[Date]

Maksim Livingstone

SQA AH COURSEWORK

URL path finder

Table of Contents

[Analysis 2](#_Toc32348510)

[Description of the Problem 2](#_Toc32348511)

[Requirements 3](#_Toc32348512)

[End user requirements 3](#_Toc32348513)

[Functional requirements 3](#_Toc32348514)

[Scope 6](#_Toc32348515)

[Boundaries 6](#_Toc32348516)

[Constraints 7](#_Toc32348517)

[Survey 8](#_Toc32348518)

[UML 10](#_Toc32348519)

[Project plan 11](#_Toc32348520)

[Identified tasks 11](#_Toc32348521)

[Resources required 11](#_Toc32348522)

[Estimate of timings 12](#_Toc32348523)

[Design 13](#_Toc32348524)

[Top level Flow chart 13](#_Toc32348525)

[Data structures. 14](#_Toc32348526)

[Pseudocode for the UI 15](#_Toc32348527)

[Pseudocode for NoodleMap 16](#_Toc32348528)

[Pseudocode for Functions 24](#_Toc32348529)

[Pseudocode for start up 27](#_Toc32348530)

[Design of integration 28](#_Toc32348531)

[Data Dictionary 28](#_Toc32348532)

[Query design 28](#_Toc32348533)

[Note to self 30](#_Toc32348534)

# Analysis

## Description of the Problem

I intend to design a program to find paths between webpages. The program will include the following features: a help screen, text-based UI to help user find their way, a database of webpages crawled that will be created using SQL, with a linked database of pages they have linked to and a pathfinding system.

The intended end users of this program will be people looking for orphan 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 print all the URL’s found on scraped webpages.

## Requirements

### End user requirements

TODO: change this to be more like actual program

* 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 number of moves they wish it to be done in.
* 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 without the pathfinder 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.
* 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 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(optional), number of moves and mode (input).
* The program will be able to crawl a URL and find all links on the URL, follow them, and repeat the process until the maximum jumps is achieved.
* 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.
* URLs need to be stored and retrieved in a database using SQL from within the program#
* 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 program, whether it be pathfinder or just showing the results, 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 program needs to be able to drop all the tables in it.

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 |  |

Generate UI

|  |  |
| --- | --- |
| Inputs |  |
| Processes |  |
| 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 |  |

### 

### 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.
* ~~Scope the clearly defined outline of what the solution will deliver in terms of functionality~~

### Boundaries

* Will not follow links onto a different domain.
* Max number of jumps in any direction will be 300 as too not slow down the computer.

UI will be text based only and not a graphical interface.

* 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 on
* The path will not always be the shortest as paths may be reused from previous scraping.

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 9th of March 2020 due to SQA requirements.
* This project can only use open source modules for data scraping and no commercial 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.

~~Constraints the restrictions that apply to the development.~~

### 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.

## UML

<https://www.lucidchart.com/invitations/accept/fcc0b091-b583-4fe0-82f1-93118e9e6ccf>

## Project plan

### Identified tasks

TODO: insert from gantt chart

### 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) |
| 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 official scrapy documentation and tutorial * Access to MySQL documentation. |
| Testing |  |
| Evaluation |  |

### Estimate of timings



# Design

## Top level Flow chart

<https://www.draw.io/#G1ClvWVKe-mLrl4Z8ucf_uFjW-MpjKrxc_>

## Data structures.

|  |
| --- |
| 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) |

## 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.

### 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.

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

TODO: finish modes.

### Errors

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



The program then exits.

## 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

Next this method is used to set the contents

    REGION setters

### METHOD setContents

        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 TRUE)

            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." 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 itteration]

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 can not 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()

    INITIALISE start AS STRING

    INITIALISE end AS STRING

    INITIALISE domain AS STRING

    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.

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

        CALL scraper.runScrape(start)

    END IF

    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

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

    if RECIEVE userInput from user = "y":

        CALL scraper.runScrape(start)

    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

### PROCEDURE help

See design of UI for the plan of help

PROCEDURE help()

    print help information

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

    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 on, 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(300) NOT NULL, Hyperlink VARCHAR(300) 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

## Design of integration

### Data Dictionary

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Key | Data Type | Size | Required | Validation |
| AutoID | PK | Integer |  | Y | Auto Increment |
| OriginURL |  | varchar | 300 | Y |  |
| Hyperlink |  | varchar | 300 | Y |  |

### 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(300) NOT NULL,

Hyperlink VARCHAR(300) NOT NULL);

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

DROP TABLE %s;

#### Inserting into tables

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

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 OriginalURL, 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;

# Implementation

## Research

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.

## Ongoing testing

|  |  |  |  |
| --- | --- | --- | --- |
| **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 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 isnt 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> |
| 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. |  |
| 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 atm 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. |  |

# Testing

## 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.

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

The method above loads this normal test data.

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']

The test result returns ['test.arg', 'www.B.random/test#', 'www.B.random/test-results/passed.html?GET=', 'F'] which is what was expected as the program only needs to return **a** path.

#### Exceptional data

For this we will use the same dataset as above, but not 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.

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

### 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 method to test for these.

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")

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement** | **Test** | **Input** | **Expected result** | **Actual result** | **Further steps** |
| Sorting “URLs” alphabetically using insertion sort | Display output without writing to file | 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 | 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 |  |
| 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 |  |
| 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 |  |

### 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 acknowledges success | --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] |
| Load information from database into NoodleMap | 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() |
|  | Load contents into noodlemap object | Loop through result and store into instance variable | All contents is written to the edges instance variable without duplication | --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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 | --expected |  |
| Database does not have any tables populating it | Program Says database is empty. | No tables are shown but the user is still asked if they want to delete them. | Create second iteration of delete table procedure which includes a check for an empty table. |
| 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 |  |
| Table does not exist | SQL statement is not fully executed as it is “IF EXISTS | --expected |  |
| Write to database | Write to newly created table | Write simple text to table | New record is written to the database | --expected |  |
| Write complex text with many characters to table | New record is written to the database, and all symbols appear correctly | --expected |  |

### Scraping call

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

TODO: this

Key:

Green background – Normal data

Yellow background – Extreme data

Orange Background – Exceptional data

## Error images



Figure 1

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

## Requirements Testing

TODO: this

## Testing with Persona and Test Cases.

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.

TODO: THIS

# Note to self

Show inputs etc on wireframe.

Don’t forget to do identified tasks

Suggestion: sort values as well as by key.