Project Report

By: Anjish Pradhananga and Iman Ullah

<u>Running the project:</u>

To run the project, start by running the GUI.java file. Doing so will present the user with a Graphical User Interface in the shape of a box containing two text fields. Start by typing in the search query into the text field labeled search. Then indicate the number of search results you desire by typing in an integer in the text field labeled amount of results. If you want your search results to have a page rank boost applied, please select the boost button. When selected, the boost button should be blue. After completing all these steps, click search, and your results should appear. Please note that if you were to give invalid data, you would be presented with an error message asking you to provide an integer. Do not provide string-type data such as "Two" or numbers with decimals such as "10.5". Make sure the number is a whole number. In the case where you do provide invalid data, all you have to do is provide valid data and click the search again.

<u>Functionality:</u>

Completed Functionality:

The project successfully can efficiently crawl and parse the title, words, and links of an input URL from its webpage and all the multi-linked URLs contained in the seed URL. It can also efficiently calculate and store data used by a search engine to generate a tier of sorted relevant web pages.

It consists of a GUI that allows a user-friendly way to interact with the program: input the number of results and select if the pagerank factor is to be considered whilst generating the results.

The GUI is made in the MVC model which ensures that any modification upon one part of the project (the logic or the interface) doesn't affect the other.

It handles exceptions well to ensure the program doesn't crash upon unforeseen errors.

The project is appropriately structured, and the codebase is documented well such that it's easier for other developers to understand the code and the rationale behind implementations and variables used.

The crawler and the search are fast and efficient, and once data has been crawled, the search engine can query the existing data independently and provide results.

Functionality not working:
Currently, the crawler is incomplete as it assumes much simplified HTML page elements and structure, which is the contrast to the real-world pages that have a much more complex structure and a wide range of dynamically used elements and attributes.

File Description:

Constants.java-
A class that is used to store the constant variables used throughout the project.
Using this class has the following advantages:
- Prevents accidental changes of important variables used throughout the project.
- Makes code easily adjustable as if any change is needed, the variable only has to be changed once as opposed to changing it everywhere it is used in the project (not possible in large projects).
- Gives other developers a good idea of how the perimeters of the project are set, as all of them are easily visible in one place.

Crawl.java-
A class that crawls data from a webpage, parses the HTML data and stores them whilst calculating relevant data for future access by the search engine. All the calculations are done here to search faster, as the engine now only has to retrieve the required data and can generate the results much quicker. This is much better for user-friendliness. This also makes sense as crawling is done only once (in our project), while search queries are done many times.

CrawlHelpers.java-
A class that primarily consists of static helper class methods which are used by the Crawl class whilst crawling, parsing, storing, and generating data from the entered webpage. These have been separated as while they are used by the crawler, they are simple tasks that can be used by other parts of the program. The name might have to be changed into just helperFunctions.java though.

Data.java-
This is an immutable record structure that stores all parsed data. It makes sure that your parsed data doesn't get changed by accident while performing analysis. Also, if

there's many elements of data parsed (in an actual webpage, there are countless HTML elements), this makes it easier to understand the data and access them as well.

PageRankData.java-
A class that implements the SearchResult Interface that stores the calculated search score and title of a URL while searching along with methods to get and set the scores. It helps in defining what data to store while searching and the behavior of how to store and display these data.

SearchResult.java
An interface that consists of methods to retrieve the score and title of a search result.

ProjectTester.java
An interface that defines the methods required while testing the search and crawl functionality. It consists of methods that crawl a url, perform a search query on the crawled data, and retrieve various pieces of data generated during crawl.

ProjectTesterImp.java
A class that implements the ProjectTester Interface and glues all of the crawl and search classes and methods to carry out tests on the overall project and even its specific functionalities. By making an instance of this class, you can test the search result, incoming/outgoing links, tf, tfidf values and even expand the testing with other factors/test cases if desired by simply adding on to the Interface.

SearchEngine.java
This class implements the search functionality using crawled data. The class is called by the view class, to update the GUI using the data calculated using this class. The class has 2 main functions. Get_query_tfidf is responsible for taking in a search query and returning the tf idf of the words in the search query.
It also has another important method, which is responsible for the actual search part of the project. It takes in 3 parameters, a search query, a boolean for the page rank boost and also an integer representing how many results we want. This is the method called by the view class. While the user is in the GUI, he is expected to provide these 3 parameters through the 2 text fields and the boost button.

SearchResultComparator.java
A class that implements the inbuilt Comparator interface used to sort the search results objects based on score in descending order and lexicographically if the scores are the same.

WebRequester.java
Responsible for retrieving the html data from a given URL.

View.java
This class is the view class of the project from the MVC model. This class implements all the elements you will see in the GUI, such as the text fields, labels, listviews and so on. It is also expected to update itself based on the data collected by the GUI or the controller class of this project. It is also responsible for calling the methods from the model classes to properly update itself.

GUI.java
As the controller class of this project, this class is responsible for interacting with the user to collect inputs, such as mouse and button clicks. The main functions of this class is to generate the GUI and run the search and page rank boost buttons. When the user clicks the search button, this class is expected to make sure that the view class changes as it should.

Project Design:

Directory structure:
In the root directory, there are specific directories to store the parsed data, the pagerank, idf, tf and tfidf data parsed and calculated during the crawling process which are stored in '.txt.' files.
For idf data, each unique word found during crawl has its own file with the idf value.
The tf, tfidf, and pagerank data are stored in files broken down based on the url.
The parsed data are also stored in individual files for each url where different elements of the parsed data (eg. url, title, words, links) are written in a new line.
Breaking down the data like this, as opposed to storing them all in one file, allows us to have faster retrieval of the required piece of information as we do not need to search through all the urls' data to access a single url. Separating the parsed data from the calculations also reduces processing time as we do not have to retrieve unnecessary data just to retrieve one small piece (to retrieve tf value, we do not need

to go through the parsed data, tfidf, or pagerank values). Opening and reading a specific file takes constant time.
This way memory and time is conserved. There is a tradeoff of disk space and RAM but computational memory is of greater priority as user friendliness is more important.
The src directory has two packages: the "projectClasses" and "testingResources". The testing files are separated from the project files to prevent file cluttering and accidental deletion of project files. This makes maintenance and workflow easier as everything is organized.

Design:
We have used the GUI to hide away the abstract details of the complex implementation of the crawler and the search engine. To the user who interacts with the project, the interface is simple and friendly and all the technical details are abstracted to only present the functionality. Full-scale crawlers and search engines are much more complex. Here, we were able to abstract them into a much simpler form and create the structure of the overall project. It currently serves our purpose well and the classes can be further appended with more attributes and methods and progressively can be built into a complete crawler and search engine as the base is well-built.

There are countless important attributes and methods that are used in the classes with a very specific purpose and behavior. Haphazard change can cause the program to behave unexpectedly and to prevent this, we have encapsulated these attributes and methods.
For example, we have defined an immutable record structure "Data" that stores all the immediately parsed data. This data shouldn't be changed while using it to carry out other operations so defining it as an immutable record prevents that. Also, in a large-scale crawler where there are many pieces of parsed data, it's easier to understand what we've parsed and also retrieve and use the required data specifically. So it makes sense to create a separate Record structure instead of storing these important data in a simple mutable data structure like arrays.
Methods like generating the idf, pagerank and other classes are encapsulated as we do not want to call them unnecessary in an unrequired part of the program with the incorrect data and risk completely overwriting our existing database with the wrong information. So, these data calculating methods are only accessible to the main method like the crawl method in the "Crawl.java" class which ensures that all calculations are carried out correctly.
Attributes are also similar. In the Pagerank class that stores the search result, we want to store all accurate data. So, by defining the url and title as private attributes without any setter functions, they can only be retrieved and cannot be accidentally modified which could affect the result. The score is also encapsulated and since it has a setter

function that always stores the score in 3 decimal places(or other ways in the future as required), there is no other way for the score to be stored. So, the behavior we want is always preserved.
As the project gets more larger and undoubtedly complex, the methods get more complex with more operations and thus such preservation of behavior is valuable as it ensures us that the attribute/ method will always stay/behave the desired way.

In the searchEngine class, the search method returns a list of SearchResult objects from "dataList" which is a list of PageRankData objects. Since PageRankData implements the SearchResult interface we can typecast it into the interface. This is useful as we only need the getTitle() and the getScore() methods and polymorphism makes things simpler. Also, with increased complexity and scale of the searchEngine, we might have multiple objects that implement the SearchResult interface. Here, polymorphism is even more useful as we can simply generalize the classes to use the methods we need and generate the desired result instead of going through and identifying each class which would rather make the code long, unmanageable, error-prone and take more processing time.
Also having the ProjectTester interface is very useful as we can test a variety of different implementations of the project(only one present right now(ProjectTesterImp.java) without worrying about the specifics as the interface already defines what is needed and tested. So testing is generalized and easily possible for any variation.

Fragmenting the major components of the projects into different classes helps us create a good image of how the project works and ties together. As someone looking at it for the first time, just by looking at the classes, they can tell what's going on in the project: "crawl class probably crawls the data, searchEngine probably searches the data, view class probably defines how the GUI looks, searchResults store url, title, and score data, etc.". This also greatly helped in working on the project as once we divided the classes, it was easier to divide the tasks and wrap them all together without any accidental code deletion, git merge conflicts, etc. as they're all independent and tied together in another class that instantiates our classes wherever needed. So, this makes it scalable too as complexity can be added on to each of the classes as needed without it directly affecting other functionality.