COMP 1405 Intro to Computer Science I Fall 2021

Final Project: Implementation of Search Engine

Karan Patel

Department of Computer Science, Carleton University

Comp 1405

David Mckenney

Nove,ber 3rd, 2021

crawler.py Functions:

crawl(seed): This function is responsible for crawling the web starting from the given seed. This function acts more like a central command where it calls and coordinates all the operation functions smoothly. PLEASE note that from the approximately 130s it takes to run this function(fruits 1 - 5 Test Case), 100s is just to read URL and the other 30s is for the actual calculation.

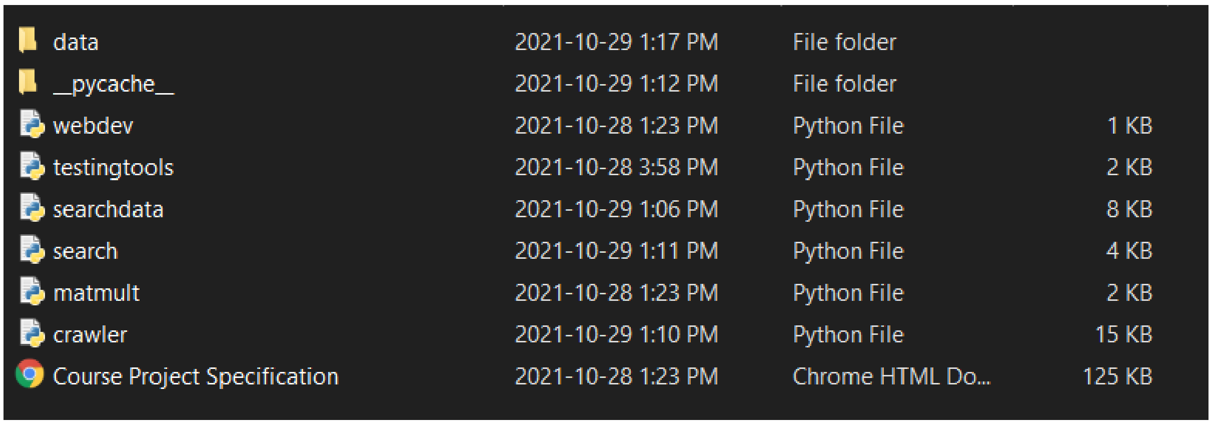
---

delDir(directory): This function is to recursively delete the given directory. This is used to reset any previously crawled data inside the data dictionary.

---

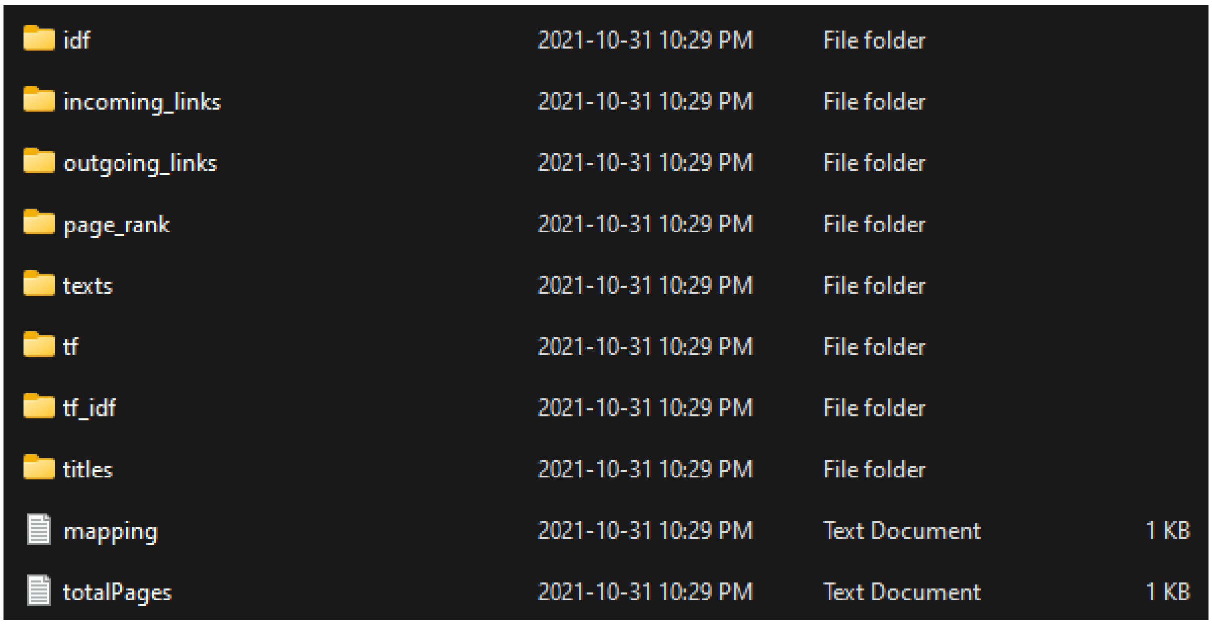
setup(): This function is to create the directory structure. It will make all the folders for tfs, text, idf, tf_idf, etc inside the main folder "data".

Main project path that hosts all .py files.



data	2021-10-29 1:17 PM	File folder	
__pycache__	2021-10-29 1:12 PM	File folder	
webdev	2021-10-28 1:23 PM	Python File	1 KB
testingtools	2021-10-28 3:58 PM	Python File	2 KB
searchdata	2021-10-29 1:06 PM	Python File	8 KB
search	2021-10-29 1:11 PM	Python File	4 KB
matmult	2021-10-28 1:23 PM	Python File	2 KB
crawler	2021-10-29 1:10 PM	Python File	15 KB
Course Project Specification	2021-10-28 1:23 PM	Chrome HTML Do...	125 KB

Inside data folder

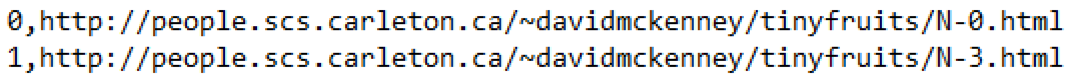


idf	2021-10-31 10:29 PM	File folder	
incoming_links	2021-10-31 10:29 PM	File folder	
outgoing_links	2021-10-31 10:29 PM	File folder	
page_rank	2021-10-31 10:29 PM	File folder	
texts	2021-10-31 10:29 PM	File folder	
tf	2021-10-31 10:29 PM	File folder	
tf_idf	2021-10-31 10:29 PM	File folder	
titles	2021-10-31 10:29 PM	File folder	
mapping	2021-10-31 10:29 PM	Text Document	1 KB
totalPages	2021-10-31 10:29 PM	Text Document	1 KB

The organization of files in different directories was not required, I could have all the files in one path. However, the implementation of directories helped me have similar name structures for each different type of file. For example, tf and tf_idf I had the same naming structure (word_index.txt), but since they're in different directories they won't conflict with each other. The time complexity of this function is O(1) because it is only creating 8 directories each taking constant time

---

mapURL(index, URL): This function is to store the mapping of all URLs and indexes in a mapping.txt file. Whenever a URL is read it is given a corresponding index code(from 0 to n) which is stored in the mapping.txt file. This file is extremely important because many times within the project I am needing to switch from index to URL and URL to index, so having a track of which index corresponds to which URL is crucial.
Within the mapping.txt file, the index and URL are stored in the form "index,URL".

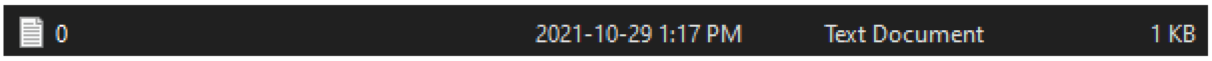


```
0,http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-0.html
1,http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-3.html
```

This function takes O(1) time as it only does 3 operations: open, write and close each taking constant time to run.

---

saveTitle(title, index): This function is to store all titles. Title files are stored in the titles directory where each file starts with the index.txt. Inside the file, the title name will be stored on the first line.



```
0                                  2021-10-29 1:17 PM      Text Document        1 KB
```

Inside → `N-0`
This function takes O(1) time as it only does 3 operations: open, write and close, each taking constant time to run.

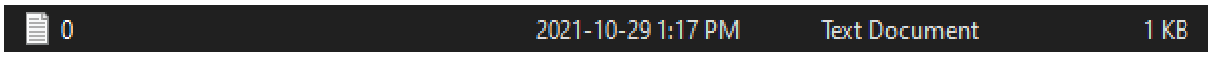
---

saveTextsAndTf(htmlContent, index): This function is to combine all texts and call the termFrequency function. This function takes in the Html Content and index. It then split the HTML content using a paragraph starter tag ex. [...,<p>....,<p>.....,<p>.....]. Then it takes all the words from the <p> tags and creates a variable combinedText which is all the words separated by "\n". Then the function calls the termFrequency function (below) and passes combined text and index. I had decided not to save the texts from each page into a directory because there is no use for the texts after calculatiting the term frequency. This function runs in O(n) time and space complexity of O(1).

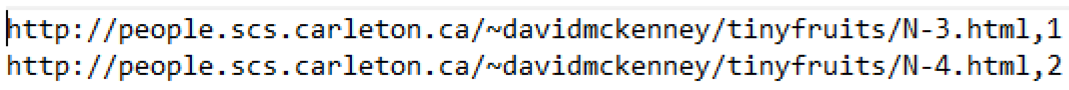
---

termFrequency(text, index): This function is to store Tf values, update uniqueWords and update pagesWithUniqueWords[word]. uniqueWords is a dictionary of all unique Words where key = word and values = word frequency. pagesWithUniqueWords[word] is a dictionary of all unique Words where key = word and values = number of pages the word appears in. These dictionaries are important for calculating idf and tf_idf values later on. This function runs in O(n) time and has a space complexity of O(n).

saveOutgoingAndIncomingLink(currentUrl, HTML content, index): This function is to store all outgoing links and call saveIncomingLink function. This function takes the given HTML content and finds the outgoing links within it. Then stores all the outgoing links into the outgoing links file.
Each URL has its own outgoing links file in the form index.txt where the content of the file is the outgoing links URL, index. The reason I include the index in this file is that I need to access the outgoing link files twice: once for the index and once for the pure URL. This is why I stored both so I won't need to find the index from the mapping file if I only stored the URL, or find the URL if I only start the index multiple times.



```
0                    2021-10-29 1:17 PM    Text Document    1 KB
```



Inside →
```
http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-3.html,1
http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-4.html,2
```

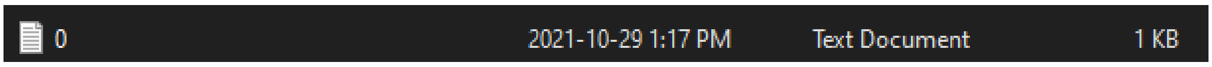
This function will also add all the new outgoing links and their index to a global queue. If this link is already in the queue or has already been crawled then it won't be added, otherwise, it will be on it. This function will run in $O(n^2)$ time and have a space complexity of O(n)

---

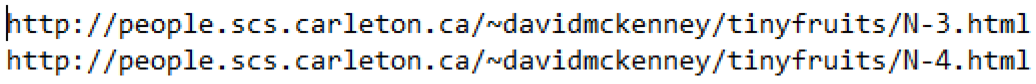
existInQueue(queue, URL): This function is to check if the given URL is in the given queue. For example, if we needed to check if an outgoing link has already been crawled we would give this function the global completedQueue as input.
This function runs in O(n) time as it has to loop for all elements in the queue(n).

---

saveIncomingLink(index, URL): This function is to store the incoming links. This function simply creates a file in the incoming links directory in the form index.txt. where the content of the file is the incoming links URL. Unlike outgoing links, incoming links will only be accessed to find the URL, therefore I only need to store the URL.



```
0                    2021-10-29 1:17 PM    Text Document    1 KB
```



Inside →
```
http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-3.html
http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-4.html
```

This function takes O(1) time as it only does 3 operations: open, write and close each taking constant time to run.

---

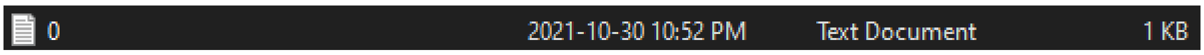
findIndexInQueue(URL): This function is to find the index of the given URL in either queue (queue or completedQueue). Since the queue elements are a list where Queue[i][0] is the URL and Queue[i][1] is the index, It is easily accessible. Otherwise, if I have to find the URL from the queue then find the index from the mapping file, it would increase time efficiency. This function runs in O(n) time as it has to loop for all elements in the queue and the completeQueue.

---

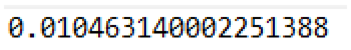
saveIdfAndTfIdf(): This function is to store/calculate Idf values and TfIdf values. This function uses the global variables uniqueWords and pagesWithUniqueWords to easily calculate the idf and tf_idf. This function runs in O(n^2) as it has to loop for all words in uniqueWords then loop for all pages

---

pageRank(): This function is to calculate and store the page rank. pageRank is stored in the page_rank directory where each file starts with the index.txt. Inside the file, the pageRank value will be stored on the first line.



```
0    2021-10-30 10:52 PM    Text Document    1 KB
```

Inside → 

```
0.010463140002251388
```

This function runs in O(n^2) time. This makes sense as it has to loop through the 2d array multiple times to calculate/update the values.

---

searchData.py Functions:

get_mapping_index(URL): This function is to get the mapping index given the URL. The function loops through all the lines in the file and when it finds a match with the given URL it returns the mapping index. This function runs in O(n) time and has a space complexity of O(1)

---

get_url(index): This function is to get the URL given the mapping index. The function loops through all the lines in the file and when it finds a match with the given index it returns the mapping index. This function runs in O(n) time and has a space complexity of O(1).

---

get_title(index): This function is to get the title of the URL given the mapping index. The title is on the first line of the corresponding title file(if it exists), therefore the function runs in O(1) time and has a space complexity of O(1)**.**

---

get_outgoing_links(URL): This function is to get the outgoing links of the URL given the URL. Since all the outgoing links files are titled with mapping indexes, we first use the get_mapping_index to find the mapping index. Then this function reads all the lines in the corresponding outgoing links file and returns a list of the outgoing links. This function runs in O(n) time and has a space complexity of O(n)

---

get_incoming_links(URL): This function is to get the incoming links of the URL given the URL. Since all the incoming links files are titled with mapping indexes, we first use the get_mapping_index to find the mapping index. Then this function reads all the lines in the corresponding incoming links file and returns a list of the incoming links. This function runs in O(n) time and has a space complexity of O(n)

---

get_page_rank(URL): This function is to get the pageRank of a URL given the URL. Since all the page rank files are titled with mapping indexes, we first use the get_mapping_index to find the mapping index. The pageRank value is on the first line of the corresponding pageRank file (if it exists), therefore the function runs in O(n) time and has a space complexity of O(1)**.**

---

get_page_rank_by_index(index): This function is to get the pageRank of a URL given the mapping index. The pageRank value is on the first line of the corresponding pageRank file (if it exists), therefore the function runs in O(1) time and has a space complexity of O(1)**.**

---

get_idf(word): This function is to get the idf value of a word given the word. If the word file exists the function will return the first line of the file(idf value). This function runs in O(1) time and has a space complexity of O(1)

---

get_tf(URL, word): This function is to get the tf value of a word given the word. Firstly the URL will be converted into the mapping index. If the tf file exists the function will return the first line of the file(tf value). This function runs in O(n) time and has a space complexity of O(1).

---

get_tf_idf(URL, word): This function is to get the tf_idf value of a word given the word. Firstly the URL will be converted into the mapping index. If the tf_idf file exists the function will return the first line of the file(tf_idf value). This function runs in O(n) time and has a space complexity of O(1).

---

largestValue(list): This function is to get the largest element in the list given the list. This function will come in handy for the search.py file. This function runs in O(n) time and has a space complexity of O(1).

---

search.py Functions:

search(phrase, boost): This function is responsible to respond to search queries. Similar to the crawler function, this function acts more like a central command where it calls and coordinates all the operation functions smoothly. Since this function is pretty large I will break it down into parts.

Part 1: This part interprets the query and creates queryVector. Using the given formula the queryVector is calculated stored. This part also keeps track of uniqueWords and adds them to uniquePhraseWordsVector. This part runs in O(n) time.

Part 2: This part loops for all pages gathered in the crawler to create a page vector for each page. This is done by looping through all the uniqueWords from the query and accessing the tf_idf value for each word. This part runs in O(n^2) time

Part 3: This part loop through all pages gathered in the crawler, and calculates/stores the cosine similarity between the page vector and query vector using the formula given. If requested this function will also multiply the cosine similarity by the page rank value to get a more accurate cosine similarity. This part run in O(n^2) time.

Part 4: This part creates a ranked list of 10 directories the contain search results in order of relatability to query. This is done by using the largestValue function from searchdata.py to find the page with the highest cosine similarity. Once the page index is found the other components of the search result will be found using functions within searchdata.py (get_url, etc.). This process will be repeated 10 times to find the 10 most related results. This part run in O(n^2) time

All together this program will run in O(n^2) time and have a space complexity of O(n).