CMSC 676 Project phase 1 Akarsh Kashamshetty

**Summary:**

Using the modules os, re, sys, bs4, glob, time I am able to parse the documents from the command line.

command: python3 main.py /files /outputDir

Submitted files:

Input Directory: /files

Output Directory: /outputLog

Sort by tokens: sortTokens.txt

Sort by frequency: sortFreq.txt

Tokenizer: main.py

The first and last 50 in sort by tokens and frequency: first50sortedTokens.txt, first50sortedTokens.txt, last50sortedFreq.txt, last50sortedFreq.txt

Partner files: partner\_lexical\_sort(1).txt, partner\_frequency\_sort(1).txt

**Approach**

* Beautiful Soup is a library for parsing html files.
* Used this library for getting text from the html file while traversing through each file from the input directory.
* While storing the tokens from each document, the tokens are also being added to the global dictionary.
* The key is the token name and value is the initialized with 1 and incremented whenever it is occurred again.
* Once all the files from the input directory is processed for token documents the global dictionary is populated with all the tokens with frequencies.

**Another approach** :

* I have used first to store all the text from all the documents in a single string as whole string.
* Then preprocessed the whole string for punctuations and numbers.
* Later tokenized them. Now created another set of unique strings using a set then using a loop traversed with each token in unique set on whole string to count the frequency of each token. This process is very inefficient.
* As there are almost 0.7 million tokens in whole string and almost 0.12 million are unique.
* To traverse them it takes 0.7 \* 0.12 million times and the estimated time seemed to take hours.
* So initially without knowing why program is taking time I spent a lot of time waiting for the program to complete. After further analysis I figured why it is going wrong and changed approach.

**Handling punctuations and numbers**

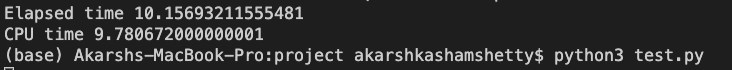
* The punctuations and numbers are handled using regular expression by replacing it with space.
* When used with the null character so many words are unable to get tokenized for example names from the mail address which get separated by @ For example, [johnson@gmail.com](mailto:johnson@gmail.com).
* The hyphen “-” is being replaced with null character because most of the cases it is important in giving the meaning to the word. For example, the word “water-jet” does not add value to context if we replace “-“with null and split it later.
* the ‘ is replaced with space so that the names like John’s will become John. This is more meaningful than Johns.
* And remaining punctuations like (, ), {, }, !. @, #, $,%, ? ..etc are replaced with space using regex.
* When tried to handle numbers I tried to keep only 4 and 10 digit numbers in the tokens so that any years or phone numbers will be not ignored but there are certain cases where the number is with attached with other word and is not making sense.
* So, I tried for so much time working on that but I could not do that. So replaced the numbers with null character.

**Incorrect Tokens**

* Some words are incorrectly tokenized due to the html files contain those words as it is.
* I am unable to handle the words which contains numbers. And while handling other words for ‘ some words like don’t became dont like that.
* And the words inside email address are not handled correctly so incorrect tokens are generated.

Efficiency of the frequency program in terms of magnitude and timings(cpu time, elapsed time)

The program is able to parse the html files in 10.9 seconds of elapsed time, 10.5 seconds cpu time.



Below is the documents vs time graph with documents on x- axis and time in seconds on y axis.

Chart, line chart

Description automatically generated

**Differences**:

My partner is Akshay Reddy Akkati. His Tokenizer is taking 13.36 seconds for parsing the html files in files directory. Where as my program is taking 10.1 seconds of elapsed time.

The approach my partner followed is the above approach I have explained during my approach about the other approach. Where he used string to store all the tokens from the all text from the html docs. Later used a for loop traversing all the tokens while also updating the frequency of each. As I mentioned above about the pitfalls of this tokenizer approach this makes it slow. But not as slow as the above approach because he used one traversal for counting frequency. My partner also handled words like cant, don’t ->can not, do not

However, we have got the same results in the final sorted tokens documents.

**Improvements**

Some improvements can be done in my program which can produce better results. If the numbers are handled correctly for years using regex, removing the tokens like ‘s’ when we handle John’s like words, ..etc.