COMP3009J Information Retrieval

Worksheet 4

Download the following files from Brightspace:

- npl-doc-text.txt (a document collection for this task).
- stopwords.txt (a list of stopwords with one word per line).
- porter.py (an implementation of the Porter stemming algorithm: instructions on how to use it are included at the beginning of that file).

You should write all of your code in a single .py file.
Do not use any ‘import’ statements, except for “porter”.

File format:

- The npl-doc-text.txt file contains 11,429 small documents.
- Each document begins with a document ID on a line by itself (in fact, all lines that begin with numbers are document IDs).
- The end of each document is marked by a ‘/’ character with three spaces before it.
- The end of the document collection is marked by a ‘/’ character at the start of a line.

Write a Python program to do the following:

1. Read the stopwords list into a data structure that is suitable for quickly checking whether a word is a stopword.
2. Read the documents from the “npl-doc-text.txt” file.
3. For each document in the collection, divide it into its terms. Stopwords should not be included, and every term should be stemmed. These details must be stored in a suitable data structure so that you can perform the tasks that follow.
4. Count the number of times each term occurs in each document (the *frequency* of each term in each document).
5. For each document in the collection, print the frequencies its terms, in the following way:
    - First print the document ID.
    - Only print details of documents that have a term with a frequency of ≥ 2.
    - In each document, do not print terms with a frequency of only 1.
    - The terms should be printed in descending order of frequency (i.e. with the most frequent term first).

Sample output (first 10 documents):

```
1: capac (2)
2: comput (2)
3: electron (2) coordin (2)
5: logic (3) digit (2) system (2) coupl (2) circuit (2)
6: circuit (2)
7: side (2)
8: core (2)
9: circuit (4) switch (2) bidirect (2) nonlinear (2) binari (2)
11: circuit (3) switch (2)
14: reson (2) multipl (2)
```

Advanced

If you finish the above tasks, you can try the following:

Try to make this program run faster. On my computer (2.9GHz Intel Core i5, 2016 MacBook Pro) it runs in less than 1 second after making some optimisations.

You have already seen how `timeit` can be used to benchmark how long a piece of code takes to run. Another (simple) way is as follows:

```
import time

start_time = time.process_time()

# the code to time is here

end_time = time.process_time()

print( 'Time is {} seconds'.format( end_time - start_time ) );
```

Try to identify which part of the code (or which operation(s)) take the most time for your program (**Hint**: how many times does the word "frequency" appear in the document collection?). Typically, we can optimise programs if we avoid repeating the same operations again and again.