COMP 1406Z – Fall 2023
Course Project: Analysis Report

Kassem Taha & Barnabé Frimaudeau

Instructions

1. Starting the Program:
    - Open the program by running "GUI.java" file with JavaFX.
2. Using Existing Crawl Data or Crawling New Data:
    - When you start, the program will try to use any info it already has in the "resources" folder.
    - If there's not enough or no data, it will ask you to give it a new website link to gather data. Put the link in the box and press the "crawl" button.
3. Searching for Information:
    - If it has the data, type what you want to search for in the search box.
    - Click the "search" button to see 10 results related to your search.
    - To boost, Click the "page rank" checkbox first, then press "search." This will give you results with the pagerank boost applied.

Functionality

Functionality	*Complete / Incomplete*
Scrape through all interlinked web pages	**COMPLETE**
Store necessary data in directories and files	**COMPLETE**
Read all the necessary data in a reasonable amount of time	**COMPLETE**
Correctly calculate the search results asked by the user and sort them in a reasonable amount of time	**COMPLETE**
Display all of the results using a neat GUI	**COMPLETE**
Added a crawl button for extra functionality	**COMPLETE**
Created somewhat robust error-handling	**COMPLETE**

Classes Outline

Classes

OsUtil
This is our file handling class. It helps us create, read, and append files without having to worry about every detail. It helps with code readability, efficiency, and reuse.

Crawler
The crawler class does most of the analyzing and computation in our program. It is responsible for gathering all the data we need for searching. It includes many variables which are only used in this class and can efficiently crawl 1000 pages. This has very complicated code and it's best kept in its own class.

ProjectTesterImp/Searcher
This class utilizes all the crawling data and is our de facto main class. In this class, we read the files we stored our data in, calculate search score, and initialize everything in the program.

SearchResult
This is a custom data type we made to house our SearchResult. It has two main properties: title, and score which we use to output and sort our search results. It has a third property url which is used in case you want to display url with title and score. It is currently unimplemented.

Controller
This class is part of the Model-View-Controller architecture pattern and it represents the controller component. Its responsibility is to handle the logic between the graphical user interface and the model (crawlMain & SearchResult).

GUI
This class is also part of the MVC architecture, but this one represents the view component. Its responsibility is to display information to the screen in the form of a window. It takes in String input (crawl seed and search query), user button clicks (crawl and search), and a boolean checkbox for the usage of PageRank boost. It then displays the search results in a vertical list view, which lists the top 10 results from most to least relevant.

MathHelper
This class is responsible for handling any complex math operations not found in the base Math library found in Java. Includes methods like matrix multiplication and euclidean distance to help with calculating pageranks. Supports further development in case we need more complicated math help.

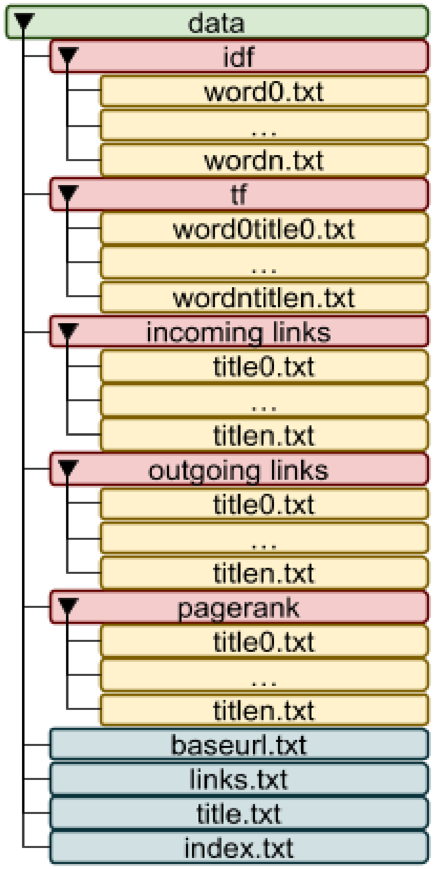
Others

WebRequester
This is the base class given to us to acquire the string we parsed in crawler.

ProjectTester
The interface provided to us which ProjectTesterImp/Searcher implements.

TestingTools
Testing tools provided by you to help with testing. Kept here for your assistance.

Overall Design



During the programming phase of this iteration of the project, our primary focus was on transitioning our existing codebase. As a result, our class structure closely resembles that of the 1405 version. This similarity is particularly evident in our file structure, which remains largely unchanged, with minimal changes made to naming conventions. We opted to keep our previous file structure as it best suited our requirements and was the least complex arrangement we had in mind.

The osutil class was an important abstraction we developed. Given the difficulty involved in file manipulation, such as adding, deleting, and appending files, we found it counterproductive to directly interact with files at all times. Hence, the Osutil class was the initial class we created from scratch to help with the next classes like Crawler.

The Crawler class parses and organizes data before handing it over to the Osutil class for storage. The Crawler performs a lot of computations and requires many variables, prompting us to make it its own class. Through encapsulation, we do not worry about the larger number of lists, arrays, dictionaries, and other variables generated during its processes. This encapsulation prevents the need to worry about the many variables within our main class, enhancing code efficiency and readability.

This leads us to our "main" class Searcher/ProjectTesterImp. In this class, we capitalized on inheritance and implemented the provided ProjectTester interface. By doing so, we created a “main class” that effectively utilizes both the crawler and osutil to initiate the crawling process and subsequently read the obtained files. The ProjectTester interface proves beneficial as it amalgamates the functionalities of Search and SearchData from the 1405 version.

With OOP in this iteration of our project, every time we want to make the crawler parse more data from other HTML tags we simply just alter the crawler function without having to worry about the other classes. If we want to calculate the IDF*TF*Search Score we do not need to alter the crawler class, we can simply read the data we already have. If we want to add more utility with files we simply add a new method to Osutil.

For the graphical user interface, we used the pane element from JavaFX. This lets us move and resize elements to our liking for a clear and concise user interface. With MVC paradigm in mind, the GUI was split into two different files, one for the controller and one for the view. Thanks to abstraction, the process of linking the GUI to the model was simple. After a bit of tweaking in the model and controller, we implemented error checking for the seed input when using the crawler.

In conclusion, our approach to this project centered on using the working structure from our previous iteration (1405 version) while making necessary adaptations to use new concepts like OOP and create new features like the GUI. By applying modularity through class structures like the Osutil and crawler classes, we achieved significant improvements in code organization, readability, and efficiency.