CPS842

SEARCH ENGINE

0.1 Implementation Details

In the following we will discuss different steps of our implementation details of the search engine.

0.1.1 Crawler

We use an existing crawler from github, **https://github.com/mirkomantovani/web-search-engine-UIC/blob/master/crawler.py**, to collect some web data which is a multithreaded crawler with 20 worker nodes. We choose **https://www.ryerson.ca** as our *seed url* and restrict the data collection to only *ryerson* domain. To comply with politeness of crawlers, each request to server has a *10 sec* timeout. We crawled over 12,000 pages; however, we only processed 10,000 pages due to the hardware limitation.

0.1.2 Ranking algorithm

For ranking algorithm, we implemented **vector space model**. To calculate document weights, we implemented only *IDF*. The final score is calculated based on the following score (d, q) calculation:

$$\texttt{score (d, q) = w1 } * \texttt{cosine-similarity(d, q) + w2} * \texttt{pagerank (d)} \quad (1)$$

We find that $w_1 = 0.3$ and $w_2 = 0.7$ give us better results. In order to perform the page ranking we had to create a graph of the network

We apply the page ranking algorithm in our *parser* class which does the parsing of *html* files and pre-process the result. We run the page ranking algorithm 200 times; however, it converged after *150 iteration*. Every intermediate result which we need for searching e.g., Dictionary, posting list, web graph are stored in pickle files and are loaded during the searching process.

0.1.3 Inverted index

After pre-processing the web data, we created tuples of (`word, docID`) and from that we created dictionary and posting list. We further calculated the length of each document in our data set and stored in a dictionary (`doc-length-dict`) in order to expedite the searching process.

0.1.4 Challenges

The most difficult part was to find a crawler that was easier to use and was appropriate for our machine. We spent days working with *Apache nutch* to use it as a crawler but we could not figure it out. We also tried working with other open source crawlers and the only crawler that worked for us was **https://github.com/mirkomantovani/web-search-engine-UIC/blob/master/crawler.py**. So we decided to use the same page ranking as well from this directory. We originally wanted to crawl more web pages and process more data; however, our machines hardware limitation did not allow us. We put the crawler on to work overnight however every time it crashed after 1 or 2 hours of working.

0.1.5 Running the program

To run the program, the following steps are followed:

1. To start searching :
    - Run search.py
2. To create crawl :
    - Run run_multithreaded.py
3. To Preprocess
    - Run m_invert.py
    - Run run_page_rank.py

Once the program is executed, a simple UI will appear which will take in a query and display the first 100 results related to ryerson. If the user double clicks the link, the program should take the user to the website,as shown below with screenshots.We have also added screenshot (Figure 10, and Figure 11) for the query which has no results such as when we search for "Saghar Jiantavana", the program will print "No result", since there is no such name in https://www.ryerson.ca.

0.1.6 Screenshots and Results from the Program

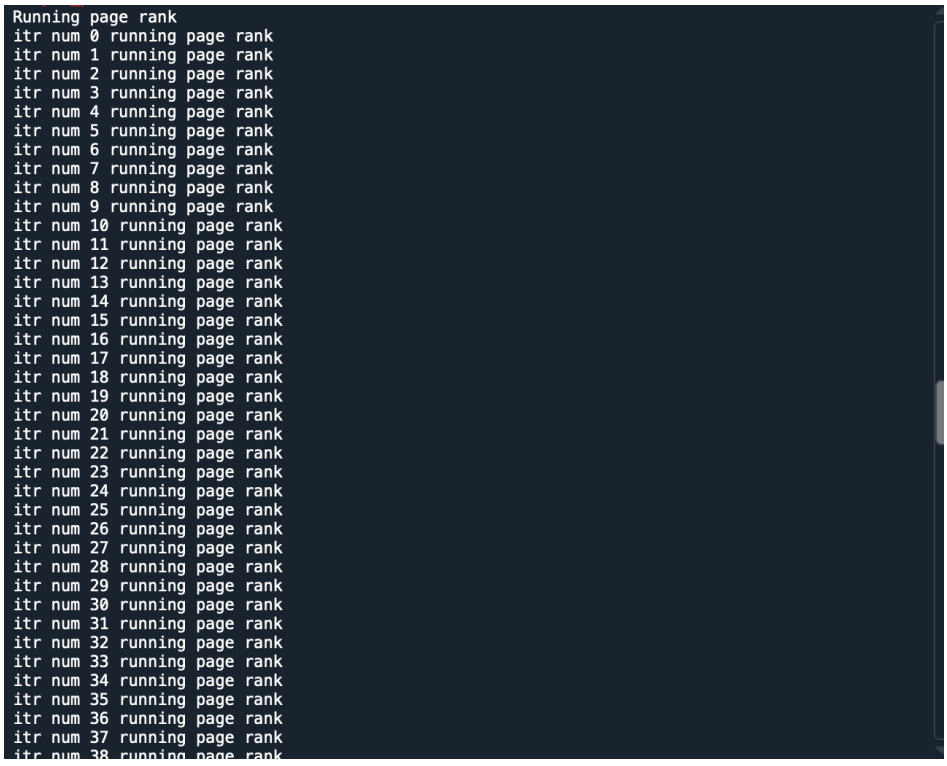

Figure 1: Page Rank

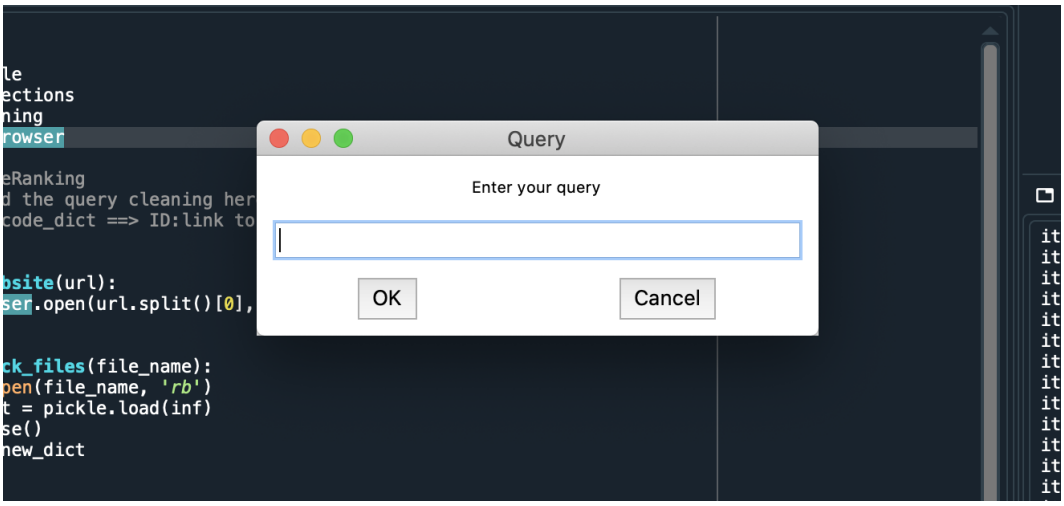

Figure 2: Query UI

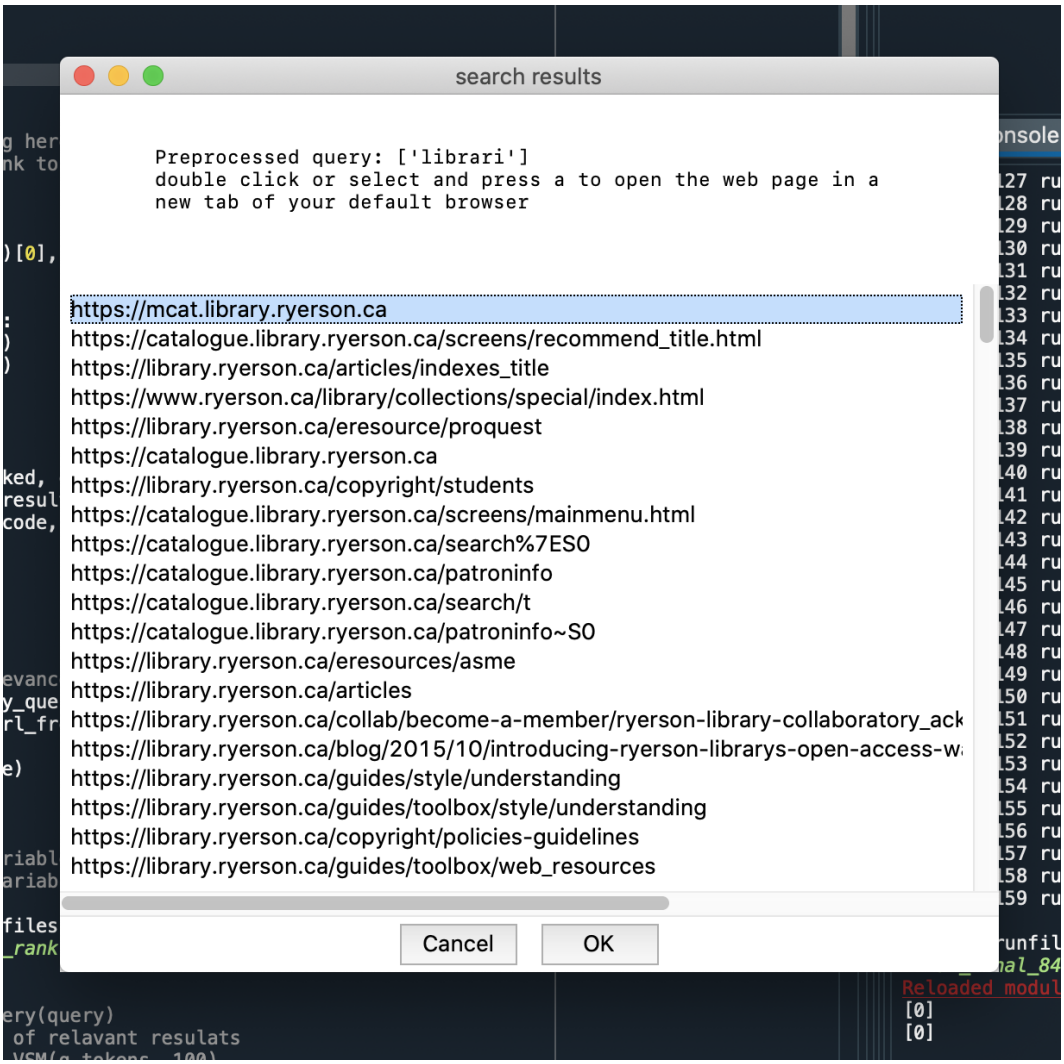

Figure 3: Search Results

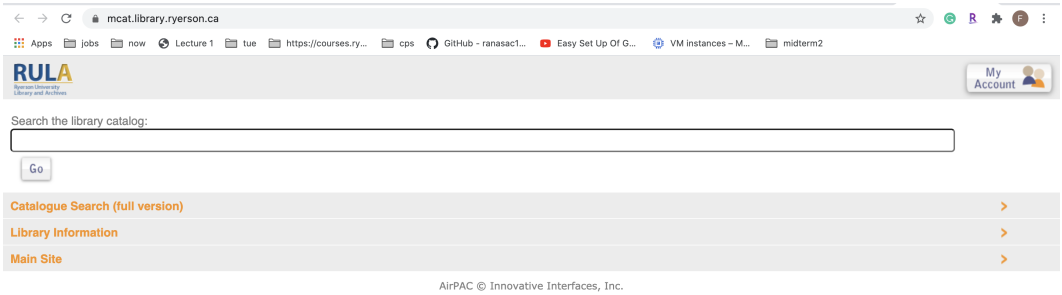

Figure 4: Web Results

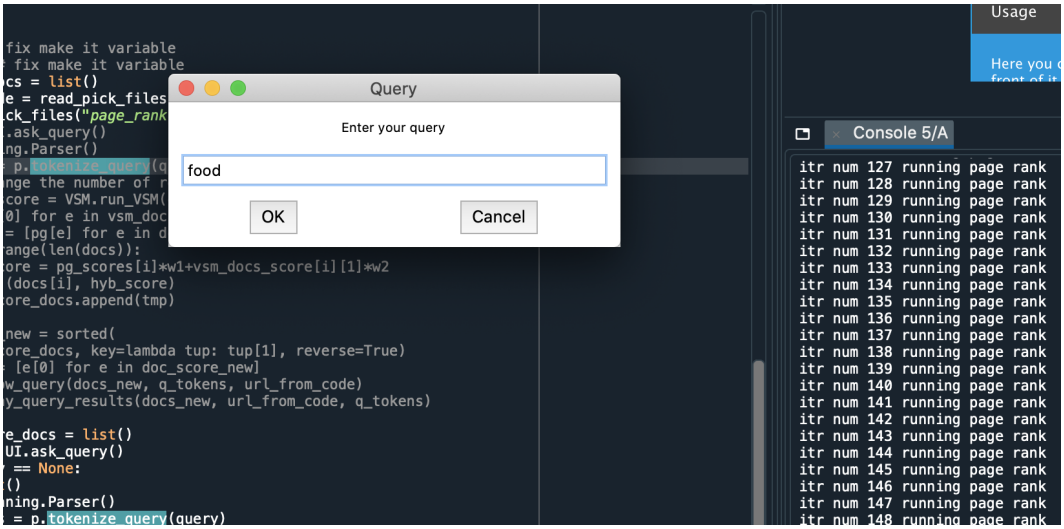

Figure 5: Query word 'food'

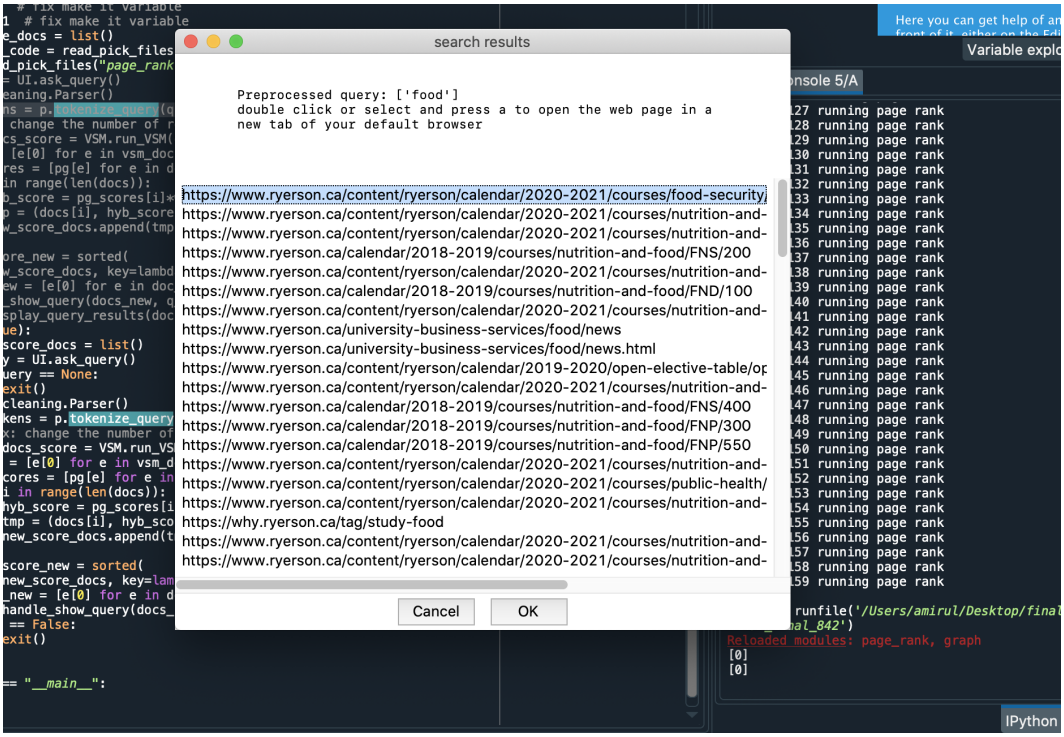

Figure 6: Search Result for 'food'

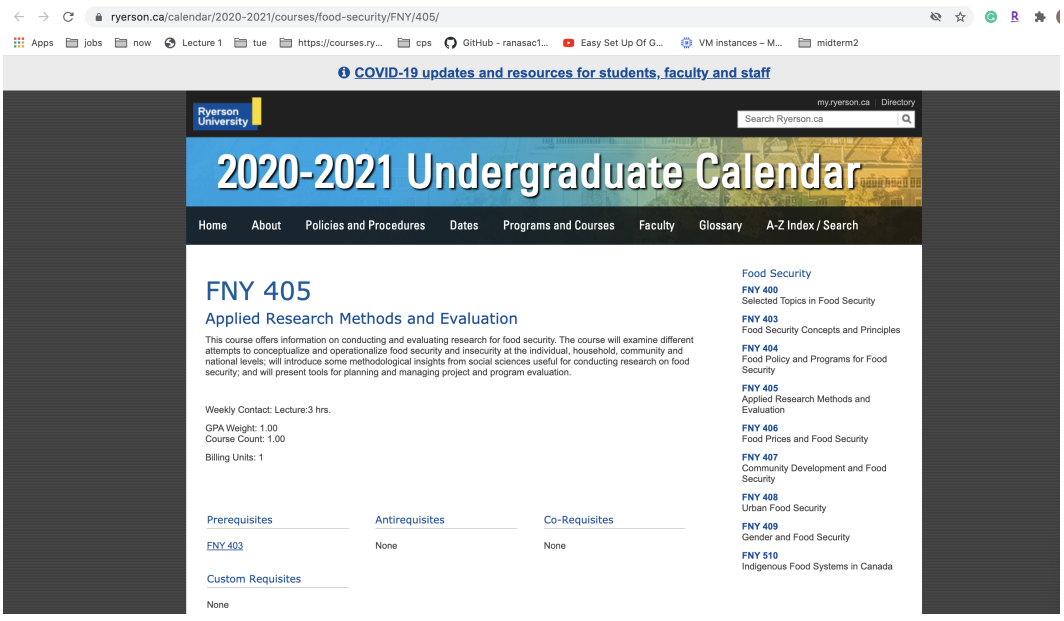

Figure 7: Web Result for 'food'

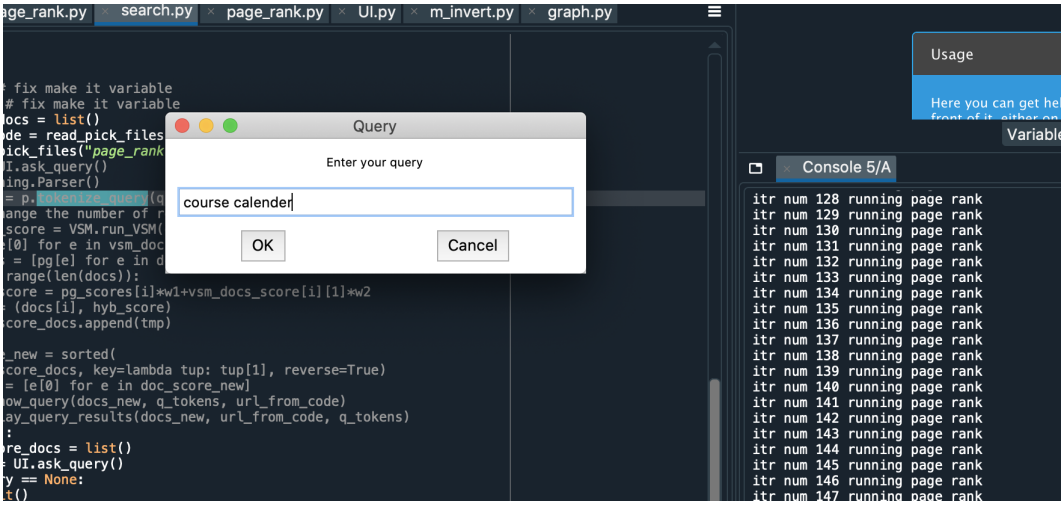

Figure 8: Searching 'course calender'

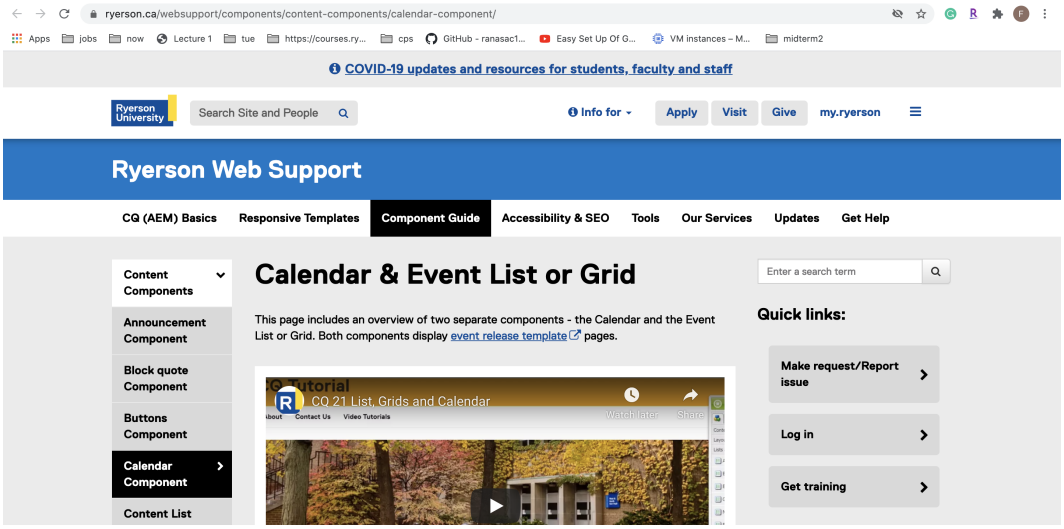

Figure 9: Web Result for 'course calender'

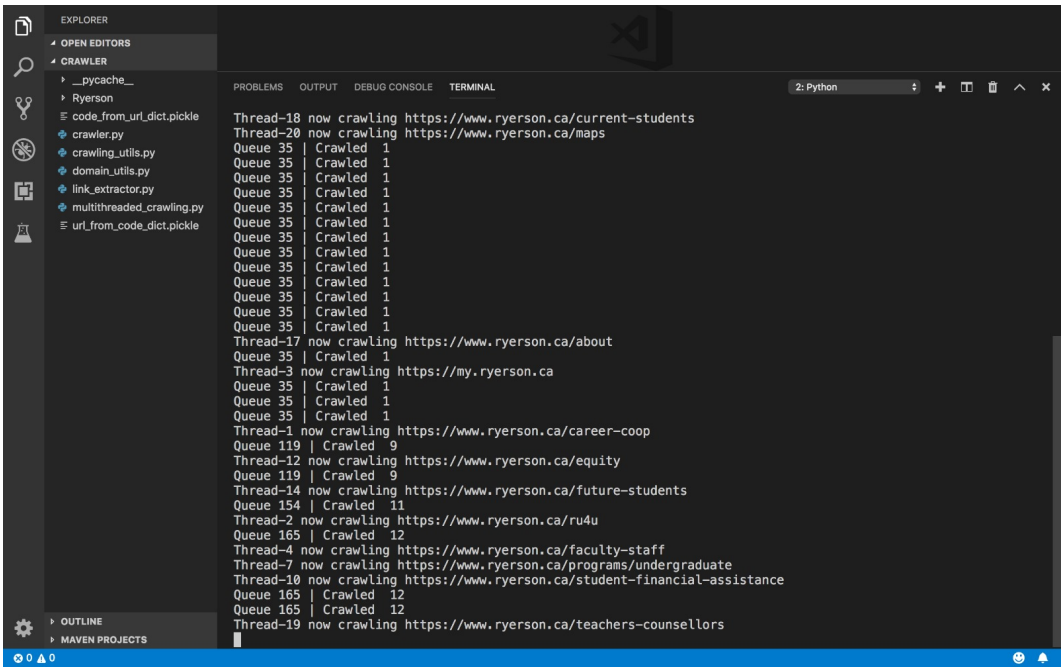

Figure 10: Crawler