



Java Web Crawler Analysis

PREPARED BY

Jason Wang

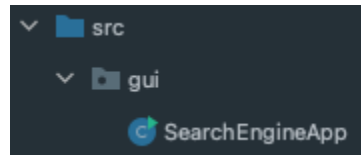
December 11, 2022

Table of Contents

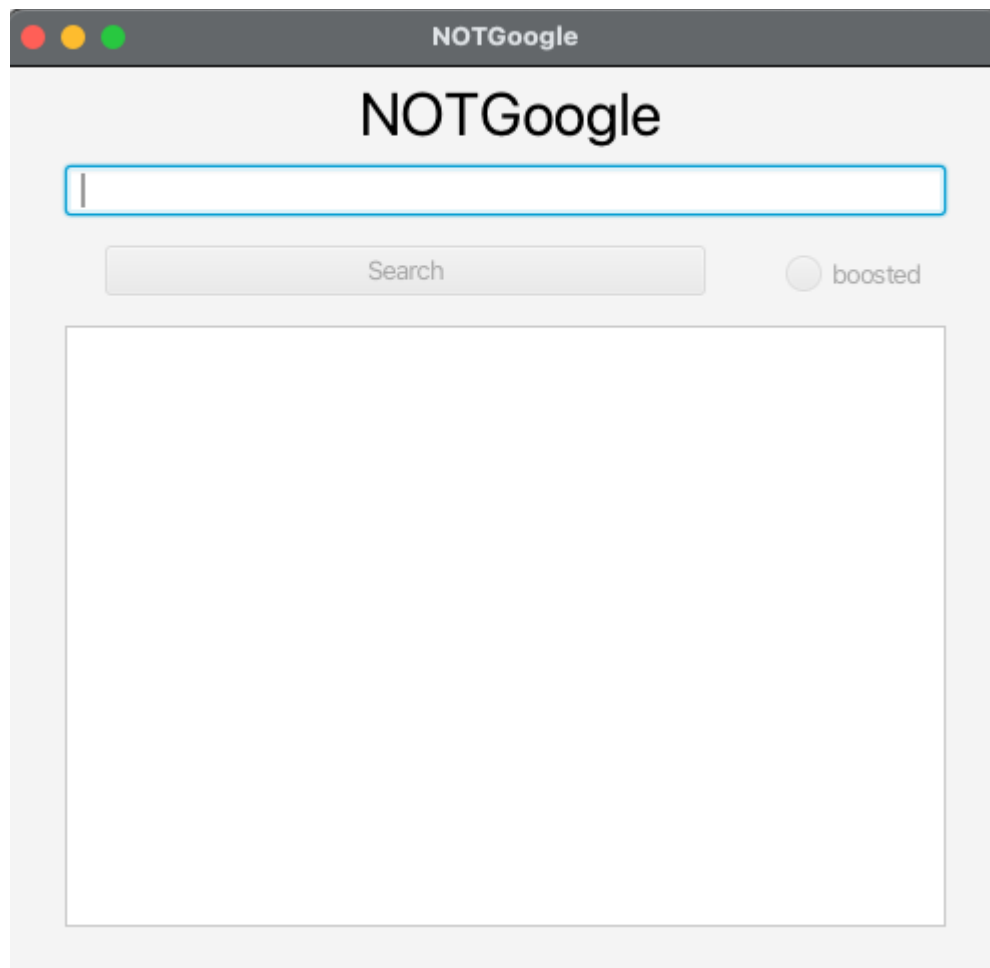
Matrix Multiplication	6
Crawler	6
Webpage	7
CrawlerAnalysis	8
Savable (Abstract Class)	9
Readable (Abstract Class)	11
SearchData	12
Search	12
Webpage Result	13
SearchEngine	14

GUI Instructions

The controller portion of the GUI can be found under `src/gui/SearchEngineApp.java`. The JavaFX files need to be imported manually before running this java file.



Once you run this file, the GUI window shown below will pop up.



To search, simply type the query within the search bar. Choose the option with the boosted radio button, then click search. The ListView on the bottom will display the top 10 web pages with their titles and their according page rank scores.

In order to use the GUI, a crawl needs to run first, which can be accomplished by running any of the test files under src/testFiles.

Note that if any additional test files are added to the testFiles packages, the two lines of code below need to be added to the top. The instance of the project tester would also need to be modified to SearchEngine:

```
package testFiles;
```

```
import model.*;
```

Functionality Completion

All functionality within the project specification has been implemented.

* Note that there is one failed test in the search of fruits3 tester because there is a rounding error of 0.0001.

Also another quick note. My crawler may run a lot slower because I decided to decrease the euclidean distance before stopping the multiplication of the vector and adjacency matrix, resulting in more accurate results to pass the tests with less failure.

File Structure

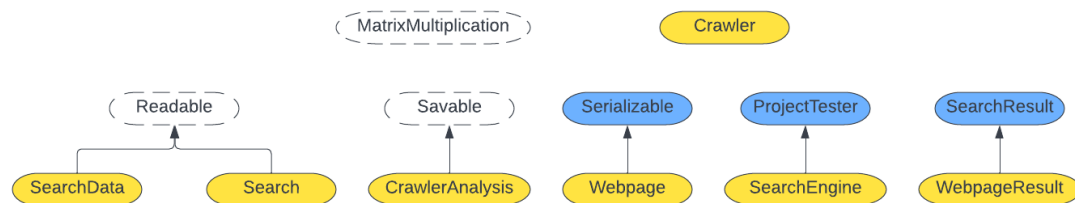
- resources
 - ldf - file
 - modified URL 1 - directory
 - externalLinks - file
 - pageRank - file
 - referenceLinks - file
 - tf - file
 - tfidf - file
 - title - file
 - modified URL 2 - directory
 - externalLinks - file
 - pageRank - file
 - referenceLinks - file
 - tf - file
 - tfidf - file

- title - file
- modified URL x - directory
 - externalLinks - file
 - pageRank - file
 - referenceLinks - file
 - tf - file
 - tfidf - file
 - title - file



The file system that I have used for this project is identical to that one that me and my partner implemented within the COMP1405 project. My original plan for the file storage system for this project was to store each webpage instance directly to file, but reading that from file would be wasting unnecessary RAM. In my opinion, the file system that I have implemented does not waste too much unnecessary RAM when reading from a file, and is able to read specific values from a list of data very efficiently, which is why I have gone with this file storage convention.


Class Outline

To introduce my implementation of the class hierarchy, I have created the UML below to give an idea of how my classes work together. Throughout my program, I have privatized all attributes of classes, which is following encapsulation of the OOP paradigm.



Click on the image to get zoomed version

	This represents Classes
	This represents Interfaces

	This represents Abstract Classes
---	----------------------------------

Matrix Multiplication

Analysis

This class can be seen as a basic toolbox that is identical to `matmult.py` that we implemented in COMP1405. I have chosen to make all of the methods within this class static so I can call the methods directly without instantiation.

Runtime

Method	Runtime
<code>scalarMultiplication</code>	$O(n^2)$
<code>multiplyMatrix</code>	$O(n^3)$
<code>euclideanDistance</code>	$O(n)$

The high runtimes of these methods cannot be avoided as the entire vector or matrices need to be iterated to perform the computation required.

Crawler

Analysis

In terms of the entire crawling process of the project, I decided to break it up into two parts. This is because the crawling process has two major parts; the data retrieval and the calculation. The class here is responsible for data retrieval only. This is an example of encapsulation as I am grouping different functionality and encapsulating them into different classes. This allows for reusability of my code. For example, if there is a more efficient method of retrieving the HTML from each webpage, the data retrieval class (`Crawler.java`) can be changed, while the data calculation (`CrawlerAnalysis.java`) can still be used without modification as long as the basic attributes and behaviours of the data retrieval stays the same.

In this class, I also created a generic method called `searchHtml` that can retrieve the data between two given tags. This allows for extensible code. For example, if we wanted data

within list tags of each web page, a new method can simply be added to call this searchHtml and get the data needed, then the new method can be called within the crawl method along with the other method calls.

I have also written modular code in this class as each one of the helper methods encapsulates a specific function, while the crawl method uses all of those building blocks to retrieve data from each unique web page.

All of the helper methods within this class are private to invoke abstraction of this class. This means that CrawlerAnalysis – which utilises this class – does not know the implementation of this class, creating loosely coupled code.

Runtime

Method	Runtime	Analysis
getHtml	$O(1)$	N/A
readHtml	$O(n^2)$	This method needs to iterate through the entire page's html and uses indexOf within a while loop, so $O(n^2)$ is the most efficient way possible.
addPageTtile	$O(n^2)$	Calls readHtml.
getWebData	$O(n^2)$	Calls readHtml.
getReferenceLinks	$O(n^2)$	This does not implement the readHtml method because it reads an attribute within anchor tags, which readHtml would not work. Though it iterates through the entire web page's html like readHtml.
crawl	$O(n^3)$	This method calls all of the previously mentioned methods, and on top of that, it needs to recursively call itself for each new unique webpage found, making a runtime of $O(n^3)$.

Webpage

Analysis

This class is essentially a storage container storing each unique webpage as an instance that contains the pages calculated values and other data such as words and html content. I am essentially encapsulating all of the different data that need to be stored within instances of this class.

Additionally, this class implements the Serializable interface because it is essentially a data container, so it is fitting to save the object directly to file (though I have not implemented this because it would take up additional memory space when reading from file), which enables extensibility; that is, if someone prefers or sees benefit in writing this object directly to file, they can implement that within the Savable and Readable classes mentioned later.

This method also overrides the equals method because Java's default implementation compares the instances based on their location in memory. But two Webpage instances may have the same url, which would result in equivalent instances within the HashMap that I have implemented within the Crawl class.

Even though this class is a storage container, all attributes are still private and accessed through setters and getters to maintain the rule of encapsulation.

Runtime

Runtime analysis is redundant for this class.

CrawlerAnalysis

Analysis

As previously mentioned within the Crawler class, the crawling process is defined by the data retrieval portion and the calculation portion. This class is responsible for the calculation of the crawl process. The structure of this class is very similar to that of the Crawl class, meaning that it is modular with different functions encapsulated within its own method, and finally called in one method, which is analysis.

In this class, I have a very clear use of abstraction, which means that the Crawler and CrawlerAnalysis classes don't know the implementation details for each other. This makes for reusable code as changing implementation details of either class will not break the other.

Most of the classes within this method calculate the value for a singular web page. In this way, new methods that may calculate for something else can be added very easily for a singular webpage (though getPageRank is an exception because it requires all web pages). It can then be added to the for loop within the analysis method, which will do the calculation for each web page. This makes my code more extensible.

This class is also a child class of the abstract class Savable, allowing the ability to save data to file. More about my reasoning for implementing it this way will be mentioned there. But as seen in the analysis method, save and saveList methods are used here in order to save the idf HashMap and the attributes of each Webpage instance to file.

Overall with the use of encapsulation and abstraction, I have enabled the crawler portion of my program to be loosely coupled, extensible, and re-usable.

Runtime

Method	Runtime	Analysis
analysis	$O(n^3)$	Calls all of the helper methods below, and saves each webpage and idf values to file.
getExternalLinks	$O(n)$	Though this method can be avoided by doing this computation within the Crawler class, that is not separating the concerns of each class. So in other words, I am sacrificing computation to make my code more organised.
getPageTf	$O(n)$	Iterates through each word within the Webpage instance argument, and calculates the tf value.
getIdf	$O(n)$	Iterates through each webpage, checking the number of times an argument word appears, then calculates the idf value for the argument word.
getTfIdf	$O(n)$	Iterates through each word within the Webpage instance argument, and calculates the tf-idf value.
getPageRank	$O(n^3)$	Since the page rank calculation process utilises every web page, this functionality cannot be broken down to calculate for each web page individually. The runtime and implementation detail is the same as the COMP1405 project, so naturally the runtime is equivalent.

Savable (Abstract Class)

Analysis

The name of this class is quite self-explanatory and seeing its implementation, it can be easily converted as an interface over an abstract class. The reason why I have gone with an abstract class over an interface is because I find that this enables for less duplication of code. A lot of the implementations within the Savable and Readable classes are simply reading or saving lists and values into values. So creating abstract classes means that I am making reusable code. If someone decided to create a class that may save lists to files, they can simply extend this class.

This class is of course abstract because there is no reason to ever create an instance of Savable, so it is deemed as abstract.

In this class, I have used the protected keyword for some of the methods, and that is because the functionalities for saving to file do not need to be revealed in the instances of the child classes, which is a case of abstraction. They are also not private because the child classes need to access these methods.

Throughout this entire method, I have provided fileName and path as string arguments which could have been removed to implement a concrete directory path within this method. But the way that I have implemented these arguments enables code reusability. If someone wants to save additional data to a file other than the concrete ones specified within the save method, they can decide the directory path and the name for the file themselves.

Overall, with the use of inheritance, I am able to reduce code repetition in the child classes, and create reusable code between my crawler and file management.

Runtime

Method	Runtime	Analysis
save	$O(n^2)$	The save class utilises the 3 methods below in order to save all Webpage instance attributes to file, but since all of the other methods are also protected, if someone decides to create a different implementation of the crawler calculation class, they can choose to call this generic save method, or to manually save the contents using the 3 other methods below.
saveValue	$O(1)$	Since only 1 double value is being written to file,

		this operation can be considered as $O(1)$.
saveString	$O(1)$	Since only 1 String value is being written to file, this operation can be considered as $O(1)$.
saveList	$O(n)$	<p>Here an entire object (HashMap) is being written to file. The overridden method is identical, except one is slightly modified to write the idf values within the correct directory location. Rather than writing all of the values using DataOutputStream to a text file. This implementation of directly writing the HashMap is more efficient as I will have $O(1)$ look-up times when obtaining the value being searched. Here the n would represent the bytes of data that the HashMap is broken down into and written to file.</p> <p>Note that the implementation of this method makes it so that any object can be written to file, but I have purposely done this to avoid code repetition (implementation for writing HashSet and HashMap are essentially the same).</p>
resetDirectory	$O(n^2)$	Call the two methods below to reset all of the cached files from the previous crawl.
makeDirectory	$O(1)$	Creates a new directory based on the File argument being passed.
deleteDirectory	$O(n^2)$	This method deletes an entire directory by recursively iterating through all content within the directory, and deleting it. In my use of this method, the runtime is $O(n^2)$ because there is one layer of folders within the base resources folder, but this can vary depending on the number of nested directories/folders.

Readable (Abstract Class)

Analysis

As previously mentioned, there is a reason why this class is made abstract. This class is very similar to the Savable class, except it provides the ability to read data from files. Most

of the reasoning for my implementation are the same as those mentioned in the Savable class.

Runtime

Most of the $O(n)$ runtime throughout these methods are due to the fact that the url argument needs to be modified using the `String.replace()` method, but since urls generally have an upper-bound, they can also be considered as an $O(1)$ operation as well.

Method	Runtime	Analysis
<code>readHashSet</code>	$O(n)$	A generic method can be used to read objects in general along with the method below, but I have created these specific methods to avoid casting errors if someone using my code decides to use these methods. This is unlike the <code>readList</code> method in the previous class because here it can cause errors.
<code>readHashMap</code>	$O(n)$	Same implementation as previous method, except it is designed for HashMaps.
<code>readValue</code>	$O(n)$	Reads a singular file from the specified argument directory path and file name.
<code>readString</code>	$O(n)$	Reads a singular string from the specified argument directory path and file name.
<code>readIdfList</code>	$O(n)$	The implementation is the equivalent to that of the <code>readHashMap</code> , but it is specifically designed to read the idf HashMap from file because it is stored within a different location within the resources folder.

SearchData

Analysis

This class is equivalent to the `searchdata.py` file specification that was implemented in the COMP1405 project. This means that this class could be an abstract class with static methods, but because I wanted to avoid using methods, I decided to implement the `SearchData` as a concrete class.

Runtime

All of the methods within this class simply call the methods from the parent class (Readable), so they all have equivalent runtime complexity to the parent method being called.

Search

Analysis

Once again, this class is equivalent to the search.py file specification that was implemented in the COMP1405 project. All of the implementations have been simply translated from Python, except methods have been encapsulated to a portion of computation needed to search the cached data. As seen in my code, there is a clear definition of each method with their role. Modularity can also be seen as all of the helper methods are called within the search method for the instance to call in order to search a query. The sorting method has also been slightly altered from the previous merge sort that me and my partner used. Instead, I decided to use selection sort with the specifications for sorting the search results for this project (Note that since most tests take the top 10 score, the runtime efficiency is about the same for all of the large fruits web page sets – $10 \cdot 1000$ (selection sort) $\approx 1000 \log_2(1000)$ (merge sort)).

Within the search method, you can see that I have returned a List of SearchResult rather than WebpageResult, which is an instance of polymorphism. Since my WebpageResult storage class simply extends the attributes required within the SearchResult interface, the instances of WebpageResults can be interpreted as SearchResults when it is implicitly casted when added to the list of search results.

Runtime

Method	Runtime	Analysis
selectionSort	$O(n^2)$	This selection sort has been altered to sort only the top x argument search results so that runtime is not wasted.
search	$O(n)$	Utilises all of the methods within this class to return a top x of SearchResult instances. This method also checks to ensure that the x argument is valid.
getQueryVector	$O(n)$	Based on the query string provided within the

		argument, it will iterate through each word within the query ($O(n)$) and calculates the query vector of the equation, and determine the order of the vector (aka placing all the words in an arraylist in order). Part of the denominator is also calculated within this method.
getPageScores	$O(n^2)$	After the query vector has been established, the cosine similarity of each page will then be calculated using the query vector obtained from the previous method, and with the vector order, calculate the vector score of each page.
reset	$O(1)$	N/A

Webpage Result

Analysis

This is somewhat similar to the Webpage class, in that it acts as a storage container. Instances of this class implements the SearchResult interface and contains the required attributes to act as an appropriate storage container for a search result.

Runtime

Runtime analysis is redundant for this class.

SearchEngine

Analysis

This class simply implements the ProjectTester that was provided for the project. Each method in this class does exactly what the specification requires within the ProjectTester interface.

Runtime

Runtime analysis is redundant for this class.