COMP 1405Z
Fall 2023 – Course Project

Project Background

The goal for the course project is to implement a web crawler and a search engine. The project will require you to implement three modules: crawler.py, searchdata.py, and search.py. This document outlines the specifications of each module within the project, which must be followed precisely. You are expected to use good programming practices throughout the project and should also pay attention to the runtime and space complexity of your solutions.

Constraints and Assumptions

1. You may not use additional modules other than the provided webdev.py module and the built-in *os*, *json*, and *math* modules of Python. The webdev.py module is included on the project page within Brightspace and can be used within your web crawler to read web pages. You cannot modify the provided webdev.py file.
2. **You should aim to optimize the efficiency of all your operations (i.e., both crawl and search). However, your primary goal should be to minimize the runtime and space complexity of your search implementation**. This is because crawling is typically completed once and then many searches will be executed on the crawled data. This means you can perform extra processing during the crawl to produce additional data that will aid in improving your search efficiency (e.g., pre-computing and saving word frequency data, PageRank values, etc.).
3. This document lists functions that are required for each module you will implement. You may add additional functions as you see fit but must support the required function signatures. If you add any additional functions, you should include a comment above that function explaining what the function's goal is, as well as what inputs and outputs are involved.
4. You may assume the HTML is well-formatted (e.g., each <a> or <p> tag has a matching closing </a> or </p> tag).
5. You may assume that there will be no additional HTML inside of the <a>…</a> or <p>…</p> tags.
6. You may assume that all links include an "href" property and the link location is contained inside quotation marks after href= inside of the link definition.

7. You may assume that the title for a page will be the text in between the <title> and </title> tags within the page. **You may not assume the title matches the last part of the URL for the page.**
8. Each URL will contain either an absolute or a relative URL. All absolute URLs will start with "http://". Links that do not start with "http://" will be relative links and you will have to compute the full URL using a combination of the link address and the current page URL. You may assume that all relative links will start with "./". **You may not assume that all pages share the same base URL** (e.g., http://people.scs.carleton.ca/~davidmckenney/fruits2/) and will have to programmatically build the URL using the current page's URL and the relative link.
9. You may assume that all words within a page will be contained between a <p> start tag and </p> end tag. Words may be separated by spaces or new-line characters (i.e., "\n").
10. **While all pages in the test data are named N-x.html, you may not assume that this is generally true for all pages (i.e., pages may have any name).**
11. You cannot use regular expressions for string parsing and manipulation.

Test Files

A number of test files are included in the testing-resources.zip file, which can be found on the project page within Brightspace. These files can be used to automatically test portions of your project code using different datasets. There are six different datasets that you can test your implementation on (tinyfruits, fruits, fruits2, fruits3, fruits4, and fruits5). If you want to test your program manually, you can start your crawl at the following links to crawl the pages associated with each dataset:

1. http://people.scs.carleton.ca/~davidmckenney/tinyfruits/N-0.html
2. http://people.scs.carleton.ca/~davidmckenney/fruits/N-0.html
3. http://people.scs.carleton.ca/~davidmckenney/fruits2/N-0.html
4. http://people.scs.carleton.ca/~davidmckenney/fruits3/N-0.html
5. http://people.scs.carleton.ca/~davidmckenney/fruits4/N-0.html
6. http://people.scs.carleton.ca/~davidmckenney/fruits5/N-0.html

The tinyfruits dataset contains only 10 interlinked web pages and is a good starting point to test your crawler/search functionality. The remainder of the datasets contain 1000 interlinked web pages and will take longer to crawl/test. Some of the test files evaluate specific functions (e.g., x-idf-test.py, x-page-rank-test.py, etc.). Other test files contain "all" in the name and will run all the tests on your code using the given dataset. Assuming your code does not crash, each test program will output a passed text file and a failed text file that will contain information about which tests your code passed/failed. You can use these files to evaluate the correctness of your implementations.

The provided test files will execute a crawl every time you run the file. If you will be running the test file repeatedly without changing your crawler, it may be worth commenting out the crawler.crawl(url) function call near the start of the test file after you have run your crawler on the dataset once. This will allow you to avoid performing the crawl to produce identical data each time you test your functionality (note: you will need to crawl again if you have changed your crawler in any way, or your data will not be consistent).

Lecture Materials and Readings

Some short lectures have been included within the Course Project module in Brightspace. These lectures will discuss the basics of crawling and search engines, as well as the algorithms and computations involved within the project (e.g., PageRank, cosine similarity, tf-idf).

One important clarification: slide #7 in the “Vector Space Model and Cosine Similarity” slides should state “Each document vector has one component for each **unique** term”. If a word appears more than once, it will affect the TF/IDF values for the document, but will not make the vector larger.

Part 1: Web Crawler (crawler.py)

This module will be responsible for performing the web crawling required for your search engine. You must implement a **crawl(seed)** function that accepts a single string argument representing the URL to start the crawl at. This function must reset any existing data (i.e., delete all files and information from any previous crawl) and then perform a web crawl starting at the seed URL to find and parse all pages that can be reached through links starting from the seed URL. This crawl process is responsible for generating all data required for the other parts of the project and saving that data to files (i.e., you should never read a web page from the searchdata or search modules). The crawler should not revisit any pages (i.e., once a page has been parsed, it is not parsed again during this crawl). Once complete, this function must return the number of total pages found during the crawl.

Part 2: Data Required for Search (searchdata.py)

This module will contain functions to produce the data required by your search engine. This module will also be used to test the correctness of the data produced by your crawler module. You must implement each of the functions below, all of which will be necessary for the overall functionality of your search engine. These functions should read the information that your crawler saved in files to produce the correct outputs.

get_outgoing_links(URL)
Accepts a single string argument representing a URL and returns a list of other URLs that the page with the given URL links to. The list does not need to be sorted in any way. The URLs returned should be absolute URLs (i.e., start with http://). If the given URL was not found during the crawling process, this function must return the value None.

get_incoming_links(URL)
Accepts a single string argument representing a URL and returns a list of URLs for pages that link to the page with the given URL. The list does not need to be sorted in any way. The URLs returned should be absolute URLs (i.e., start with http://). If the given URL was not found during the crawling process, this function must return the value None.

get_page_rank(URL)
Accepts a single string argument representing a URL and returns the PageRank value of the page with that URL. If the given URL was not found during the crawling process, this function must return the value -1. The algorithm for computing PageRank is discussed in one of the lecture recordings included on Brightspace. You should use the matmult module you created as part of tutorial #4 to perform the matrix operations required for PageRank calculations. You should use an alpha value of 0.1 for the PageRank calculation and stop the iteration when the Euclidean distance between t_{x-1} and t_x vectors is less than or equal to 0.0001.

get_idf(word)
Accepts a single string argument representing a word and returns the inverse document frequency of that word within the crawled pages. The inverse document frequency of a word *w* is calculated using the equation below (note, all logarithms within the project should be calculated using base 2):

$$idf_w = \log\left(\frac{Total\ \#\ Documents}{1 + (\#\ of\ documents\ w\ appears\ in)}\right)$$

If the word was not present in any crawled documents, this function must return 0.

get_tf(URL, word)
Accepts two string arguments: a URL and a word. This function must return the term frequency of that word within the page with the given URL. The term frequency of a word *w* within a document *d* is calculated using the equation below:

$$tf_{w,d} = \frac{\#\ occurences\ of\ w\ in\ d}{Total\ \#\ of\ words\ in\ d}$$

If the word does not appear in the page with the given URL, or the URL has not been found during the crawl, this function must return 0.

get_tf_idf(URL, word)
Accepts two string arguments: a URL and a word. This function must return the tf-idf weight for the given word within the page represented by the given URL. The tf-idf weight of a word *w* in a document *d* can be calculated using the equation below:

$$tfidf_{w,d} = log(1 + tf_{w,d}) \times idf_w$$

Part 3: Search (search.py)

This module will be responsible for answering search queries. The module must implement a **search(phrase, boost)** function that accepts the following two parameters:
1. phrase - A string representing a phrase searched by a user. This string may contain multiple words, separated by spaces.
2. boost - A boolean value representing whether the content score for each page should be boosted by that page's PageRank value (True) or not (False). If the boost value is True, the content score (i.e., cosine similarity score) for a page must be multiplied by the page's PageRank value to determine the page’s overall search score for the query.

This function must perform the necessary search calculations to produce ranked results using the vector space model and cosine similarity approach (see associated recording for details). The function must return a list of the top 10 ranked search results, sorted from highest score to lowest. Each entry in this list must be a dictionary with the following keys and associated values:

1. url - The URL of the page
2. title - The title of the page
3. score – The total search score for the page

There may be ties in some of this data, which could produce spurious failed test cases when evaluating the search functionality. If one of your search test cases failed but most of the results look correct, it could be worth temporarily returning more than 10 ranked results to check whether the 11th/12th/etc. results are tied with the 10th result in score and contain the correct page. If this is the case, and the scores have been calculated correctly, the search results can be considered correct.

Part 4: Analysis Report (analysis.pdf)

Write an analysis report that explains the runtime and space complexity of the functions contained in your crawler, search and searchdata modules. Within the report, you should also:

1. Include a list of functionality that is complete and incomplete.
2. Describe and justify the file structure you have used to store the data
3. Discuss what design decisions you made to improve the efficiency of your implementation.
4. Highlight any other qualities of your project that you think are worth noting.

It is important to clarify in your report *why* you have implemented your solution the way you have. Focus your discussion on this rather than explaining the code. A few example reports from a previous term have been added to the project page on Brightspace. These reports can provide some guidance about what you could discuss in your own report. You shouldn't copy their formats directly, as they are not perfect and the requirements this term are slightly different.

Submission

Before submitting, ensure you remove any print statements and other code that will be executed when your modules are imported (i.e., nothing should be executed when one of your modules is imported). Add all the required resources for your project and your analysis PDF to a single .zip file named "project.zip" and submit it to Brightspace. If you have run the provided testers, you should include the passed/failed output in your zip. You must also include a README.txt file that contains your name and your partner's name (if applicable), along with instructions for running your crawler and search engine from the command line. Ensure that your file is a .zip and has the proper name. Download your submission afterward and check the contents to ensure everything worked as expected. You should verify that all resources required for crawling and search are included in the zip (e.g., crawler.py, searchdata.py, search.py, matmult module, any other additional modules you created, etc.). You are encouraged to run the tests again on the downloaded zip to ensure you get the expected results.