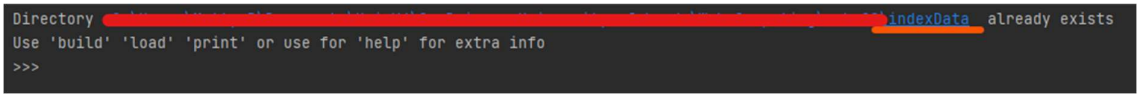
How to invoke and use the tool

There is a single python file to be run; it is called ‘invIndex.py’.

When run, the program will attempt to create a subdirectory called ‘indexData’; the system uses this to save the index data as the system generates the inverted indexes scraping the website. This is for visualisation purposes only to show how the index is built, and these files are **not** used to generate the final inverted index; this data is kept within the program whilst running. If the directory already exists, it will not create a new one and will instead use the ‘indexData’ directory

The program uses a command-line interface styling to interact with the user.



Crawling the website

Data structures

- ‘queue’ is an array that holds the URLs which have not yet been visited
- ‘visited’ is an array that holds the URLs which have been visited

Methods

When the user inputs build, the program will scrape the website. The program will retrieve all the links on a page. The system checks that the link does not contain ‘iso’ or ‘edit’ or ‘login’ as any link containing these is redundant. If the link does not contain these, it will then look to see if it has already been visited or whether it is already in the queue; if not, it will add it to the queue.

It will then analyse the text within the page. First, it removes numbers and any punctuation. It then splits the text by ‘ ‘ (white-space). A dictionary of words is created, and the frequency of occurrence is recorded.

This dictionary is then passed upwards and added to the unsorted array of documents.

Once the queue is empty, the application will then create the complete inverted index.

Algorithms

- build, calling function which starts the process
- CrawlSite, used to look at the website until queue is empty.
- CrawlPage, used to analyse individual pages, extracts the links, adds them to the queue, and extracts text.
- AnalyseText, used to analyse the text within a page

Creating the inverted index

Data structures

- Documents, a dict that contains the text dictionaries for each page. It uses the doc name (e.g. doc_123) as the key.
- pageIndexer, a dict that contains the URLs for each doc. For example, it could look like
  { ‘doc_123’: ‘/places/france’, ‘doc_122’: ‘/places/germany’ }

The structure looks like this:

```
{
        indexes : {}
        data : {}
}
```

Example of inverted index:

```
{
        "indexes": {
                "doc_0": "/places/default/view/Afghanistan-1",
                "doc_1": "/places/default/view/Albania-3",
                "doc_2": "/places/default/view/American-Samoa-5",
        }
        "data": {
                "flag": [
                        ["doc_0", 1],
                        ["doc_1", 1],
                        ["doc_2", 1],
                        ["doc_3", 1]
                ]
        }
}
```

Methods

The program will create dictionaries for each page it visits.
Once all pages have been visited, the inverted index is created. It will invert the document dictionaries and will instead represent which documents contain specific words.

The inverted index uses the pageIndexer to save which document represents which page/URL. The inverted index data dictionary is within 'data'.

Once created, the inverted index is saved as 'InvertedIndex.txt'

This can then be loaded into the program by using the 'load' command.

Computing the scores of pages when processing a search query

Data structures

- validWords, contains all of the users search terms which could be found in the invertedIndex.
- ranking, contains the search results for the query

Methods

find, and **print** commands are treated differently, but they begin the same

- It will first check whether the invertedIndex has been loaded. If not, it will inform the user.
- The program then takes the users input and attempts to find a match with the dictionary keys
- If a match is found then it is added to the 'validWords' array

print

- print will then retrieve the document names from the invertedIndex, it will also present the user with the frequency of use.
- An example of this can be seen below

```
>>>print Peso
+---------+-----------+
|   Doc   | Frequency |
+---------+-----------+
|  doc_8  |     1     |
| doc_69  |     1     |
| doc_73  |     1     |
| doc_140 |     1     |
| doc_198 |     1     |
| doc_203 |     1     |
| doc_221 |     1     |
| doc_239 |     1     |
+---------+-----------+
Use 'build' 'load' 'print' or use for 'help' for extra info
>>>
```

find

- The find function will loop through all of the valid words and add each document within to a dict called ranking
- Each entry is held under the document name and contains
    - The URL to document
    - A counter is used to count the number of search words found on the page
    - And frequency, which is a sum of the total number of occurrences of all of the words
- Once looped through all of the valid words, the dict is then sorted by the number of search words each page contains and the frequency of the words (both in descending order).
- An example can be seen below using the term 'find Euro eu fr'

```
>>>find Euro eu fr
Searching...
+---------+-----------------------------------------------------------------------------------------------+-------------+-----------+
|   Doc   |                                              URL                                              | Words Found | Frequency |
+---------+-----------------------------------------------------------------------------------------------+-------------+-----------+
| doc_173 |             http://example.python-scraping.com/places/default/view/France-74                  |   3 of 3    |     4     |
| doc_21  |             http://example.python-scraping.com/places/default/view/Greece-84                  |   2 of 3    |     2     |
| doc_24  |             http://example.python-scraping.com/places/default/view/Italy-105                  |   2 of 3    |     2     |
| doc_27  |           http://example.python-scraping.com/places/default/view/Luxembourg-125               |   2 of 3    |     2     |
| doc_29  |             http://example.python-scraping.com/places/default/view/Monaco-141                 |   2 of 3    |     2     |
| doc_54  |            http://example.python-scraping.com/places/default/view/Mayotte-137                |   2 of 3    |     2     |
| doc_81  |             http://example.python-scraping.com/places/default/view/Belgium-22                 |   2 of 3    |     2     |
| doc_93  |            http://example.python-scraping.com/places/default/view/Vatican-239                |   2 of 3    |     2     |
| doc_109 |          http://example.python-scraping.com/places/default/view/French-Guiana-75              |   2 of 3    |     2     |
| doc_137 |            http://example.python-scraping.com/places/default/view/Reunion-178                |   2 of 3    |     2     |
| doc_175 |             http://example.python-scraping.com/places/default/view/Spain-208                 |   2 of 3    |     2     |
| doc_200 |           http://example.python-scraping.com/places/default/view/Guadeloupe-87               |   2 of 3    |     2     |
| doc_206 |        http://example.python-scraping.com/places/default/view/Saint-Barthelemy-182           |   2 of 3    |     2     |
| doc_208 |    http://example.python-scraping.com/places/default/view/Saint-Pierre-and-Miquelon-187       |   2 of 3    |     2     |
| doc_230 |          http://example.python-scraping.com/places/default/view/Saint-Martin-186             |   2 of 3    |     2     |
| doc_232 |    http://example.python-scraping.com/places/default/view/French-Southern-Territories-77      |   2 of 3    |     2     |
| doc_254 |           http://example.python-scraping.com/places/default/view/Martinique-134              |   2 of 3    |     2     |
| doc_10  |             http://example.python-scraping.com/places/default/view/Austria-15                |   1 of 3    |     1     |
| doc_18  |             http://example.python-scraping.com/places/default/view/Estonia-69                |   1 of 3    |     1     |
| doc_19  |             http://example.python-scraping.com/places/default/view/Finland-73                |   1 of 3    |     1     |
| doc_20  |             http://example.python-scraping.com/places/default/view/Germany-81                |   1 of 3    |     1     |
```