are facing the pandemic of COVID-19. Hospitals are full of patients. If we are not feeling well and thinks having some minor disease other than COVID-19, it is not a good choice to go to the hospital. Instead, we are more willing to choose to search online and find out what disease could we possibly get. Besides that, it will be better for us if we could find a cure for minor illness online.

However, the search engine of existed medical sites have some problems. The first problem is that the search within the site will focus more about news. If we search for specific symptoms, the results of the search are not that satisfactory. Therefore, I want to make an integrated search engine that focus on the search on symptom keywords.

I have searched for three different medical websites and trying to find the best seed for our search engine. The first one is the official site of Centers for Disease Control and Prevention (CDC). The site contains information for known infectious disease and other common diseases such as diabetes, Cancer and influenza, as well as the real time data for the coronavirus. The website contains symptoms of a disease, required actions after being infected and some useful tips and guidance for patient. However, there are few problems that make the website not the best seed to crawl. The first problem is that the site contains languages assistance. It contains pages of languages other than English that increase the work of data extractor and indexer. The second problem is that it contains many other information that is not useful for our search engine, for example, food safety, school health and tobacco use. The third problem is that the data in a page is separated. For example, if we want to know more about coronavirus, we need to click “Symptoms” link to go to other pages. Therefore, I did not choose CDC for the seed. Same problems with WHO official website.

My final choice for the seed is MedlinePlus, US national library of medicine. The website contains relative information about common and uncommon diseases, and each page are organized well so that it contains a lot more information. Besides that, it contains some explanations toward symptom keywords, some information about medicines, and an encyclopedia for medical terms.

Implementation

A classic search engine consists of four parts: user interface, web crawler, indexer and ranking system. User interface is where user interact with the search engine: entering query text and examining the result set. Web crawler crawls the web page, examine the content of the page and process the text properly for the convenience of indexer. Indexer build up an inverted index where each term will have a posting list, which contains the documents that the term appears, and possibly some other information, such as the position of term in the documents, next pointers and so on. The ranking system will sort the result in order of relevance by different ranking algorithm. In reality, a web application contains front end and back end. The front end of the search engine will be the user interface. A query extender might be included in the user interface in order to helper the search engine more user-friendly and intelligent. The backend logic will process user’s query text, analysis the text and accessing the index (generated by the indexer), generate a result set and the ranking system will rank the result set based on the ranking algorithms, and return the result set to the user interface. Caches, clusters, disturbed computing, machine learning and many other

techniques might involve to improve the correctness and efficiency of the search engine. The communication between the front end and the back end is typically HTTP request. The front end evokes remote procedure calls of the back end, and the backend has a dispatcher to divided different request to different business logic. Indexer and web crawler could be either implemented as backend business logic, or it could be implemented as independent microservices. The main business logic could invoke the crawler service and indexer service periodically and hereby accomplishes the following benefits: making the code highly maintainable and testable, loosely coupled, independently deployable and organized around business capabilities.

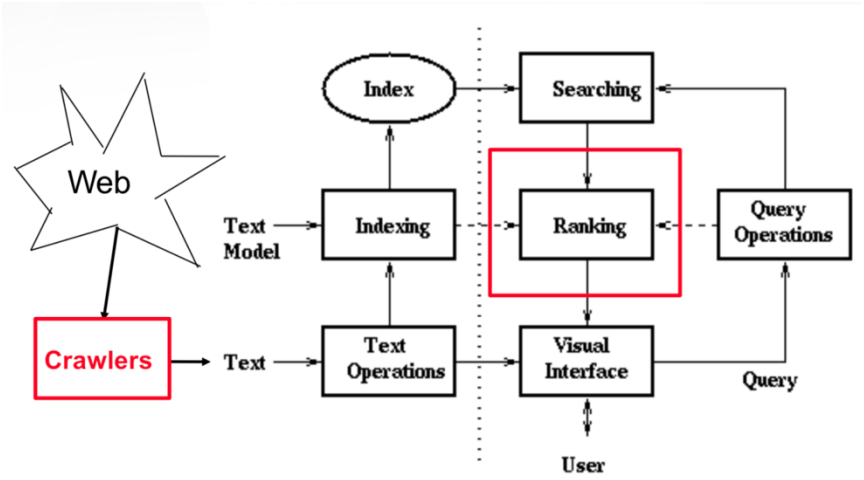

Figure 2: Structure of a classic search engine

In the following session, the detailed introduction of different module in my search engine will be presented.

User Interface

The job of the search engine user interface is to help the user to express their need and form their queries, and help the user to understand what is returned by the search engine, and record down user's interaction with the search result and collect the data, such as user's clicking, re-entering the query, and other interactive data to improve the quality of search engine. The interface is usually in a simple form, where a search bar, a search button and a result list consist of its major components, for the following three reasons:

- User do not want their thought to be disturbed by an intrusive interface when seeking information for a larger task.
- It is not possible for user to read and to think about something else when they are reading the retrieved information,
- The interface design must be understandable and appealing to a wide variety of users of all ages, cultures and backgrounds, applied to an enormous variety of information needs.

Therefore, the mentioned three elements are sufficient in a user interface. Some user interface will include a query expansion module that selecting and adding terms to the user's query with the goal of minimizing query-document mismatch and thereby improving retrieval performance. Those

extra query terms are often obtained by spell check, stemming, and retrieving relative words from cache. For the logic part, the user interface should contain a function that could record down user's input in the search bar, and send the input to backend via HTTP Post or Get method. The method is usually handled by AJAX, Asynchronous JavaScript And XML module. The benefit of using asynchronous request is to avoid the overhead consumed by rendering the whole page another time. After user enters the query and clicks the "Search" button, the query content will be put in the parameter of a HTTP GET method or a field of the body in a HTTP POST method, along with other searching conditions or constrains (in parameters or in field of the body). The request is sent to the backend, and the backend returns the search result. The remaining job of the user interface is to list the result, a sorted list of result in order of decreasing relevance based on the ranking system, in a simple and succinct manner for user to inspect.

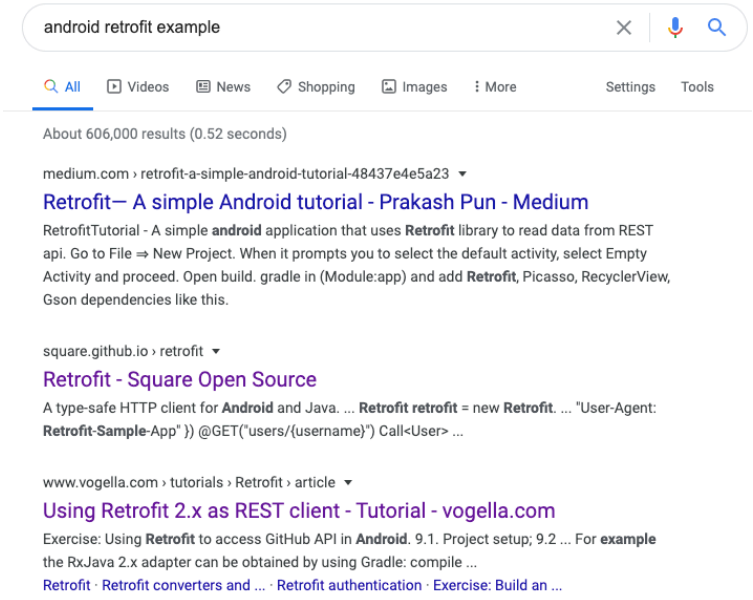

Figure 3: A classic user interface of a famous search engine consisting of a search bar, a search button and a list view of relevant result.

Since I have lack of experience of developing front-end code with JavaScript, in the final presentation my project only consists of some python scripts running in the system's command line interface. However, I have ignored an important fact: user interface is not restricted to the web interface. Mobile App's user interface is also considered as user interface for a search engine. I have experience on developing mobile, especially Android Apps so that I should have included a workable user interface in mobile App. The main activity of the interface is very simple: it consists of a text bar and a query button. I set a OnClickListener to the search button that will extract the text content in the text bar and send a HTTP Get request to the backend server via Retrofit. Then the response, in JSON format is passed to the next Activity as extra. In the second activity where the search result is displayed, the response is parsed into several items and these items are to be put into an adapter, and the list view will use this adapter to display the search result. Since I did not have that during the final presentation and what is presented in the final presentation should be considered as the final version of our search engine, I will not include the code to my GitHub repository.

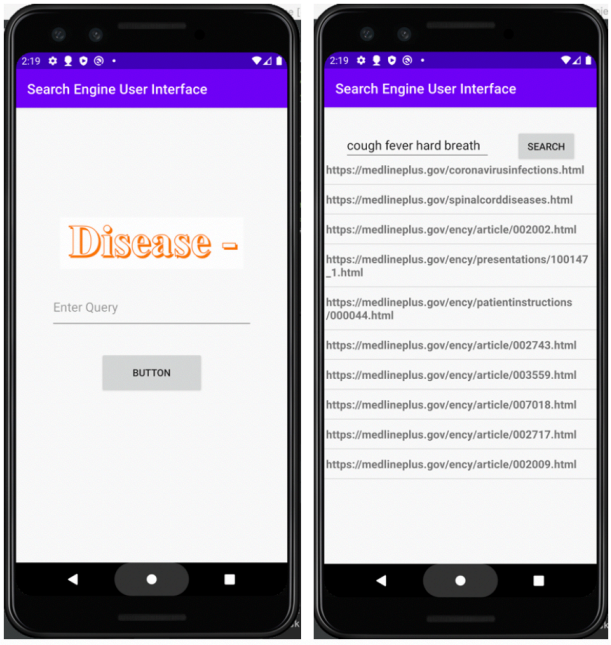

Figure 4: An Android User Interface for my search engine.

Web Crawler

A Web crawler, sometimes called a spider and often shortened to crawler, is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing (web spidering). Web crawler starts from a specific seed, extract all texts and metadata from the webpage, and extract all reference links of the page and go to next page after finishes the page. The texts, possible other multi-media data will be passed to indexer or stored to storage for the indexer to do the job.

My web crawler is simply a python script without famous web scrapping tools such as Scrapy. The web crawler consists of a set storing visited page, and a queue storing all expanded yet visited page and the current depth of a page. The main function of the crawler is basically running a breath-first search algorithm, popping out a link from the queue (URL frontier). For each link popped, the crawler will first examine the current depth. I set the maximum depth for the crawler to be 3. Then, the crawler will extract all links in the a-tag and attribute “h-ref” using regular expression matching. For each extracted link, the crawler will first check whether the link is belongs to the site, and check whether the page is in Spanish or not. After that, the content of the page is stored as a text file locally, along with the extracted meta data of the page. Finally, the link is pushed into the queue, and the crawler will pop another link from the queue and repeat the whole process. User agent proxy code (prevent the site from banning the crawler) are included in the crawler but never used, since my crawler is crawling in a moderate speed and will not trigger the protection of the website. Multi-threading code are also included in the crawler. After running the crawler overnight, my crawler successfully crawls up to 10,000 pages with a maximum crawling depth 3. All pages are stored locally with a proper assigned document ID.

Indexer

Indexer is the module that generated inverted index. An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. A complex inverted index would contain other information, such as the positions of each term in each document for the convenience of phase query. However, such inverted index would cost a lot of space. The inverted index of my search engine is of the simplest form: the key is a term; the value is a set for document Ids which the term appeared in the document.

The logic of the indexer is quite simple. The input is a directory, and for each text file in the document, the indexer will use regular expression to extract all words. The indexer filters out some words in the stop word list and generate a document-term index. Then, the index is converted into an inverted index, and meanwhile some characteristic such as term frequency, document frequency inverted document frequency, document vectors are calculated. When the indexing is done, the index and other useful data are all cached into local files as binary file. For the next time the search engine can get those data from the binary file and save a lot of time. In reality, the cache will be updated in a reasonable period to adapt the frequently changing web contents.

Ranking System

Ranking is the module where the search engine will use some pre-defined characteristics to decide the relevance of a documents, and sort the documents in the order of decreased relevance. In my implementation, I use TF-IDF and cosine similarity to define the relevance of documents upon a query. Term frequency, document frequency and inverse document frequency are all calculated in the indexing part, so the search engine is able to evaluate the relevance of a document in a short time. Basically, the ranking system will separate all words in the query text, getting all documents that contains those words, and calculate cosine similarity by the method described in the class. After that, the system uses a min heap to return the top 10 results. Note here the returned score are negative. This is because when I using heap, the default mode of the heap is a min heap. In order to get a max heap and return the max score first, I have to negate all scores to get the max score.

```
COSINESCORE(q)
1  float Scores[N] = 0
2  float Length[N]
3  for each query term t
4  do calculate w_{t,q} and fetch postings list for t
5      for each pair(d, tf_{t,d}) in postings list
6      do Scores[d] += w_{t,d} × w_{t,q}
7  Read the array Length
8  for each d
9  do Scores[d] = Scores[d]/Length[d]
10 return Top K components of Scores[]
```

```
def cosine_score(self, doc_ids, query):
    score = dict.fromkeys(doc_ids, 0.0)
    for word in query:
        # for query word not in docs, skip, (0)
        if word not in self.inverted_index.keys():
            continue
        # weight of query term is the idf
        weight_tq = self.indexer.idf[word]
        for doc_id in self.inverted_index[word]:
            weight_td = self.indexer.get_score(word, doc_id)
            score[doc_id] += weight_tq * weight_td

    for doc_id in doc_ids:
        length = self.indexer.vectors_length[doc_id]
        score[doc_id] /= length
```

Figure 5: Cosine similarity algorithm from textbook's pseudocode

The reason I choose this ranking method is the easiness to implement. I did not choose any search tools such as elastic search.

Evaluation

An analysis and evaluation are required for the search engine. A common evaluation method for a search engine is to calculate the precision and recall. precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances. Recall (also known as sensitivity) is the fraction of the total amount of relevant instances that were actually retrieved. Since we are not crawling all websites for the seed, the recall is difficult to calculate. And precision is difficult to define in one-word query. Therefore, we use the term frequencies of the retrieved documents to evaluate the performance of one-word query. For the one-word query, I selected five most common symptom keywords and five most common disease name as query text. For the free text query, I choose three of some popular medical searches as query text.

One-word query

The first evaluation is the search of five most common symptom keywords. They are fever, diarrhea, fatigue, aches and cough. Below is the evaluation of the searches. The first row is document name, the second row is the word count and the third row is the cosine score.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
fever	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	5452.txt	659.txt	3015.txt	4795.txt
	234	234	234	234	234	234	50	43	44	33
	0.00114	0.00114	0.00114	0.00114	0.00114	0.00114	0.00086	0.00085	0.00084	0.00082
diarrhea	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	1964.txt	6433.txt	6998.txt	5351.txt
	142	142	142	142	142	142	77	40	30	25
	0.00200	0.00200	0.00200	0.00200	0.00200	0.00200	0.00186	0.00160	0.00153	0.00143
fatigue	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	6098.txt	652.txt	6035.txt	1863.txt
	98	98	98	98	98	98	35	35	33	32
	0.00257	0.00257	0.00257	0.00257	0.00257	0.00257	0.00221	0.00218	0.00215	0.00213
aches	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	5780.txt	6832.txt	2087.txt	5422.txt
	21	21	21	21	21	21	11	6	3	3
	0.00434	0.00434	0.00434	0.00434	0.00434	0.00434	0.00398	0.00342	0.00285	0.00285
cough	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	4351.txt	1819.txt	660.txt	5942.txt
	80	80	80	80	80	80	56	24	26	20
	0.00255	0.00255	0.00255	0.00255	0.00255	0.00255	0.00237	0.00303	0.00202	0.00188

Table 1: Analysis for one-word query: common symptoms for disease. The first row is document name, the second row is the term count in the document and the third row is the score.

Figure 6: Documents about fever, left is Document 5452; Right is Document 659

Here, the documents 5617, 5611, 5620, 5609, 5603 and 5604 are all xml pages with no metadata. They are of different link name, but by comparison their contents are the same. The xml page contains plenty of words, some of them are Spanish. Those pages are considered as noise for the search engine: although it contains a lot of words, and the term frequency for words would be very

large, the page contains no useful information. In order to eliminate these noises, we need to introduce duplicate elimination algorithm in the web crawler. Besides that, adding a filter to filter out XML links would also solve the problem. The rest of the results are useful information returned by the search engine. Document 5452 in fever search is a short encyclopedia defining the symptom "fever". (https://medlineplus.gov/ency/article/003090.htm). Document 659 is a web page that describe causes, treatments and therapy about fever (https://medlineplus.gov/fever.html).

The second evaluation is the search of five most common disease name. They are influenza, pneumonia, diabetes, cancer and HIV. I am using the same schema for the first evaluation.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
influenza	5092.txt	4372.txt	694.txt	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	5546.txt
	126	36	21	29	29	29	29	29	29	18
	0.00463	0.00350	0.00326	0.00313	0.00313	0.00313	0.00313	0.00313	0.00313	0.00310
pneumonia	5617.txt	5611.txt	5620.txt	5609.txt	5603.txt	5604.txt	171.txt	2812.txt	2815.txt	2810.txt
	84	84	84	84	84	84	59	44	47	31
	0.00266	0.00266	0.00266	0.00266	0.00266	0.00266	0.00243	0.00243	0.00243	0.00231
diabetes	3841	6759	185	5842	6687	3525	6589	4695	3606	6696
	1	1	1	1	1	1	1	1	1	1
	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008
cancer	3878	2203	2403	2878	6151	2423	3841	3463	55	6281
	1	1	1	1	1	1	1	1	1	1
	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002
HIV	No Results									

Table 2: Analysis for one-word query: common disease name. The first row is document name, the second row is the term count in the document and the third row is the score.

As we can see, documents 5617, 5611, 5620, 5609, 5603 and 5604, which introduced in the previous evaluation, also affects some of the search. Note that the search of diabetes and cancer only returns documents with term frequency 1. By examine these documents, I found that all of these documents only mention the search term in side list bar or navigation bar. The contents of the documents are nothing about the search term. The search of HIV returns nothing. This is due to the fact that my web crawler did not crawl the seed thoroughly. The maximum depth of the crawler is 3. A large portion of the sites is not crawled and the search engine will not retrieve those documents. This can be fixed by increasing the crawl depth. Document 5092 in influenza search is a page about flu shot. (https://medlineplus.gov/flushot.html). Document 4372 is a page about flu (influenza) test. (https://medlineplus.gov/lab-tests/flu-influenza-test/). Although these pages are not exactly what we want (we possibly want a page for introduction, treatment and symptoms of influenza), they contains the word and they are relative to the term.

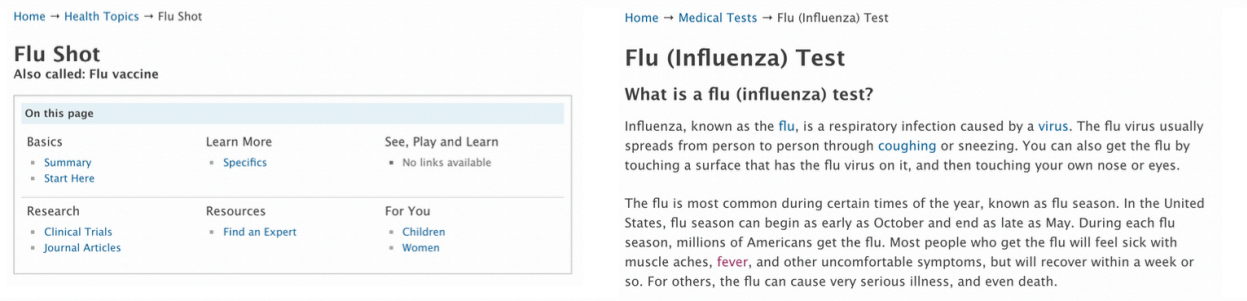

Figure 7: Documents about influenza, left is Document 5092; Right is Document 4372

Free-text Query

To evaluate free-text query, I select some most searched query of medical query and list the top three retrieved documents. Then, the document name, link and descripted is presented.

Query text: what are the symptoms for coronavirus

Document: 4406.txt, Ranking: 1st, Score: 0.00594
Description: Coronaviruses infect the respiratory system. Coronavirus infections are common and are usually not serious. In recent years, some coronaviruses have changed into more dangerous forms. These new viruses are SARS, MERS, and the newly discovered COVID-19. Testing helps prevent the spread of disease. Learn more.
Link: https://medlineplus.gov/lab-tests/coronavirus-testing/

Document: 68.txt, Ranking: 2nd, Score: 0.00429
Description (keywords): Lungs and Breathing.
Link: https://medlineplus.gov/lungsandbreathing.html

Document: 77.txt, Ranking: 3rd, Score: 0.00413
Description: News about MedlinePlus, including announcements of updates, changes, and new content.
Link: https://medlineplus.gov/whatsnew/

Query text: What can I do to stop snoring?

Document: 423.txt, Ranking: 1st, Score: 0.00575
Description: Snoring is the sound you make when your breathing is blocked while you are asleep. Learn common causes. Find treatments available to reduce snoring.
Link: https://medlineplus.gov/snoring.html

Document: 4537.txt, Ranking: 2nd, Score: 0.00544
Description: Snoring is a loud, hoarse, harsh breathing sound that occurs during sleep. Snoring is common in adults.
Link: https://medlineplus.gov/ency/patientinstructions/000720.htm

Document: 4536.txt, Ranking: 3rd, Score: 0.00493
Description: Snoring affects many of people during their sleep. Often, people do not even realize they are snoring.
Link: https://medlineplus.gov/ency/anatomyvideos/000119.htm

Query text: What causes kidney stones?

Document: 4162.txt, Ranking: 1st, Score: 0.00364
Description: A kidney stone is a solid mass made up of tiny crystals. One or more stones can be in the kidney or ureter at the same time.
Link: https://medlineplus.gov/ency/article/000458.htm

Document: 3816.txt, Ranking: 2nd, Score: 0.00346
Description: Bladder stones are hard buildups of minerals. These form in the urinary bladder.
Link: https://medlineplus.gov/ency/article/001275.htm

Document: 4163.txt, Ranking: 3rd, Score: 0.00367
Description: Cystinuria is a rare condition in which stones made from an amino acid called cysteine form in the kidney, ureter, and bladder. Cystine is formed when two molecules of an amino acid called cysteine are…
Link: https://medlineplus.gov/ency/article/000346.htm

Generally speaking, free-text query would return our information needed. However, since the ranking system is using TF-IDF and cosine similarity, without any support of phase query, the query result might not be that accurate and powerful. When we query a free text, the search engine will divide the query into several terms, retrieving relative documents for each query word, and finally calculated the score on the union of all documents. Therefore, sometimes the result might become the ORed result of several one-word queries. This can be solved by introducing a more powerful ranking algorithm, such as BM25.

Running instruction

Go to the script folder and execute the following command in command line interface:

python3 Main.py

The script supports the following argument:
optional arguments:

-h, --help show this help message and exit
-c, --crawl optional: crawl the content from cdc again. Warning: it will take a lot of time
-i, --index optional: index the content from local file again. Warning: it will take a lot of time

Conclusion

In conclusion, Disease- is a simple search engine that focus on disease and symptoms and consists of a crawler, an indexer and a ranking system. During the presentation there is no user interface presented. I have implemented a user interface in android system but I will not include it in the final project repository. The search engine is mostly implemented by myself, with little assistance with public tools and algorithms and. The performance is limited to the duplicated XML pages that are considered noise, the number of pages the crawler crawled and the simplicity of ranking algorithms. However, Disease- will present relevance information for the user. In the future, I would considered rewrite the application in Java and construct a HTML user interface, with tools like elastic search or scrapy. The project is a final presentation of what I have learnt in this semester. Hopefully the lacking of web interface will not cause a great point deduction.

Reference

https://en.wikipedia.org/wiki/Web_search_engine
https://www.reliablesoft.net/top-10-search-engines-in-the-world/
https://microservices.io/
https://searchuserinterfaces.com/book/sui_ch1_design.html
https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_947
https://en.wikipedia.org/wiki/Web_crawler
https://www.geeksforgeeks.org/inverted-index/
https://en.wikipedia.org/wiki/Precision_and_recall
https://www.health.harvard.edu/blog/google-top-10-health-searches-2017-2018022113300