120942514

March 15, 2020

Artemis Livingstone

SQA AH COURSEWORK

URL path finder

# Table of Contents

[Table of Contents 1](#_Toc35201711)

[Analysis 4](#_Toc35201712)

[Description of the Problem 4](#_Toc35201713)

[Requirements 5](#_Toc35201714)

[End user requirements 5](#_Toc35201715)

[Functional requirements 5](#_Toc35201716)

[Scope 10](#_Toc35201717)

[Boundaries 10](#_Toc35201718)

[Constraints 11](#_Toc35201719)

[Survey 12](#_Toc35201720)

[UML 14](#_Toc35201721)

[Project plan 16](#_Toc35201722)

[Resources required 16](#_Toc35201723)

[Estimate of timings 18](#_Toc35201724)

[Identified tasks 20](#_Toc35201725)

[Design 22](#_Toc35201726)

[Top level Flow chart 22](#_Toc35201727)

[Data structures. 24](#_Toc35201728)

[Design of integration 25](#_Toc35201729)

[Data Dictionary 25](#_Toc35201730)

[Query design 25](#_Toc35201731)

[UI Design 27](#_Toc35201732)

[Execution through terminal with no generated UI 27](#_Toc35201733)

[Execution through text-based UI in terminal 28](#_Toc35201734)

[Database connection error 29](#_Toc35201735)

[Pseudocode for the UI 30](#_Toc35201736)

[METHOD UI 30](#_Toc35201737)

[METHOD setContents 30](#_Toc35201738)

[METHOD setCommands 31](#_Toc35201739)

[METHOD showUi 31](#_Toc35201740)

[Pseudocode for NoodleMap 32](#_Toc35201741)

[METHOD Noodlemap 32](#_Toc35201742)

[METHOD addEdge 32](#_Toc35201743)

[METHOD loadCSV 33](#_Toc35201744)

[METHOD loadDatabase 34](#_Toc35201745)

[METHOD dijkstra 36](#_Toc35201746)

[METHOD returnMap 37](#_Toc35201747)

[METHOD mergeSort 38](#_Toc35201748)

[METHOD merge 39](#_Toc35201749)

[METHOD insertSort 40](#_Toc35201750)

[Pseudocode for Functions 41](#_Toc35201751)

[PROCEDURE pathfinder 41](#_Toc35201752)

[PROCEDURE clearDatabases 42](#_Toc35201753)

[PROCEDUTE sort 43](#_Toc35201754)

[PROCEDURE quit 46](#_Toc35201755)

[Pseudocode for Start Up 46](#_Toc35201756)

[Pseudocode for Scraper 47](#_Toc35201757)

[PROCEDURE runScrape 47](#_Toc35201758)

[Trimming URL to domain form 50](#_Toc35201759)

[Implementation 52](#_Toc35201760)

[Research and Skills Learnt. 52](#_Toc35201761)

[All Required Modules 53](#_Toc35201762)

[Ongoing testing 57](#_Toc35201763)

[References used during implementation: 64](#_Toc35201764)

[For checking fastest comparisons for strings: 64](#_Toc35201765)

[For understanding how MySQL handles input with different characters. 64](#_Toc35201766)

[For understanding how to loop through dictionary’s in Python 64](#_Toc35201767)

[For implementing arguments of unknown length in Python 65](#_Toc35201768)

[For calling functions from dictionaries 65](#_Toc35201769)

[For implementation of merge sort 65](#_Toc35201770)

[For implementation of the argument parsing system for passing in parameters into code 65](#_Toc35201771)

[For installation and setting up of MySQL database on my system. 66](#_Toc35201772)

[Testing 67](#_Toc35201773)

[Test plan 67](#_Toc35201774)

[Test plan for Component Testing 67](#_Toc35201775)

[Test Plan for Integration Testing 84](#_Toc35201776)

[Test Plan for Test Cases. 89](#_Toc35201777)

[Testing Plan for Persona Testing 90](#_Toc35201778)

[End User Testing Design 92](#_Toc35201779)

[Component Testing 94](#_Toc35201780)

[Component Testing for NoodleMap() 94](#_Toc35201781)

[Connecting to database and executing commands 99](#_Toc35201782)

[Component Testing for UI 105](#_Toc35201783)

[Component Testing for clearDatabases() 107](#_Toc35201784)

[Component testing for Scraper 111](#_Toc35201785)

[Integration Testing for pathfinder() procedure 118](#_Toc35201786)

[Integration Testing for sort() procedure 122](#_Toc35201787)

[Error images 129](#_Toc35201788)

[Test cases. 130](#_Toc35201789)

[Test Case 1 130](#_Toc35201790)

[Test Case 2 132](#_Toc35201791)

[Test Case 3 135](#_Toc35201792)

[Test Case 4 137](#_Toc35201793)

[Test Case 5 137](#_Toc35201794)

[Conclusion 139](#_Toc35201795)

[Testing with Persona 140](#_Toc35201796)

[End user testing 143](#_Toc35201797)

[Evaluation 145](#_Toc35201798)

[End user Testing Analysis 145](#_Toc35201799)

[Overall Analysis of Testing 146](#_Toc35201800)

[Requirements Specification Evaluation. 146](#_Toc35201801)

[End User Requirements Evaluation 146](#_Toc35201802)

[Functional Requirements Evaluation 148](#_Toc35201803)

[Robustness of Program 149](#_Toc35201804)

[Maintainability of Program 150](#_Toc35201805)

[Further steps 151](#_Toc35201806)

[Proof of Iteration 152](#_Toc35201807)

# Analysis

## Description of the Problem

I intend to design and implement a program that finds paths between webpages and can return the URLs found on each webpage. These functionalities combined will allow a developer to test their website for orphan pages, or how the pages connect to each other. By getting the paths between webpages, the developer can evaluate the usability of their website by seeing how easy it is to get to certain pages. The program will include the following features: a help screen, text-based UI to help users find their way, a database of webpages crawled that will be created using SQL with a linked database of pages they have linked to, a pathfinding system and a system to return all the found webpages and all the URLs found on them.

The intended end users of this program will be people looking for orphan pages and trying to understand the ease of getting to pages, therefore will probably be tech literate developers, who are comfortable with using terminal based tools.

My project meets the Advanced Higher Computing Science requirements as it will have a UI suitable for tech literate users with validation for database connections. The project will be classed under the procedural programming section of the course as it contains multiple 2d arrays. My project will interface with an SQL database, creating a database and writing and reading URL’s from it. My project will also have a sorting algorithm to sort all the URL’s found on scraped webpages before printing to screen.

This development meets the requirements of the Advanced Higher project, because:

It is based on the Software Design content:

* Procedural programming is being used (there are objects, but this is being treated as a procedural programming project)
* 2 dimensional arrays are being used throughout the project
* An insertion sort is used to sort the URL’s alphabetically

It integrates with the Database Design part of the course:

* Details of the URL’s are stored in a database with each table being an individual Domain
* The program will automatically create tables with the domain name if they don’t exist using SQL
* SQL is used to write to these tables
* SQL is used to drop any saved tables to free up storage
* SQL is used to read from these tables

## Requirements

### End user requirements

* User must be able to use text-based UI to input a starting website, and an end website to find a path to.
* User must be able to input the maximum number of jumps while scraping a website for the first time.
* The user must be able to view the requested path or receive an error message that there is no path.
* The user must be able to view a help page from the text-based UI.
* The user must be able to view the pages that the spider finds if they wish.
* The found pages must be sorted alphabetically for readability, and the user must have the option to write the output to file.
* If the user wishes to save to file, and selects a csv file, make the output be comma separated.
* The user must be able to clear the stored websites, to free up storage.

### Functional requirements

* The program must display a text-based UI that can take in a starting page, end page(only if the path finding mode is selected), number of moves and mode (input).
* The program will be able to scrape a URL and find all links on the URL, follow them, and repeat the process until the maximum jumps is achieved.
  + Maximum number of jumps must be positive and, of course, an integer.
* The webpage’s URL is to be stored then all the links leading off also need to be stored in a database using SQL.
  + This is done by making the first field the current URL and the second field will be a link that leads off.
  + If the table already exists (aka has already been scraped) then drop the table.
* URLs need to be stored and retrieved in a database using SQL from within the program
  + When retrieving the program must confirm that the table does in fact exist.
* Show the results of the Scraper if requested by the user
  + Call the scraper and write the output to the database
    - Connect to database
    - Create a new table with the domain name, dropping the table if it already existed
    - Write the webpage URL and any found URLs to database
  + Get all the output from the server by fetching it from the database.
  + Load the output into a node like structure inside of an object
  + Sort the results of the scraper alphabetically using ~~insert sort~~ merge sort (see testing as to why this was changed)
* Calculate a path between two URLs
  + Call the scraper and write the output to the database
    - Connect to database
    - Create a new table with the domain name, dropping the table if it already existed
    - Write the webpage URL and any found URLs to database
  + Get all the output from the server by fetching it from the database.
  + Load the output into a node like structure inside of an object
  + Use Dijkstra’s algorithm to calculate a path between two given nodes or display an error message if there is no path.
* The output of the showing the sorted results of the scraper, can be saved in a file type of the user’s choice. If they are just showing the results and use a .csv extension, then the system will output a CSV.
* The program must be able to reuse an existing table of URLs
  + This is done by not calling the scraper and simply calling the table name for the URL required. The table must exist, or the program will notify the user that it does not.
* The program needs to be able to drop all the tables in it.

#### Input and output.

Start Up

|  |  |
| --- | --- |
| Inputs | Any parameters being passed in terminal |
| Processes | Check if program is being started in terminal or with text-based UI |
| Outputs | Start-up state |

If start up state is with UI

|  |  |
| --- | --- |
| Inputs | Start-up state |
| Processes | Generate UI |
| Outputs | print UI to screen. |

Once UI is displayed right after start up

|  |  |
| --- | --- |
| Inputs | User inputs mode |
| Processes | Checks if mode exists in dictionary of commands and If it does, it calls the associated function. |
| Outputs |  |

If the path finder is called (even if UI isn’t specified):

|  |  |
| --- | --- |
| Inputs | Start web page and end webpage and number of jumps and whether reindexing is required |
| Processes | If reindexing is required:   * Call web scraper * initialise connection to database * Write output of scraper to database. * Load database into csv * Close connection to database * Execute pathfinder with start point and end point * Generate UI with pathfinder results. * Call UI object with contents   If reindexing is not required by the user:   * initialise connection to database * Load existing database into csv * Close connection to database. * Execute pathfinder with start point and end point * Generate UI with pathfinder results. * Call UI generator method with contents |
| Outputs |  |

When the results of scraping are called

|  |  |
| --- | --- |
| Inputs |  |
| Processes | If reindexing is required:   * Call web scraper * initialise connection to database * Write output of scraper to database. * Load database into csv * Close connection to database * Sort the webpages alphabetically * Generate UI with sorted map results. * Call UI generator method with contents   If reindexing is not required by the user:   * initialise connection to database * Load existing database into csv * Close connection to database. * Sort the results alphabetically if required by user (defaults to sorting it) * Generate UI with sorted map results. * Call UI generator method with contents |
| Outputs | Print the list of pages (alphabetically sorted) and the URLs found on them. |

Generate UI

|  |  |
| --- | --- |
| * Inputs |  |
| * Processes | * Check if input is required from user. |
| * Outputs | * Print stored Text * Print prompt if required. |

Get prompts from user

|  |  |
| --- | --- |
| Inputs | Get user input |
| Processes | Check if user input is valid  Call related function from dictionary |
| Outputs |  |

Dropping all cached tables

|  |  |
| --- | --- |
| Inputs | User selects the deleteTables option from the menu |
| Processes | Connect to database.  Get all tables.  Get confirmation from user that they wish to drop all tables.  Loop through tables and drop them all. |
| Outputs | All table names  Confirmation of database deletion |

### Scope

* A full design including pseudocode and data dictionary and query design for creating databases.
* A working data scraper and path finder, both being called from one system
* A help file that can be displayed in command prompt
* A test plan including test persona, test cases and outputs when different data is inputted and the results of the testing.
* End user survey to design functional requirements.
* An evaluation of the solution.
* The program will be tested and made for Python 3.7 on a windows 10 machine
* This program will largely be tested on [www.wikipedia.com](http://www.wikipedia.com) as [www.wikipedia.com/robots.txt](http://www.wikipedia.com/robots.txt) says “# Friendly, low-speed bots are welcome viewing article pages, but not dynamically-generated pages please.” Which gives permission for bots to scrape [www.wikipedia.com](http://www.wikipedia.com) if the time between pings is reasonable.
* The program will also be largely tested on quotes.toscrape.com and books.toscrape.com as legal permission has been obtained from the owner of the domain (see testing with persona). This will probably be more viable than Wikipedia, as there are a lot less links per page, meaning it is easier to check if all the links are being collected.

### Boundaries

* Will not follow links onto a different domain.
* It is not going to be designed nor tested for macOS or any windows installation prior to windows 10 or Linux installation.
* There will be no graphical representation of the possible paths due to time constraints.
* The program will not be packaged up into an installer so modules will need to be added manually.
* The program will use a simple text-based UI.
* The program will not be tested with URLs that contain spaces as RFC 1738: Uniform Resource Locators (URL) (which has now been superseded by RFC 3986: Uniform Resource Identifier (URI)) states that a space is an unsafe character.

### Constraints

Some technical, legal and time constraints apply to this project.

* This project will need to be completed between 1st of October and 16th of March 2020 due to SQA requirements.
* This project can only use open source modules for data scraping and no commercial only ones.
* I will be using primarily windows 10 and Linux Fedora for development, so any modules that are used need to work on either system.
* The project will be primarily written in Python 3.7 as it is the language I am most proficient in, with the exception of the SQL that is being used to store and fetch the URL’s from the database and any external modules that are programmed in another language.
* This program has only been tested on websites with prior permission, due to the relatively large quantity of pings resulting from the scraping. Doing so without prior permission of the website holder may break the Computer Misuse Act.

### Survey

An example survey that was used for the creation of the end user requirements.



A common request was for a help function with the text-based UI as it’s not the most common method of using a system for many people. This was one of the examples that helped me create my end user requirements. A large proportion of the people who said they were familiar or very familiar, said they would be interested in such a solution, and when questioned further said that they would be likely to use this for their own projects.

## UML

## Project plan

### Resources required

The resources that will be required at each stage of development are listed here:

|  |  |
| --- | --- |
| Analysis | * Gantt project 2.8.10 (windows) * Google chrome (windows) * Firefox (Debian) * Microsoft word (windows) * Survey monkey. * Internet connection. * Balsamiq mock-ups 3 |
| Design | * Microsoft word (windows) * Balsamiq mock-ups 3 * Draw.io for flow charts. |
| Implementation | * Visual studio code 1.38.1 * Python 3.7 with scrapy and MySQL installed * Anaconda environment * Google chrome * Git 2.20.1.windows.1 for backup and version control * GitHub account * Access to Github.com * Access to official scrapy documentation and tutorial * Access to MySQL documentation. * Access to stack overflow. * MySQL monitor and server * Access to <http://toscrape.com> * Computer with at least 6 GB of ram, preferable over 8 GB * System with MySQL installed * All the modules as specified in requirements.txt (see implementation) * References to websites as specified in ongoing implementation |
| Testing | * Snipping tool * Anaconda navigator/ Conda environment manager * Command prompt/ terminal * Microsoft word (windows) * Survey monkey * Access to end users |
| Evaluation | * Microsoft word (windows) |

For testing, I decided to use a windows system, with relatively high specs, as the machine will be running the database as well as doing large sorts and finding paths. The machines specifications were as such:

32 GB of DDR4 ram

Intel I7-7700k CPU running at 4.2 Gigahertz

Projects were executed from a Hybrid Drive, with the Python installation being on an SSD.

### Estimate of timings

### Identified tasks

These identified tasks have been imported from the Gantt chart.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Name | Begin date | End date | Duration  (days) | Predecessors | Outline number |
| 5 | Analysis | 08/10/2019 | 21/10/2019 | 10 |  | 1 |
| 41 | create scope boundaries and constraints | 08/10/2019 | 08/10/2019 | 1 |  | 1.1 |
| 14 | Functional requirements | 08/10/2019 | 09/10/2019 | 2 |  | 1.2 |
| 109 | Scope and Boundaries | 08/10/2019 | 09/10/2019 | 2 |  | 1.3 |
| 8 | Create end user questionnaire | 08/10/2019 | 11/10/2019 | 4 |  | 1.4 |
| 10 | create end user requirements | 14/10/2019 | 15/10/2019 | 2 | 8 | 1.5 |
| 32 | UML | 14/10/2019 | 21/10/2019 | 6 | 8 | 1.6 |
| 28 | Design | 22/10/2019 | 25/10/2019 | 4 | 5 | 2 |
| 104 | create top level flow chart | 22/10/2019 | 22/10/2019 | 1 |  | 2.1 |
| 86 | Design and adapt Dijkstra’s algorithm | 22/10/2019 | 22/10/2019 | 1 |  | 2.2 |
| 83 | Design UI | 22/10/2019 | 24/10/2019 | 3 |  | 2.3 |
| 93 | Design UI method | 25/10/2019 | 25/10/2019 | 1 | 83 | 2.4 |
| 1 | Implementation | 28/10/2019 | 17/01/2020 | 60 | 5;28 | 3 |
| 0 | Create Dijkstra’s algorithm | 28/10/2019 | 31/10/2019 | 4 | 28 | 3.1 |
| 21 | Create crawler | 01/11/2019 | 21/11/2019 | 15 | 0 | 3.2 |
| 27 | SQL reader and writer | 22/11/2019 | 12/12/2019 | 15 | 21 | 3.3 |
| 43 | Create UI | 13/12/2019 | 19/12/2019 | 5 | 27 | 3.4 |
| 47 | Integrate parts | 20/12/2019 | 17/01/2020 | 21 | 43 | 3.5 |
| 110 | Testing | 20/01/2020 | 28/02/2020 | 30 | 47 | 4 |
| 117 | Create test plan for Components | 20/01/2020 | 28/01/2020 | 7 |  | 4.1 |
| 116 | Create test plan for integration testing | 29/01/2020 | 03/02/2020 | 4 | 117 | 4.2 |
| 120 | Create test plan for test cases | 04/02/2020 | 05/02/2020 | 2 | 116 | 4.3 |
| 124 | Complete test plan for end user testing | 06/02/2020 | 06/02/2020 | 1 | 120 | 4.4 |
| 127 | Work through all testing, taking screenshots | 07/02/2020 | 26/02/2020 | 14 | 124 | 4.5 |
| 130 | Go back through and fix any errors found | 27/02/2020 | 28/02/2020 | 2 | 127 | 4.6 |
| 134 | Evaluation | 02/03/2020 | 03/03/2020 | 2 | 130 | 5 |
| 137 | Evaluate requirement specification | 02/03/2020 | 02/03/2020 | 1 |  | 5.1 |
| 140 | Evaluate testing | 03/03/2020 | 03/03/2020 | 1 | 137 | 5.2 |
| 143 | Final polishing off | 04/03/2020 | 16/03/2020 | 9 | 140 | 6 |

# Design

## Top level Flow chart

Drawn using draw.io

## Data structures.

The UI object is used to store and display a basic text-based UI. It can store all the function names of related commands and can accept user input if required.

NoodleMap is used to store the edges of the map, and contains the methods to sort, find a path, and load a database into it. The reason it is called Noodle, is while the edges can be almost treated as “nodes”, they are not actually nodes by definition, and thus are not called as such. To make sure I still knew what they were and remembered that they could be treated as nodes for most operations, I decided to call them noodles.

|  |
| --- |
| UI |
| +sectionName  -contents  -prompt  -commands |
| +UI(section)  +setContents(contentText)  +setCommands(prompt, \*\*kwargs)  +ShowUI(acceptCommands) |

|  |
| --- |
| NoodleMap |
| +edges  -Matrix |
| + NoodleMap()  - add\_edge(originNoodle, destinationNoodle)  -merge(left, right)  -mergeSort(array)  +loadCSV(filename)  +loadDatabase(TableName)  + dijkstra(startNoodle, endNoodle)  + returnMap(sort) |

## Design of integration

### Data Dictionary

Table name: \*domain name\* (this will be changed by the program)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Key | Data Type | Size | Required | Validation |
| AutoID | PK | Integer |  | Y | Auto Increment |
| OriginURL |  | varchar | 2048 | Y |  |
| Hyperlink |  | varchar | 2048 | Y |  |

The reason the varchars have a length of 2048 is that the max length of a URL is 2048 characters as it is the max old IE 8 could take, and thus has become more or less the standard.

### Query design

%s makes these values parameters that can be treated as variables

#### To create database

Create database Websites;

#### If a table does not exist for the domain

CREATE TABLE %s (

AutoID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

OriginURL VARCHAR(2048) NOT NULL,

Hyperlink VARCHAR(2048) NOT NULL);

#### If the table does exist and to delete table in general:

|  |  |
| --- | --- |
| DROP | Table name |

DROP TABLE IF EXISTS `%s`;

#### Inserting into tables

The original idea: To insert values into table with a specified domain name.

|  |  |
| --- | --- |
| INSERT | OriginURL, Hyperlink |
| TABLE | (stored domain) |
| Values | (URL),(found link) |

INSERT INTO %s (OriginURL, Hyperlink) VALUES (%s, %s);

However due to limitations of MySQL and the way it sanitises input made it impossible to pass a parameter into the table name without breaking the values to be inserted, while sanitising the values broke the table name. So now the way it is done:

First, we Store SQL as string

"INSERT INTO `"+ domain + "` VALUES (NULL, %s, %s);"

We concatenate table name into the query. While this does pose a threat of SQL injection, the user already has almost full control over the database already. I have not been able to find a way to sanitise this input without making it impossible to successfully store full domain names, as some contain characters that will be removed by the sanitising system provided with MySQL.

Then we execute the SQL and pass the parameters in.

#### To retrieve values from the table

|  |  |
| --- | --- |
| SELECT | OriginURL, Hyperlink |
| FROM | (domain) |

Select OriginURL, Hyperlink FROM %s;

#### Dropping all tables

SHOW TABLES;

Then we loop through all the results inserting the table names that have been returned as %s

DROP TABLE %s;

## UI Design

The UI for this project is intended to be simple and entirely executable through command prompt/ terminal. The idea behind having the two modes, is that if you simply use the terminal mode with no UI then you can pipe the output into a file, which documents all the results of the scrape, and the path or the map of the linked webpages. One of reasons I made this program have a very basic UI, is that this program can be integrated into another program as a module. By including a GUI this would bloat the program and require the user to install more dependencies, such as modules for GUI interaction. This would increase the size of the solution as all the modules would need to be packaged up for the user, and not only the parts that are being used.

### Execution through terminal with no generated UI

The purpose of the execution through the terminal without the UI is to allow for easy piping into programs or using this code as a module within another program.

This is the help screen that is displayed upon passing -h or –help to the program.



Note how the program has exited after this. The terminal mode requires correct input from the user to execute. If the user inputs the incorrect parameters, then an error message will be shown that shows the basic usage and then says error: unrecognized arguments: and what the user inputted.

Upon the user putting all the correct parameters, the first part of the Start-up is executed (see Pseudocode for Start Up).

The user types their commands into the terminal as is shown by the >\_

Actual execution for the pathfinder would be executed as such: Python dijkstra.py --mode pathfinder --reindex True --start <https://en.wikipedia.org/wiki/Computer_science> --end <https://en.wikipedia.org/wiki/Computer_architecture> --jumps 2

All these parameters will be passed into the start-up section and parsed into variables which will then be passed to the scraper and the objects required to execute the function.

Note that the dropping of all the tables is not available through command line due to it being too easy for the user to commit an action without confirming it.

### Execution through text-based UI in terminal

This is the UI that is shown upon start up, when no parameters are passed to the program.



The UI is generated by the showUi method of the UI class. In this case it is being generated by the mainMenu object. The UI will take in the users input and then check if their input is in the commands instance variable of the mainMenu object and call the associated function. If the command is not in commands then the program will ask to insert a valid option and it will show the main menu again.

The help function will display what the commands do.



The program will then quit without receiving user input.

If the pathfinder mode is selected the pathfinder() procedure is run, and the output is the path between two nodes.

If the returnMap mode is selected the sort() procedure is run, and the links found on each webpage are outputted.

If the deleteTables mode is selected, then the clearDatabases procedure is executed, and the program will find and delete all the tables stored in the database after user confirmation.

If the help mode is selected, the showUi() method of the helpScreen object is called, and the output is the help screen.

### Database connection error

If the connection to the database is unsuccessful, then the program will print out some common ways to fix the connection.



The idea of this is to help the user set up or fix any problems they may have. This was made after suffering through many hours trying to set up and fix MySQL with the correct authentication etc. I am hoping to make this as easy as is possible for the user, to make the code more robust and accessible.

## Pseudocode for the UI

This is the constructor method that initialises the properties and sets the section name instance variable to the parameter passed to it. The section name is more for help with debugging and if any new features were to be added.

CLASS UI()

### METHOD UI

METHOD UI(section)

        INITIALISE PUBLIC STRING sectionName

        INITIALISE PRIVATE STRING contents

        INITIALISE PRIVATE STRING prompt

        INITIALISE PRIVATE DICTIONARY commands

        SET ME.sectionName = section

    END METHOD

### METHOD setContents

Next this method is used to set the contents

    REGION setters

        PUBLIC METHOD setContents(contentsText)

            SET ME.contents = contentsText

        END METHOD

### METHOD setCommands

This populates the commands dictionary with the name of the command as the key and the name of the function that it needs to call as the value

        PUBLIC METHOD setCommands(prompt, kwargs AS DICTIONARY)

            ME.prompt = prompt

As kwargs is a dictionary, we loop through all the keys, with their respective array being stored in the value variable.

            FOR EACH key, value IN kwargs

                SET ME.commands[lowercase(key)] = value

            END FOR

        END METHOD

    END REGION

The first thing the show UI does is clear the screen and then display the contents of object.

### METHOD showUi

REGION getters

        PUBLIC METHOD showUi(acceptCommands AS BOOLEAN DEFAULT FALSE)

            CLEARSCREEN()

            INITIALISE userInput AS STRING

            SEND ME.contents TO DISPLAY

If the contents of the acceptCommands is True, then it receives input from the user, and then checks if it is in the commands. If it is, then the program uses the user input as the key for the dictionary. The value is then called as a function. This assumes that the setting of the instance variable containing the function name is valid. If the key is not in the dictionary, then it displays an error message.

            IF acceptCommands = TRUE

                RECIEVE userInput FROM KEYBOARD

                IF userInput IS IN ME.commands

                    CALL ME.commands[userInput.lower()]()

                ELSE

                    SEND "Please select a valid option. Press enter to continue" TO DISPLAY

                    WAIT FOR KEY

                    CALL ME.showUI()

                END IF

            END IF

        END METHOD

## Pseudocode for NoodleMap

Constructor for the class that instantiates the two instance variables. The second one, matrix is a 2D array, with a length of (0,0).

CLASS Noodlemap()

### METHOD Noodlemap

This is the constructor method that is called on initialisation.

    METHOD Noodlemap()

        INITIALISE PRIVATE DICTIONARY edges

        INITIALISE PRIVATE ARRAY OF ARRAY OF STRING matrix INITIALLY [[""]]

    END METHOD

### METHOD addEdge

Appends the parameters into the edge list. This is used for the Dijkstra method. The first parameter is the “webpage” while the second is the link found on it. Note that these only go in one way, the normal way to do these would be to make them bidirectional, but there is no guarantee that there is a link back to the previous webpage.

    PRIVATE METHOD addEdge(origin\_noodle, destination\_noodle)

        APPEND destination\_noodle TO edges[origin\_noodle]

    END METHOD

### METHOD loadCSV

This method was used during ongoing testing to load test data

METHOD loadCSV(self, filename)

Opens the filename passed as a parameter

        SET lines = OPEN filename IN READ MODE

This makes lines an array of the lines in the file name.

        SET lines = lines.split(new line)

as Python variables are hard typed, this is declaring a 2d array populated entirely by zeros

To see how this works in greater detail, see the loadDatabase() function

        SET cols\_count = 2

        SET rows\_count = len(lines)

        SET ME.matrix = [["" FOR x = 0 to cols\_count - 1]

                         FOR y = 0 to rows\_count - 1]

        SET innerloop = 0  # index of dimension 1

        SET outerloop = 0  # index of dimension 2

        FOR EACH singleLine IN lines

makes sure that there are no unnecessary spaces in the line by replacing and spaces with nothing.

            SET singleLine = singleLine.replace(" ", "")

            SET innerloop = 0

splits up the two arguments and removes any new line characters. It then loops through creating the matrix of the values by using the outerloop as the y value and innerloop as the x value

            FOR y IN singleLine.split(','):

                SET ME.matrix[outerloop][innerloop] = y

                SET innerloop = innerLoop + 1

            END FOR

            SET outerloop = outerloop + 1

        END FOR

        FOR index = 0 TO rows\_count:

This adds to the dictionary of edges by calling the addEdge() method

            CALL ME.addEdge(self.\_\_matrix[index][0], self.\_\_matrix[index] [1])

        END FOR

    END METHOD

### METHOD loadDatabase

This method loads the database contents into the matrix property.

    PRIVATE METHOD loadDatabase(tableName as string)

        INITIALISE domain AS STRING

        connect to database "websites"

        domain = format tablename to domain form

[first iteration]

IF domain not in result

            SEND “The table you have entered does not exist. Please try again" TO DISPLAY

            Exit program

[/first iteration]

[second iteration]

During testing it was found that this would cause a type error if there were no tables in the database as the program cannot check if a value is in an array if there is no array.

IF result = None and domain not in result

            SEND “The table you have entered does not exist. Please try again" TO DISPLAY

            Exit program

[/second iteration]

The design of this query can be seen in the design section.

        execute "SELECT OriginURL, Hyperlink FROM parameter" where parameter = domain and store result in 2D array called result

        INITIALISE cols\_count AS INTEGER = 2

        INITIALISE rows\_count AS INTEGER = len(result)

This redeclares the 2D array called matrix by creating an array of values = “”. This array will have a length of cols\_count. Then an array of those arrays will be created with the length of rows\_count. These combined creates an array of arrays which is a 2D array. The first parameter ends up being along the x-axis while the second is for the y-axis.

        ME.matrix = [["" for x = 0 to cols\_count-1] for y = 0 to rows\_count -1]

        INITIALISE innerLoop AS INTEGER = 0

        INITIALISE outerLoop as INTEGER = 0

This is then looping through the values from the SQL statement with an inner and outer loop and writing to the 2D matrix array.

        FOR EACH row IN result DO

            SET innerLoop = 0

            FOR each value in row DO

                SET ME.Matrix[outerloop][innerloop] = value

                SET innerLoop = innerLoop + 1

            END FOR

            SET outerLoop = outerLoop + 1

        END FOR

This then adds the matrix values into the edge property by calling the private method called addEdge() (see above)

        FOR index = 0 to rows\_count - 1 DO

            Me.addEdge(ME.matrix[index][0], ME.addEdge[index][1])

        END FOR

        CLOSE CONNECTION

    END METHOD

### METHOD dijkstra

This is a general Dijkstra’s algorithm, but it is changed to ignore weight values.

    PUBLIC METHOD dijkstra(initial, final\_destination)

        INITIALISE shortest\_paths AS DICTIONARY

        INITIALISE current\_noodle AS STRING = initial

Set is being used as it has an inbuilt function to check if it contains values.

        INITIALISE visited AS DEFAULT SET #this is being done as it has a method to check if it contains things

There is an exit condition within the WHILE loop as otherwise if the end point did not exist then an infinite loop would occur.

        WHILE current\_noodle NOT = final\_destination DO

            ADD current\_noodle TO visited SET

            INITIALISE destinations AS ARRAY INITIALLY ME.edges[current\_noodle]

            FOR EACH next\_noodles IN destinations DO

                IF next\_noodles is not in shortest\_paths

The hard-typed value is used here as the original algorithm that was used was made for finding paths between weighted nodes.

                    SET shortest\_paths[next\_noodles] = (current\_noodle, 1)

                ELSE

                    INITIALISE current\_shortest\_weight AS STRING

                    SET current\_shortest\_weight = shortest\_paths[next\_noodles][1]

                    IF current\_shortest\_weight > 1:

                        SET shortest\_paths[next\_noodles] =(current\_noodle,1)

                    END IF

                END IF

            END FOR

            INITIALISE possible\_noodle AS DEFAULT DICTIONARY

            FOR EACH noodle in shortest\_paths

                IF noodle not in visited

                    SET possible\_noodle[noodle] = shortest\_paths[noodle]

                END IF

            END FOR

            IF possible\_noodle is empty

                SEND message that there is no path TO DISPLAY

            END IF

This part will be implemented by using lambda functions to change the inbuilt min function in Python. The lambda function is necessary as we only want the minimum of index 1 and not the other indexes too.

            SET current\_noodle = smallest value between possible\_noodle and all the values at index one of all keys in the

        END WHILE

        INITIALISE path AS ARRAY OF DICTIONARY

        WHILE current\_noodle has values in it

            APPEND current\_noodle TO path

            SET next\_noodles = shortest\_paths[current\_noodle][0]

            SET current\_noodle = next\_noodles

        END WHILE

As the way the path has been appended means it would display in reverse without this.

        SET path = reverse path

        RETURN path

    END METHOD

### METHOD returnMap

This includes an optional variable to not sort. In a future version of this program it could be used to get values much quicker, rather than waiting to sort.

    PUBLIC METHOD returnMap(sort INITIALLY TRUE)

        INITIALISE unsorted\_list[] AS ARRAY

Allows for sorting of the key values as a 2D array cannot be passed to it.

        SET unsorted\_list = ARRAY of keys of ME.edges

        IF sort = TRUE

            INITIALISE sorted\_list AS DEFAULT DICTIONARY

Merge sort returns the sorted version of the array that is passed to it. It then creates a new dictionary and populates it in the order of the sorted list that has been returned.

            FOR EACH key IN ME.mergeSort(unsorted\_list):

                SET sorted\_list[key] = ME.edges[key]

            END FOR

            RETURN sorted\_list

        ELSE:

            RETURN ME.edges

        END IF

    END METHOD

### METHOD mergeSort

This is a recursive merge sort.

    PRIVATE METHOD mergeSort(array)

This makes sure that once the sub array is of length zero it doesn’t try and half it again.

        IF length of array <= 1

            RETURN array

        END IF

        INITIALISE left AS ARRAY OF STRING

        INITIALISE right AS ARRAY OF STRING

        INITIALISE counter AS INTEGER INITIALLY 0

This loop puts the first half of the values in the left array, and the second half to the right

        FOR EACH value IN array

            IF counter < (length of array) / 2 rounded down

                APPEND value TO left

            ELSE:

                APPEND value TO right

            END IF

            SET counter = counter + 1

        END FOR

Recursively calls the merge sort again.

        SET left = ME.mergeSort(left)

        SET right = ME.mergeSort(right)

Once we have arrays of length one, the merge will begin going up the stack.

        return ME.Merge(left,right)

    END METHOD

### METHOD merge

This next method begins to combine the left and right arrays going up the stack.

    PRIVATE METHOD merge(left, right)

        INITIALISE result AS ARRAY

This continues while both the left and right arrays have values.

        WHILE length of left NOT = 0 AND length of right NOT = 0

This loop is for being able to compare the letters one after each other in the case of letters being equal. The smallest length is being used to avoid index errors.

            FOR letter = 0 to (smallest between length of left[0] and length of right[0])

Here, we are always calling left[0] and right[0] as the first result is being popped out, so it’s no longer in the array. Letter is being used as an array, as strictly speaking string is an array of char.

                IF left[0][letter] comes before right[0][letter]

                    pop left[0] and append it to result

                ELSE IF right[0][letter] comes before left[0][letter]

                    pop right[0] and append it to result

If both have the same letters to the very end of the for loop, then the shorter one comes first. There is no possibility of the two being equal, as two keys in a dictionary cannot be equal.

                ELSE IF length of left[0] < length of right[0] and (smallest between length of left[0] and length of right[0]) - 1

                    pop left[0] and append it to result

                ELSE IF length of left[0] > length of right[0] and (smallest between length of left[0] and length of right[0]) - 1

                    pop right[0] and append it to result

                END IF

            END FOR

        END WHILE

Either left or right may have elements remaining, so we consume them (Only one of the following loops will be entered.)

        WHILE length of left NOT = 0

            pop left[0] and append it to result

        END WHILE

        WHILE length of right NOT = 0

            pop right[0] and append it to result

        END WHILE

    END METHOD

### METHOD insertSort

This insertion sort has not ended up being used as part of the program, due to the length of time taken being a few orders of magnitude larger.

 PRIVATE METHOD insertSort(unsorted\_list):

        FOR start\_value = 1 to lenght of unsorted\_list

            FOR current\_value = start\_value to 0 STEP -1

                FOR letter = 0 to smallest between length of unsorted\_list[current\_value] and length of unsorted\_list[current\_value-1]

                    IF unsorted\_list[current\_value][letter] comes before unsorted\_list[current\_noodle][letter]

                        SWAP unsorted\_list[current\_value] and unsorted\_list[current\_value-1]

The next line breaks the for loop. The logic behind this is that if a condition has been met, then there is no need to check the further letters and the loop breaks.

                        BREAK FOR LOOP

This If statement has many statements so each one will be explained individually. 1. First statement checks if the length of the unsorted\_list value stored at current\_value is lower than the one directly before it. 2. Second statement checks if the letters that are being compared are equal in stored in them. The last statement checks if this is the last letter that can possibly be checked (this is so that the two values will only be sorted by length in the chance that no other letters matched.) keep in mind that the last value of letter is dependent on the shortest word.

                    ELSE IF length of unsorted\_list[current\_value] < length of unsorted\_list[current\_value - 1] and unsorted\_list[current\_value][letter] = unsorted\_list[current\_value - 1][letter] AND letter = smallest value between length of unsorted\_list[current\_value] and length of unsorted\_list[current\_value]

                        SWAP unsorted\_list[current\_value] and unsorted\_list[current\_value-1]

                    END IF

                END FOR

            END FOR

        END FOR

        RETURN unsorted\_list

    END METHOD

END CLASS

## Pseudocode for Functions

This is the pseudocode for the functions that are called by the UI object, and allows for the difference in terminal calls and calls through the UI

### PROCEDURE pathfinder

This procedure is made to call the scraper with the required parameters while using the object.

PROCEDURE pathfinder()

    RECIEVE start FROM KEYBOARD

    RECIEVE end FROM KEYBOARD

SEND "Would you like to reindex the database? (y/n)” TO DISPLAY

The next line calls the scraper if the user specifies that they want the website to be reindexed.

[First iteration]

    IF first character of RECIEVE userInput FROM KEYBOARD = "y"

        CALL scraper.runScrape(start)

    END IF

[/first iteration]

[second iteration]

SET valid = False

WHILE not valid

SEND "Would you like to reindex the database? (y/n) \n" TO DISPLAY

GET FROM userInput FROM KEYBOARD

IF len(userInput) > 0

     IF userInput == "y"

         CALL scraper.runScrape(start)

          SET valid = True

     ELSE IF userInput = "n"

          SET valid = True

            END IF

     END IF

IF :

         SEND "Please input a valid option" TO DISPLAY

END IF

    END WHILE

[/second iteration]

    domain =  format start into domain form

this loads the now created/ updated database of URL’s into the noodles object.

    CALL noodles.loadDatabase(domain)

    SEND noodles.dijkstra(start, end) TO DISPLAY

END PROCEDURE

### PROCEDURE clearDatabases

This procedure drops all existing tables in the database, thus clearing the stored URLs.

PROCEDURE clearDatabases()

    mydb = connection to database

This gets all the tables in the database by looping through the results of SHOW TABLES

    execute "SHOW TABLES" and store results in array called result

[Second iteration] This if statement below was inserted after testing found that if there were no tables, the program would still try and display them and ask the user for input on whether to delete them or not.

    IF LENGTH of result = 0

        print("There are no websites cached on this system")

        CLOSE DATABASE CONNECTION

        quit()

END IF

    FOR EACH table IN result

        SEND table TO DISPLAY

    END FOR

SEND “Are you sure you want to delete all archived websites (y/n)?”

    GET userCheck FROM KEYBOARD

    IF first character of userCheck = "y"

This loops through the result array which contains all the table names, and concatenates them into the drop statement

        FOR EACH table IN result

            SEND "Deleting " + table + "..." TO DISPLAY

            exectute "Drop table if exists parameter" where parameter is table

            SEND table + " deleted"

        END FOR

        SEND "All cached Databases deleted." TO DISPLAY

    ELSE

Displays the main menu with all the options again. The connection to the database is closed to make sure there is no memory leak, and as we are not using the connection until the user selects an option from the main menu there is no need to keep it open.

        CLOSE CONNECTION

mainMenu.showUI()

    END IF

    CLOSE CONNECTION

END PROCEDURE

### PROCEDUTE sort

This procedure is used to get the results from the scraper, sort them, and then if the user wishes to, writes the output to file.

PROCEDURE sort()

    SEND "please enter the start page" TO DISPLAY

    RECIEVE start FROM KEYBOARD

[first iteration]

    SEND "Would you like to reindex the database? (y/n)" TO DISPLAY

    IF RECIEVE userInput from user = "y"

        CALL scraper.runScrape(start)

END IF

[/first iteration]

[second iteration]

SET valid = False

WHILE not valid

SEND "Would you like to reindex the database? (y/n) \n" TO DISPLAY

GET FROM userInput FROM KEYBOARD

IF len(userInput) > 0

     IF userInput == "y"

         CALL scraper.runScrape(start)

          SET valid = True

     ELSE IF userInput = "n"

          SET valid = True

            END IF

     END IF

IF :

         SEND "Please input a valid option" TO DISPLAY

END IF

    END WHILE

[/second iteration]

    SET domain = format start into domain form

    SEND "Loading database..." TO DISPLAY

    CALL noodles.loadDatabase(domain)

    SEND "Database loaded." TO DISPLAY

This next part loops until the user input is valid.

    SET validInput = False

    WHILE validInput = False

        SEND ""Do you wish to write output to file? (y/n)" TO DISPLAY

        GET write\_to\_file FROM KEYBOARD

        IF write\_to\_file = "y":

            SET validInput = True

            SEND "Please input the name of the file you wish to write output to" TO DISPLAY

            GET writeFileName FROM KEYBOARD

            SET openedFile = open writeFileName in write mode

            SEND "Sorting and writing to file." TO DISPLAY

If the user puts in a file name with the last 4 characters being “.csv” (thus being a csv file) the program adds the commas and makes sure that the output is correctly sorted.

            IF last 4 characters of writeFileName = ".csv"

As noodles.returnMap() returns a dictionary, we loop through all the keys, with their respective array being stored in the array variable.

                FOR EACH key AND array in noodles.returnMap()

                    CALL openedFile.write(key)

                    FOR value IN array

                        write to openedFile ", " + value

                    END FOR

                    write to openedFile("\n")

                    SEND key + ": " array TO DISPLAY

            ELSE:

                FOR EACH key AND array IN noodles.returnMap()

                    write to openedFile key + ": " + array" + newline

                    SEND key + ": " + array" TO DISPLAY

                END FOR

            END IF

        ELSE IF write\_to\_file = "n"

            SET validInput = True

            SEND "Sorting and printing to terminal." TO DISPLAY

            FOR EACH key AND array in noodles.returnMap()

            SEND key + ": " + array"

        ELSE:

            SEND "Please enter valid input." TO DISPLAY

        END IF

    SEND "Complete" TO DISPLAY

END PROCEDURE

### PROCEDURE quit

Does what it says on the tin. This quits the program with no errors.

PROCEDURE quit()

    quit program

END PROCEDURE

## Pseudocode for Start Up

Purpose of this is checking if program is running from command line arguments or not.

IF length of arguments passed in command line > 1

    IF mode argument is passed as "pathfinder"

        INITIALISE domain AS STRING

        SET domain = format startPage argument to domain form

        IF reindex argument = TRUE

            run the web scraper with parameters

            CALL noodles.loadDatabase(domain)

        END IF

        SEND noodles.dijkstra(startPage argument, endPage argument) TO DISPLAY

    ELSE IF mode argument = "returnmap"

        IF reindex argument = TRUE

            run the web scraper with parameters

            CALL noodles.loadDatabase(domain)

        END IF

        CALL noodles.loadDatabase(startPage argument)

        FOR EACH key and value IN noodles.returnMap()

            SEND key + ": " + value TO DISPLAY

        END FOR

    ELSE

        SEND "Command not recognised" TO DISPLAY

        QUIT PROGRAM

    END IF

ELSE

These lines handle the instantiation of both the noodles and mainMenu objects and then load the data into the mainMenu object by using the setter methods.

    INSTANCIATE noodles as new NoodleMap object

    INSTANCIATE mainMenu as new UI("mainMenu") object

INSTANCIATE helpScreen as new UI("help") object

CALL helpScreen.setContents("""Pathfinder: This will find the path between two webpages on a domain if they exist. The scraping of websites will often take a long time.

\nReturnMap: This will show all the pages that are linked to the given page. The number of jumps given will make this recursive, and will show all the pages that are linked to the linked pages and so on.

\nDeleteTables: This will delete all the cached URLs on this system.

\nThe reindex option is used to scrape a webpage. It clears all previously scraped data off the domain.""")

    CALL mainMenu.setContents('Welcome to PathFinder! To see help, type: help \n Options: \n pathfinder: Finds a path between two URLs \n ReturnMap: View all found links.')

    CALL mainMenu.setCommands(Pick option', pathfinder() procedure, returnMap() procedure, help() procedure, quit() procedure)

    CALL mainMenu.showUI()

END IF

## Pseudocode for Scraper

### PROCEDURE runScrape

This is the pseudocode for calling the scraper and writing the output into a database.

PROCEDURE runScrape(page INITIALLY "", jumps INITIALLY 0)

The next line checks if the max number of jumps has been modified from the default value or is invalid

    IF jumps <= 0

        SET validInput = False

        WHILE validInput == False:

            SEND "Please input the max number of jumps to be performed by the scraper" TO DISPLAY

            RECIEVE jumps FROM KEYBOARD

            TRY:

Tries to convert the user input to an integer. If a type error occurs then the input was not an integer and it needs to be received from user again

                convert jumps to int

                SET validInput = True

            EXCEPT ValueError:

                SET validInput = False

                SEND "Please input a valid positive integer." TO DISPLAY

            END TRY

END IF

  checks if the page has been passed as a parameter and if it hasn’t, gets input from user.

    IF page = "":

        SET website = input("Please input the website you wish to scrape: ")

    ELSE:

        SET website = page

    END IF

 Trims away anything that trails the first / and all references to http or https making it into a domain

    domain = format website to domain form

    PASS ALL REQUIRED START UP INFORMATION TO SCRAPER

The scraper returns a dictionary called dictOfUrl with the key being the page the links are found on, and the array being all the links found. This includes anchors, both relative and absolute links and php submissions. These are all handled later, so that only relative and absolute links are handled.

START UP SCRAPER

    CONNECT TO DATABASE

Drops the table if it already exists. Each domain is stored as a table.

    EXECUTE "DROP TABLE IF EXISTS `parameter`;" where parameter = domain

creates a table with the name of the domain being scraped.

    EXECUTE (“CREATE TABLE `parameter`(AutoID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY, OriginURL VARCHAR(2048) NOT NULL, Hyperlink VARCHAR(2048) NOT NULL);" where parameter = domain

backticks are used so that any character can be accepted aka the . in the url. The surrounding '' are used so that MySQL doesn’t mistake them for table references. This has been changed since original design. The original was just simply passing in domain as a parameter. This does however open the program up to SQL injection, however the user already has almost full control over the database, and they can only inject what they can already do.

    query = "INSERT INTO `"+ domain + "` VALUES (NULL, parameter, parameter);"

This is so that any values that have the domain appended later will also contain the domains transfer protocol. This is necessary to make the links useable and readable by the system.

    IF "https://" in website:

        domain = "https://" + domain

    ELSE IF "http://" in website:

        domain = "http://" + domain

    END IF

    SEND "Writing to database" TO DISPLAY

    FOR EACH originURL, hyperlinks IN dictOfUrl

        FOR EACH item IN hyperlinks

            IF item != "" AND item != "/"

ignores all same page anchor links as they will not have http in their name

                IF len(item) > 1 OR "http" IN item:

                    IF "http" in item

                        SET queryParameters = (originURL, item,)

                        EXECUTE query with parameter=queryParameters

if http is not in the item, / is the first char we can assume that it is a relative link

                    ELSE IF item[0] != '#' and item[0] == "/"

Appends the domain name to the relative paths to create a usable link.

                        SET queryParameters = (originURL, domain+item,)

executes the sql statement with the parameters place of parameter. This method also removes any SQL injection attempts from the parameters being passed through.

                        EXECUTE query with parameter=queryParameters

                    ELSE IF item != '#'

appends the domain name and a slash to relative paths that are using interactive link. Seems to be rare but some websites do have it

                        SET queryParameters =(originURL, domain+'/'+item,)

                        EXECUTE query with parameter=queryParameters

                    END IF

                END IF

            END IF

        END FOR

    END FOR

    CLOSE DATABASE CONNECTION

END PROCEDURE

## Trimming URL to domain form

For the program to work we need to be able to trim the given URL by the user to be in the form of a domain.

Originally, I planned just to delete everything after the first slash, but I realised that we need to remove the http or https to allow the program to write to system and allow the program to create full URLs from a relative hyperlink.

Let the URL be stored in url. So first we want to remove the https:// if it exists. So, we use the replace predefined function.

domain = url.replace(“https://”, “”)

Next, we remove the http:// if that exists:

domain = url.replace(“https://”, “”).replace(“http://”, “”)

This should leave us with a string that looks like “www.thisisanexample.co.uk/this/is/a/test.html”

Next, we would like to get everything before the first /. So, by splitting the string by the first / we would get two strings, and if we take the first one then we should have our domain!

domain = url.replace("https://", "").replace("http://", "").split("/", 1)[0]

So, when we put this in a function it ends up looking something like this:

FUNCTION trimUrl(Url)

domain = url.replace("https://", "").replace("http://", "").split("/", 1)[0]

    return(domain)

END FUNCTION

# Implementation

## Research and Skills Learnt.

When I set myself this project, I wanted to expand on my knowledge of how web scraping systems functioned, and how to implement a system that could get information from websites using Python. I was also interested in how pathfinding algorithms worked, especially after seeing mentions of them in books such as Mathew Lane’s Power up, a book explaining the mathematics behind video games.

For web scraping I ended up researching a module called Scrapy. I originally intended to use beautiful soup, due to experience, but I then realised that its scraping mechanism could end up aggressive and wasn’t suitable due to the large volume of pings it produced, and thus could be breaching the Computer Misuse Act of 1990. Beautiful Soup would also result in a lot of work needing to be done and wasn’t realistic in the timeframe that was set (see constraints). Thus, I decided to research and implement a scraper through Scrapy. I used the documentation, together with the beginner tutorials on the same website to learn the basics, as well as messing around on websites specifically made for scraping. I located websites to scrape on this thread:

<https://www.reddit.com/r/scrapinghub/comments/5p6gpz/web_sites_for_scraping_practice/> and used <http://books.toscrape.com/> most of all. This is where I discovered the need to check for domains, anchor links and other similar quirks. My research of what characters were possible in domains and URLs (so I could find how to sanitise input into the database) led me on to reading about the generic syntax of URLs (<https://tools.ietf.org/html/rfc3986> and <https://www.ietf.org/rfc/rfc1738.txt>), and how certain characters are used for passing information, similar to how a php GET works. I had to be careful to not remove any of these characters when formatting the URLS, as that would make the URL invalid and one would not be able to follow it.

For Dijkstra’s algorithm I had to look for an algorithm, not already finished code. I ended up finding an interesting article, that contained the basic algorithm and a method for creating nodes. The bulk of this page however was discussing the logistics of the travelling salesman, and how the difficulty of finding the shortest path increases as more nodes are added, as there are now more paths to be tested. Sadly, this website has been seemingly taken down, and I cannot find it in my history, as I was on a school issued laptop. However, the algorithm that I found used weighted nodes, to simulate distance between them. As I had no use for this kind of functionality, I had to remove it, and by doing so I learnt a lot about lambda functions, and how they can be theoretically passed to other functions that do not normally take functions as arguments. This can be seen in my code from when I use a lambda functions to get the minimum value from a 2-dimensional array where I use the lambda function to loop through all possible values of k in possible\_noodle[k][1]. Finding information on how to use lambda functions for this situation was quite difficult, as they are firstly not particularly common, and secondly, they are very rarely passed into predefined functions. I also ended up learning quite a bit about some nuances of MySQL, where I was stumped for quite a long time when trying to figure out how to make characters in URL’s not escape the strings (see continuous testing).

I also had to learn how to use source control such as Git, so I could maintain my project across multiple computers and operating systems. As I also enjoy messing around with Python outside of the school course, I also had to learn how to use Conda, a system that is used for environment management. It allowed me to isolate the packages I need for this project, set a definite Python version, and stop any external modules other than the ones I chose to be called from my program.

## All Required Modules

This is the contents of requirements.txt . All of these can easily be installed by the user by simple typing pip install -r requirements.txt

asn1crypto==1.3.0

atomicwrites==1.3.0

attrs==19.3.0

Automat==0.8.0

autopep8==1.5

bcrypt==3.1.7

certifi==2019.11.28

cffi==1.14.0

colorama==0.4.3

constantly==15.1.0

cryptography==2.8

cssselect==1.1.0

hyperlink==19.0.0

idna==2.8

importlib-metadata==1.5.0

incremental==17.5.0

lxml==4.5.0

more-itertools==8.2.0

mysql-connector==2.2.9

packaging==20.1

parsel==1.5.2

pluggy==0.13.1

Protego==0.1.15

py==1.8.1

pyasn1==0.4.8

pyasn1-modules==0.2.7

pycodestyle==2.5.0

pycparser==2.19

PyDispatcher==2.0.5

PyHamcrest==1.9.0

pyOpenSSL==19.1.0

pyparsing==2.4.6

pytest==5.3.5

pytest-runner==5.2

pywin32==227

queuelib==1.5.0

Scrapy==2.0.0

service-identity==18.1.0

six==1.14.0

Twisted==19.10.0

w3lib==1.21.0

wcwidth==0.1.8

wincertstore==0.2

zipp==2.2.0

zope.interface==4.7.1

**See Overleaf For Code.**

Dijkstra.py is responsible for the majority of the program’s functions.

Scraper.py handles all of the scrapers functionality and writing to database.

All the functionality of the program is documented in testing.

## Ongoing testing

This was taken from the GitHub issues page for this project, that I used to keep track of all major problems that crept up. Therefore, the terminology may not be quite correct, as the primary use of this was to make sure I remembered exactly what went wrong, and any ideas I had to fix it. I tended to only document large issues that would take longer than a single day to fix, as a way of remembering what to fix, and to remind myself of what I had tried, and what was yet to be done.

|  |  |  |  |
| --- | --- | --- | --- |
| **Component** | **Issue** | **Solution** | **References** |
| NoodleMap.Dijkstra() | The pathfinding algorithm only returns the first two nodes. This seems to be occurring as one of the loops isn't properly looping.  The loop counter is either bugged, or the exit condition is incorrect. | It was found that a code block was incorrectly indented, thus causing the exit condition to be True |  |
| Scraper credentials | The program keeps throwing mysql.connector.errors.NotSupportedError: Authentication plugin 'caching\_sha2\_password' is not supported whenever it tries to connect to the database. I have checked and confirmed that the credentials are correct, and the user has full access to the ports and is admin in the database. | I needed to specify for the program to use the auth\_plugin argument when connecting, as the default for the database is caching\_sha2\_password which is not currently supported by the Python plug in, so I needed to add auth\_plugin='mysql\_native\_password' to all the database connections. |  |
| Scraper database access | SQL statement in scraper throws this error  mysql.connector.errors.ProgrammingError: 1054 (42S22): Unknown column https://www.example.org/' in 'field list'  Might be to do with special characters existing in the URL causing syntax errors.  seems that the syntax for the table creation has been slightly off or the select query isn’t calling the right table  mysql.connector.errors.ProgrammingError: 1146 (42S02): Table 'websites.websites.'example.org'' doesn't exist  notice the extra ' around the example | The issue has been resolved.  The fix was done by wrapping the values in a backtick then an apostrophe.  query = "INSERT INTO `%s` VALUES (NULL, '`%s`', '`%s`')"  queryParameters = (domain, originURL, item,)  mycursor.execute(query % queryParameters)    mydb.commit()  Splitting the statement into a query and the values to be put into the query also seemed to help. Also, notice the mydb.commit() as this is necessary to push any changes done by an insert statement. | <https://dev.mysql.com/doc/refman/8.0/en/string-literals.html>  not exactly my issue but: <https://stackoverflow.com/questions/9628571/mysql-table-name-with>  <https://stackoverflow.com/questions/776123/does-mysql-allows-to-create-database-with-dot>  [https://stackoverflow.com/questions/20463333/mysqldb-Python-insert-d-and-s](https://stackoverflow.com/questions/20463333/mysqldb-python-insert-d-and-s) |
| This new method seems to work more often but is often crashes due to MySQL trying to sanitise the input and ending up trying to insert what it thinks is a table name into the value section. | query = "INSERT INTO \`"+ domain + "\` VALUES (NULL, %s, %s);"  ........  queryParameters = (originURL, domain+item,)  mycursor.execute(query, queryParameters)  mydb.commit()  The first line will append the domain into the query with backticks surrounding it, making sure that any . or other command characters are not read by the system. This could mean however that the user can close the backtick and put in malicious code, however after spending over a month on this issue, this will have to be the method that is used. The next lines then do the actual work and input the parameters in place of the %s, by using the inbuilt method of the mycursor object, which sanitises input. | <https://stackoverflow.com/questions/50557234/authentication-plugin-caching-sha2-password-is-not-supported> |
| NoodleMap.mergeSort() | Seemingly randomly becomes a dictionary upon reaching a length of 1 with a key and a value with neither being none. occurs on line 156 at the moment but will probably crop up elsewhere as well.  Traceback (most recent call last):  File "c:\Users\Owner\.vscode\extensions\ms-Python.Python-2019.11.49689\PythonFiles\ptvsd\_launcher.py", line 43, in <module>  main(ptvsdArgs)  File "c:\Users\Owner\.vscode\extensions\ms-Python.Python-2019.11.49689\PythonFiles\lib\Python\old\_ptvsd\ptvsd\\_\_main\_\_.py", line 432, in main  run()  File "c:\Users\Owner\.vscode\extensions\ms-Python.Python-2019.11.49689\PythonFiles\lib\Python\old\_ptvsd\ptvsd\\_\_main\_\_.py", line 316, in run\_file  runpy.run\_path(target, run\_name='\_\_main\_\_')  File "E:\anaconda\envs\wikigame\lib\runpy.py", line 263, in run\_path  pkg\_name=pkg\_name, script\_name=fname)  File "E:\anaconda\envs\wikigame\lib\runpy.py", line 96, in \_run\_module\_code  mod\_name, mod\_spec, pkg\_name, script\_name)  File "E:\anaconda\envs\wikigame\lib\runpy.py", line 85, in \_run\_code  exec(code, run\_globals)  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 271, in <module>  sort()  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 238, in sort  for x, y in noodles.returnMap().items():  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 99, in returnMap  return self.MergeSort(unsorted\_list)  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 138, in MergeSort  left = self.MergeSort(left)  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 138, in MergeSort  left = self.MergeSort(left)  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 143, in MergeSort  for key in self.merge(left, right):  File "e:\Users\Owner\Documents\Work\comp\wikigame\wikigame\dijkstra.py", line 156, in merge  if ord(left[0][letter].lower()) <= ord(right[0][letter].lower()):  IndexError: list index out of range | The issue was due to recursion attempting to concatenate into a dictionary before completion. Fixed by implementing concatenation in another procedure after executing the sort. |  |
| All of URL sanitation. | first letters are being cut off occasionally. and so are trailing parts of the URL. | first letters are being cut off occasionally. and so are trailing parts of the URL. |  |
| trailing slashes have seemingly been fixed, however there is an issue determining the difference between http and https when concatenating. | realised that the https and http can be taken from the URL passed by the user and then concatenated later. |  |
| At that point, the program was just Dijkstra’s algorithm with weights still intact. | Dijkstra was outputting (x, a) instead of expected output seems code is largely identical to the algorithm, and despite an error being spotted in variable names, the issue still wasn't resolved | turns out the return part of the function was not properly indented and therefore the while loop would not execute. Indentation has been fixed, and now pathfinder works as expected. |  |
| NoodleMap.insertSort() | Insert sort takes over an hour to sort any large sized scrape. According to Wikipedia, the time taken to conduct an insert sort is О(n2) comparisons. Therefore, in a webpage in excess of 500000 pages, it is very easy for this to take a ridiculous amount of time. This resulted in a single passthrough of an already scraped website take over 4 hours on my laptop. | Have changed to a merge sort, but left in the insert sort method, to show proof of understanding of Advanced Higher concepts. The merge sort is a lot faster, by orders of magnitude. |  |
| Scraper data base creation | The create table command seems to work, aka raises no errors, but when something tried to write to the newly created database, the tables has miraculously vanished. | So, it turns out that the way MySQL works is it sends the command to the database which then executes it.  mycursor.execute("DROP TABLE IF EXISTS `%s`;" % domain)  mycursor.execute("CREATE TABLE `%s`(AutoID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY, OriginURL VARCHAR(300) NOT NULL, Hyperlink VARCHAR(300) NOT NULL);" % (domain))  mydb.commit()  The reason this doesn’t work, is that when the create table is running, the table still exists as the delete hasn’t been completed, and thus it isn’t created, so once the table is dropped, nothing takes its place, and we cannot write to it.  This can easily be fixed by inserting a time.sleep(.25) This makes sure that the delete is executed, and the new table is created. |  |
| Scraper relative link parser | It turns out that relative links can use both ./ and ../ which are relative from the current directory, and the directory above respectively. This creates a problem during scraping, as we need to figure out the absolute address, and we do not necessarily know where the page the link is found on links to.  I have figured out that it is possible to figure out the absolute link by splitting the current page URL by the slashes and taking the part before the slashes and appending it to the relative link with the dots removed, will try implementing | The following implementation works. This was written on the back of a post-it note in the rain while waiting for a bus, but it works on the first implementation and seems to be quite efficient.  for next\_page in response.css('a::attr(href)'):  #this allows for the scraper to create absolute links from relative links  slashCounter = str(next\_page.root).count("../") #this is checking how many directories up the link goes  if slashCounter > 0: #if it is relative.  #splits the pages url by "/" from the right. The slashcounter + 1 is due to the first slash on the right just being the current directory and we want to move up one.  #the first item will be the trimmed URL to append to the relative link.  #we multiply the slash counter by 3 as there are 3 characters in "../" we take away one as we want to keep the last "/" so we dont have to insert one ourselves  #appends the found URL to the key which is the webpage it was found on  dictOfUrl[response.url].append(response.url.rsplit("/", slashCounter + 1)[0] + str(next\_page.root)[(3\*slashCounter) - 1:])  else:  dictOfUrl[response.url].append(str(next\_page.root))  yield response.follow(next\_page, self.parse) |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Scraper: appending to dictionary | The URL of some pages are ending with a slash, however the link to the same page will not. This means that the Dijkstra’s algorithm cannot find ­a path between them, as it is comparing the URLs as strings. | After some playing around, I realised that if a URL ends with a slash, entering it with a slash will also work. However, if a page does not have a slash, and there is a slash in the link, link will not work. Therefore, I decided to remove all slashes at the end of links, as it still works if used as a link. |  |

## References used during implementation:

### For checking fastest comparisons for strings:

<https://stackoverflow.com/questions/7571635/fastest-way-to-check-if-a-value-exists-in-a-list>

### For understanding how MySQL handles input with different characters.

<https://dev.mysql.com/doc/refman/8.0/en/string-literals.html>

not exactly my issue but: <https://stackoverflow.com/questions/9628571/mysql-table-name-with>

<https://stackoverflow.com/questions/776123/does-mysql-allows-to-create-database-with-dot>

[https://stackoverflow.com/questions/20463333/mysqldb-Python-insert-d-and-s](https://stackoverflow.com/questions/20463333/mysqldb-python-insert-d-and-s)

### For understanding how to loop through dictionary’s in Python

[https://docs.Python.org/3/tutorial/datastructures.html](https://docs.python.org/3/tutorial/datastructures.html) see 5.6 Looping Techniques

### For implementing arguments of unknown length in Python

[https://www.digitalocean.com/community/tutorials/how-to-use-args-and-kwargs-in-Python-3](https://www.digitalocean.com/community/tutorials/how-to-use-args-and-kwargs-in-python-3)

### For calling functions from dictionaries

A slight war broke out in the answers of how to make this readable and simple, but I took a slightly less robust option as the dictionary is only being populated by the programmer and thus we can assume the programmer enters the correct function names to call.

<https://stackoverflow.com/questions/9168340/using-a-dictionary-to-select-function-to-execute>

### For implementation of merge sort

<https://en.wikipedia.org/wiki/Merge_sort> This is using the top down list approach algorithm, which I then adapted to be able to compare strings.

[https://www.geeksforgeeks.org/ord-function-Python/](https://www.geeksforgeeks.org/ord-function-python/) together with <http://www.asciitable.com/> allowed me to create a way to compare strings based on their letters and numbers in their name

### For implementation of the argument parsing system for passing in parameters into code

[https://docs.Python.org/3/howto/argparse.html#id1](https://docs.python.org/3/howto/argparse.html#id1) this explains how to do the very basics of creating the parser object

[https://docs.Python.org/3/library/argparse.html](https://docs.python.org/3/library/argparse.html) I then used this to get a deeper understanding of how to use the parameters to show good help dialogue etc.

### For installation and setting up of MySQL database on my system.

[https://www.w3schools.com/Python/Python\_mysql\_getstarted.asp](https://www.w3schools.com/python/python_mysql_getstarted.asp) after the importing of the module, all the setting up and allowing ports, making sure all systems had correct credentials was largely done by trial and error, with the exception of the understanding how to use the auth plug in where the following website was used.

<https://stackoverflow.com/questions/50557234/authentication-plugin-caching-sha2-password-is-not-supported>

# Testing

## Test plan

### Test plan for Component Testing

#### Test plan for Component Testing for NoodleMap()

##### Dijkstra’s algorithm

Due to the nature of websites having a huge number of webpages, each with hundreds of URLs, it would be almost impossible to find a website where every link and page is known. So, to test the pathfinder, I created a small method to load a csv of “URLs” that contained many of the characters I had found during my research to make sure it worked. I also created a method to load the CSVs into any NoodleMap object (shown below):

def loadCSV(self, filename):

        lines = open(filename, 'r').readlines()

        cols\_count = 2

        rows\_count = len(lines)

        self.\_\_matrix = [["" for x in range(cols\_count)]

                         for y in range(rows\_count)]

        innerloop = 0  # index of dimension 1

        outerloop = 0  # index of dimension 2

        for singleLine in lines:

            # makes sure that there is no unnecessary spaces in the csv

            singleLine = singleLine.replace(" ", "")

            innerloop = 0

            # splits up the two arguments and removes any new line characters.

            for y in singleLine.strip().split(','):

                self.\_\_matrix[outerloop][innerloop] = y

                innerloop += 1

            outerloop += 1

        for index in range(0, rows\_count):

            # adds to the dictionary of edges

            self.\_\_addEdge(self.\_\_matrix[index]

                           [0], self.\_\_matrix[index][1])

##### Normal Data

This is the csv that is being loaded in for these tests:

test.arg                                   ,www.B.random/test#

test.arg                                   ,C

test.arg                                   ,E

test.arg                                   ,G

test.arg                                   ,A/finder

abcd                                       ,A/finder

acd                                        ,A/finder

abc                                        ,A/finder

ab                                         ,A/finder

www.B.random/test#                         ,www.B.random/test-results/passed.html?GET=

www.B.random/test#                         ,H

A/finder                                   ,www.B.random/test#

A/finder                                   ,www.B.random/test-results/passed.html?GET=

C                                          ,L

www.B.random/test-results/passed.html?GET= ,F

F                                          ,H

I                                          ,J

I                                          ,K

I                                          ,L

G                                          ,H

H                                          ,G

K                                          ,Y

G                                          ,Y

J                                          ,L

Which can be shown with an actual map:

To test the Dijkstra’s algorithm, I wrote a quick test to load that csv and return the path between test.arg and F.

noodles = Noodlemap()

noodles.loadCSV("UnitMap.csv")

print(noodles.dijkstra("test.arg", "F"))

The expected output from this would be ['test.arg', 'www.B.random/test#', 'www.B.random/test-results/passed.html?GET=', 'F'] or ['test.arg', 'A/finder', 'www.B.random/test-results/passed.html?GET=','F']

##### Exceptional data

For this we will use the same dataset as above, but deleting the link 'www.B.random/test-results/passed.html?GET=' to 'F'

Our expected output would be to receive a message saying that there is no path.

##### Return map

To test the sorts and writing to file, I have used the same map of nodes as for the normal data above, and thus had to write another function to test for these. This is almost identical to the actual sort() however it has been adapted to work with loading CSVs. This tests the NoodleMap.returnMap() method, as it is just formatting the output of it and writing it to file to let me see if it is functional.

def testingSort():

    noodles.loadCSV("UnitMap.csv")

    validInput = False

    while validInput == False:

        write\_to\_file = input("Do you wish to write output to file? (y/n) \n")

        cls()  # clears screen

        if write\_to\_file.lower() == "y":

            validInput = True

            writeFileName = input(

                "Please input the name of the file you wish to write output to \n")

            openedFile = open(writeFileName, "w")

            print("Sorting and writing to file.")

            # if the file is a csv then it will write as if it is a csv

            if writeFileName[-4:] == ".csv":

                # loops through the dictionary printing the key followed by a comma and then appends the values stored at that key

                #gets the resulting dictionary of the merge sort and then stores the key and array of the key into their respective variable by using item()

                # this loops through the keys in the result of noodles.returnMap() and stores the key's respective array in the array variable.

                for key, array in noodles.returnMap().items():

                    openedFile.write(key)

                    for value in array:  # this loops through the array associated with the key.

                        # puts a comma in to make it a csv

                        openedFile.write(", " + value)

                    openedFile.write("\n")

                    print("%s: %s" % (key, array))

            else:

                for key, array in noodles.returnMap().items():

                    openedFile.write("%s: %s \n" % (key, array))

                    print("%s : %s" % (key, array))

        elif write\_to\_file == "n":

            validInput = True

            print("Sorting and printing to terminal.")

            for key, array in noodles.returnMap().items():

                print("%s: %s" % (key, array))

        else:

            print("Please enter valid input.")

    print("\n Complete")

By utilising this stub, I can test the returnMap() method to check if it returns the expected values. I have designed the following test table.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Sorting “URLs” alphabetically using insertion sort | Call returnMap() to display data | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. |  |  |
| Enter invalid data for output prompt | Unsorted data and “test” to output prompt | Message appears asking user to use valid input |  |  |
| Display output and write to test.txt | Unsorted data, and “y” to output prompt, and test.txt for file name | Noodles.returnMap() called and sorted output returned. New file called test.txt is created with the contents of the output |  |  |
| Display output and write to test.csv | Unsorted data, and “y” to output prompt, and test.csv for file name | Noodles.returnMap() called and sorted output returned. New file called test.csv is created with the contents of the output, with the values being comma separated |  |  |
| Sorting “URLs” alphabetically using merge sort | Display output without writing to file | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. Merge sort will be called recursively and not infinitely loop |  |  |
| returnMap calls \_\_merge() | \_\_merge is called with the unsorted array of the dictionary keys as the parameter | merge() will recursively call itself until it is left with arrays/ lists of length one. Then \_\_mergeSort will be called recursively back through the call stack, merging these smaller arrays. The final merged array is returned |  |  |
| Use the sorted keys of the dictionary to sort the dictionary | Use sorted key array | Dictionary is now alphabetically sorted by key. |  |  |
| Enter invalid data for output prompt | Unsorted data and “test” to output prompt | Message appears asking user to use valid input |  |  |
| Display output and write to test.txt | Unsorted data, and “y” to output prompt, and test.txt for file name | Noodles.returnMap() called and sorted output returned. New file called test.txt is created with the contents of the output |  |  |
| Display output and write to test.csv | Unsorted data, and “y” to output prompt, and test.csv for file name | Noodles.returnMap() called and sorted output returned. New file called test.csv is created with the contents of the output, with the values being comma separated |  |  |

#### Test plan for connecting to database and executing commands.

These following tests check the individual database operations. This was done by copy and pasting the database connection and command code from my project and pasting it into separate Python scripts. I would then check the contents of the variables using break points or by executing select and show table statements to check the contents of the table/database respectively.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Connect to database | Attempt to connect to database | Correct credentials for database | Program acknowledges success |  |  |
| Incorrect credentials for database | Program displays error message |  |  |
| Load information from database into Noodles object | Read information from table | Table exists | Cursor executes select query successfully |  |  |
| Table does not exist, but other tables do | An error message is displayed to the user and the program exits |  |  |
| No tables exist in the database | An error message is displayed to the user and then the program exits |  |  |
| Load contents into noodlemap object | Loop through result and store into instance variable | All contents are written to the edges instance variable without duplication |  |  |
| Drop all tables | Get a list of all tables | Execute “show tables” | Get array of tables |  |  |
| Check if Length of array of tables is greater than zero | Length of array is greater than zero | Continue the program |  |  |
| Length of array is zero | Tell the user that there are no tables and quit the program |  |  |
| Loop through the array of table and drop found tables | Pass in the tables into the drop statement | Database is no longer populated by any tables |  |  |
| Writing to table | Concatenate table name into query | Table name | Concatenates table name into SQL query with ` surrounding it. |  |  |
| Input values into insert query and then execute | Values and query into mycursor.execute() | Query is sanitised and executed successfully. |  |  |

#### Test plan for component testing for UI

The UI is relatively simple and thus can be quite simply tested. I am just using the default start up to test the display and letting the calling of breakpoints show that the function calls are successful.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Display UI | Write contents to instance variable called contents | Contents of Main Menu | Saves contents to mainMenu object |  |  |
| Display contents of mainMenu | Previously saved contents | Prints Contents and waits for user input |  |  |
| Get user command | Get user input and execute required command | User inputs a valid command that exists | Procedure associated with command is called |  |  |
| User inputs a command that does not exist | User is asked to input command again |  |  |

#### Test Plan for clearDatabases()

This is simply checking if the logic of the commands is correct. To see the actual testing of the database, see Connecting to database and executing commands.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Connect to database | Check if connection to database is successful | Valid credentials are passed. | Connection is successful and program continues |  |  |
| Invalid credentials are passed, or the database is down | Connection fails, and an error message is shown. |  |  |
| Drop all tables | Get a list of all tables | Execute “show tables” | Get array of tables |  |  |
| Display tables and ask user if they want to drop all tables | User enters “y” | Continue |  |  |
| User enters “n” | Program quits |  |  |
| Check if Length of array of tables is greater than zero | Length of array is greater than zero | Continue the program |  |  |
| Length of array is zero | Tell the user that there are no tables and quit the program |  |  |
| Loop through the array of table and drop found tables | Pass in the tables into the drop statement | Database is no longer populated by any tables |  |  |

#### Test Plan for Scraper

These are the individual tests that were carried out for the components of the scraper module I made, before any full integrated tests were made

##### Database integration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if database is running | Connect to database | Valid credentials and database is running | Program continues but connection is shut down to save memory |  |  |
| Non-Valid credentials and database is running | Program quits with custom error message |  |  |
| Valid credentials and database is down. | Program quits with custom error message |  |  |
| Non-Valid credentials and database is not running | Program quits with custom error message |  |  |
| Create a new table with the domain name, dropping the table if it already existed | Attempt to get all tables and drop them tables | Database is populated with tables. | Program displays all tables and drops them |  |  |
| Database does not have any tables populating it | Program Says database is empty. |  |  |
| Attempt to create table | Just simple text as table name | New table appear in database |  |  |
| Complex table name with different symbols | New table appears in database with no errors |  |  |
| Drop Single table if it exists | Table exists | Database is dropped with no errors |  |  |
| Table does not exist | SQL statement is not fully executed as it is “IF EXISTS |  |  |
| Write to database | Write to newly created table | Write simple text to table | New record is written to the database |  |  |
| Write complex text with many characters to table | New record is written to the database, and all symbols appear correctly |  |  |

##### Scraping call

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Load all required parameters into scraper | Get number of jumps from user. | No input from user | Ask user for input |  |  |
| User inputs a number less than zero | An error message is displayed, and program asks for input again |  |  |
| Text is inserted | An error message is displayed, and program asks for input again |  |  |
| Initialise scraper object and load requirements into the **class** for when the object is initialised | Send website details, depth limit, and depth priority to scraper class | No errors, and class has been changed |  |  |
| Initialise object called process of the CrawlerProccess() class, and set it up with the ScraperWithLimit | Process is ready to be started |  |  |
| Run the scraper | Start the scraper | Connect to given website | No errors |  |  |
| Write found URLs to dictionary | Data being found | Dictionary is becoming populated, with no duplication |  |  |
| Follow found links and ignore other domains | Links found | No outgoing connections to unauthorised domains |  |  |

### Test Plan for Integration Testing

#### Test Plan for pathfinder() procedure

This procedure integrates the scraper and the addEdge(), loadDatabase(), dijkstra() methods of the noodleMap class. See design stage for detailed explanation of what this does.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if user wants to re-index | Get input from user to check if they wish to re-index | User inputs “y” | Scraper is started up |  |  |
| User inputs “n” | Scraper is not started up and program continues |  |  |
| User inputs anything else | Scraper is not started and user in asked to re-enter their answer |  |  |
| Scrape webpages | Run scrape if the user requested | User inputs y above | Call scraper.runStart(start)  With start being the users entered webpage. Scraper is started (see component testing for scraper) or test cases |  |  |
| Get path between webpages | Trim URL into domain form | Send users input to trimUrl() | Trim URL is called, and a string formatted t domain form is returned |  |  |
| Load table into noodles object | Pass trimmed URL into noodles.loadDatabase() | noodles.loadDatabase() is called with the trimmed domain as the parameter, and the database connection is successful. The required table is read and stored in noodles.\_\_edges |  |  |
|  | The trimmed URL that is being passed does not exist, as the user didn’t call the scraper and the page hasn’t been scraped before | noodleMap.loadDatabase() is called with the with the trimmed domain as the parameter. The program will check if the table exists. As it does not exist, the user will be told it does not exist, and the program will quit |  |  |
| Print result of noodleMap.Dijkstra() | Pass the users start input and end input into the method | Method is called and path is returned. It should then be printed. |  |  |

#### Test Plan for sort() procedure

This procedure integrates the scraper and the addEdge(), loadDatabase(),returnMap(), merge() and mergeSort() methods of the NoodleMap class. See design stage for detailed explanation of what this does.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if user wants to re-index | Get input from user to check if they wish to re-index | User inputs “y” | Scraper is started up |  |  |
|  | User inputs “n” | Scraper is not started up and program continues |  |  |
|  | User inputs anything else | Scraper is not started and user in asked to re-enter their answer |  |  |
| Scrape webpages | Run scrape if the user requested | User inputs y above | Call scraper.runStart(start)  With start being the users entered webpage. Scraper is started (see component testing for scraper) or test cases |  |  |
| Print sorted output of pages. | Display output without writing to file | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. Merge sort will be called recursively and not infinitely loop |  |  |
| Trim URL into domain form | Send users input to trimUrl() | Trim URL is called, and a string formatted t domain form is returned |  |  |
| Load table into noodleMap object | Pass trimmed URL into noodleMap.loadDatabase() | noodleMap.loadDatabase() is called with the trimmed domain as the parameter, and the database connection is successful. The required table is read and stored in noodleMap.\_\_edges |  |  |
| Sorting “URLs” alphabetically using merge sort (this is the one used, rather than insertion sort) | returnMap calls \_\_merge() | \_\_merge is called with the unsorted array of the dictionary keys as the parameter | merge() will recursively call itself until it is left with arrays/ lists of length one. Then \_\_mergeSort will be called recursively back through the call stack, merging these smaller arrays. The final merged array is returned |  |  |
| Use the sorted keys of the dictionary to sort the dictionary | Use sorted key array | Dictionary is now alphabetically sorted by key. |  |  |
| Enter invalid data for output prompt | Unsorted data and “test” to output prompt | Message appears asking user to use valid input |  |  |
| Display output and write to test.txt | Unsorted data, and “y” to output prompt, and test.txt for file name | Noodles.returnMap() called and sorted output returned. New file called test.txt is created with the contents of the output |  |  |
| Display output and write to test.csv | Unsorted data, and “y” to output prompt, and test.csv for file name | Noodles.returnMap() called and sorted output returned. New file called test.csv is created with the contents of the output, with the values being comma separated |  |  |

### Test Plan for Test Cases.

#### First Test Case

I will get the path between <http://books.toscrape.com/> and <http://books.toscrape.com/catalogue/twenty-love-poems-and-a-song-of-despair_91/index.html> when this page hasn’t been scraped before.

The steps that will need to be taken:

* Select pathfinder in the menu
* Input the start page
* Input the end page
* Confirm re-scraping
* Input number of jumps (three, as this is how many we know it should take.)
* Get output

~~I have confirmed that it is possible to get between these two pages in 4 jumps, might be possible to do in less.~~ After checking it is possible to do it in 3.

#### Second Test Case

I will get all the links on pages starting from <http://books.toscrape.com> assuming it has not been scraped before. Also write the output to books.csv.

The steps that will need to be taken:

* Select returnMap in the menu
* Input the start page
* Confirm re-scraping
* Input number of jumps (5 as we want a broader overview of the website.)
* Confirm writing to file
* Insert books.csv as name of file

#### Third Test Case

I will run the pathfinder to get the path between two pages after we have scraped the page previously. We want to find a path between <http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html> and <http://books.toscrape.com/catalogue/shakespeares-sonnets_989/index.html>

The steps that will need to be taken:

* Select pathfinder in the menu
* Input the start page
* Input the end page
* Confirm no re-scraping
* Get output

#### Fourth Test Case

I will run delete the cached webpages to free up storage.

The steps that will need to be taken:

* Select deleteTables in the menu
* See what tables there are on the system
* Confirm the deletion
* See confirmation

#### Fifth Test Case

Getting the path between <http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html> and <http://books.toscrape.com/catalogue/shakespeares-sonnets_989/index.html> from the terminal.

This is the command entered to get the path with the scraper active.

Python dijksta.py --mode pathfinder --start <http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html> --end [http://books.toscrape.com/catalogue/shakespeares-sonnets\_989/index.html --jumps 5](http://books.toscrape.com/catalogue/shakespeares-sonnets_989/index.html%20--jumps%205)

If this all works, then this shows that the terminal-based control system is functional.

### Testing Plan for Persona Testing

The test persona shall be Caroline. She is a newly graduated web developer, and always leaves orphan links in her webpage. She heard about this program through a discord server and is testing it out, as it will save her time in figuring out if her links work, and make sure she can land a new job by creating a portfolio that does not have any orphan pages. To do this she wants to see how all the pages link to each other. She also wants to make sure she can reach the contact page in at least 3 jumps from her home page. After she is finished she wishes to clear the cached webpages to clear up storage.

Tasks:

* View the help page
* Run the returnMap command with reindexing, and save to file
* Run the pathfinder command
* Delete the stored tables

### End User Testing Design

My end user testing is designed to test primarily the usability of my program. As the program is using a text-based UI, it can be used with read aloud systems, however this raises a problem with the scraper outputting way too much information (see Evaluation - further steps). As my program also uses terminal to display itself, one can use the options within terminal/ command prompt to change the colour scheme, to make it more suitable for colour blind users.

The people selected for the end user testing will be asked to complete three tasks, and complete a small survey for each one, and be asked how they feel the program could be improved.

I have chosen to do think aloud testing as this will let me know the thought process of the user and allow me to gauge their understanding of how to use the program after reading the help screen, and if the program is intuitive.

The three tasks will be:

* Get the path between <http://books.toscrape.com/catalogue/chronicles-vol-1_462/index.html> and <http://books.toscrape.com/catalogue/why-the-right-went-wrong-conservatism-from-goldwater-to-the-tea-party-and-beyond_781/index.html>
* Write the possible paths from <http://books.toscrape.com/catalogue/chronicles-vol-1_462/index.html> to a file called userTesting.csv.
* Delete all the cached webpages from their system.

They will be recommended to use the help command to do these, and will be given limited help by me, but this will be recorded so I can understand what parts of the instructions are unclear. I will talk them through the installation of the requirements of the program.

#### Survey

## Component Testing

The component test is similar to the integration testing as almost all the objects and functions communicate through the database, if the communication to the database is correct, then the procedures and methods will work.

Key:

Green background – Normal data

Yellow background – Extreme data

Orange Background – Exceptional data

### Component Testing for NoodleMap()

#### Testing using normal data

The test outlined in the Test plan for Dijkstra’s algorithm returns ['test.arg', 'www.B.random/test#', 'www.B.random/test-results/passed.html?GET=', 'F'] which is what was expected as despite there being multiple paths possible, so the program only needs to return **a** path.



#### Exceptional data

Our expected output would be to receive a message saying that there is no path for the exceptional data, as there is no path between these nodes.

The program returns No route can be found from test.arg to F, so we have received our expected results



#### Return map

By using the test data together with the adapted sort() procedure, and following the test plan, we can test if NoodleMap.returnMap() returns the correct sorted data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Sorting “URLs” alphabetically using insertion sort | Call returnMap() to display data | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. | --expected |  |
| Enter invalid data for output prompt | Unsorted data and “test” to output prompt. (aka invalid input) | Message appears asking user to use valid input | --expected |  |
| Display output and write to test.txt | Unsorted data, and “y” to output prompt, and test.txt for file name | Noodles.returnMap() called and sorted output returned. New file called test.txt is created with the contents of the output, and contents is also printed to screen | --expected |  |
| Display output and write to test.csv | Unsorted data, and “y” to output prompt, and test.csv for file name | Noodles.returnMap() called and sorted output returned. New file called test.csv is created with the contents of the output, with the values being comma separated | --expected |  |
| Sorting “URLs” alphabetically using merge sort | Display output without writing to file | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. Merge sort will be called recursively and not infinitely loop | --expected |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | returnMap calls \_\_merge() | \_\_merge is called with the unsorted array of the dictionary keys as the parameter | merge() will recursively call itself until it is left with arrays/ lists of length one. Then \_\_mergeSort will be called recursively back through the call stack, merging these smaller arrays. The final merged array is returned | --expected |  |
|  | Use the sorted keys of the dictionary to sort the dictionary | | Use sorted key array | Dictionary is now alphabetically sorted by key. | --expected |  |

### Connecting to database and executing commands

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Connect to database | Attempt to connect to database | Correct credentials for database | Program continues. | --expected |  |
| Incorrect credentials for database | Program displays error message | --expected, but the default error message gives too much information on the database such as name, ports etc… | Create custom error message to help explain to the user how to fix these common problems. See [second iteration] |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | [second iteration]  Incorrect credentials for database | Program displays error message | --expected |  |
| Load information from database into Noodles object | Read information from table | Table exists | Cursor executes select query successfully | --expected |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Table does not exist, but other tables do | An error message is displayed to the user and the program exits | --expected |  |
| No tables exist in the database | An error message is displayed to the user and then the program exits | Program crashes with type error | So it turns out that when there are no tables, the show all tables and then check if the domain is in them does not work, as if there are no tables then the SQL query returns None, which causes a type error (see figures 1 and 2):  if domain not in result:              print("The table you have entered does not exist. Please try again")              quit()  To fix this, we just need to add another condition:  if result == None or (domain,) not in result:  print("The table you have entered does not exist. Please try again")  quit()  The (domain,) is necessary as MySQL returns an array of arrays where the second dimension of arrays has length of 1 (weird I know, but this does the job of comparing it) |
| Load contents into noodlemap object | Loop through result and store into instance variable | All contents are written to the edges instance variable without duplication | --expected |  |
| Drop all tables | Get a list of all tables | Execute “show tables” | Get array of tables | --expected |  |
| Check if Length of array of tables is greater than zero | Length of array is greater than zero | Continue the program | --expected |  |
| Length of array is zero | Tell the user that there are no tables and quit the program | --expected |  |
| Loop through the array of table and drop found tables | Pass in the tables into the drop statement | Database is no longer populated by any tables | --expected  Before:  After: |  |
| Writing to table | Concatenate table name into query | Table name | Concatenates table name into SQL query with ` surrounding it. | --expected |  |
| Input values into insert query and then execute | Values and query into mycursor.execute() | Query is sanitised and executed successfully. | --expected  Before:    After: |  |

### Component Testing for UI

The UI is relatively simple and thus can be quite simply tested.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Display UI | Write contents to instance variable called contents | Contents of Main Menu | Saves contents to mainMenu object | --expected |  |
| Display contents of mainMenu | Previously saved contents | Prints Contents and waits for user input | --expected |  |
| Get user command | Get user input and execute required command | User inputs a valid command that exists  (in this case deleteTables) | Procedure associated with command is called | --expected |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | User inputs a command that does not exist | User is asked to input command again | --expected |  |

### Component Testing for clearDatabases()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Connect to database | Check if connection to database is successful | Valid credentials are passed. | Connection is successful and program continues | --expected |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Invalid credentials are passed, or the database is down | Connection fails, and an error message is shown. | --expected |  |
| Drop all tables | Get a list of all tables | Execute “show tables” | Get array of tables | --expected |  |
| Display tables and ask user if they want to drop all tables | User enters “y” | Continue | --expected |  |
| User enters “n” | Program quits | --expected |  |
| Check if Length of array of tables is greater than zero | Length of array is greater than zero | Continue the program | --expected |  |
|  |  | Length of array is zero | Tell the user that there are no tables and quit the program | --expected |  |
| Loop through the array of table and drop found tables | Pass in the tables into the drop statement | Database is no longer populated by any tables | --expected |  |

### Component testing for Scraper

These are the individual tests that were carried out for the components of the scraper module I made, before any full integrated tests were made

#### Database integration

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if database is running | Connect to database | Valid credentials and database is running | Program continues but connection is shut down to save memory | --expected |  |
| Non-Valid credentials and database is running | Program quits with custom error message | --expected |  |
| Valid credentials and database is down. | Program quits with custom error message | --expected |  |
| Non-Valid credentials and database is not running | Program quits with custom error message | --expected |  |
| Create a new table with the domain name, dropping the table if it already existed | Attempt to create table | Just simple text as table name | New table appear in database | --expected |  |
| Complex table name with different symbols | New table appears in database with no errors | --expected |  |
| Drop Single table if it exists | Table exists | Database is dropped with no errors | --expected  Before:    After: |  |
|  |  | Table does not exist | SQL statement is not executed as statement contains “IF EXISTS” | --expected    Try and drop a table that does not exist:    No errors occur. |  | |
| Write to database | Write to newly created table | Write simple text to table | New record is written to the database | --expected  Statement:    Result: |  | |
| Write complex text with many characters to table | New record is written to the database, and all symbols appear correctly | --expected    Result | Slightly strangely, the AutoID didn’t seem to increment correctly despite being auto incremented. This is the first time this has appeared, and I couldn’t seem to replicate it. | |

#### Scraping call

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | | **Further steps** |
| Load all required parameters into scraper | Get number of jumps from user. | No input from user | Ask user for input and await integer | --expected | |  |
| User inputs a number less than zero | An error message is displayed, and program asks for input again | --expected | |  |
|  | Text is inserted | An error message is displayed, and program asks for input again | --expected |  | |
| Initialise scraper object and load requirements into the **class** (not object) for when the object is initialised | Send website details, depth limit, and depth priority to scraper class | No errors, and class has been changed | --expected | These two are impossible to show, as the process object inherits over 148 attributes, which are used to change how the scraper works. Therefore, it is nigh impossible to show all of it. | |
| Instantiate object called process of the CrawlerProccess() class, and set it up with the ScraperWithLimit | Process is ready to be started | --expected |
| Run the scraper | Start the scraper | Connect to given website | No errors | --expected |  | |
| Write found URLs to dictionary | Data being found | Dictionary is becoming populated, with no duplication except for trailing slashes (done to speed up scraper) | --expected |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Follow found links and ignore other domains | Links found | No outgoing connections to unauthorised domains | --expected (done by checking through the dictionary, and no external domains were found) | Yet again, impossible to show, due to sheer volume of webpages being scraped |

## Integration Testing for pathfinder() procedure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if user wants to re-index | Get input from user to check if they wish to re-index | User inputs “y” | Scraper is started up | --expected |  |
| User inputs “n” | Scraper is not started up and program continues | --expected |  |
| User inputs anything else | Scraper is not started and user in asked to re-enter their answer | program continues without calling the scraper but, it just ignores the input | Go back and create user validation. See [second iteration] |
| [second iteration] User inputs anything else | Scraper is not started and user in asked to re-enter their answer | --expected |  |
| Scrape webpages | Run scrape if the user requested | User inputs y above | Call scraper.runStart(start)  With start being the users entered webpage. Scraper is started (see component testing for scraper) or test cases | --expected |  |
| Get path between webpages | Trim URL into domain form | Send users input to trimUrl() | Trim URL is called, and a string formatted into domain form returned | --expected    And the start string is changed to domain form |  |
|  | Load table into noodles object | Pass trimmed URL into noodles.loadDatabase() | noodles.loadDatabase() is called with the trimmed domain as the parameter, and the database connection is successful. The required table is read and stored in noodles.\_\_edges | --expected  Noodles.\_\_edges is now populated |  |
|  | The trimmed URL that is being passed does not exist, as the user didn’t call the scraper and the page hasn’t been scraped before | noodleMap. loadDatabase() is called with the with the trimmed domain as the parameter. The program will check if the table exists. As it does not exist, the user will be told it does not exist, and the program will quit | --expected |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Print result of noodleMap.Dijkstra() | Pass the users start input and end input into the dijkstra method | Method is called and path is returned. path should then be printed. | --expected  Call stack showing method being called:    Output: |  |

## Integration Testing for sort() procedure

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Check if user wants to re-index | Get input from user to check if they wish to re-index | User inputs “y” | Scraper is started up | --expected |  |
| User inputs “n” | Scraper is not started up and program continues | --expected |  |
| User inputs anything else including blank | Scraper is not started and user in asked to re-enter their answer | program continues without calling the scraper but, it just ignores the input | Go back and create user validation. See [second iteration] |
|  |  | [second iteration] User inputs anything else including blank | Scraper is not started and user in asked to re-enter their answer | --expected |  |
| Scrape webpages | Run scrape if the user requested | User inputs y above | Call scraper.runStart(start)  With start being the users entered webpage. Scraper is started (see component testing for scraper) or test cases | --expected |  |
| Print sorted output of pages. | Display output without writing to file | Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. Merge sort will be called recursively and not infinitely loop | --expected |  |
| Trim URL into domain form | Send users input to trimUrl() | Trim URL is called, and a string formatted into domain form is returned | --expected    And the start string is changed to domain form |  |
| Load table into noodleMap object | Pass trimmed URL into noodleMap.loadDatabase() | noodleMap.loadDatabase() is called with the trimmed domain as the parameter, and the database connection is successful. The required table is read and stored in noodleMap.\_\_edges | Noodles.\_\_edges is now populated |  |
| Sorting “URLs” alphabetically using merge sort (this is the one used, rather than insertion sort) | returnMap calls \_\_merge() | \_\_merge is called with the unsorted array of the dictionary keys as the parameter | merge() will recursively call itself until it is left with arrays/ lists of length one. Then \_\_mergeSort will be called recursively back through the call stack, merging these smaller arrays. The final merged array is returned | --expected  Proof that merge sort is being called recursively: |  |
| Use the sorted keys of the dictionary to sort the dictionary | Use sorted key array | Dictionary is now alphabetically sorted by key. | --expected |  |
| Enter invalid data for output prompt | Unsorted data and “test” to output prompt | Message appears asking user to use valid input | --expected |  |
| Display output and write to test.txt | Unsorted data, and “y” to  output prompt, and test.txt for file name | Noodles.returnMap() called and sorted output returned. New file called test.txt is created with the contents of the output | --expected |  |
| Display output and write to test.csv | Unsorted data, and “y” to output prompt, and test.csv for file name | Noodles.returnMap() called and sorted output returned. New file called test.csv is created with the contents of the output, with the values being comma separated | --expected |  |
| Unsorted data, and “n” to output prompt | Noodles.returnMap() called and sorted output returned. Merge sort will be called recursively and not infinitely loop | --expected |  |

## Error images



Figure 1

Figure 2 – the error that occurs when there are no tables in a database when checking if a table exists

## Test cases.

### Test Case 1

Getting the path between <http://books.toscrape.com/> and <http://books.toscrape.com/catalogue/twenty-love-poems-and-a-song-of-despair_91/index.html> when this page hasn’t been scraped before.

When the UI displays, we pick the option pathfinder:



Next, we input <http://books.toscrape.com/>



Now we input the page we want our path to terminate at and put in y for the re-scrape prompt:



We now put in an estimated number of jumps that will be needed.



This is some sample output during the scraping:



When the scraper is finished, it prints basic information, and then starts writing to the database.



The program finds a path in exactly 3 jumps! 

This proves that there is a quick way to get to this book from the home page in three jumps.

### Test Case 2

Get all the links on pages starting from <http://books.toscrape.com> assuming it has not been scraped before. Also write the output to books.csv.

Once the UI displays, we enter returnMap



Next we input our URL we want to scrape



Next we input “y” to re-scrape the website.



And we insert 5 as we want a broader outlook on the website.



Sample output of scraper: 

Now that the scraper has finished we wait for the write to finish. 

We input “y” as we want to write this output to books.csv



We input the name of the file.



The program has now sorted and written the output to file.

This is a small sample of the contents within the output. 

The reason there are duplicates, is that there are two of that URL on the webpage.

### Test Case 3

We want to run the pathfinder to get the path between two pages after we have scraped the page previously. We want to find a path between <http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html> and <http://books.toscrape.com/catalogue/shakespeares-sonnets_989/index.html>

The background and font. Colour has been changed to save ink during printing

First we pick pathfinder.



And now we put in the webpage we decided to start on and the one we end on



This time we do not want to re-index, so we input “n”



And we get the path!



### Test Case 4

Drop all cached pages.

We select to drop all tables.



We now see the tables that are stored and can confirm our choice.

All the pages are deleted



### Test Case 5

Getting the path between <http://books.toscrape.com/catalogue/a-light-in-the-attic_1000/index.html> and <http://books.toscrape.com/catalogue/shakespeares-sonnets_989/index.html> from the terminal.

This is the command entered to get the path with the scraper active.

Sample output 

The scraper has finished and is now writing to database

And we get the output!

### Conclusion

All test cases were successful, and no errors occurred. As expected output was achieved, there is evidence that the program has passed integration testing. As we can see that return map works successfully, we know that the scraper is successfully writing to the database and the data is being successfully loaded into noodle object which means that the database integration is working correctly. I have confirmed that the paths that are being given returned by the pathfinder are in fact correct by following the links to check if they exist on the page. The above 5 test cases show the proof that the implementation functions as intended as the scraper writes from the database, and the noodle object and the clearDatabases() function both read from the scraper, shoving that my integration has been successful.

## Testing with Persona

Caroline is a newly graduated web developer, and she always leaves orphan links in her webpage. She heard about this program through a discord server and is testing it as it will save her time in figuring out if her links work, and make sure she can land a new job by creating a portfolio that does not have any orphan pages. To do this she wants to see how all the pages link to each other. She also wants to make sure she can reach the contact page in at least 3 jumps from her home page. After she is finished she wishes to clear the cached webpages to clear up storage.

Once she has installed the program and set up the prerequisites, she starts up the database on her machine and inputs the credentials necessary into. She now starts up the program.



As this is her first time selects the help option to understand how to use it.





She reads the instructions and decides that she needs to see all the linked pages on her website. So, she starts up the program again and inputs “returnmpa” by accident when she fumbles the keys.





Caroline re-enters the command, and viola, it works.



Now she enters her hosted website: <http://quotes.toscrape.com/>



As she has not yet scraped the website, she selects “y” for the re-index.



She wants a full overview of her site, so she will select up to 10 jumps.



Once the scraper has finished, she selects “y” to print it to file, so she can see what pages link to what. She also decides to call the file links.csv so she can easily see them in a spreadsheet package of her choice.



When reading through the file, she thinks that it may be hard to get information on Dr Seuss, as she doesn’t have any links leading to his info page on her home screen. To check this she decides to find a path between the home page([http://quotes.toscrape.com](http://quotes.toscrape.com/author/Dr-Seuss/)), and the Dr Seuss page (<http://quotes.toscrape.com/author/Dr-Seuss/>), so she starts the program again and selects pathfinder.

Next she inputs the start and end webpage.



As she just indexed the database, she selects no on the re-index prompt.



The program finds the path and prints it.



This lets Caroline know that there is a path to the Dr Seuss information page, but it’s not easily accessible from her home page. This will let her go back and make her page more accessible.

As Caroline doesn’t want to use up unnecessary storage space on her computer, she decides to delete all the cached tables by using the deleteTables command.



As she only indexed the one page, there is only one webpage stored. She confirms that she does want to delete the webpage, so she inputs “y”.



The cached system is now deleted.



Thank you to scrapinghub (<https://scrapinghub.com/>) for giving explicit permission to use their website as an exemplar for the test persona and end users.

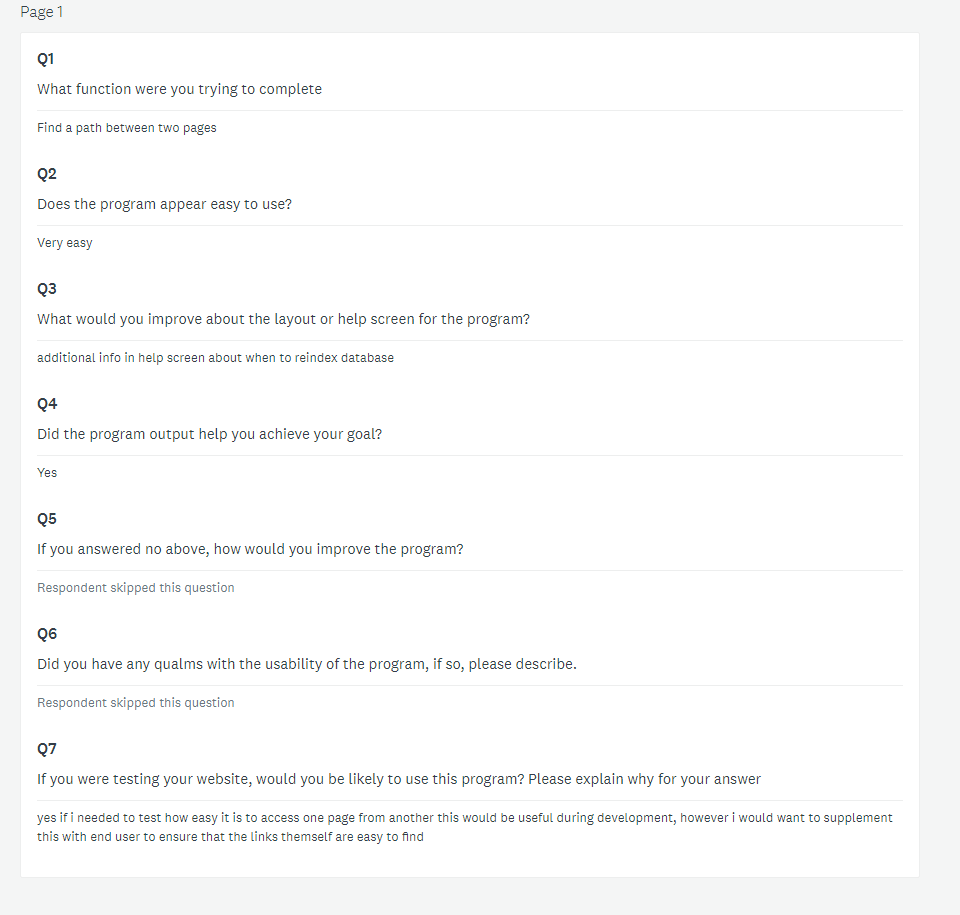


## End user testing

The end users gave me common feedback on a few points:

* Help is not conveniently displayed, so it is easy to not realise it is an option.
* Wording of the return map in the help function is not particularly clear.
* Program shouldn’t quit after displaying help.
* Explain what the re-index option does.
* Explain what the jumps are.
* Warn the user that writing to database takes a long time.

A sample survey that was taken:



A few suggestions that were given by one or two users:

* Tell the user that the program will change the output depending on file type specified.
* Make a clear installation guide as installation was quite lengthy and required a lot of input from me.

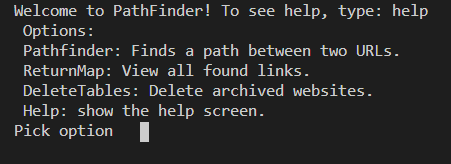
# Evaluation

## End user Testing Analysis

To recap, some of the common comments I received from the users were:

* Help is not conveniently displayed, so it is easy to not realise it is an option.
* Wording of the returnmap in the help function is not particularly clear.
* Program shouldn’t quit after displaying help.
* Explain what the re-index option does.
* Explain what the jumps are.
* Warn the user that writing to database takes a long time.

I have decided that it is easy enough to add the help to the main menu as an option, so it was impromptu added.

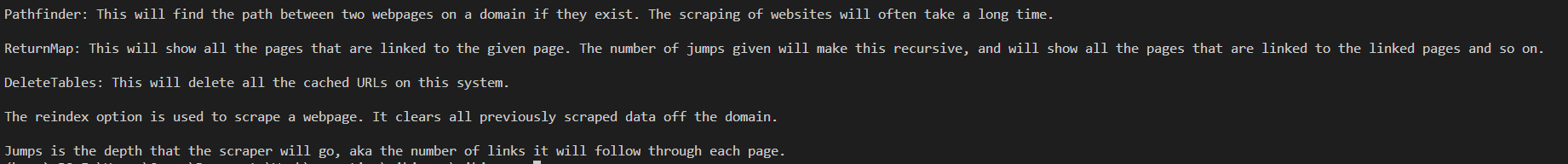


This makes it easier for the user to see the help option.

The wording of the help function should be reworked to something simpler and more self-explanatory and will be reworked during corrective maintenance.

A better explanation for the re-index should be added during Perfective Maintenance, as the wording needs to be reworked.

An explanation for what the Jumps is relatively simple and can be added immediately.



The option to warn the user that writing to database takes a long time is also easy to implement, and thus was changed immediately.



A detailed installation guide will be made during maintenance, however as it is not part of the advanced higher course, it will not be completed intime for the deadline, and it was not included in the specifications of the program.

Overall, end user testing showed that the program delivers on the planned functionality and works as expected. However, some elements of the UI such as how information is conveyed to the user could be improved to be clearer and/or more concise.

## Overall Analysis of Testing

For my end user testing I did not have a particularly large pool of test subjects, as it was difficult to find people with prior experience in web development. This meant that the opinions were relatively narrow for the demographic, and I ended up selecting people from the same group of university students, who had similar experience with web development, and thus had similar experience, and similar opinions. This means I didn’t get as good a range of testing as if I picked from different demographics. I also did not have any people with special needs test the program, which means that other than my personal evaluation of accessibility, there was no external input. This means there could be small oversights from my lack of knowledge in this topic.

I tried to document all the operations the program can do in the Test Cases, where it shows the steps a user would take to complete a task. This shows almost all the functionality of the program and proves functionality.

## Requirements Specification Evaluation.

### End User Requirements Evaluation

User must be able to use text-based UI to input a starting website, and an end website to find a path to.

The program displays a basic text-based UI that has options for each operation, which will ask the user for the required input that is needed for the mode that they select.

User must be able to input the maximum number of jumps while scraping a website for the first time.

When the user decides to scrape/re-scrape a webpage, the program asks them how many jumps the user wishes the scraper to do.

The user must be able to view the requested path or receive an error message that there is no path.

As shown during testing, the program will display a path between two pages if it is requested. If there are no jumps, then the program prints that there is no path.

The user must be able to view a help page from the text-based UI.

There is a help option in the main menu, and I also expanded on this by adding a help screen that displays common solutions for when the database is down.

The user must be able to view the pages that the spider finds without the pathfinder if they wish.

The returnMap option in the main menu returns the result of the scraper, whether it be stored or just scraper.

The found pages must be sorted alphabetically for readability, and the user must have the option to write the output to file.

The output of return map will be sorted alphabetically by a merge sort.

If the user wishes to save to file, and selects a csv file, make the output be comma separated.

The output of returnMap can be written to file, and if the filename ends with .csv then it will add a comma between all the values.

The user must be able to clear the stored websites, to free up storage.

The clearTables command loops through all the stored tables and drops them if the user confirms that they wish to delete them.

As can be seen above, my program matches the end user requirements, and even expands on them.

### Functional Requirements Evaluation

The program must display a text-based UI that can take in a starting page, end page(only if the path finding mode is selected), number of moves and mode (input).

This requirement is matched, but not exactly. The program displays a menu of all the operations that the it can complete, and then only asks for what it needs to execute it.

The program will be able to scrape a URL and find all links on the URL, follow them, and repeat the process until the maximum jumps is achieved.

The scraper does follow links but doesn’t loop into webpages it has already visited. It stops following links when it reaches the maximum depth (jumps).

The webpage’s URL is to be stored then all the links leading off also need to be stored in a database using SQL.

The domain is created as a table and the pages with the URLs leading off are saved as records. If the table already exists then it will be dropped, which makes sure there is no unnecessary duplication.

URLs need to be stored and retrieved in a database using SQL from within the program

The loadDatabase method in the NoodleMap class loads in databases, and checks to make sure the database exists, which makes sure the user input is a valid table.

Show the results of the Scraper if requested by the user

This is implemented in the returnMap method in the NoodleMap class.

Calculate a path between two URLs

The dijkstra method in the NoodleMap class calculates the path between two URLs, after the table has been loaded by the loadDatabase method. If there is no path then the program does not get stuck in an infinite loop, and instead tells the user that there is no path.

The output of the showing the sorted results of the scraper, can be saved in a file type of the user’s choice. If they are just showing the results and use a .csv extension, then the system will output a CSV.

The output of the sorted contents can be written to file. If the user requests a csv, then the values will be written in a comma separated fashion.

The program must be able to reuse an existing table of URLs

When the user inputs a URL, they are given the option to re-index. If they select not to re-index then the program will attempt to call the domain as a table name. If the table name does not exist, the user is notified, and the program exits.

The program needs to be able to drop all the tables in it.

The program can drop all the tables within it by utilising the clearDatabases procedure. It asks for confirmation of action before dropping the table.

The program matches almost all of the functional requirements, and creates a more user friendly version of “The program must display a text-based UI that can take in a starting page, end page(only if the path finding mode is selected), number of moves and mode (input).” The version that was implemented makes it easier for the user as they don’t need to input all the parameters even if some are unnecessary

## Robustness of Program

While I was trying to make my program more robust, I was looking for ways to make sure the scraper could be used in a way that does not print much output, and does not display port numbers, etc. During this, I realised that due to the complexity of the scraper I could not easily implement a method that would make the scraper more robust. Simple things such as entering a URL incorrectly will make the scraper return nothing, but still create a table, which could be potentially used to attack the machine hosting it by flooding it with requests. The scraper will also show information about the machine that it is being run on, some of which is normally not always available to normal users such as telnet password, and CPU architecture. This would need to be fixed during either corrective or perfective maintenance.

Otherwise my program is quite robust. For example, all database connections are encompassed in a try catch statement which catches the specific error that is caused if the Database connection is unsuccessful. The custom error message recommends simple fixes that the user could use to fix the connection.

The program also compensates for slower systems during the dropping and creation of tables within the scraper. Sometimes the creation of the table would finish executing in the database before the drop statement. This would often result in the program crashing as there would no longer be a table to write values to. This seemed to happen more often on a slower system, which I combatted by inserting a sleep of .25 seconds to make sure the operation would be completed on even the slowest of systems.

My program also checks if the database contains the requested table if the user tries to call from the cached websites. Without this the program would immediately crash whenever the user tried to call something non-existent from cache.

By creating a dictionary of functions in the mainMenu object, I have made sure that the user can only call the correct function, rather than any function of their choosing. This makes sure that the user’s input is an assigned function that exists.

Overall, I would say that my program is quite robust, as is detailed above. Perfective maintenance would be needed to patch the scraper to make sure it doesn’t display too much.

## Maintainability of Program

As the program is very modular, it was very easy to locate where bugs originate, making bug hunting quite simple. This combined with complicated parts of the code being heavily commented allowed me to fix logic errors and remember the use of small obscure code snippets. For example, the three lines that I used to create absolute links from relative links is syntactically simple, however the logic behind it is quite complicated, so there is a substantial number of comments to make sure anyone can understand it. This means that it was decided to try and optimise these lines (as specified in further steps), one could understand the original logic, and rewrite it.

As I also used objects for the display of the UI, it is easy to add new menus, allowing for easier perfective maintenance and allowing me to add new functions with ease. For example if I decided I wished for there to be another option on the main menu, it’s very easy to just add the name the user has to type, plus the function call, rather than going and creating many if statements to loop through all the possible commands.

I have explained all the steps that were taken to modify the URL into something that can be reliably read and compared, as they are relatively complex and don’t make much sense without explanation. This was done so that if any issues were to be found with how the URLs are modified it would be relatively easy to insert some new string manipulation to fix it.

I am also using a text file to store the credentials of the database, meaning that if something changes (e.g. username or password) the user doesn’t have to go through all the database connections and change all the values. This also makes it more maintainable on other users’ systems, as they can just put in the username and password that they used, rather than create one especially for this program.

Also, by using Argparse, it has made the viewing, and understanding of what the parameters do a lot easier and would also allow for easy additions of more parameters if the program was to be expanded to include additional functionality.

I would therefore evaluate my program to be very maintainable.

## Further steps

When I created this program, I intended it to be a tool that was easy to deploy and was not particularly bloated by modules. Due to this I decided to stick with a very simple, terminal based UI, so that I would not need to use any modules to display a more advanced GUI. The program was designed to be integrated as a module if the user wanted. This means that in the future I could carry out perfective maintenance and create a second program that displays a pretty UI but utilises my program to do all the scraping and path finding and returns the output to the GUI.

Another good idea may be to encrypt the credentials file. Currently the database credentials are stored in plain text. However, if the credentials are to be encrypted this would require programming a system to create and read encrypted files. This could be implemented during the Perfective maintenance phase. This may be disproportionate to the work required to implement it, as the user could limit the permissions of the user that is linked to the credentials.

When looking over how accessible my program was, I noticed a distinct issue. Although my program is almost entirely based around text-based UI, a screen reader could have issues due to the scraper. This adds another reason to make sure that the scraper does not spout a ton of output, and to rework the way the scraper works during perfective maintenance.

When looking over how accessible my program was, I noticed a distinct issue. Although my program is almost entirely based around text-based UI, a screen reader could have issues due to the scraper. This adds another reason to make sure that the scraper does not spout a ton of output, and to rework the way the scraper works during perfective maintenance.

During implementation I discovered that relative links would need to be converted into absolute links, therefore I wrote a quick bit of code for string manipulation. While necessary, this slows down the scraper, as it needs to process the string between following each link. This could be optimised during Perfective maintenance, where it could be optimised, perhaps by using regex or writing a small snippet in C (as this would be faster than Python and can be integrated)

Another change that could be implemented is to sort the values of the array, and not just the key during return map. This could be implemented during the Perfective maintenance, but this would add a lot of time to the execution of the program and might not be worth it.

Due to the way the domain is concatenated into the SQL statement, it is possible to insert malicious SQL code. This creates a security flaw in the program, allowing the user to edit the database. However, as the program already requires admin access and the user would need to input the credentials for the program to work in the first place, therefore it is safe to assume the user would already have be able to modify the database. However, it would be a good idea to fix this during corrective maintenance.

# Proof of Iteration

See overleaf for code that was taken from a previous commit of my project. This shows that the project has been changed over time, with errors being fixed, parts being optimised, and variables names being made more concise.

Proof that the project has been edited many times:

