Report of Phase One

Steps to run the program:

You can run this program in two different ways:

1. You can run the python file using python simple.py <input_dir_path> <output_dir_path>
2. You can run the shell script using ./tokenize <input_dir_path> <output_dir_path>

For both the above-mentioned ways, you need to have current working directory as the one which has python and shell files and you need to have beautifulsoup package installed.

Implementation:

I have used beautifulsoup package available in python to parse the html documents. I have maintained two different dictionaries named local_dict and global_dict to keep track of the tokens. I have read through each html file using a for loop and used ISO-8859-1 encoding format to read through the html documents. After reading through each html document I have used html.parser available in beautifulsoup to parse the html document and remove all the html tags which we don't need when we are tokenizing the document. Then I have used get_text() method available in beautifulsoup package to extract the text from the html document. I have then used regular expression package i.e., re to remove all the digits and remove punctuations like apostrophe, dashes so we can compress the words like "country's" to "countrys". I have then used findall function available in re package to find all the words and stored them in a variable 'final_words'. I have then looped through this variable 'final_words' to convert all the words into lower case. And then I have used the final_words variable to load up both the local and global dictionaries. Finally, I have written the output in an output text file using file stream commands of python. After looping through all the documents and the global dictionary is loaded with all the tokens available in all the html documents, we sort the global dictionary by token and then by the frequency and load up two different files named 'globaldictsortedbykeys' and 'globaldictsortedbyvalues' respectively.

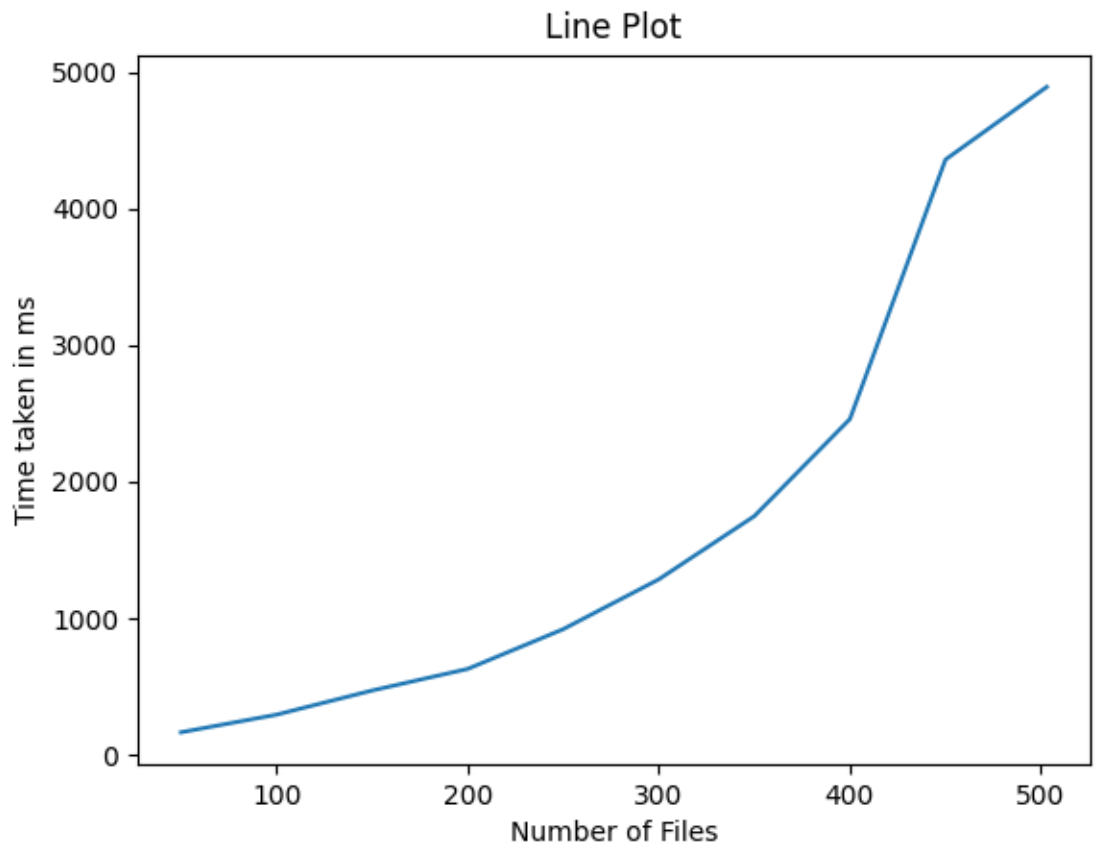
Drawbacks:

One of the issues I have seen in the tokenizer is that whenever it encounters a full stop it ends the word there. So, words like U.S.A becomes u, s, a which is a kind of drawback. If I were to remove the full stop and squeeze the letters together then words of the sentences that are near full stops will be affected. This is the reason why my tokenizer couldn't handle them properly.

Performance:

Number of documents	Time in Milliseconds
50	165
100	294
150	471
200	629
250	920
300	1285
350	1748
400	2459
450	4359
503	4891

The following graph demonstrates the graphical representation of the above table.



Comparison:

I have compared my implementation with Chelsea. She has also used beautifulsoup package to parse the html documents but she has used html2text package to extract the text from the document whereas I used get_text() method available in beautifulsoup package to extract the text. The difference between the two processes is that html2text preservers the structure of the html document whereas get_text() method does not. But since we only need the text to tokenize the documents I have used get_text() method. Since she used html2text package the elapsed time is more than twice as mine.

She has also used nltk package to tokenize the words which would require one to download 'punkt' tokenizer to do so. I have used regular expression package and findall function available in it to extract the tokens from the text. This might have also contributed to the increased time complexity of her code.