CSC326
Lab 1 Report
Development Phase 1

Group #: 4
Names: Anny Kim Ly (997581718)
Kevin Gumba (997585117)

Section 1: Frontend (by Anny Kim Ly)

1.0 Description of Front End

There are three functions in the Front End code:

- QueryPage()
  HTML and CSS is used to make the Query WebPage, containing the title and logo of our search engine, as well as an input area along with a search button. This function will then call on the keywords_db(), passing in an empty keywords list and a 1 to indicate that it is the query page that is calling this function. The top 20 most popular keywords searched will then be outputted as a table onto the Query WebPage. If there is less than 20, only those will be displayed, and if there are none (first run), then there will be no table.

- SearchPage()
  This function retrieves the user input from the Query WebPage and makes use of the string.split() function to tokenize the input into a list of keywords. This list is then passed to keywords_db(),along with a 2 to indicate that it is the search page that is calling this function. The function will then output a table containing the inputted keyword(s) in a table with their number of occurrences. If a phrase is entered, the function will also output the number of keywords listed in that phrase.

- keywords_db(keywords, page)
  SQL, HTML and CSS is used in this function. To run, 2 inputs are required: a list of keywords and an integer to indicate which function is its caller.

  If the integer is 1, that means the QueryPage is the caller, and this function displays the top 20 most popular keywords searched. If there is less than 20, than it will output what it has. If there was no previous search, no table will be displayed.

  If page is 2, then the caller is SearchPage. The function will then attempt to insert the input keywords as a new entry into the Keywords database with a count of 1. If that fails, then the keyword entry already exists, thus only the count needs to be increased by 1. The function will then display the input keywords in a table along with their counts. As well a button to return to the Query Page has been implemented. Once returned, the top keywords table is displayed.

1.1 Design Decisions

In addition to HTML and Python, CSS and SQL was used to complete Lab 1. The bonuses for the Front End has been completed.

CSS was incorporated into HTML for formatting and alignment purposes. It fulfilled the user-friendly aspect of the front-end design, as in the elements of the page react corresponding to the user resizing the window. CSS was used to center the text, input box and tables. The tools and commands used were ensured to produce an interface that had relatively consistent appearance across the different browsers.

SQL was utilized for storing the keywords and their number of occurrences (counts). This permits the program to keep the history of the data past a single run, thus allowing the completion of the bonus of storing and displaying the top 20 most popular keywords searched onto the query page.

1.2 Instructions to Run Code

To run the program, **simply type "python <file name>.py" in the command line** to start up the search webpage. In this case the file name is "SpaceExplorer.py". On execution of file, the query page can be accessed through "http://localhost:8080/". The user will be presented with a simple interface to submit a keyword or phrase. The query page will also present a table with a list of the top 20 searched keywords and their number of occurrences (bonus). Once the "search" button is clicked, the user will be redirected to the results page. There, they can then find a table with the inputted keyword(s) and its number of occurrences. If a phrase containing multiple keywords (separated by a sentence), the program will also output the number of keywords listed in that phrase.

Section 2: Backend (by Kevin Gumba)

2.0 Design Decisions

2.0.1 Storing data

There are 3 main data files created within the modified crawler.

- lexicon (variable name: lex)
    - contains word ID and corresponding word
    - one dictionary data structure
        - key: word IDs
        - values: corresponding words
    - implemented in word_id function
- document index (docIndex)
    - contains document ID and corresponding URL
    - one dictionary data structure
        - key: document IDs
        - values: corresponding URL
    - implemented in crawl function
- inverted index (invIndex)
    - contains word ID and document IDs containing that word
    - pne dictionary data structure
        - key: word IDs
        - values: list of corresponding document IDs
    - implemented in _add_text function

All three main data files use dictionaries since the data structure allow the use of keys, making quick insertion and retrieval of data. This is a a very useful feature for data handling in search engines since it reduces wait time for users. Moreover, since BeautifulSoup API creates unique word IDs and document IDs, there is no need to continually modify the key and value pairs once inserted. Therefore, data structures like lists are not required since its main use is for continual addition, modification, and deletion of data.

The tradeoff for using dictionaries is that the data structure does not have built in functions to modify stored data (such as list.sort() or append). Therefore, if needed, all values must be moved to another data structure, such as lists, and then be used for data handling. However, dictionaries allow lists as its value (as seen with our inverted index). Therefore, if needed, dictionaries can later be modified to include this feature.

The inverted index contains a dictionary with a list as its value since multiple document IDs can correspond to a single word ID key. A list in a dictionary allows for many document IDs to be appended without overwriting the previous values.

For the bonus mark, the URL's title and page descriptions are stored.

- URL title (variable name: docTitle)
    - contains doc IDs and corresponding URL title
    - one dictionary data structure
        - key: document IDs

        - values: URL titles
    - implemented in _visit_title function
- URL description (docParDescript)
    - contains sentences in the web page
    - one dictionary data structure with lists as value
        - key: document IDs
        - value: list containing texts contained in <p>, <P>, and meta tags
    - implemented in _curr_parag function

Two separate data structures are created for the bonus mark to allow for simplicity of code. Separate dictionaries allow for independence of data, decreasing the chance for errors when coding. Moreover, dictionaries allow the use of keys, making quick access to data.

The main tradeoff for creating two separate data structures for URL titles and descriptions is memory use. These information could have been stored in the same data structure as the document index. However, this would lead to more complex code which is prone to bugs. Additional data handling would have been required (ex. more for loops), possibly increasing the wait time for data retrieval. Also, since access to data is more of a priority than memory, this feature of having one main dictionary was not taken into consideration.

2.1 Instructions to Run Code

The main web pages used to crawl the web are crawler test pages. This was chosen for its simplicity, allowing for easy debugging of code. Use of more complex web pages may hide bugs and would make fixing code a lot harder. Once the basics of the crawler work on the test pages, for the next phase, a more complex URL could be used.

- urls.txt:
    - http://www.codedread.com/test-crawlers.html
    - http://www.york.ac.uk/teaching/cws/wws/webpage1.html

Two main functions are required in order for the user to access crawled data.
To run the function, **simply type "python <file name>.py" in the command line** to receive the output. Note that depth is set to 1 in the main function.

- crawler.get_inverted_index()
    - lab1tester1.py
    - Returns a dictionary
        - key: word IDs
        - values: set of corresponding document IDs
- crawler.get_resolved_inverted_index()
    - lab1tester2.py
    - Returns a dictionary
        - key: document IDs
        - values: set of corresponding URLs

For both functions, for loops are created to access all keys in the lexicon, document index, and/or inverted index.

The following are bonus mark functions to retrieve URLs' titles and/or descriptions.

- crawler.get_url_title()
    - lab1tester3.py
    - Returns a dictionary
        - key: URL
        - value: set of corresponding URL title
- crawler.get_url_description()
    - lab1tester4.py
    - Returns a dictionary
        - key: URL
        - values: set of corresponding strings inside web page

BeautifulSoup API is used in order to parse tags such as <title>, <p>, <P>, and meta tags for descriptions. As much usable strings are stored rather than limiting to just three strings so that more data can be utilized.

It was decided that our crawler will not use self._enter since it does not allow access to BeautifulSoup API. Also, when debugging, not all tags (<title>, <p>, <P>, meta) were recognized with self._enter functions. Therefore, a separate function is called within the crawl.