COMP 1406Z- Fall 2022

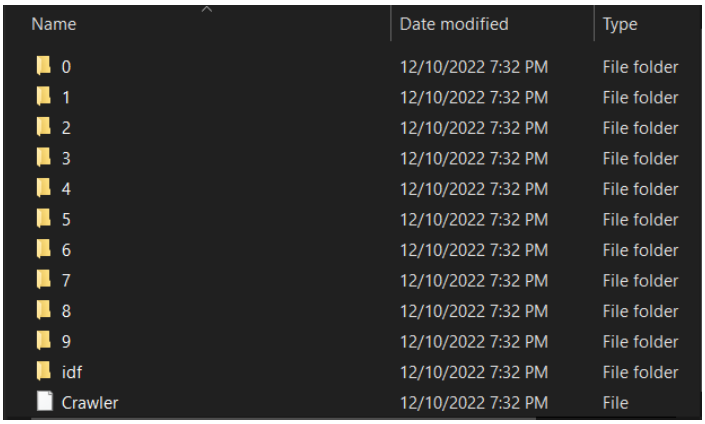
Course Project: Analysis Report

Kevin Lai

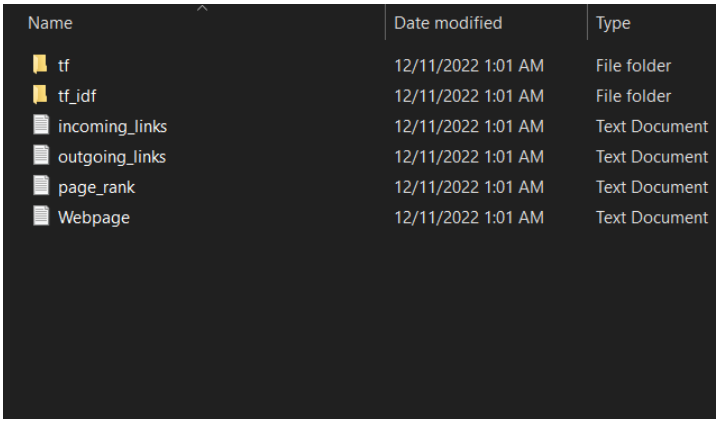
Crawler Class (Crawler.java)

I designed my Crawler class to perform most of the data processing and store them into text files to be accessed from other classes (SearchData, Search). The Crawler class has one attribute urlToNum which maps URLs to indexes (HashMap). My Crawler Class will crawl through all URL pages, each page will only be read once, and the title,href links will be saved as attributes in a URL Object. While the words contained in the paragraph tags will be saved in an ArrayList. Each webpage (Webpage Object) is saved in a directory labeled by indexes starting from 0. This works well because all URLs are mapped to an index (diagram A below), all idf calculations are also stored in a directory within the crawl_data directory. A Crawler object is created so the urlToNum attribute can be accessed when running the GUI since the GUI is performing searches on existing crawled data. The outgoing_links, incoming_links and title are all saved as attributes of the Webpage Object (Diagram B below). This makes it easier to access any of the data when retrieving it by URL for the Search Data class.

File Structure (Diagram A)

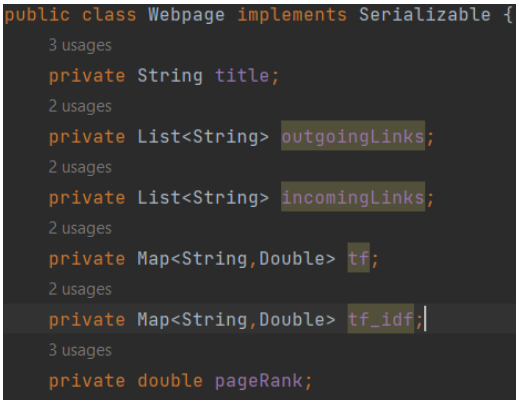


Name	Date modified	Type
0	12/10/2022 7:32 PM	File folder
1	12/10/2022 7:32 PM	File folder
2	12/10/2022 7:32 PM	File folder
3	12/10/2022 7:32 PM	File folder
4	12/10/2022 7:32 PM	File folder
5	12/10/2022 7:32 PM	File folder
6	12/10/2022 7:32 PM	File folder
7	12/10/2022 7:32 PM	File folder
8	12/10/2022 7:32 PM	File folder
9	12/10/2022 7:32 PM	File folder
idf	12/10/2022 7:32 PM	File folder
Crawler	12/10/2022 7:32 PM	File

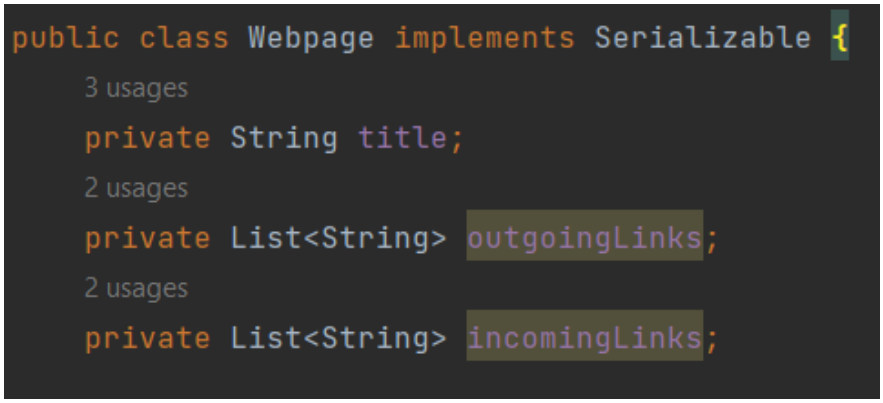


Name	Date modified	Type
tf	12/11/2022 1:01 AM	File folder
tf_idf	12/11/2022 1:01 AM	File folder
incoming_links	12/11/2022 1:01 AM	Text Document
outgoing_links	12/11/2022 1:01 AM	Text Document
page_rank	12/11/2022 1:01 AM	Text Document
Webpage	12/11/2022 1:01 AM	Text Document

File Structure (Diagram B) Version 1 and updated version on the right



```
public class Webpage implements Serializable {
    3 usages
    private String title;
    2 usages
    private List<String> outgoingLinks;
    2 usages
    private List<String> incomingLinks;
    2 usages
    private Map<String,Double> tf;
    2 usages
    private Map<String,Double> tf_idf;
    3 usages
    private double pageRank;
```



```
public class Webpage implements Serializable {
    3 usages
    private String title;
    2 usages
    private List<String> outgoingLinks;
    2 usages
    private List<String> incomingLinks;
```

SearchData Class (Search Data.java)

This Class is responsible for fetching all the data needed for the search engine. Since most of the data processing was done in the Crawler class, I was able to reduce the time complexity of my Search Data Methods. The Methods retrieve data by opening a file path and reading a text file.

Search Class (Search.Java)

This Class is responsible for answering search queries by utilizing the Methods in the Search Data class to rank the top X searches based on the query from highest to lowest. If the similarity scores are the same (rounded to 3 decimal places) then the comparator will sort the pages based on lexicographical order. I mainly worked on optimizing my search methods to ensure that I minimized the time complexity. All the data that are used to calculate the top X most relevant pages are stored in text files from the crawl process and retrieved from the Search Data Class. This class is the model part of our MVC application. It deals with calculating the cosine similarity that ranks our webpages and returning the data back to the controller which will return it back to the view.

SearchApplication Class (SearchApplication.java)

This class is the controller part of our MVC Application, it deals with the application logic and acts as the coordinator between the view (SearchView.java) and model (Search.java). This class extends Application as it's the main entry point for our JavaFX application. All Initialization and starting the crawl process setting the scene and handlers are done in this class. The initialize method in this class deletes all the directories recursively before starting a new crawl process and all attributes that need to be initialized. The listener created in this class is for the search button. When a user clicks the search button, the handler is invoked and information is passed to the search method in the Search class: the search query, page rank being used, top X most relevant pages. When the search method is done processing the returning pages are then passed to the view to display to the user.

SearchView Class (SearchView.java)

This class is used to create the UI and display data to the user. This is the view part of our MVC Application. The UI is created all on one pane. I also styled the search buttons, text field and text area with CSS.

UnitTester Class (UnitTester.java)

This class implements the ProjectTester class and is used for unit testing. The class has 2 attributes, searchApp(SearchApplication) and searchData(SearchData). The class uses these 2 attributes to invoke all methods for the unit testing.

Webpage Class (Webpage.java)

The class is used to simulate a webpage, it implements Serializable so that Object can be written out as files. There are 4 attributes in this class: title, outgoingLinks, incomingLinks, score. The outgoing and incoming links are saved as ArrayList of Strings. The class also implements the SearchResult interface in order to display data to the GUI.

WebRequester Class (WebRequester.java)

This class is used to read a URL and return the String representation of it the webpage.

SearchResult Class (SearchResult.java)

This interface is a template for how the data should be returned from the Search Class(model) to the SearchView(view) and then displayed to the users in the format of title(webpage title) and score (cosine similarity score) which ranks the most relevant pages .

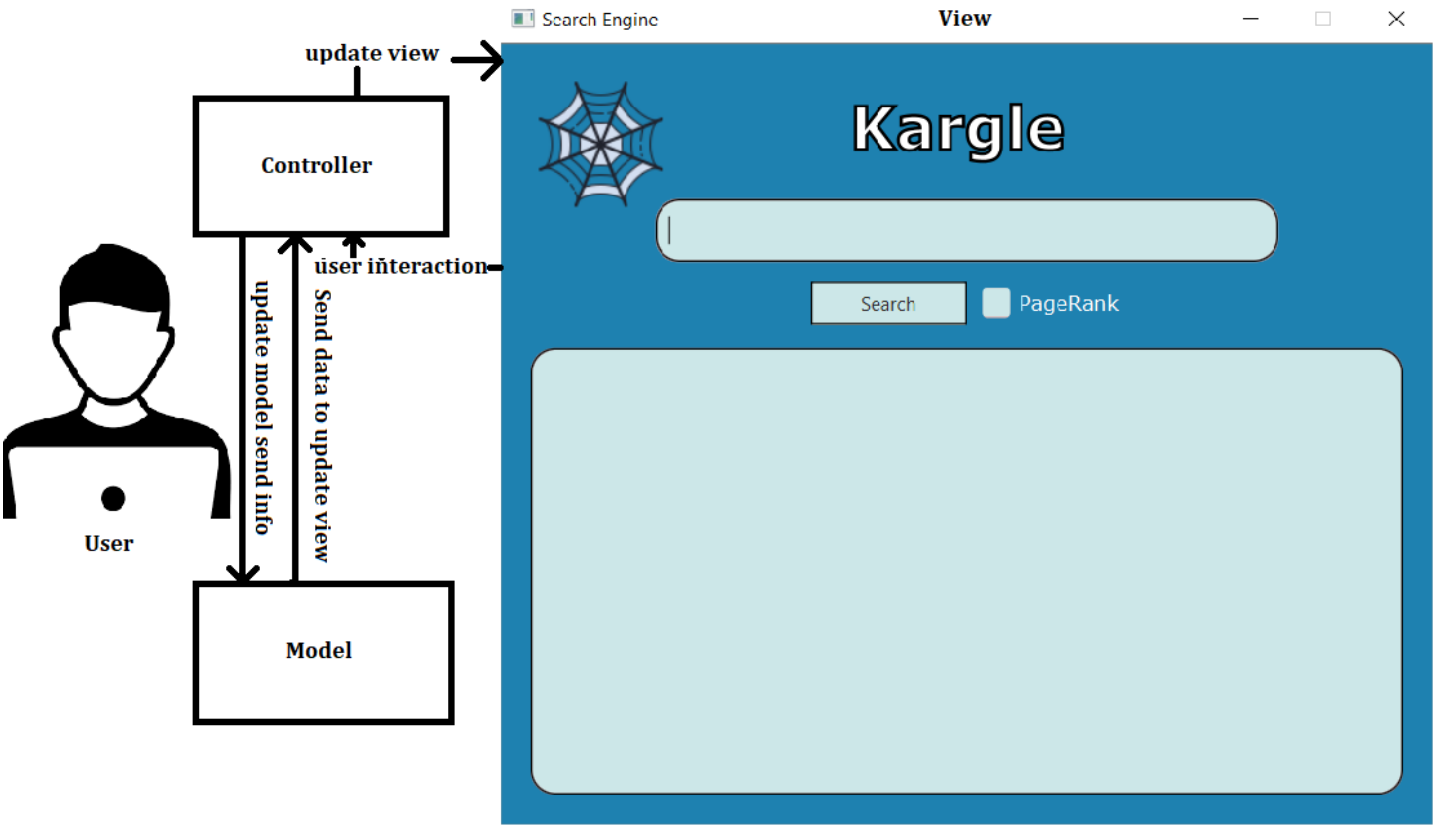
ProjectTester Class (ProjectTester.java)

This interface is a template for how the unit testing class should be. The UnitTester class implements this interface and defines all the methods.

MatMult Class (MatMult.java)

Class that is used for matrix multiplication and calculating Euclidean distance between two vectors. The methods are used for calculating the page rank.

Overall Design



The design of my web crawler application starts with the user making a request by interacting with the Graphical User Interface (GUI)/ view (SearchView). The user will type a search word or sentence and select whether they want to use page rank boost. Once the user clicks the search button the event handler for the search button invokes the handler method in the controller (SearchApplication.java). The handler method then sends the search query along with a Boolean to indicate if the page rank is used to the model (Search.java). The model will then return a list of the most relevant webpages to the users search query based on the cosine similarity calculation of each webpage. The List is then returned to the controller which will return it back to the view (UI) to be displayed to the user in the format (page title, cosine similarity score).

I will now discuss the changes I made to my Project. The first changes were to convert the project to OOP. I did this by creating classes for my webpages. The reason I decided to make my webpages an object is because I noticed from my last project that a lot of the data correlated to a specific page. Each webpage had their own tf, tf_idf, outgoing links, incoming links, and page rank, so I made each webpage object contain all these attributes. My plan for this project was to make my webpage object serializable so that I can write the object out to files. I made sure to make my tf and tf_idf a hash map so when retrieving the data for a specific word it would be constant time. I also made my Search Data a class to make it easier to unit test my application as my UnitTester class which implements the ProjectTester interface uses the Search Data class to run its unit testing. My Search class(model) also uses the Search Data methods to calculate the cosine similarity score for each webpage in relation to a search query. I initially had all my cosine similarity calculations all in one method but have since separated it into methods to improve readability. Change was also made to my parsing of the webpages in my Crawler class. I had originally read each character of the String represented webpage and manually checked for the tags but made changes to use the String library find method to reduce the amount of code, the run time hasn't changed it is still O(n). Overall changing the project to OOP has made the application more organized as each class has its role in the application and attributes, behaviors that belong to it.

Updated Version 1.1

Initially I had my tf, tf_idf as attributes of my webpage class. My runtime was taking upwards of 10+ minutes to run all the test cases in some of the larger datasets. I realized that even though I was having constant time for lookup for my tf,idf. I was reading in the URL object along with all the attributes which took much longer since I would have to read in all attributes for the URL object before looking through the hashmap that contained the tf, tf_idf value for the current word. I decided to change my tf and tf_idf back to its own directories with text files for each of the words which improved my runtime for searching.

Instruction for running GUI

- Navigate to the file tab and under Project structure add javafx lib

- Navigate to run tab and select edit configurations. The Search Application should be shown on the left side under Applications. On the right side under build and run make sure to change the vm option to the path where your javafx jdk is.

- Under the start method in SearchApplication is the handler for the Search button

```
view.setTopPages(model.search(view.getSearchField().getText(), boost: true, X: 10));
```

The value 10 can be changed depending on how many relevant pages you want the search engine to return.

- Launch the application by right clicking on the SearchApplication and selecting run SearchApplication main().

- The GUI will use the data from the last performed crawl, if no crawl has been made then the GUI will notify the user that there is no data crawled to perform the search.

- The GUI should launch up and you may type in a sentence of search words and select whether you want to use page rank boost. Click the search button and your results should display in the TextArea.