build a search tool

COURSEWORK 2

SC17DPC@LEEDS.AC.UK

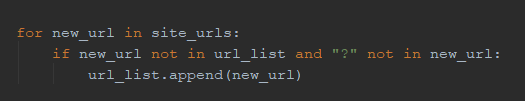
2201101857

Daniel Perez Cascon

Crawling the Site

The crawler is fed an initial url and then checks the robots.txt file. The file is parsed to allow us to find out which files are disallowed and which we can access.

Before each attempt at crawling, we ensure that the url we are about to crawl is not in the disallowed list that we formed using the robots.txt file.

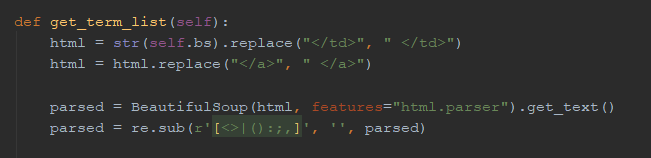
Our tool finds the URLs that are contained in the websites crawled and adds them onto a list of URLs if they are not in said list already. If the URL contains a “?” symbol, we assume that the query will redirect us to another url on the website – therefore we don’t add it onto the URL list. We will traverse this list until the end is reached and all URLs have been crawled This is shown below:

The for-loop loops over all the URLs found (which are returned by the parser using the get\_urls() function) and ignores any url which contains a query. Furthermore, it also ignores URLs which are already in the URL list.

The parser will fetch all the URLs on the current website by using Beautiful Soup and parsing for any <a> tag. Once a list of <a> tags is found we will parse for href= to allow us to find the actual path of the link in the tag. We then generate a list of the new URLs found and return it to the main process to be appended to the main URL list.

Creating the Inverted Index

The inverted index is created using a Parser class. Our parser class has 2 main functions, get\_urls() and get\_term\_list(). The way in which get\_urls() works is explained in the previous section.

The original way in which get\_term\_list() worked was that it would take the HTML of the page and parse the text to find strings. However, we found that the way the pages were formatted did not give spacing between items found in <td> and <a> tags. This meant that using the above method we would get a large amount of words being concatenated when they did not need to be. To solve this issue, we modified the original HTML to find all the above-mentioned problematic tags and replaced them with a version of the tag which had a space in it. As seen below:

We then used Beautiful Soup to parse the HTML of the page and got all the strings within the page. This set of terms was then analysed to see how frequently the occurred in the page and then both the lists of term frequency and of terms were returned to the main function.

When this process was completed for all URLs we used a file\_manager class to store our necessary information into a txt file. The information stored was 1) A list of URLs we had visited, 2) A list of the words that were contained in the documents, this list was made up of a dictionary which used the words as the key and the values stored under said key were tuples made up of (document index, word frequency in said index). 3) The raw information on the websites crawled including lists of the terms on the website in the order they appeard in.

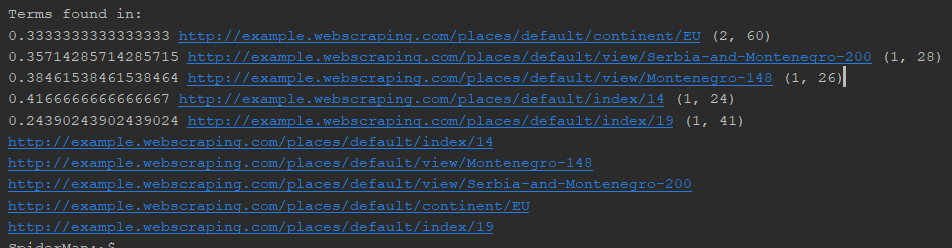
Before the data was stored to the txt file we had to compile the data. This data was compiled as a dictionary with the keys being the URLs and the values being a list of 2 lists, the first list being a list of tuples containing the term and its frequency and the second list being a list of terms in the order they appeared on the web page.

Computing Search Queries

Our algorithm lets us compute scores differently for single word queries than for multiple word queries.

For multiple word queries we query the inverted index and find the documents where each term in the search query appears, we then compare the returned lists together and find the documents where all the terms appear. We then check the frequency of each of the query terms in each of the documents which contain all the terms. The frequencies are summated and organised from highest to lowest and then the URLs associated with the frequencies are displayed to the user in size order. This is stored in a dictionary with the key being the URL and the frequencies being the values.

For single searched words we do the same, but we also take into account the first instance of the search query. We use an equation where the frequency is divided by the index of the first occurrence of the searched term in the document. This gives us a higher value for documents which have a higher frequency and where the searched term appears earlier on. This system is shown below. With the output showing the score, URL and a tuple containing the frequency and the index of the term.



Tool Usage

The modules: requests, time, bs4, re, json and operator will have to be present in the environment before the tool can be run. This can all be done with the pip package manager.

The tool is run in python so once launched using python/python3 with the main.py file we will be able to run the commands.

Commands:

load: Will load the inverted index and other required lists from reversed\_index.txt

build: Builds the reversed index and saves it to the reversed\_index.txt file

print term: Prints the location and frequency of a given term in the reversed index

find term1 [term2] [term3] : finds the location of the terms provided and ranks them