Algorithms

1 CRAWLER

It keeps track of two things: List of crawled pages and queue of to crawl pages. It runs multi threads. Each thread is responsible for downloading an html document and does each of the following:

1) Deques a URL from the to crawl queue and checks three things
   a. The URL is not in the crawled pages.
   b. The URL is an html document.
   c. The URL's robots.txt allows us to crawl it.
2) If all checks passed it continues with the following steps, else it goes to the next URL.
3) It downloads the html page of that URL using jsoup.
4) Finds all the anchor tags and gets the href property and adds it to the to crawl.
5) Adds the URL to the crawled pages and goes to the next one.
6) In case it got interrupted, it saves the crawled list and to crawl queue to a file.

The next time we run the file it first checks the crawled file, if it is empty that means it's the first time crawling and loads the initial seeds to the to crawl pages. If it's not empty then it loads both the crawled and to crawl pages and continues working from this state.

How we determine if we are allowed to crawl based on robots.txt:

1) Starts by identifying the path of the page intended for crawling.
2) Locates the start of each user-agent group.
3) Looks for the user-agent under which our search agent falls (*).
4) Splits the permissions into 2 categories: Allowed and Disallowed.
5) Since, allowed paths are of a higher priority, it first checks if the current path is any of the allowed paths, and in such case, returns true and allows crawling.
6) Next, it roams the disallowed paths, and checks if the current path is any of them. If so, returns false and prevents crawling.
7) Otherwise, returns true.

While saving the file to the documents folder we needed to save it by a name that would allow the indexer to identify the URL, but since some of the URL characters are disallowed to be a part of a file name, we couldn't save it by the URL name directly instead, we implemented an encoding algorithm that converts the URL to a name allowed to be a file name then decoding it to the original URL in the indexer. The encoding algorithm is simply a mapping of each character that could be found in a URL to a unique number. And the decoding does the reverse process.

2 INDEXER

It initially reads all the html files in the documents folder (the folder containing the crawled pages.) It runs multi threads. Each thread works at one URL at a time and does the following:

1) Take one document from the crawled documents.
2) Creates a new doc with the URL of the document. A doc is a class that has a URL a word

3) Get the title, and all the paragraph and header (from 1 to 6) tags. And for each type does the following:
    a. Passes the text of the tag to the preprocessor and gets back an array of words.
    b. For each word it checks if that word was encountered before for that URL
        i. If Yes: it increases the type (p, h1, h2…etc.) of that word.
        ii. If No: it adds that word to the array of words belonging to that URL and sets the type encountered by one. And all other types as zero.
4) For each unique word and URL together, it calculates the TF and adds that row to the database.
5) It moves the finished document from the documents folder to the indexed documents folder, so that if the indexer got interrupted it doesn't repeat all the work again and indexes only the remaining documents.

After inserting all the documents in the database, the algorithm retrieves all of them at once and group by the word to count how many times each word is repeated in all the documents. From this it calculates the DF and IDF for each word and inserts them all at once in another table in the database with the word being the unique identifier.

3 QUERY PROCESSOR

The algorithm takes in three parameters from the request sent to the /search endpoint: The search query, the page number and the limit per page. It passes the search query to the preprocessor module and gets back an array of words. Search the rows in the database that has one or more of these words and returns multiple rows. We obtain only the unique URLs and those unique URLs represent the count of all the search results founds for this search query. We then go to the other table that contains the details of each URL like the website name, the website text…etc. and find all the rows that has any matching URL and we limit the returned results to be only equal to the received limit in the request parameters and we skip by an amount equal to the (page-1) * limit.

The algorithm here is required to return the snippet from the whole text of the URL based on the search query and it does so as follow:

The idea is to use a queue to make blocks and the shortest block at the end is the on containing the most words in the shortest length.

The way we fill the blocks is just by going through each word sequentially if the search query contains that word, we add it to the end of the block. The first word in the current block must exist only once, so if while inserting that will not be true, we deque the first element from the queue and check for the next first one of them satisfies the condition.

The query processor is also required to preprocess the words and it does that as follow:

Takes the raw text of the search query or the html tags text and does the following:

1) Checks the text character by character against the punctuations and removes them.
2) Examines the text word by word and removes the all the stop words.
3) Stems each word by an algorithm found called Porter Stemmer.
4) Returns an array of preprocessed words.