COURSE PROJECT ANALYSIS REPORT

COMP 1406Z – FALL 2022

Aria Zhang

Instructions for running GUI

The controller is the SearchApp class, the view is the SearchAppView Class, and the model is other classes discussed below in Crawler Module and Search Module.

First please select the website you want to crawl and then enter the crawl button to crawl all the data needed in the search process. Then you can fill in the query we want to search, enter the boost button if we want the result to be boosted by PageRank, and then enter the Doogle button to execute the search.

Crawler Module

Overall Design

1) **WebPage Class**

   This class represents the web pages that have been crawled during the crawling process. It has attributes such as URL, the title of the page, index(each web page has a unique index), body(the body content in the page), and pageRank(the page rank of this web page).

   It also has tf, tfidf attributes both of which are HashMaps that store a single word of body content as key and tf/tfidf as value.

   After creating new objects of this class, I store the object in the file named after its index in the directory that is named after its index too.

   Though each web page also has its outgoing links and incoming links, I didn't set them as the attributes because compared with body content, a webpage may have much greater numbers of those links and it will make the object data hard to manage. So, I stored those links to separate txt files in the same directory with the object. This will allow me to access the data more efficiently later and will make my code more readable.

   I created two **constructors** for this class, one with no parameter, through which I can initiate a new web page. The other one is with url parameter, through which I can read the stored object file and get access to its attributes.

2) WebExtractor class

This class is to extract all required information from the web page, following calculations are based on the output information of this class, after executing this class, I'll no longer need to get any other information from any websites.

In this class, I created urlMap, urlArray, and uniquewords instance variables. urlMap, urlArray are to give every URL a unique id. The urlArray is to record the sequence of URLs since the map may not have a precise sequence of how URLs are added. And in the urlMap, the key is the URL, and the value is the sequence of the URL in the urlArray. When generating the search result, we need to output the title of each url, so I also created a titleArray whose sequence is the same as urlArray. I store the unique words that appeared in the body content into an ArrayList so that I can calculate each word's tf, idf, tfidf later, and can help me avoid duplicate calculations.

After creating objects of this class, the four variables will be stored in the memory and be easily accessed later.

crawl(String seedURL) Methods:

I append the seed into the urlMap and urlArray that I mentioned above and append the seed to a Queue named urlQueue which will be used to get the first URL in the list, parse it then remove it from the queue until the queue is empty. While urlQueue is not empty, I need to continue parsing the first URL inside it using the **WebRequester Class.**

During the whole crawling process, each time I crawl a new web page, I created a **webpage object** of the WebPage class. And then store the url link, title, index, and body attributes of the webpage object.

3) WebCalculator Class

After collecting all the basic information we need, we can start calculating the required parameter during the search process. In this class, we have the getPageRank, idFrequency, tfAndidf methods to calculate different indexes. After calculating, I read and get the previously stored webpage object and update its pagerank, tf, and tfidf attributes, then store it in the same directory again.

The reason why I choose to create a new calculator class for calculating instead of adding these methods into **WebExtractor class** is that it will be easier to see the whole calculating process and update it if possible. **WebExtractor class** has already crawled all information a WebPage could have. So if I want to get more parameters for each page, I can easily use the output information from **WebExtractor class** and add new method in the **WebCalculator Class** to calculate or analyze. Then the new parameter can be updated into **WebPage object** attributes.

4) WebReader Class

This class is responsible for reading the data that we extracted and stored in **WebExtractor class**, or calculating and stored in **WebCalculator class**.

In this class, I created two attributes: String url and **HashMap<String, WebPage> webPages**;

The **constructor** of this class is to get the url from the signature and check whether the **webPages HashMap** contains this url key. If the hashmap is null or doesn't contain this url key, I'll call the **webPages class constructor**(with the url parameter one) to create a webpage object with this url, read the local object file information, and add this whole object to the **webPages HashMap**.

Then when I want to read specific data of objects, like pagerank, tf, tfidf, I can get the object in the HashMap using it's url and then read the attributes of this object.

The advantage of this design is that after I read an object file, I don't have to read the local file again when I need other information associated with it because I have already stored its data in my HashMap, which saves my reading time as well as my search time a lot. In the beginning, my design is to read the local object file and then get its data every time I want some information. Compared with the previous design, the new design saved more than 10 times runtime for the search module.

And in **ProjectImplement Class**, I create a WebReader object to get the required webpage object in the webpages hashmap, then get the required data in the webpage's attributes. This design significantly improved the running speed of testers.

The **extractor-calculator-reader** design is quite clear for a crawler project and easy to extend. When I want to get more information from a website, like images, I'll revise the extractor class to get the image and find a way to store it. If I want to have a new way to analyze a website, like how the sequence of words appeared in a website affects its score, then I can add a new method in the calculator class using the information from the **Extractor Class**. Then if I want to read the new data I created, I can update the reader class.

Besides, **taking a webpage as an object** is also clearer than store separated information in different files, especially when the number of webpages is huge. In the search module, when we want to calculate the score of a webpage, we need different information about this webpage, like title, tfidf, pageRank. If we store information in separate files, we need to repeat the open file - read data - close file process several times, which is time-consuming, and also it will decrease the readability of our code.

5) WebCrawler class

This class represents the main web crawler controller. The class controls the above classes to crawl a given URL, extract links from the page, calculate the required index and store the page information. So that when I want to add a new function to the crawler program, all I need to

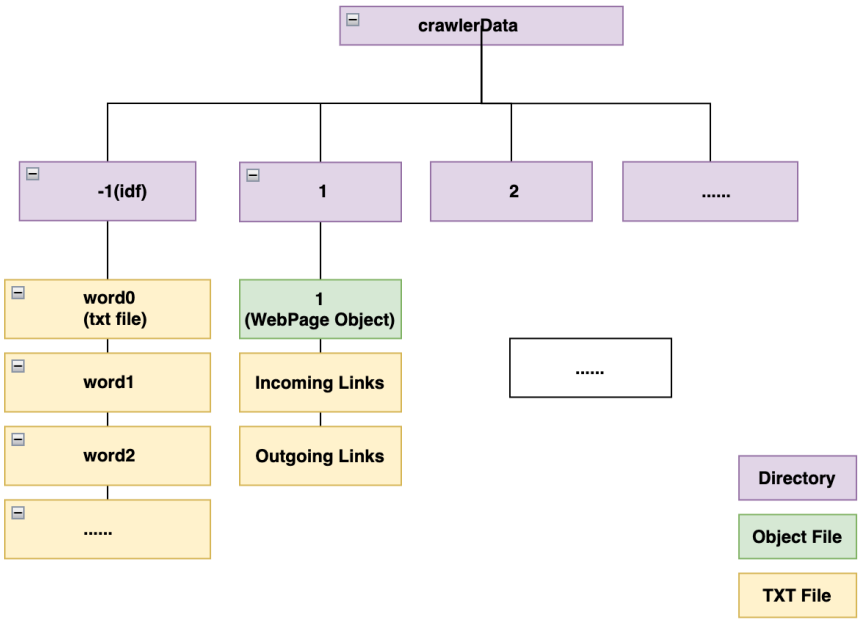
do is to create the function itself in a separate class, and then use this class to control executing the new function, which will make my project easier to be extended.

6) **WebFinder Class** and **WebFileEditor Class**
WebFinder Class is a helper of WebExtractor class in which there is findTitle, findURL, and findBody method. These methods are used during the extractor process.

The reason I created a separate class for these methods instead of putting them into the WebExtractor class is that when I want to find more information on a webpage, it will be easier to write a new method in this finder class, and it will also be clearer to see what information we can get from the current crawling program.

WebFileEditor Class is also a helper class to create/read text or object files. Dividing these methods into a new class will make my code easier to read.

File Structure



Search Module

Overall Design

1) SearchResultImp Class

This class takes title and score as attributes and implements a compareTo method to sort SearchResult Array.

2) SearchQuery Class

This class represents every query that has been searched. It has attributes such as words, searchPhraseList (put every word in the query sentence into a list), searchPhraseSet(remove duplicate words in the query), and size(the size of searchPhraseList). Then I create getTFIDF() method to calculate each word's tfidf regarding the query words.

The reason why I want each query to be an object is that first, I can calculate each word's tfidf inside the query class, which will make my code more readable. If I want to analyze the query itself later, for example, if the query is very long, we can delete unimportant words, and I can add different methods into this class to accomplish it.

3) SearchIndexer Class

This class is to initiatively get the cosine similarity of each web page.

In this class, I created a **SearchQuery** object and then get the tfidf of each word in the query. Then I use URLCosineSimilary() method to calculate the cosine similarity of each url that has been crawled before. In this method, I create **WebReader Object** for each url in the urlArray static variable and then get its tfidf and pagerank. The calculation sequence is the same as the urlArray variable I created in **WebExtractor Class**, so I simply output an array to store the cosine similarity instead of using a HashMap.

This class is the interjection between **SearchQuery Object** with **WebReader Object**. This is another reason why I put the cosineSimilarity calculation into an independent class. The calculation itself can be changed to generate different results when the query and crawler remain the same. Or I can keep the calculation remains the same but change the other two objects. It will be clearer to see how the result is generated.

4) SearchEngine Class

In this class, I created a searchIndexer object to execute the calculation, after getting the cosine similarity array, I assign the title and cosinesimilarity to the **SearchResultImp object** and add the searchResultImp object to an array. Then I sort the array using the compareTo method in the searchResultImp class and output the first X result.

The reason why I don't want to use the Comparable interface in WebPage Class is that a **WebPage object** has more attributes, when adding a WebPage object into an array and then sorting the array will take up more memory space.

Similar to the Crawler Module, I want to have a structure that different classes in the Search Module take different, independent responsibilities so that the code is easy to be extended. In this case, when I want to change the way of parsing the query sentence, it's easier to revise the code in the **SearchQuery Class**. When I want to change the way to calculate the score of each web page or add more limits to this calculation, I only need to revise the **SearchIndexer Class**. And then I use searchEngine Class to output the result I want. If I want to have a different format of the output, I just need to change the SearchResultImp class, and then revise the SearchEngine Class. Other classes can remain unchanged.

I planned to add a function in the SearchIndexer Class like the WebReader Class, that is, create a SearchQuery HashMap as well as a SearchQuery ArrayList, when executing a new search, I add the new SearchQuery into the HashMap and ArrayList and control their size to be a fixed number. When the size exceeds the fixed number, I'll delete the front element of the HashMap and the ArrayList. After adding the function, I can improve the efficiency when the user wants to search the same content, and I can analyze the similarity between the user's different searches using this HashMap and then optimize the search results. However, this function is unrelated to the course project specification and the code is too long, so I deleted it.