Information Retrieval Homework 1

Pradeep Agrawal (IQ57213)

Language used: Python

Following is the step-by-step explanation of how I did this:

1. I downloaded the tarfile and then unzipped it. By doing this, I got 503 HTML files. I put all the files in a directory named 'files'.
2. Then I wrote a python code in which I used os.scandir to iterate through all the HTML files. In each iteration, I was checking if the current iteration holds a file.
3. For each valid file, I was calling a method 'extract_tokens' by passing the filepath. This method was responsible for extracting the words out of the HTML file.
4. For this word extraction, I am using BeautifulSoup (a python library). This library is providing me all the text present in the file. Once I got the text, I broke that into words by using split() method. After getting the words, I am using a Regular Expression to check if the words have any special characters or numbers. If they had such things, then I removed those characters. And then I took only those words which were greater than 2 in length and lowercase all of them.
5. Once I got those tokens, I passed those tokens to a method called 'create_token_files'. This method also requires output directory path. So, this method will check if the output directory exists. If not, will create that directory and then write these tokens into a text file separated by newline. So, this will result in having 503 txt files of tokens.

```python
def extract_tokens(file_path):
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as fp:
        soup = BeautifulSoup(fp, features="html.parser")
    for script in soup(["script", "style"]):
        script.extract()
    text = soup.get_text()
    words = text.split()
    words = [re.sub(r'[0-9]+|\W+|_', '', word) for word in words]
    words = [word.lower() for word in words if len(word) > 2]
    return words
```

```python
def create_token_files(output_dir, filename, words):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
        print("The new directory is created!")
    filepath = output_dir + '/' + filename.split('.')[0]+'.txt'
    with open(filepath, 'w') as f:
        for word in words:
            f.write(word)
            f.write('\n')
```

6. I wrote a method 'create_frequency_files' to create frequency files. So, when I got the token files, I called this method by passing output_dir as parameter. This method iterates over these text files and read words in them. Here I am using python dictionary to hold frequency of each word. So, if the word does not exist in the dictionary, then I add that word as key with 1 as its value. And if it exists then I increase its count by 1. At the end of this loop, I get frequencies of every word respectively in this dictionary. Using this, I created 2 files. One having words sorted by tokens alphabetically and second sorted by their respective frequency.

Code Execution Guide:

We just need to run the python file with 2 command line arguments (input files path, output files path). Keep in mind that we have to have the input-directory in the same path where we have the tokenize_search.py file.

python tokenize_search.py <input-directory> <output-directory>
Ex. python tokenize_search.py html-files-dir text-files-dir

In above command ‘tokenize_search.py’ if the python file name. ‘html-files-dir’ if the directory which holds all the HTML files and then ‘text-files-dir’ is the directory name where we will put all the text token files.
Following are screenshots of the output:

```
(search_engine@3.9.10) ➜  search_engine python3 tokenize_search.py files text_files_dir
All the times are in milliseconds
Parsed Files 100 and Time taken 1123.539
Parsed Files 200 and Time taken 881.482
Parsed Files 300 and Time taken 819.741
Parsed Files 400 and Time taken 685.798
Parsed Files 500 and Time taken 759.315
Total elapsed time: 4487.913
(search_engine@3.9.10) ➜  search_engine
```

Above shown the time taken in parsing those HTML files and converting them into token files. And finally, creating 2 frequency files.

Code Execution Flow:
When we run the command, it will start execution and will call the ‘main’ method that will run the entire script in the required sequence as shown in the screenshot.

```
def main(input_dir, output_dir):
    file_count = 0
    start = time.process_time_ns()
    for filename in os.scandir(input_dir):
        if filename.is_file():
            file_count += 1
            tokens = extract_tokens(filename.path)
            create_token_files(output_dir, filename.name, tokens)
        if file_count%100 == 0:
            end = time.process_time_ns()
            print(f'Parsed Files {file_count} and Time taken {(end - start)/1000000}')
            start = time.process_time_ns()
    create_frequency_files(output_dir)
```

Incorrectly Tokenized Words:
After this entire process, I investigated both the generated frequency files and noticed that there are some random words that does not make any sense are there. Those words are some random collection of characters.
Ex. abaloldalizasnak, abortusztanacsadas, aceliparreorganizacios, acsaladtamogatasok etc

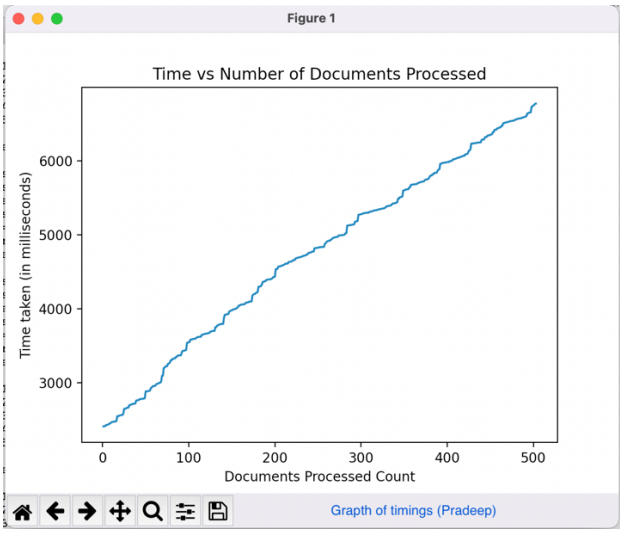
It seems like there were some special characters in between these words inside somewhere which got removed while cleaning the data and I end up getting such words.

Results Comparison:

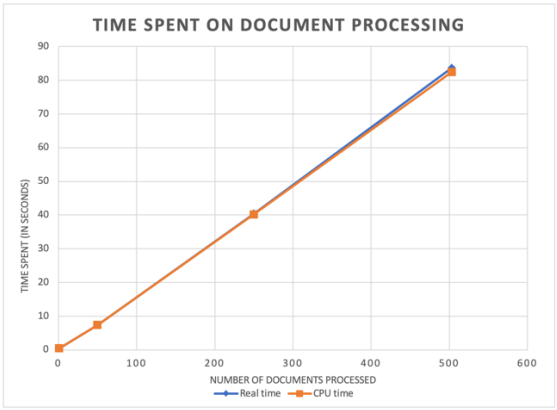
- I compared my results with Vinayak’s and Koby’s works. And I noticed that Vinayak is using Jsoup parser to parse the HTML files while I used BeautifulSoup.
- Vinayak is removing all the special characters and converting it into lowercase and then splitting them by whitespace. While I am removing special characters, numbers and symbols. In addition to that I am keeping only the tokens which have character length of greater than 2.

- Koby's frequency file have tokens like 'a', ''s', '79yr', '€' etc. While my frequency file contains only words which does not have any integers in them.
- Koby's code is taking approx. 83 seconds in the execution of code. While my code takes ~6 seconds in worst case.

Graphs:



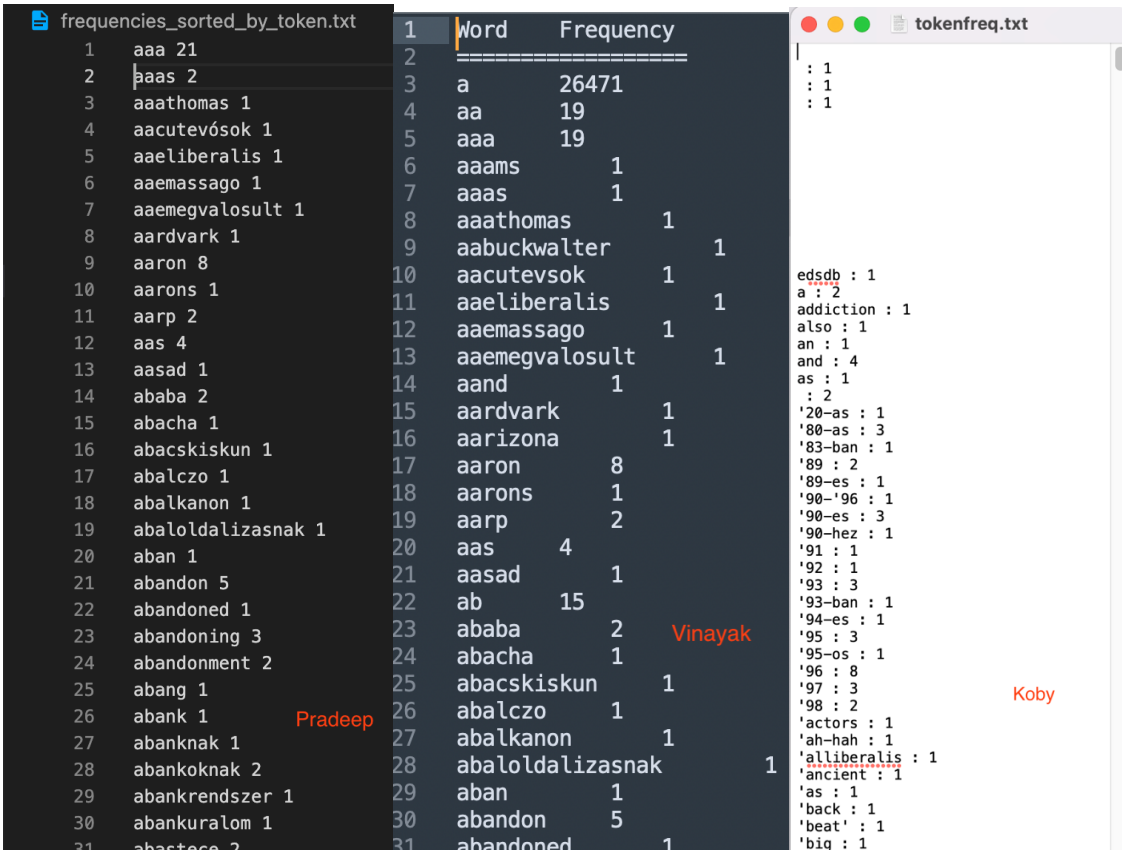
Graph of timings (Pradeep)



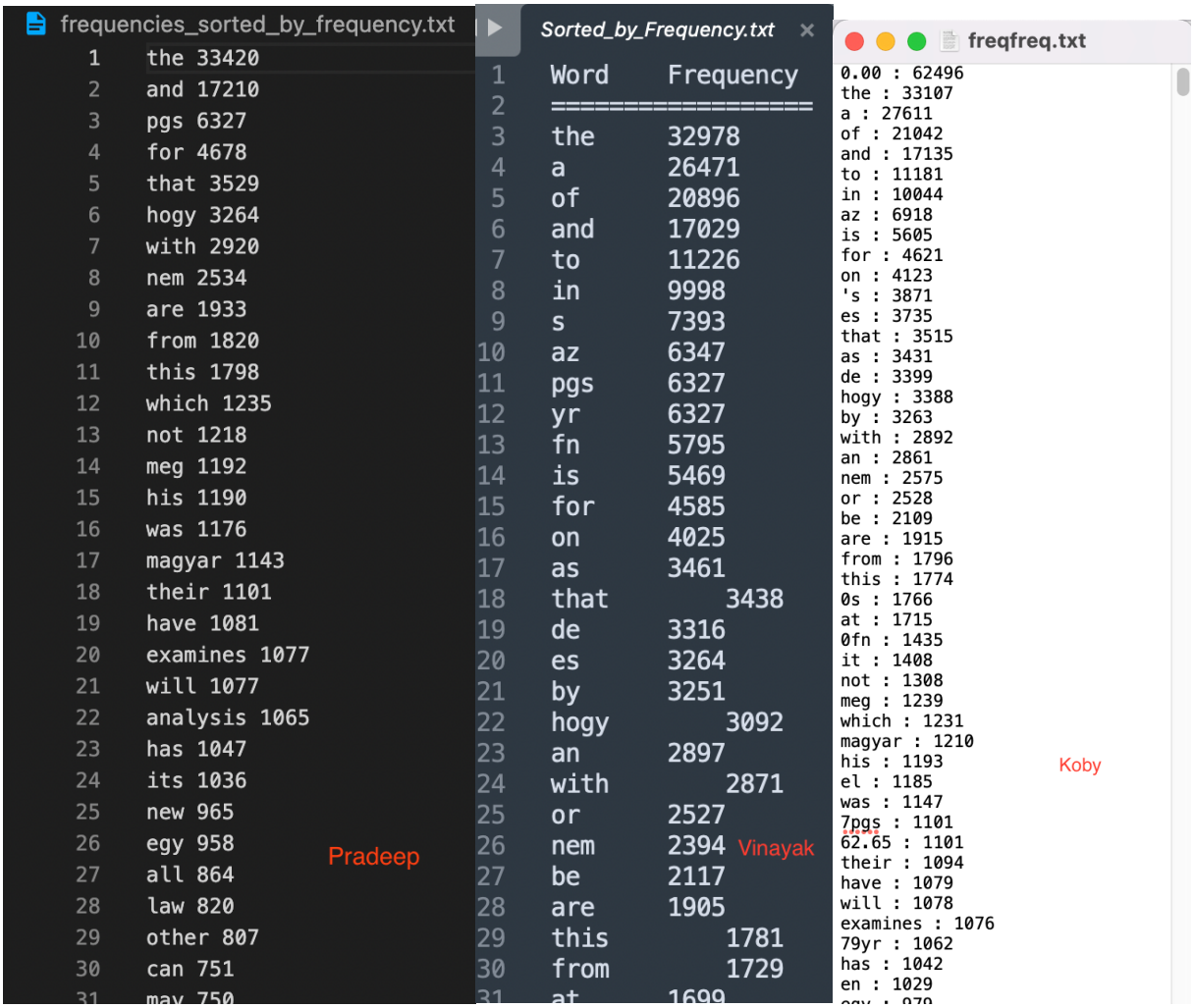
No. of docs	Real time	CPU time
1	0.46607866	0.460198
50	7.4119276	7.354634
250	40.4371991	40.240882
503	83.6204944	82.405139

Graph of timings (Koby)

Token files sorted by alphabets:



Token files sorted by frequencies:



frequencies_sorted_by_frequency.txt

```
1   the 33420
2   and 17210
3   pgs 6327
4   for 4678
5   that 3529
6   hogy 3264
7   with 2920
8   nem 2534
9   are 1933
10  from 1820
11  this 1798
12  which 1235
13  not 1218
14  meg 1192
15  his 1190
16  was 1176
17  magyar 1143
18  their 1101
19  have 1081
20  examines 1077
21  will 1077
22  analysis 1065
23  has 1047
24  its 1036
25  new 965
26  egy 958
27  all 864
28  law 820
29  other 807
30  can 751
31  may 750
```

Pradeep

Sorted_by_Frequency.txt

	Word	Frequency
1		
2	==========	==========
3	the	32978
4	a	26471
5	of	20896
6	and	17029
7	to	11226
8	in	9998
9	s	7393
10	az	6347
11	pgs	6327
12	yr	6327
13	fn	5795
14	is	5469
15	for	4585
16	on	4025
17	as	3461
18	that	3438
19	de	3316
20	es	3264
21	by	3251
22	hogy	3092
23	an	2897
24	with	2871
25	or	2527
26	nem	2394
27	be	2117
28	are	1905
29	this	1781
30	from	1729
31	at	1699

Vinayak

freqfreq.txt

```
0.00 : 62496
the : 33107
a : 27611
of : 21042
and : 17135
to : 11181
in : 10044
az : 6918
is : 5605
for : 4621
on : 4123
's : 3871
es : 3735
that : 3515
as : 3431
de : 3399
hogy : 3388
by : 3263
with : 2892
an : 2861
nem : 2575
or : 2528
be : 2109
are : 1915
from : 1796
this : 1774
0s : 1766
at : 1715
0fn : 1435
it : 1408
not : 1308
meg : 1239
which : 1231
magyar : 1210
his : 1193
el : 1185
was : 1147
7pgs : 1101
62.65 : 1101
their : 1094
have : 1079
will : 1078
examines : 1076
79yr : 1062
has : 1042
en : 1029
egy : 979
```

Koby