Assignment 4 – Extracting Term Frequencies for Indexing Tool

Preprocessing is an important task in search tools, but it is only the initial task, which is followed by the tasks of document indexing and document search. After your work with the preprocessing pipeline, your team lead gave you the task of building an indexing tool for text documents derived from web pages.

First, you will start from the result of your preprocessing pipeline, which are texts in lowercase, with no punctuation, without stop words, and keeping only the word stems. Each word stem will be called a *term*. Your task, then, is to index all the *terms* from a preprocessed text file, which we will call a *document*.

Your colleague at the team is building another part of this tool, which is a simple dictionary (or map) that maps a *web link* onto a *document*. Thus, you will not need to worry now with that mapping process, just stick with your document indexing tool.

Your indexing work will be split into parts. In this initial part of the indexer, you will create a new file containing the *term frequencies* of each *term* inside a *document*. That is, you will create a new file with one line for each *term*, and each line will contain the *term* and the *term frequency*, which is the number of times that *term* appear in the *document*. In a sense, this is a mapping inside a document, which may be aided by a dictionary in Python. After building a dictionary, you may simply dump its contents to an output file.

Programming environment

For this assignment, please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself one or two days before the due date to iron out any bugs in the Python programs you have uploaded to a lab workstation. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

Start your Assignment 4 by copying the files provided in **`/home/rbittencourt/seng265/a4/`** into your **`a2/`** directory inside the working copy of your local repository. Notice that the **`a2/`** directory is your project directory for you second project (the indexing tool, developed during Assignments 4, 5 and 6), and it not related to Assignment 2.

Finally, when you finish Assignment 4, tag its final commit with a release name **a4** and send it to the remote repo, so then you will be able to start working comfortably with Assignment 5, which will use the same **`a2/`** directory.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want **to** discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Roberto).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact the course instructor as soon as possible. (Code-similarity analysis tools will be used to examine submitted work.) The URLs of significant code fragments you have found online and used in your solution must be cited in comments just before where such code has been used.

Description of the task

You will develop one program in Python that reads text from the standard input and produces text sent to the standard output. The idea is simple and similar to previous assignments. Suppose you **`cat`** from a text file and pipe the results into your program and then redirect the output of your program to another file. Doing this provides general *term frequency counting* functionality and different parts of your code may be reused in other scenarios.

List 1 shows an original text extracted from a web page. It was preprocessed with your preprocessing pipeline, as shown in List 2. This preprocessed text will be the input to your *term frequency counter* tool, which should produce an output like the one in List 3, with each term from List 2 and its frequency, lexicographically sorted by term.

List 1: Example of contents from a text originated from a web page.

```
You were hired to work in a team that is building a new web search tool.
Without much experience on the subject, and counting only with a small team,
your team lead decided to take small steps.
Another teammate is building the crawler, which produces a file with web links.
On the other hand, you will be responsible for checking such text files with web links,
downloading the files, and organizing them in a local Unix filesystem.
```

List 2: Example of contents of a preprocessed text, which will be the input to the term frequency counter program.

```
you were hire work in team that is build new web search engin
without much experi on subject count onli with small team
your team lead decid take small step
anoth teammat is build crawler which produc file with web link
on other hand you will be respons check such text file with web link
download organ file in local unix filesystem
```

List 3: Example of contents of the output term frequency count generated by the term frequency counter program.

```
anoth 1
be 1
build 2
check 1
count 1
crawler 1
decid 1
download 1
engin 1
experi 1
file 3
filesystem 1
hand 1
hire 1
in 2
is 2
lead 1
link 2
local 1
much 1
new 1
on 2
onli 1
organ 1
other 1
produc 1
respons 1
search 1
small 2
step 1
subject 1
such 1
take 1
team 3
teammat 1
text 1
that 1
unix 1
web 3
were 1
which 1
will 1
with 3
without 1
work 1
you 2
your 1
```

Your program will be run from the Unix command line. Input is expected from **`stdin`**, and output is expected at **`stdout`**. **You must not provide filenames to the program, nor hardcode input and output file names**.

Implementing your program

The Python program you will develop in this assignment:

- reads line by line from the standard input and stores the count of each *term* in a dictionary that uses the *term* as key and the *term count* as value;
- whenever it finds a repeated term existent in the dictionary, it does not erase the existing key, just updates the current value associated to that key;
- stores the keys sorted lexicographically in the dictionary;
- finally, dumps all key value pairs separated by a space to the standard output, each pair in a different line.

Assuming your current directory contains your **`term-frequency-counter.py`** script, which has a shebang added by you at the beginning of your script file to find your Python interpreter, and a **`tfc-tests/`** directory containing the assignment's test files is also in the current directory, then the command to run your script will be.

```
% cat tfc-tests/in01.txt | ./term-frequency-counter.py
```

In the command above, output will appear on the console. You may want to capture the output to a temporary file, and then compare it with the expected output. The **`diff`** command allows comparing two files and showing the differences between them.

```
% cat tfc-tests/in01.txt | ./term-frequency-counter.py > temp.txt
% diff tfc-tests/out01.txt temp.txt
```

The same thing (i.e., producing output and comparing it with the expected output) can be combined into a one-liner:

```
% cat tfc-tests/in01.txt | ./term-frequency-counter.py | diff tfc-tests/out01.txt -
```

The ending `hyphen/dash` informs **`diff`** that it must compare the contents of **`tests/out07.txt`** with the input piped into **`diff`**'s **`stdin`**.

You should commit your code whenever you finish some functional part of your it. This helps you keep track of your changes and return to previous snapshots in case you regret a change. When you are confident that your term frequency counter code is working correctly, push it to the remote repository. Do not forget to tag your final commit with a release name **a4** and send it to the remote repo.

Exercises for this assignment

You may develop your code the way that suits you best. Our suggestions here are more for facilitating your learning than as a requirement for your work. You may not need to do the exercises in the item 1 below if you want to practice deeper problem solving skills. But, in case you get stuck, you may look at them as a reference. On the other hand, if Python programming seems difficult to you, you may use them as a script to learn and practice.

1. Write your program **`term-frequency-counter.py`** program in the **`a2/`** directory within your git working copy (Recall that you are starting a new indexing tool, which is different from the preprocessing tool from Assignments 1 and 2, and that is the reason why you are using a new project directory named **`a2`**).
    a. Practice with **`stdin`** by reading it line by line with a for loop and **`sys.stdin.`** and printing the output with **`print()`**.
    b. Play with standard input redirection, by redirecting input to read from a file instead of reading from the keyboard.
    c. Play with the standard output, using **`print()`** to print on your terminal screen the output as it is required.
    d. Play with standard output redirection, by redirecting **`stdout`** to save data in a file instead of sending them to the console.
    e. Learn to remove the newline character from a line by either using **`strip()`** or Python slices.

f. Learn to tokenize a line in Python by using **`split()`**. As an example, print each token in a different line to see your tokenizer working.
g. Practice with Python dictionaries to add new key-value pairs and with the dictionary **`.get()`** method to deal with existing keys in a dictionary.
h. Create a dictionary in your program, and store the values for each term as you tokenize them.
i. Check whether your dictionary shows your results sorted, iterating over your keys. Recall that if you use a different Python version in your laptop, that may be different from the version in the lab environment. If you need to sort a dictionary output, learn to use the **`sorted()`** function.
j. Finally, use a for loop to send the output requested by the assignment from your dictionary to the standard output.
k. Test your program using pipes and the **`diff`** tool with the first test file as explained in the section above.
l. Have you thought about modularizing and commenting your code during the development? If not, now it would be a good time to separate parts of your code into different functions and document your code as well, in case you want an "A" grade.

2. Commit your code frequently (**`git add`** and **`git commit`**), so you do not lose your work. We will look at the code commits you did in this assignment. We require at least three different commits (you may either use the split of your work into parts as suggested at the start of the previous section or do your own split). Our final grading will take that into account.
3. When you are done with your commits, do not forget to **`git push`** them into your repo.
4. Use the test files in the lab-workstation filesystem located in **`/home/rbittencourt/seng265/a4/`** (i.e., the **`tfc-tests/`** subdirectories of **`a4/`**, all inside lab-workstation filesystem) to guide your implementation effort. Start with simple cases (for example, the one described in this write-up). In general, lower-numbered tests are simpler than higher-numbered tests. Refrain from writing the program all at once, and budget time to anticipate for when "things go wrong". There are two pairs of test files in the directory.

What you must submit

- You must submit 1 (one) single Python source file named **`term-frequency-counter.py`**, within your **`git`** repository (and within its **`a2/`** subdirectory) containing a solution to this assignment (We are using the **`a2/`** subdirectory both for assignments #4, #5 and #6. Ensure your work is committed to your local repository and pushed to the remote before the due date/time. (You may keep extra files used during development within the repository, there is no problem doing that. But notice that the graders will only analyze your **`term-frequency-counter.py`** file.) Do not forget to tag your final commit with a release name **a4** and send it to the remote repo.

Evaluation

Our grading scheme is relatively simple and is out of 100 points. Assignment 4 grading rubric is split into five parts.

1) Modularization - 10 points - the code should have appropriate modularization, dividing the larger task into simpler tasks (and subtasks, if needed);
2) Documentation - 15 points - code comments (enough comments explaining the hardest parts (loops, for instance), no need to comment each line), function comments (explain function purpose, parameters and return value if existent), adequate indentation;
3) Version control - 15 points – Appropriate committing practices will be evaluated, i.e., your work cannot be pushed in just one single commit, its evolution will have to happen gradually;
4) Tests: Part 1 - 30 points - passing test 1 in **`tfc-tests/`** directory: **`in01.txt`** and **`out01.txt`**;
5) Tests: Part 2 - 30 points - passing test 2 in **`tfc-tests/`** directory: **`in02.txt`** and **`out02.txt`**.

We will only assess your final submission sent up to the due date (previous submissions will be ignored). On the other hand, late submissions after the due date will not be assessed.