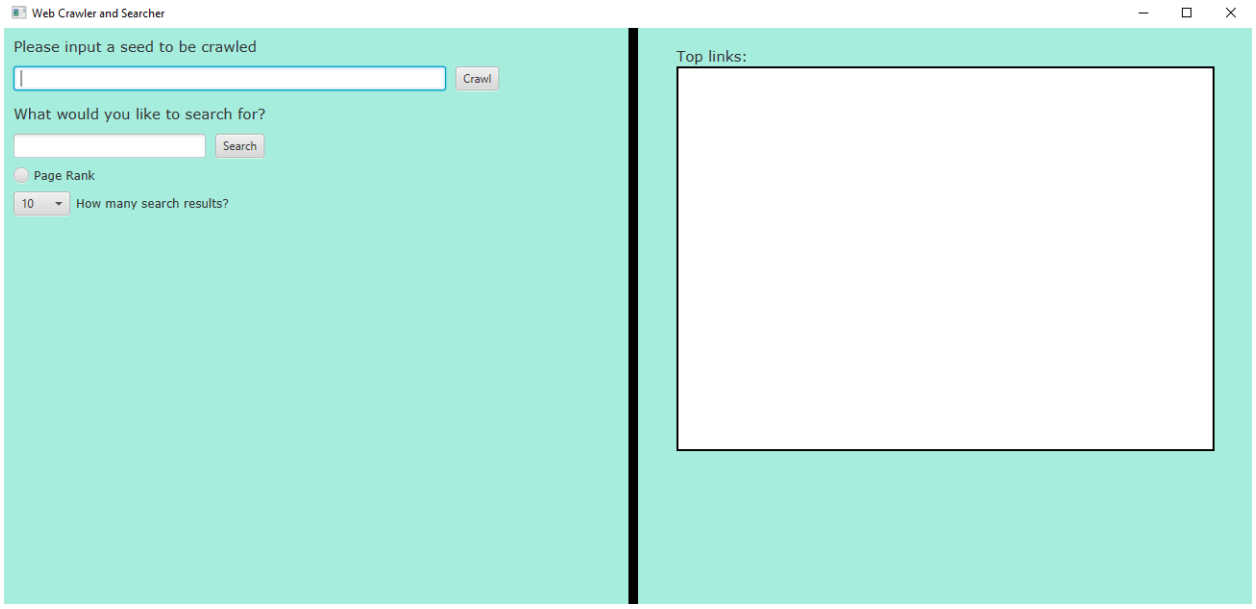
Final Project Report

By: Jonathan Pasco-Arnone

A)

GUI Instructions:

I used JavaFX to make my GUI which can be run through my “Controller” class. Once the program has started, you will be greeted with the page seen below.



You can clearly see the input locations of the link to be crawled as well as the input location of the query to be searched. Additionally, there are labeled buttons and a combo box that has a clear intended purpose.

B)
Functionality:

All functionality required in the course project outline is included in my project.

Working functions from the project tester interface:

- initialize
- crawl
- getOutgoingLinks
- getIncomingLinks
- getPageRank
- getIDF
- getTF
- getTFIDF
- search
- (more functions are added and work as intended, but were not required by the outline)

C)
Class Outline:

Controller:
The controller class represents the controller portion of the MVC paradigm. That is to say, this class connects the model and view classes. Additionally, it acts as the starting point/main class of the project that boots up the GUI and allows for its functionalities to be fully utilized. This is done through the inheritance of the application class that allows the program to receive information from events such as button clicks or screen size changes. When a button is clicked, the controller class will detect an event when it occurs, and execute its code. Moreover, when the width of the screen is changed the controller class will update the objects on the screen to ensure they are still organized correctly.

Crawler:
Performs the initialize and crawl functions. Takes a seed URL and crawls all related links, generating an organized data structure of folders to be used in future functions. During this crawl, a hashtable for the link folder locations and the number of documents a word appears in, along with an array for the page ranks, is serialized in a separate folder from the crawl data. Storing all the data from both the crawl and serialized information is made convenient by inheriting the abstract file control class.

File Control:
This abstract class is inherited by multiple classes so that every class that requires it has access to the read, write, serialize, and deserialize functions. Additionally, this class includes two constant strings representing the “crawl path” and “parsed path” (ie. the string representation of the path to both of these commonly used locations).

All Fruits:
All the fruit testers and test results from the downloaded testing resources.

Model:
The model class is the model piece of the MVC paradigm. This class glues the functionality of all the working classes together into a convenient and easy-to-understand class by receiving input from the controller, passing it into the designated class, then receiving the return value and returns it back to the controller. This class implements the project tester interface and inherits the properties from the abstract file control class.

Page View:
The page view class acts as the view part of the MVC paradigm. This class holds all the labels, buttons, text fields, etc. Furthermore, this class has update functions for when the screen width is adjusted or a search is completed. This class inherits the pane class to allow adding nodes to the main pane.

Project Tester:
The interface was downloaded from the project materials.

Search Data:
Another function integrated with the model. This class performs most of the methods outlined in the project tester interface. Inherits the abstract file control class that is used in some form during every function call.

Search Engine:
The search engine class is the last class that is used in the model. This file is responsible for the search functionality of the program. Through a complex algorithm covered in COMP 1405, this class is able to produce a list of the top X search results for a given query. This class also inherits from the file control superclass.

Search Result:
The interface was downloaded from the project materials.

Single Search Result:
This class is used primarily in the search method to serve as an object variable representing each weblink in a list. This class inherits properties from the previously mentioned search result class.

Testing Tools:
A class that was downloaded from the project materials.

Web Requester:
A class that was downloaded from the project materials, which was then used to grab the data of the HTML file representing the website.

D)
Overall Design:

The design of this project was crucial, both because of its magnitude, in comparison to any other assignment I have completed in the past, and because of the organizational requirements to conform to OOP (object-oriented programming). Some of the design was inspired by my previous final project in COMP 1405. However, most of the sections of code that I used from this past project were either altered to conform to OOP or improved in some way. The overall design of this project can be best described in the OOP principles, file organization, and efforts to improve efficiency.

One of the main challenges in designing this program was making the original code into an object-oriented application. Following the MVC paradigm, my classes use encapsulation to hide the internal details of how the crawl data is generated, stored, and used. The model class is an excellent example of this and provides an incredible level of robustness to the program. Furthermore, all the lowest-level computing classes demonstrate great modularity. Another concept of OOP that I have used is abstraction. In the “FileControl” class I have outlined specific fields and methods that are inherited for use in all of my subclasses. Abstraction is an approach that I have used to increase the extensibility of this project. Overall, OOP has made my project much easier to read and far more organized without reducing runtime excessively (will actually run faster than my Python version in some cases).

Another important part of the design was the file structure of all the data. Since the search function was required to work regardless of whether a crawl was run in the same execution, I was required to store the crawl data in files to be later accessed whenever. The structure consisted of two main folders, crawl data, which held all the relevant parsed information from the crawl, and parsed data which held all the serialized variables. I organized it this way because having all the important crawl information readily available for any class to access at any time makes coding much simpler.

Lastly, there were many steps I took to improve this project's efficiency. One example of this is my serialization of variables. I chose to do this because it made the programs outside of the crawler (where all the serialization occurs)

able to directly access the variables. The serialized variable representing a hash table of links as keys and the folder number as their value allowed for drastically improved runtime; instead of searching through 1000 folders, the code would only search one. Moreover, with this same variable, I had it passed to each working class on initialization, through their constructors, so that they would not have to deserialize the hash table every time. Another example of improved efficiency was having an abstract class for the majority of file manipulation. This class was able to handle the vast majority of error handling so I was able to simply call “writeFile()” and not have to worry.

To conclude, this project and its design were heavily influenced by OOP principles. Leveraging the MVC paradigm, I was able to create a program that is heavily modular and robust. The OOP principles enhanced the readability and even the runtime performance in some cases. Furthermore, the file structure, emphasizing the separation between the crawler's data and the parsed data, further improved the efficiency of the code. Finally, the improvements to the code resulted in reliable maintainability through reused error handling. Overall, the design was an imperative component in making this project and provided a lighter load of work in writing the program.

Additional improvements (most are minor as the main ones were already mentioned):

Improvment	**Advantage**	**Disadvantage**
Constant variables representing the string path value of the crawl path and parsed path	- Will not accidentally misspell the path - If the path changes I only have to change the string once	- Classes outside of the hierarchy cannot access these constants
Update methods of view class	- Allow for the table to be loaded or screen organization to be changed	

Sorting the search results manually	- Conforms to the project tester interface that requests a list of another provided interface that I am not allowed to edit	- Does not use a custom compareTo() function to make code look less messy
Separated crawl function into multiple separate functions	- More visually appealing - Organized - Decreased code complexity	