[Date]

Maksim Livingstone

SQA AH COURSEWORK

URL path finder

Contents

[Analysis 2](#_Toc31275769)

[Description of the Problem 2](#_Toc31275770)

[Requirements 3](#_Toc31275771)

[End user requirements 3](#_Toc31275772)

[Functional requirements 3](#_Toc31275773)

[Scope 6](#_Toc31275774)

[Boundaries 6](#_Toc31275775)

[Constraints 7](#_Toc31275776)

[Survey 8](#_Toc31275777)

[UML 10](#_Toc31275778)

[Project plan 11](#_Toc31275779)

[Identified tasks 11](#_Toc31275780)

[Resources required 11](#_Toc31275781)

[Estimate of timings 12](#_Toc31275782)

[Design 13](#_Toc31275783)

[Top level Flow chart 13](#_Toc31275784)

[Data structures. 14](#_Toc31275785)

[Design of integration 16](#_Toc31275786)

[Data Dictionary 16](#_Toc31275787)

[Query design 16](#_Toc31275788)

[Note to self 18](#_Toc31275789)

# 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 end users of my program will be people willing to find orphan links in webpages if a search system is not used, therefore will probably be tech literate.

My project meets the advanced higher computing requirements as it will have a UI suitable for tech literate users with validation for if the pages have valid URLs by using a try catch with a get() procedure and checking the code sent from the sever is not 404 and that the domain exists. 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 for a verbose mode to print the node maps.

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

### 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
* Depending on the mode that the user requested, the program will either display the contents of the database (aka the results of the spider) (sorted alphabetically or unsorted) or show the path between the requested URLS:
  + The program will then call the separate path finding algorithm.
  + SQL query’s results then need to be written into a 2D array.
  + (possible: sort 2D array so that easier to read.)
  + Then a node map object is created using the data in the 2D array.
  + Then using a pathfinding algorithm, find A path (not always shortest) from first link to second link in the maximum number of jumps given by the user.
* The user will be able to specify if they wish to re-index the database or reuse the existing one

Start Up

|  |  |
| --- | --- |
| Inputs |  |
| 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) as long as 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](http://www.ietf.org/rfc/rfc1738.txt) (which has now been superseded by RFC  3986) 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.
* 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

### 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

## 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) |

## Pseudocode

Pseudocode for Dijkstra’s Algorithm:

### 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 methods are 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

            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.

    REGION getters

#### METHOD showUi

        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

    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

        SET lines = lines.split(new line)

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

        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 is no unnecessary spaces in the csv

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

            SET innerloop = 0

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

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

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

                SET innerloop = innerLoop + 1

            END FOR

            SET outerloop += 1

        END FOR

        FOR index = 0 TO rows\_count:

            # adds to the dictionary of edges

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

        END FOR

    END METHOD

TODO: this

#### 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

        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

        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

                    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.

            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 its 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 actually 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

## 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 on the computer:

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;

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:

Store sql as string:

“INSERT INTO “ + tableName + “ (OriginURL, Hyperlink) VALUES (%s, %s);”

Then we execute the sql and pass the parameters in.

To retrieve values from the table

Select OriginalURL, Hyperlink FROM %s;

# Note to self

Show inputs etc on wireframe.

Don’t forget to do identified tasks

Suggestion: sort values as well as by key.