COMP1406Z Project Report

Jiayu Hu
2022/12/08

Instructions for Running the Program and GUI

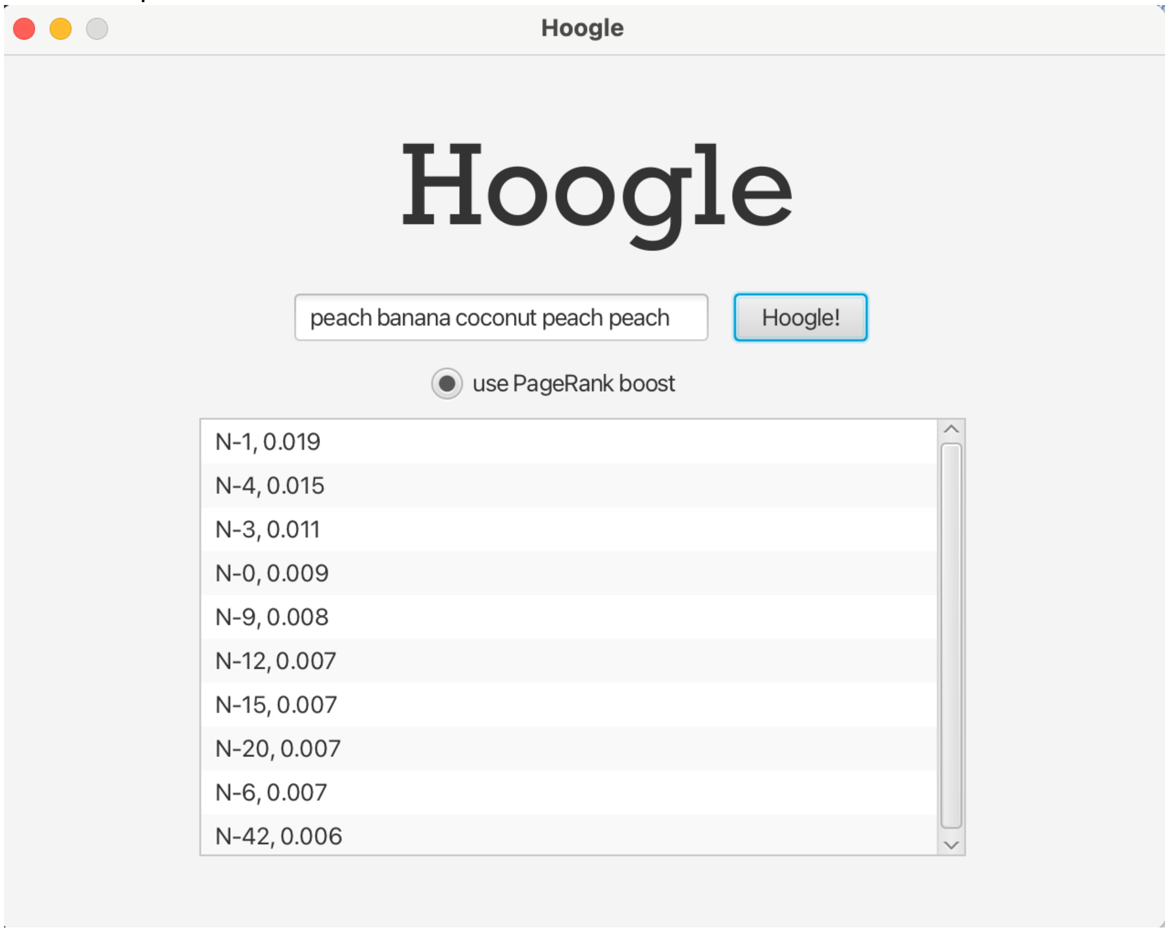
1. Go to CrawlerStart.java and change the URL in the 6th line to the seed URL you want to start at.
   For example:

```
tester.crawl("https://people.scs.carleton.ca/~davidmckenney/fruits/N-0.html");
```

2. Run CrawlerStart.java. Wait for the crawling to finish.

3. After the crawling is finished, run SearchEngineApp.java, and enter the query you want to search in the text field. Choose whether to use PageRank boost, and then press the "Hoogle!" button to start the search.

4. The top 10 results of the search will show in the list.



peach banana coconut peach peach

[✓ use PageRank boost]

```
N-1, 0.019
N-4, 0.015
N-3, 0.011
N-0, 0.009
N-9, 0.008
N-12, 0.007
N-15, 0.007
N-20, 0.007
N-6, 0.007
N-42, 0.006
```

Functionalities of the Search Engine

Functionality that completed

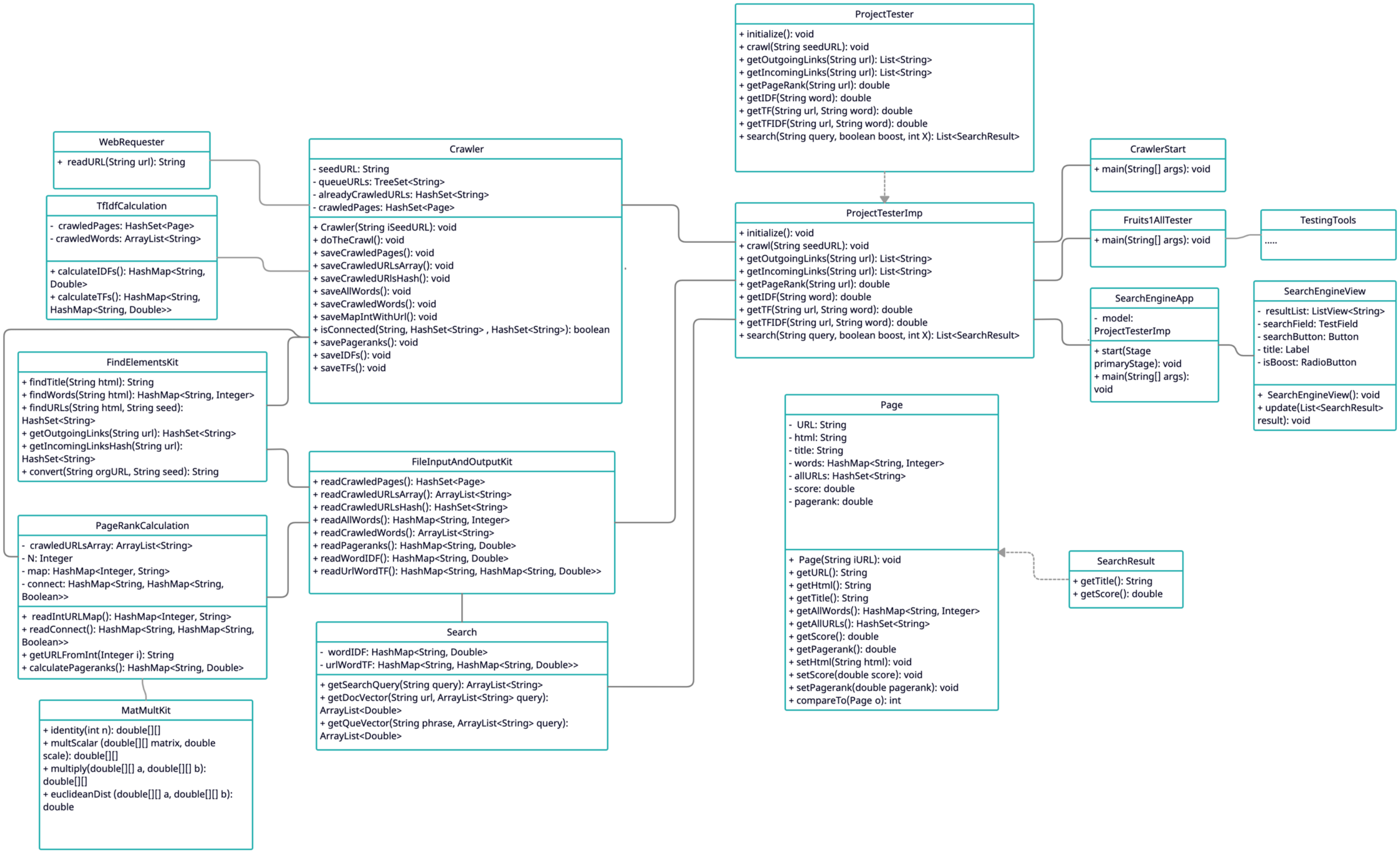
- Crawl from a specific seed URL in fruit databases
- Read HTML elements on the crawled pages in fruit databases
- Save crawled data to the local repository
- Save crawled pages as Page Objects to the local repository
- Calculate PageRank for each crawled page
- Save PageRank to the local repository
- Search string query based on whether to use PageRank boost or not, and return the top X results

Functionality that does not work

- Crawl on a larger scale of website database
- Read HTML elements on any webpage on the internet
- Change the number of crawled pages
- Search and return the results based on the words' order in the query

An Outline of Each of the Classes and Interfaces

Class or Interface	**Main responsibility**
Page	Represent a crawled page object
SearchResult	Include methods like getTitle() and getScore()
Crawler	Crawl all pages from the seed URL, and save useful data to the local repository
WebRequester	Read URL and return the HTML as a String
FindElementsKit	Include helper functions that are used to find elements on the HTML
TfIdfCalculation	Include helper functions that are used to calculate IDFs and TFs for all crawled pages and words saved in the local repository
PageRankCalculation	Include helper functions that are used to calculate PageRank for all crawled pages
MatMultKit	Include helper functions for calculations on matrix
FileInputAndOutputKit	Include helper functions to read data from the local repository and return Java objects
Search	Get the search query user entered, get the search vector and query vector for search
ProjectTesterImp	Implement ProjectTester, initialize the local repository, do the crawl, return the required result like outgoing links and incoming links, and do the search based on a string query, whether to boost or not, and return the top X results
ProjectTester	Include methods like initialized(), crawl(), etc.
CrawlerStart	The program that starts crawling on a seed URL
Fruits1AllTester, …	Testers that could start different testers and return result in .txt files
TestingTools	Helper functions for Tester class
SearchEngineApp	The controller for GUI
SearchEngineView	The view for GUI


ProjectTester
+ initialize(): void
+ crawl(String seedURL): void
+ getOutgoingLinks(String url): List<String>
+ getIncomingLinks(String url): List<String>
+ getPageRank(String url): double
+ getIDF(String word): double
+ getTF(String url, String word): double
+ getTFIDF(String url, String word): double
+ search(String query, boolean boost, int X): List<SearchResult>
WebRequester
+ readURL(String url): String
Crawler
- seedURL: String
- queueURLs: TreeSet<String>
- alreadyCrawledURLs: HashSet<String>
- crawledPages: HashSet<Page>
+ Crawler(String iSeedURL): void
+ doTheCrawl(): void
+ saveCrawledPages(): void
+ saveCrawledURLsArray(): void
+ saveCrawledURlsHash(): void
+ saveAllWords(): void
+ saveCrawledWords(): void
+ saveMapIntWithUrl(): void
+ isConnected(String, HashSet<String> , HashSet<String>): boolean
+ savePageranks(): void
+ saveIDFs(): void
+ saveTFs(): void
CrawlerStart
+ main(String[] args): void
TfIdfCalculation
- crawledPages: HashSet<Page>
- crawledWords: ArrayList<String>
+ calculateIDFs(): HashMap<String, Double>
+ calculateTFs(): HashMap<String, HashMap<String, Double>>
ProjectTesterImp
+ initialize(): void
+ crawl(String seedURL): void
+ getOutgoingLinks(String url): List<String>
+ getIncomingLinks(String url): List<String>
+ getPageRank(String url): double
+ getIDF(String word): double
+ getTF(String url, String word): double
+ getTFIDF(String url, String word): double
+ search(String query, boolean boost, int X): List<SearchResult>
Fruits1AllTester
+ main(String[] args): void
TestingTools
.....
SearchEngineApp
- model: ProjectTesterImp
+ start(Stage primaryStage): void
+ main(String[] args): void
SearchEngineView
- resultList: ListView<String>
- searchField: TestField
- searchButton: Button
- title: Label
- isBoost: RadioButton
+ SearchEngineView(): void
+ update(List<SearchResult> result): void
FindElementsKit
+ findTitle(String html): String
+ findWords(String html): HashMap<String, Integer>
+ findURLs(String html, String seed): HashSet<String>
+ getOutgoingLinks(String url): HashSet<String>
+ getIncomingLinksHash(String url): HashSet<String>
+ convert(String orgURL, String seed): String
Page
- URL: String
- html: String
- title: String
- words: HashMap<String, Integer>
- allURLs: HashSet<String>
- score: double
- pagerank: double
+ Page(String iURL): void
+ getURL(): String
+ getHtml(): String
+ getTitle(): String
+ getAllWords(): HashMap<String, Integer>
+ getAllURLs(): HashSet<String>
+ getScore(): double
+ getPagerank(): double
+ setHtml(String html): void
+ setScore(double score): void
+ setPagerank(double pagerank): void
+ compareTo(Page o): int
FileInputAndOutputKit
+ readCrawledPages(): HashSet<Page>
+ readCrawledURLsArray(): ArrayList<String>
+ readCrawledURLsHash(): HashSet<String>
+ readAllWords(): HashMap<String, Integer>
+ readCrawledWords(): ArrayList<String>
+ readPageranks(): HashMap<String, Double>
+ readWordIDF(): HashMap<String, Double>
+ readUrlWordTF(): HashMap<String, HashMap<String, Double>>
PageRankCalculation
- crawledURLsArray: ArrayList<String>
- N: Integer
- map: HashMap<Integer, String>
- connect: HashMap<String, HashMap<String, Boolean>>
+ readIntURLMap(): HashMap<Integer, String>
+ readConnect(): HashMap<String, HashMap<String, Boolean>>
+ getURLFromInt(Integer i): String
+ calculatePageranks(): HashMap<String, Double>
SearchResult
+ getTitle(): String
+ getScore(): double
Search
- wordIDF: HashMap<String, Double>
- urlWordTF: HashMap<String, HashMap<String, Double>>
+ getSearchQuery(String query): ArrayList<String>
+ getDocVector(String url, ArrayList<String> query): ArrayList<Double>
+ getQueVector(String phrase, ArrayList<String> query): ArrayList<Double>
MatMultKit
+ identity(int n): double[][]
+ multScalar (double[][] matrix, double scale): double[][]
+ multiply(double[][] a, double[][] b): double[][]
+ euclideanDist (double[][] a, double[][] b): double

The overall design of the system

How I have applied the OOP principles in my project

abstraction

- using interfaces ProjectTester.java and implementing it in ProjectTesterImp.java so that it can have the methods in its own way
- using interfaces SearchResult.java and implementing it in Page.java, so that the Page has the methods in SearchResult.java
- using abstract classes

encapsulation

- the Page object has private attributes such as String URL, which are able to be accessed via their public getter and setter methods
- other Classes have attributes that are set to private

Polymorphism

- type-casting Page object to SearchResult object when returning SearchResult
- type-casting objects when reading from files
- type-casting HashSet like outgoingLinksHash to ArrayList when it is required to return List, so that it could be returned as the proper type
- using constructors for Page with different parameters
- overriding methods in ProjectTesterImp.java so it has its own methods

How OOP Principles has improved the overall quality of my implementation

Before implementing OOP Principles, I only have three modules (crawler.py, searchdata.py, and search,py), and each module contains a lot of attributes and methods, which could be hard to find bugs or update the functionality.
After implementing OOP Principles, I have more Java classes representing different program functionality, and I can debug updates and code easily.

Before implementing OOP Principles, I have redundant codes in different python modules, which could be troublesome if I want to change them.
After implementing OOP Principles, my different code refers to the same Java project and no longer has redundant codes. It is easier to update functionality and debug.

Before implementing OOP Principles, the only way for me to get a page's elements from its URL is to read the element from a dictionary that has the URL as the key and the element as the value.
After implementing OOP Principles, I can have an Object called Page representing the webpage in real life, with attributes like URL, title, and words that the page has. This data structure is more direct and clearer than a dictionary.

How I have saved my data in files

1. When running CrawlerStart.java, the ProjectTesterImp.java will create a new Crawler object, which has saving methods to save crawled data like crawled pages, crawled URLs, crawled words, etc. The saving methods will save the objects to .dat files using ObjectOutputStream and FileOutputStream, and those saved objects will be read and used for future purposes.
2. For calculating page ranks, there are other methods such as saveMapIntWithUrl(), saveConnect() to save the data needed for the calculation.
3. After calculating page ranks for each page, the data is also saved as an object to a .dat file.
4. The IDF for each crawled word and TF for each URL are also saved as Objects into two different .dat files.

Runtime complexity Analysis

In the ProjectTesterImp Class:

Method		**Runtime complexity**
initialize()		O(n), n = # files in the current path
crawl(String seedURL)	doTheCrawl()	O(n*m), n = # crawled pages, m = # length of HTML
getOutgoingLinks(String url)		O(n), n = # crawled pages
getIncomingLinks(String url)		O(n), n = # crawled pages
getPageRank(String url)		O(1)
getIDF(String word)		O(1)
getTF(String url, String word)		O(1)
getTFIDF(String url, String word)		O(1)
search(String query, boolean boost, int X)	Search.*getSearchQuery*(query)	O(n), n = # words in the query
	Search.*getQueVector*(query, queryList)	O(n), n = # words in query
	for (double i : queVector)	O(n), n = # words in query

	for (Page p : crawledPages)
	   Search.*getDocVector*(url, queryList)
	   for (int i = 0; i < queVector.size(); i++)
	   for (double i : docVector)
		O(n * m), n = # crawled pages, m = # words in query

	if (boost)
	   for (Page p : result)
		O(n), n = # crawled pages

	while (sortedResult.size() < 10)	O(1)

	Collections.*sort*(sortedResult);	O(n*log(n))

To improve the runtime efficiency in the ProjectTesterImp Class, especially those get methods implemented from the ProjectTester Interface, I choose to save the objects into files as HashSet or HashMap, so the runtime complexity for searching a specific element would be O(1).

In the Crawler class:

Method		**Runtime complexity**
doTheCrawl()	while (queueURLs.size() != 0)	O(n), n = # crawled pages
	if (alreadyCrawledURLs.contains(curURL))	O(1)
	FindElementsKit.*findTitle*(curHtml)	O(m), m = # length of HTML
	FindElementsKit.*findWords*(curHtml)	O(m), m = # length of HTML
	FindElementsKit.*findURLs*(curHtml, seedURL);	O(m), m = # length of HTML
total		O(n*m), n = # crawled pages, m = # length of HTML
getUrl1Url2Connect()		O(n^2), n = # crawled pages

In the PageRankCalculation:

Method		**Runtime complexity**
calculatePageranks()	MatMultKit.*identity*(*N*);	O(n^2), n = # crawled pages

	for (int i=0; i < *N*; i++)
	   for (int j=0; j < *N*; j++)
		O(n^2), n = # crawled pages

	matrix = MatMultKit.*multScalar*(matrix, 1-a);	O(n^2), n = # crawled pages
total		O(n^2), n = # crawled pages

To improve the runtime efficiency when calculating pageranks, I save the information about pageranks to a file, so that the program only needs to calculate it once (O(n ^ 2), n = # crawled pages), and then read it from the file (O(1)), instead of calculating it every time it runs (O(m * (n ^ 2)), m = # running times, n = # crawled pages).