Comp 1405Z - Fall 2022
Course Project: Analysis Report
Iman Ullah

Crawler Module(crawler.py)

Overall Design

While creating the crawler module, I had the intention to do more than just crawl through the pages and store data. I also wanted to do all the necessary calculations for my query search and store them. This would increase the runtime and the space used during the crawl. But would provide a faster search result. I prioritized a faster search to minimize the time between the user entering the search query and the user receiving the results.

Because of this I have decided to store the url, outgoing links, incoming links, idftf for each link, tf for each link, idf values and page ranks.

In my crawler module, I implemented multiple smaller functions for the crawl function. I decided to go this route as it would for a more organized and readable code. It would also make it easier for me to debug as I could focus on specific functions at a time and would allow me to lower the amount of repetitive code.

This section goes in depth into various parts of the crawler module such a file organization, space complexity and the multiple functions and their time complexities

File Organization

To organize my files, I implement a folder tree. The main folder is named crawler and contains all the information gathered during the crawl. Within the crawler folder, there are multiple other folders. The number of folders are dependent on how many links are found in the crawl. There is 1 folder for the idf values of each word found in the crawl and multiple other folders which represent each link found in the crawl. Algebraically there are x + 1 amount of subfolders within the crawler folder, x being the amount of links found in the crawl.

Within the idf_values folder, you will find the idf values for all the words found during the crawl as well as a folder for page rank which stores the page rank values for each link found during the crawl. I decided to put the page ranks in the idf folder, because of code readability. Having the

page rank within the idf folder doesn't have much of an impact in runtime as you can reach for the page ranks directly through proper file path.

Within all the other subfolders contain other 2 subfolders and 3 files. The 3 files are the incoming links, the url and outgoing links for each link/page. All the incoming links are in the same file. This doesn't cause an issue for runtime as I have never needed to grab a single specific link. Every time I needed to access the incoming links, I needed to retrieve every single incoming link for the specified page. If I were to put each individual incoming link in a single file, it would not improve run time of information retrieval and would just be a waste of space. The reasoning is the page for outgoing links. The links file only contains 1 url. So overall all these files run constant time complexity as they can be retrieved through a single command.

The other 2 subfolders contain the tf and the idf tf values for the page. There is a single file for each word.

Overall the way I have my files setup allows me to retrieve any information at 0(1) time. The exceptions being the incoming and outgoing links. Because those files store multiple links. This is the reason why I have my files setup this way as it allows me to retrieve information very fast.

Function: del_data(filepath):

The function takes a directory and deletes the directory and everything within it. Used to delete all previously crawled data. The function uses the os module to properly execute it's deleting operations.

Time complexity: The time complexity of this function is O(N) where N is the amount of total files. As each file needs to be deleted once. The O(N) doesn't take into consideration that we also have to delete each folder and subfolders once. But the amount of folders and sub folders are insignificant compared to the amount of words.

Space complexity: The function uses a list to loop through the files in the folders. The space used by the list can fluctuate depending on the amount of subfolders and files and folders it has. The space complexity is therefore O(N) where N is the amount of subfolders and files in a given folder.

Function: grab_words(data):

The function takes in a URL and retrieves all the words within that url.

Time Complexity: The function loops through all the lines of the html data of that webpage. There's a few other calculations the code does, which is insignificant in the worst case. As in the worst case the webpage has a ton of lines.
The worst case time complexity is O(N) where N is the number of lines.

Space Complexity: The function uses a list to store all the words found. The space complexity of that list is O(N) where N is the number of words found in the html data.

Function: grab_link(data):

Takes a line of code which contains a link and returns that link. The function is only called when you know a line has a link.

Time complexity: O(1) since it does only one calculation

Space complexity: O(1) since it takes in the line which is stored as a string and returns that line

Function: grab_title(data):

Takes a url and returns the title from the html data.

Time complexity: The code loops through each line of html data. If the title is at the very last line then the time complexity is O(N) where N is the amount of lines

Space Complexity: Needs URL as a parameter and also returns the title. Used 2 variable so space complexity is always O(2)

Function: mult_matrix(a ,b)

Takes 2 matrices and returns the resulting matrix when matrix a is multiplied by matrix b

Time complexity: Goes through 3 loops nested within each other, so the time complexity is O(N^3). There are other smaller computations, but they are insignificant for the worst cases where the matrices are very large

Space Complexity: O(3N^2) where n is equal to the length of rows or columns of matrices, whichever is bigger. Matrices are 2 dimensional lists so they are O(N^2) space complexity and there's at most 3 matrices

Function: find-tf(word, words, title)

Takes in a **word** and returns the tf of that word within a list of **words** and stores them in a file named by the **title** parameter.

Time complexity: O(2N), the program loops through the list of words and does at most 2 computations during each iteration. There are other computation done within this function, but they are insignificant at the worst cases where there are a lot of words to loop through

Space complexity: At the worst case, the space complexity is O(N). Since in the worst case, the list of words will be very big, which makes the other variables insignificant.

Function: relative_to_absolute(url, link)

Takes in a relative link and turns it into an absolute link

Time complexity: O(N), the function loops through the url and places the relative link at the very end. There is also one other computation done, which becomes more and more insignificant as the url becomes larger.

Space Complexity: O(2), makes use of 2 variables

Function: update_incoming(link, url)

Take the **url** and add it to a list of incoming links for the passed **link**.

Time Complexity: Calls grab_title function which is O(N) time complexity where N is the amount of lines and does few other computations, which become insignificant at the worst cases. The worst case time complexity is O(N) where N is the amount of lines in the **link.**

Space Complexity: O(3), uses 3 variables

Function: store_outgoing(outgoing_links, title)

Takes in outgoing files and stores them in a txt file

Time Complexity: Needs to loop through outgoing_links list to do a single function. Worst case time complexity is O(N) as all the other computations becomes insignificant as the outgoing links list becomes bigger.

Function: store_idf(all_words, all_titles)

Takes in all words and all titles found throughout the crawl and uses them to calculate and store idfs for each word.

Time Complexity: Worst case time complexity is O(N(2M+4)) where N is the amount of total word and M is the amount of titles

Space Complexity: Uses 2 lists which are used to pass in words and titles. Worst Case time complexity is O(N+M) where N is the amount of total word and M is the amount of titles.

Function: product_store_idftf()

Doesn't take in any parameters, takes values from saved files and computes the idftf for each word in each link.

Time Complexity: worst case time complexity is O(7N*M) as it goes 2 loops nested within one another and does 7 calculations in each iteration of the nested loop. Where N is the amount of link and M is the amount of words within those links

Space Complexity: Uses 2 lists, which account for most of the ram used in the worst cases. Worst case space complexity is O(N*M). N and M being the lengths of the 2 lists.

Function: produce_page_rank()

Implemented the algorithm covered in class to determine the page ranks values of all the pages that we crawled through and save those values in files.

Time Complexity: This function has the longest runtime. I have split the function into parts and listed the run time of each part. And added them at the end to get the total runtime of the function. I did this to make it easier to analyze. All these runtimes are for worst cases.

Producing a list for mapping files in the adjacency matrix: O(2N) where n is number of links crawled

Creating empty adjacency matrix: O(N+1)^2 where N is number of links crawled

Complete adjacency matrix: O(N(3+3M+N) where N is number of links crawled and M is the number of outgoing links in the link with the highest number of outgoing links

More algorithm calculations: O(4N^2) where N is the number of links crawled

First page rank calculation iteration: O(N) where n is the number of links crawled

Page rank and ed calc after first page rank calc: uses matrix_mult function: O(Q(N^3 + N)) N is the number of links crawled at worst case and Q is the amount iterations required until ed between current and last iteration is less than 0.0001

Combining these we get: O(3N + (N+1)^2 + (N(3+3M+N) + (4N^2) + Q(N^3 + N))

=O(3NM + 6N^2 + 8N + 1 + Q(N^3 + N))

Worst case complexity: O(6N^2 + Q(N^3)) at worst cases a lot of terms become insignificant

Space Complexity: function uses 2 lists, 2 2D lists but it's just matrices with 1 column and 1 2D list. The 2 matrices with 1 column are identical to 1D lists. In the worst case O(4N + N^2)

Function: crawl(data)

Takes in a url and crawls through all related urls and extracts the data within those pages.

Time complexity: O(N(N^2+2N^2 + 6N^2 + Q(N^3)) Time complexity includes all the smaller functions we go through to get crawl data. During the worst case scenarios the run time of certain functions and calculations become insignificant. So I didn't include them

Q is the amount iterations required until ed between current and last iteration of page rank is less than 0.0001
N in worst case is the amount links we went through

Space Complexity: O(8N) as we use 8 different list to store and compute data. The single element variables become insignificant as N increases in worst case scenarios. N is for the amount of links or words(whichever is bigger in length) we search through.

Searchdata.py

Contains multiple functions which allow us to retrieve multiple information found and calculated during crawl.

Function: get_outgoing_links(URL)

Takes a url and returns a list of outgoing urls within the given url

Time complexity: O(2N) N is the length of the URL or the amount of outgoing links, if there are more outgoing links than characters in the URL.

Space Complexity: O(N) In the worst case the single element variables become insignificant. N represents the singular list used to store outgoing urls

Function: get_incoming_links(URL)

Take a url and returns a list of incoming urls within the given url

Time complexity: O(2N) N is the length of the URL or the amount of incoming links, if there are more incoming links than characters in the URL.

Space Complexity: O(N) In the worst case the single element variables become insignificant. N represents the singular list used to store outgoing urls

Function: get_page_rank(URL)

Takes the url and returns it's page rank

Time complexity = O(N), uses the grab_title function which accounts for majority of the run time in worst cases

Space Complexity: O(3), 3 variables the return variable, title and URL

Function: get_idf(word):

Takes an word and returns it's idf

Time Complexity: O(5), at most 5 calculations to retrieve the info and return it

Space Complexity: O(3), uses 3 variables

Function: get_tf(URL, word)

Takes the url and returns the tf of the word in that html page

Time complexity: O(N), calls the function grab_title, which takes up most of runtime at worst cases

Space complexity: O(5) uses 5 variables

Function: get_tf_idf(URL,word)

Takes the url and returns the tfidf of the word in that html page

Time complexity: O(N), calls the function grab_title, which takes up most of runtime at worst cases

Space complexity: O(5) uses 5 variables

Search.py

Contains the main search function used by the user. I also created more minor functions to help with the search function. This module mainly does minor math calculations and retrieves crawled data.

Function: get_query_idftf(word, word_count, length)

Calculates the query idftf

Time complexity: O(6), at worst cases 6 calculations

Space complexity: O(6), at worst cases 6 variables

Function: get_query_vector(u_words, length)

Calculates the query vector for cosign calculations

Time complexity: O(8N) n is number of u_words

Space complexity: O(2N) uses 2 lists which accounts for majority of space usage in worst cases

Function: get_file_vector(file, index_key)

Computes the file vector for the given file

Time complexity: O(3N) n is number of u_words

Space complexity: O(N+4) uses 2 lists and 4 variables

Function: cos_similarity(q_vector, f_vector)

Computes cosign similarity between query vector and the given file vector

Time complexity: O(2N) n is the length of q_vector

Space Complexity: O(2N) uses 2 lists which accounts for majority of space usage in worst cases

Function: content_score(query)

Returns the best pages based of query with page rank boost off

Time Complexity: O(10N + 5N^2) at worst cases, there are more minor calculations which become insignificant in worst cases. N represents the amount of files found during crawl

Space complexity: O(6N) N represents the amount of files found during crawl. There are 7 lists, but of them is capped to 10 values

Function: content_score_pagerank

Returns the best pages based of query with page rank boost on

Time Complexity: O(10N + 5N^2) at worst cases, there are more minor calculations which become insignificant in worst cases. N represents the amount of files found during crawl

Space complexity: O(6N) N represents the amount of files found during crawl. There are 7 lists, but of them is capped to 10 values