COMPSCI 121 / IN4MATX 141
Assignment 1 - Text Processing Functions
Due date: 1/12/2019 at 11:59pm

This assignment is to be done individually. You cannot use code written by your classmates. Use code found over the Internet at your own peril - it may not do exactly what the assignment requests. If you do end up using code you find on the Internet, you must disclose the origin of the code. **Concealing the origin of a piece of code is plagiarism**. Use the discussion section for general questions whose answers can benefit you and everyone.

General Specifications

1. Programing language: You must use either python2 or python3 for this assignment. The next homework will use crawlers written in Python 2.7.x where x is the latest version of 2.7, so we encourage using this homework to brush up your knowledge of Python.

2. Make sure to break down your program into classes/methods/functions and put enough comments for them to make the program understandable. **Otherwise, you will lose credits for the understanding part** (see evaluation criteria).

3. At points, the assignment may be underspecified. In those cases, make your own assumptions and be prepared to defend them.

4. There won't be any face to face meetings for this assignment.

5. For both part A and part B, please add a brief runtime complexity explanation for your code as a comment on top of each method or function. (See description of runtime complexity in Part B of the assignment). **This explanation and your code actual conformance with this explanation will be the basis for evaluating the performance of your program.**

6. You are allowed to use libraries like regex or NLTK, but you should know how they do work internally. **Also, pay attention that they may not satisfy this assignment requirements**. It is highly recommended that you write the code yourself to learn more in the process.

7. Exception handling is required for bad inputs. An example of bad input would be a character in a non-english language. Your code should be able to tokenize the whole input file even though there may be some bad inputs in it. You should be able to skip the bad input and continue with the rest. If your code throws an exception in the middle of tokenizing a TA's input test case, you will lose the whole credit for that test case.

8. All of the TA's input test cases will be UTF-8 encoded.
9. **Do not hard code the input file names in your code or read them from system standard input (stdin). You should get the file names from command line arguments. As the assignment will be graded using an automatic grader, not doing this will result in losing the whole credit for the assignment.**
10. Your code should work for the large files that won't fit in memory.

Part A: Word Frequencies (40 points)

Write a program that takes a text file as a command line argument and outputs the word frequency in decreasing order.

Briefly, implement the following functionality:

1. Read an input file passed as a command line argument
2. Tokenize the text of the file. A token is a sequence of alphanumeric characters (a-zA-Z0-9), independent of capitalization (so *"Apple" and "apple"* are the same token)
3. Counts the number of occurrences of each word in the tokens generated
4. Print out the word frequency counts onto the screen. The print out should be ordered by decreasing frequency. (so, highest frequency words first). Resolve ties alphabetically in ascending order.

Here is an example of an input file and corresponding output after computing the word frequency:

Input:

Here's a fun-fact! White tigers live mostly in "India".
Wild lions live mostly in "Africa".

Output:

in	2
live	2
mostly	2
a	1
africa	1
fact	1
fun	1
here	1
india	1
lions	1

s 1
tigers 1
white 1
wild 1

Each line of the output should use "[token]\t[frequency]" formatting where you need to replace [token] with the token name and [frequency] with the token frequency. For example, if you have two tokens: hello with frequency 1 and world with frequency 2, the output should be:

world\t2
hello\t1

Note that "\t" will be changed to a tab (couple of spaces) when printed on the console.

IMPORTANT NOTE: Do not print any extra lines in the output as it will result in losing credit.

The TA will use a test text file during the evaluation of your assignment. For this part, it is expected that your program will read this text file, tokenize it, count the tokens, and print out the word (token) frequencies in descending order.

Part B: Intersection of two files (60 points)

Write a program that takes two text files as arguments and outputs the number of tokens they have in common. Here is an example of input/output:

Input file 1:
We reviewed **the trip** and credited **the cancellation fee**. **The driver has** been notified.

Input file 2:
If a **trip** is cancelled more than 5 minutes after **the driver**-partner **has** confirmed **the** request, a **cancellation fee** will apply.

Output
6

IMPORTANT NOTE: Do not print any extra lines in the output as it will result in losing credit.

You can reuse code you wrote for part A.

The TA will use a test text file during the evaluation of your assignment. Note that some of the text files may be VERY LARGE.

Once you have implemented your program, please perform an analysis of its runtime complexity: does it run in linear time relative to the size of the input? Polynomial time? Exponential time?

For this part, programs that perform better will be given more credit than those that perform poorly.

Submitting Your Assignment

You must have two separate files for part A and part B of the assignment named "PartA.py" and "PartB.py" respectively. These files should be in the root of a zip file that you will upload to Canvas. Your zip file structure should be:

Assignment1.zip
---------- PartA.py
---------- PartB.py

You can replace Assignment1 with whatever name you think is appropriate. You don't need to add your UCInetID or student number to the zip file name. Canvas will do that automatically when we are downloading your assignments. Do not zip the directory containing these two files! So the following examples are not a correct structure:

Assignment1.zip
---------- Assignment 1
---------- PartA.py
---------- PartB.py

Assignment1.zip
---------- PartA
---------- PartA.py
---------- PartB
---------- PartB.py

This is necessary as the automatic grader can only work with this structure. Again, please pay attention that you only need to upload **ONE** zip file containing two python files at shown above and not one zip file for each part.

Grading Process

The correctness of your programs will be evaluated against a set of test cases using an automated grader. If necessary the results will be reviewed by a TA or Reader as well. Then, your source code will be evaluated by a TA or Reader for Understandability and Performance.

Evaluation Criteria

Your assignment will be graded on the following four criteria.

1. Correctness (40%)
   a) How well does the behavior of the program match the specification?
   b) How does your program handle bad input?
2. Understanding/structure/comments in the code (30%)
   a) Do you demonstrate understanding of the code submitted?
3. Efficiency (30%)
   a) How quickly does the program work on large inputs?