Analysis Report

Aekus Singh Trehan

No partner

Carleton University

COMP 1405 Z

Dave McKenney

Tuesday, Oct 25th, 2022

crawler.py

Functionality:

The crawler.py takes in a string URL and then goes through all connected sites and stores information about each site in files. All functionality for the crawler is complete - more depth about functionality within the explanation for each function. There are 4 functions within this file all with their analysis below. There are additionally 4 global variables, totalPages, queue, incoming_links, and allWords. The totalPages variable is an int value that keeps track of how many websites the crawler has read through for saving site information in the correct files as well as for calculations for idf and pagerank values. The queue is a list full of URLs that need to be read through, when a link is found on a website, it will check if it hasn't been added to the queue or if it's been read and then add it to the queue. incoming_links is a dictionary which keeps track of what sites have links to each specific site. At the end of the crawling process, the incoming_links dictionary will be read through and stored in files for each site. The allWords dictionary keeps track of every word and how many sites contain each word. This is used for the calculation of the idf.

The overall design is to do every calculation possible with the information from just the websites so that the search can be as fast as possible. This means that there are a lot of files created in an organized manner (file structure explained below) so when queried, there is less computation needed. All the files created are done in this file, files for titles, incoming and outgoing links, tfs, idfs and tfidfs for each word, a URL mapping file (explained below), and pagerank files for each site.

crawl(seed):

The crawl function is the ‘main’ function of the crawler. This function will take in a string URL (must be an absolute URL, not a relative URL), start by calling on the fresh_crawl function to delete past information, then get all the information about each URL from the webdev read_url function and then send it to the write_info function to be stored for later queries. It will keep repeating for all connected URLs and after going through all connected URLs, it calls on the write_final_info function for all the final calculations to occur before returning the number of sites that have been crawled. The time complexity for this function on its own is O(N), where N is the total connected URLs from the initial website. The space complexity for this function is O(N+NM+K) where N is the number of links the crawler goes through, M is the number of links within each link, and K is the total number of unique words. This is because the main space-taking variables are the queue list, read list, allWords dictionary and the incoming_links dictionary. The queue and read list together will take up a maximum of O(N) space in total as all links will go to both of those lists and all end in the read list at the end. Then the incoming_links dictionary will take up NM space because for each link (N) it will store every link that goes to that link (M), therefore being O(NM). Finally, the allWords dictionary will take up a space complexity of each unique word in all the URLs (K).

fresh_crawl():

The fresh crawl function is for deleting past crawl data and creating a data folder to keep new crawl information. The time complexity for this is O(NM) where N is the number of sites previously crawled and M is the number of words per site. This makes sense because although there are three nested for-loops, the file structure is what matters. The second for loop will only repeat 6 times for each site because the number of files within each site's folders is always the

same. The outside folders will repeat for the number of sites there are (N) and then each word within each site will be gone through twice, once for tf and once for tfidf meaning 2M therefore total time complexity is O(NM). The space complexity for the fresh crawl is O(1) as the only variable saved is dataDir.

write_info(content):

The write info function is for organizing all the info in the 'content' which is a string representing info from a URL. It will store information into files about titles, word tf's, and outgoing links as well as save information in variables for later calculations of incoming links and tfidf and add new URLs to the queue for the crawl function to go through. To make it easier to understand I will separate the time complexity of the write_info function into multiple parts. Firstly for getting and storing the title of the URL the time complexity is O(N) where N is the size of the URL information passed in. This is because the .find() built-in python function will have a time complexity of O(N*M) where N is the length of the string and M is the size of the substring but since we know that the substring is always a length of 7 it just becomes a constant leaving the time complexity to be O(N). The next part is to get all the information from inside the p tags from the site content. The time complexity would be O(NM) where N is the number of p tags there are and M is the number of words in each p tag. This is because the loop will keep going for each p tag (N) and then go through each word in each p tag (NM). It will also write all the files for each word in that URL but that ends up being less than O(NM) by default since it's only going through each unique word. The next part is the incoming, outgoing and queue links. The queue has the highest time complexity out of all three as they all go through all of the site info looking for links and then when all the links are found, the queue portion will check if each link is in read and queue which is a total time complexity of O(NML) where N is the length of

the URL content and M is the length of the queue list and L is the length of the read list. The space complexity for the write info function will also be separated to make more sense. Note that I will be assuming that the content variable doesn't count as it is taken from somewhere else. Let N represent the number of words in the URL and M represent the length of the URL content. The words dictionary is the length of each unique word in the content meaning that in its worse case it has a space complexity of O(N). The tempContent and last strings have a space complexity of O(M) as they are both initialized as the same as content before they decrease in size. The links_out list isn't as significant to the space complexity because the list has to be less than N and M - the links are only a smaller portion of all of the content from the URL.

write_final_info():

The write final info function is for doing all the calculations and writing all information that can only be done after all the connected URLs are crawled. This includes writing incoming links for every URL, writing idf values for each unique word found in the crawl, tfidf values for each word in each file, and getting and writing pagerank values for each URL. The time complexity for this function is $O(MN^2)$. This is from calculating the pagerank vector. The matrix is multiplied by a matrix of the same length adding up to 1 until the distance between the two matrices is less than 0.0001. This is calling on the mult_matrix in the matmult.py file which has a time complexity of $O(N^2)$, where N is the number of URLs, and repeating that for the number of times it takes till the distance between them is less than 0.0001 (M). The other portions of the write final info function also have their time complexity but none are at the same power as this one, therefore making this one the overall time complexity. The space complexity for different variables is described as follows. The mapping has a space complexity of O(N) where N is the

number of URLs crawled. The matrix list is a list of lists that all have N values in them, N being the number of URLs crawled making the matrix list have a space complexity of $O(N^2)$.

searchdata.py

Functionality:

The searchdata.py file is for returning information about the crawl. All functionality for the searchdata file is complete - more depth about functionality within the explanation for each function. From the overall design of the crawler file, the functions in this file are significantly faster. Since all the computation is done in the crawler, functions in this file just need to locate the information that is being asked for and return it. This can be done using specific input from the user and looking through the mapping to find where the information is located. The mapping is a global variable that stores every URL that was crawled and has it mapped to its specific file location. This mapping is a dictionary so that when a URL is given it can be converted in O(1) time.

get_title(file):

The get title is a function I made to get the title of a file easier than opening and closing files in the search.py file. The file input is a string representing the folder name for the URL that the search is going through. This made it easier when in the search file as I don't need to reverse map to get the URL to map it again and get the folder number, saving computational time. The time and space complexity of it is O(1) as it just opens the file, reads the line, and then returns the title.

get_outgoing_links(URL) and get_incoming_links(URL):

The get incoming and outgoing links functions are for returning a list of incoming or outgoing links for the specified URL. The time and space complexity for these functions are O(N) where N is the number of links incoming or outgoing for the given URL. The default return value for these functions is None.

get_page_rank(URL):

The get page rank is for getting the page rank for a specified URL from the crawl. This function will return a float value representing the pagerank if it is found and will return -1 as a default value if it isn't found. The time and space complexity for this function is O(1).

get_idf(word):

This function takes in a string of a word in any of the files and returns the idf for this word. The time and space complexity for this function is O(1) and it will return a default value of 0 if that word wasn't found during the crawl.

create_mapping():

This function is called by the crawler after it is done doing everything so that the mapping can be initiated. If this function wasn't there none of the code would work because there would be no files or directories under the name 'data' and 'url_id.txt' at the start of running the code, causing the code to break. This function has a time and space complexity of O(N) where N is the total number of sites crawled.

get_mapping():

This function is to avoid redundancy. Rather than make the mapping again in the crawler function to do calculations, the get_mapping function is used right after creating the mapping so

that the mapping only has to be created once, thus, saving time at the cost of space. The time and space complexity of this function is O(1).

get_tf(URL, word) and get_tf_idf(URL, word):

Both these functions take in a URL and a word and then output the tf or tfidf at that word for that site. The default return value for both is 0 if the word wasn't found on that site. The time and space complexity for this function is O(1).

search.py

Functionality:

The search.py file does all the calculations for each URL's 'score' to determine which URLs are the most fitting for what is queried. This file only contains one function which is the search(phrase, boost) function. It takes in a string of words representing a search query and a True or False representing boost - whether to multiply the cosine similarity score by pagerank. In this file rather than opening files and getting information itself, most of the information comes from the functions in the searchdata.py files.

search(phrase, boost):

The time complexity for this function is O(NM). This is because the bigger portion of the time is when it goes through and calculates the score for each page. Let N represent the number of pages crawled as it will go through each one. Then for each link, it will go through every word in the query to get the tfidf as well as to calculate the cosine similarity for each word - the number of words in the query representing M, therefore a time complexity of O(NM). The space complexity is a bit different. Let's assume N is the number of URLs crawled through and M is the number of words in the query. The words dictionary and the search list are just as big as the

phrase, meaning a space complexity of O(M). The queryOrder, the queryVector, docVector and toDelete lists are the same except they can be smaller than M, therefore also having the same space complexity. allFiles list is a list of the dictionaries for each URL being a space complexity of $O(N^2)$. Since we know that each dictionary only has 3 values in each, however, there is a space complexity of O(3N), meaning it has a space complexity of O(N). The reverseMapping dictionary is also the same with a space complexity of all the links crawled through O(N).

matmult.py

Functionality:

The matmult file is for some of the calculations for the pageranks and cosine similarity. There are two functions in the file, one for multiplying matrices and one for getting the distance between two matrices.

mult_matrix(a, b):

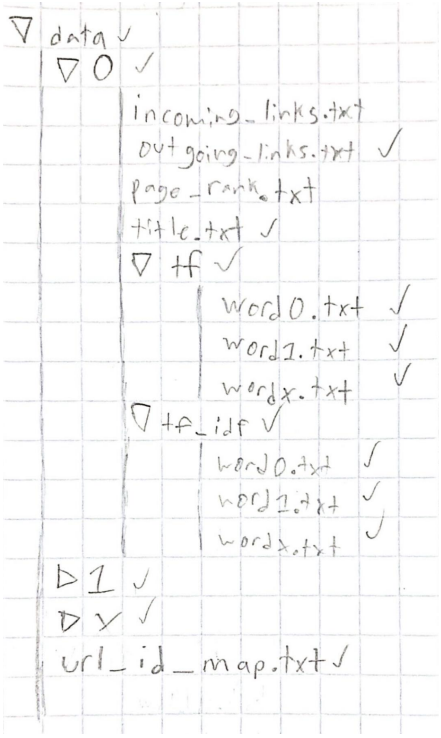
This function takes in two lists in a list (matrices), a and b, and then goes through all values in A's rows and B's columns, multiplies them and then adds them to the total. The time complexity of this function is $O(N^2M)$ where N is the length of a[0] and M is the length of b.

euclidean_dist(a,b):

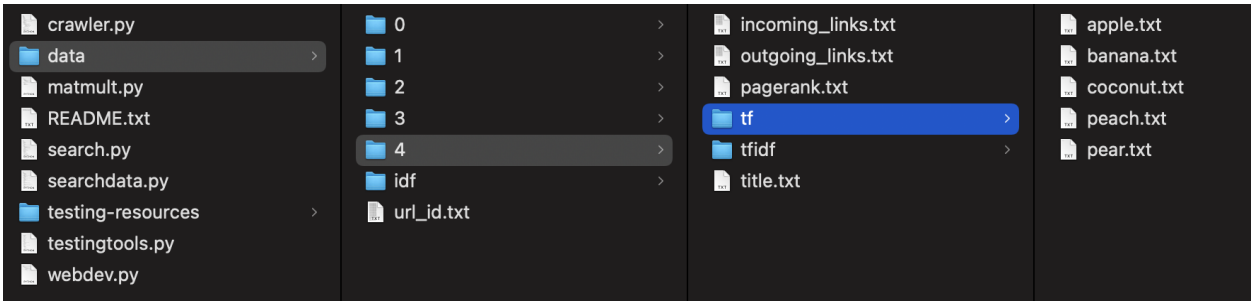
The euclidean distance function takes in two matrices that are 2 dimensional (i.e. two single-row matrices), a and b, and gets the euclidean distance between them. Note that this is very similar to calculating the distance between two points, except it's for higher dimensions. The time complexity of this function is O(N) where N is the length of a[0] (the length of the matrix)

File Structure

When thinking of the file structure I was thinking about every calculation I could do in the crawler so that the efficiency of the search would be as fast as possible. The image to the right was my first variation of the file structure. It is designed so that each URL is 'mapped' to a number. Each URL's number is remembered in the url_id.txt file (was called url_id_map before, hence the old name on the right) where the URL and the number are separated with a #. Now although there might be # in the URL itself it doesn't matter what that character is because when running the function rfind(), as long as it isn't a number, it will always appear to the right of the URL and therefore not interact with the URL when getting the number. The files in each folder are for storing calculations or numbers that are needed for efficiency. Incoming links file stores incoming links, outgoing links files store outgoing links, pagerank files store pagerank, title files store the title, and then there are the tf and tfidf folders which store those values for each word. The files are organized this way so that information is quickly and easily received and so that all the calculations that can be done before the search are done and stored.



Between the original structure above and the most recent one, the only difference is that the new one has a folder for idf values for each unique throughout the whole search. This is because I originally thought that I would calculate idf once and not need it again but that was before I realized that it is needed in page rank as well as searchdata.py. This is what the file structure looks like now:



Now you might be thinking, the tf values aren't used for any of the calculations other than tfidf so why are they kept in their files? At least that's what I thought when I first made them but after removing them I realized that it's because in the searchdata.py file there is a function that needs to return a tf value for any given site and word. I then readded these files to make that process considerably faster as the values are already saved and no calculations need to be done again.

Additional Design Decisions

I originally made another file for each crawled site called vectorDoc.txt to keep the tfidf of all words in the site in a list, i.e. a vector document for each document but I realized that it is faster to just get each word tfidf because most words in that vector wouldn't be used and it just uses more space. Rather than having a space complexity of O(NM) it has a space complexity of O(NL) where L is always less than M. This is because rather than getting every word tfidf value (M), this would just get the values for each unique word in the search (L) meaning that L will always be smaller than M, therefore, having a smaller space complexity. Note that N is each site crawled as this needs to be done for each site.