SummerProgrammingPractice2020–Search Engine

Qianyi Zhang

July 27, 2020

Abstract

making a simple search engine to search the websites of School of Information, Renmin University of China.

1 Introduction of the function

The system of search engine will crawl the pages in *http://info.ruc.edu.cn* and save them as html files in the local computer, extract the tittle and main body of each page and save as txt files. Then it will use THULAC to cut the words and tag the attributes.it will also build a dictionary of every word appeared in the pages and the inverted index of each word. At last, user could input a query and get a series of URL based on tf-idf algorithm.

2 Design ideas

2.1 spider

I use the BFS algorithm and wget to get the web pages, at the same time, build a map from the URL to the Hash unsigned int value. I will save the html file and name it with the Hash value. After I get a new URL, I will delete all the content behind the last "/", and add the new URL that is not the complete website in the html file. If I find a complete URL ,I will save it directly. I also need to figure out whether the URL was found or not.

2.2 extract

I use two methods to extract contents of web pages.First, I will find ""content"" in the web page ,because usually the main content will appear after a label "<class = "content">".Then ,I will maintain a stack which to calculate the <...> and </...> to find the end of the label mentioned above.In this way, I will get the start and end position and save all the contents between ">" and "<".However, after some tests, I find some web pages don't have the label. Therefore, I looking forward to the new label "<body>" and then find all the content marked by "<p...>" and "</p>". Thanks to the normal contents of the webpages, the program runs well.

2.3 cut

The API of the THULAC didn't work. I have to call the "system(...)" function to cut the word. And this step is easy. I use the same way to cut the query.

2.4 build index

I designed two class to build the index, *doc* and *wordindex*. The first class consists of the Hash value, frequence of the word appearing and URL. The second class consists of the word, its attribute and list of doc. The *dictionary* is the vector of the point to the *wordindex*. The core part is the "getin(doc x)" function of the class *wordindex*, which add the frequence of the *doc* or put a new *doc* in the list, as well as making sure the *doc*s in the list was sort by their Hash values.

2.5 search

First, I will cut the query into words. Then calculate the score of the doc which includes the target words using tf-idf algorithm, and sort these docs in a new class *scores* with their scores. all the *scores* will be saved in a vector called *content*. At last, sort the *content* by their scores and return the top n results according to the request.

3 kind of the query

The program could handle all kinds of queries. If there is not any words that appear in the query appearing in the doc, there will be no answer but the program will run well. However, all the query will be handled in the same way. There is not advanced search methods.

4 usage method

Before compiling the cpp codes, users must replace the file paths with their own paths in order to get the web pages and other contents in the local computer. Then compile the cpp codes with C++ 11 or above standard. The search_engine.cpp will provide a Command Line Tool for users to type the query. The test.cpp is similar with the search_engine.cpp but it will provide web server for automatically test.

5 Summary

From this practice, I have learned a lot about crawler and search engine. And get some important skills to debug. Through I don't use all the tools mentioned in the class, I am glad to complete the task in the end. However, I still have lots of problems to deal with after completing the main part.There are also lots of things need to learn after Practice.