Assignment 1: Text Processing

Due Jan 27 by 11:59pm **Points** 10 **Submitting** a file upload **File Types** zip
Available Jan 15 at 12am - Jan 28 at 11:59pm 14 days

This assignment was locked Jan 28 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 Piazza for general questions whose answers can benefit you and everyone.

General Specifications

1. You can use any programming language. The next homework will use crawlers written in Python, so you may want to use this homework to brush up your knowledge of Python.
2. Make sure to break down your program into classes/methods/functions corresponding to the parts in this specification. They will be tested separately.
3. The function signatures in this specification are informal; their purpose is to explain the inputs and outputs of the methods.
4. **Important:** At certain points, the assignment may be underspecified. In those cases, make your own assumptions and be prepared to defend them.

Part A: Word Frequencies (40 points)

- **Method/Function:** List<Token> tokenize(TextFilePath)
  Write a method/function that reads in a text file and returns a list of the tokens in that file. For the purposes of this project, a token is a sequence of alphanumeric characters, independent of capitalization (so *Apple*, *apple* are the same token).
- **Method:** Map<Token,Count> computeWordFrequencies(List<Token>)
  Write another method/function that counts the number of occurrences of each token in the token list.
- **Method:** void print(Frequencies<Token, Count>)
  Finally, write a method that prints out the word frequency counts onto the screen. The print out should be ordered by decreasing frequency. (so, highest frequency words first)

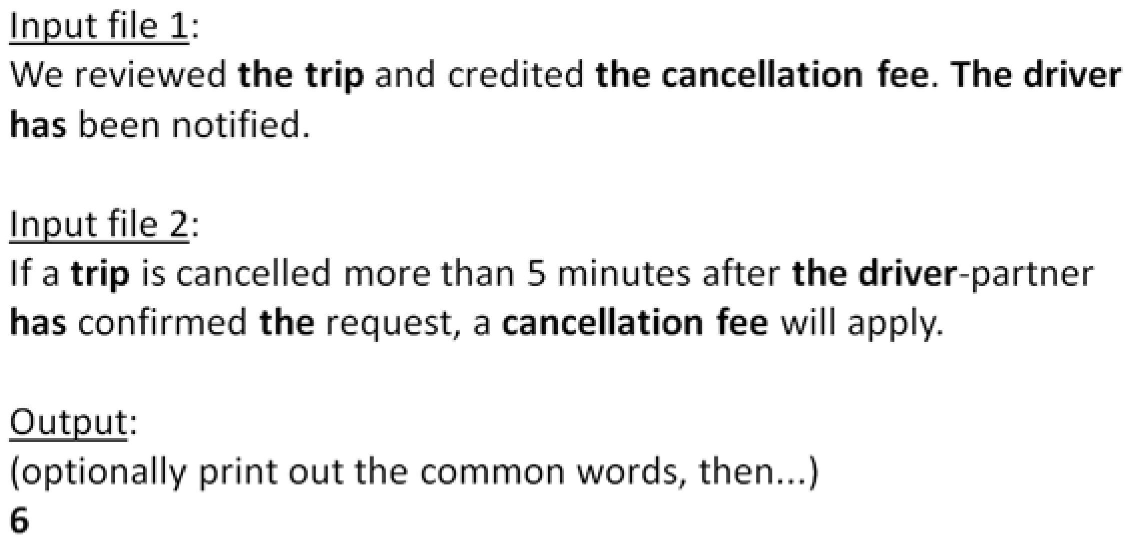
The TA will use their own test text files. For this part, it is expected that your program will read this text file, tokenize it, count the tokens, and print out the token (word) frequencies.

Please, use one of the output formats when you print out the result:

<token>\t<freq>
<token> <freq>
<token> - <freq>
<token> = <freq>
<token> > <freq>
<token> -> <freq>
<token> => <freq>

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:
(optionally print out the common words, then...)
6

- You can reuse the code you wrote for part A.
- The TA will use their own text files. Note that some of the text files may be VERY LARGE.
- For this part, programs that perform better will be given more credit than those that perform poorly.

Common Tasks

- 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 (does it run in linear time relative to the size of the input? Polynomial-time? Exponential time?). **This explanation and your code's actual conformance with this explanation will be the basis for evaluating the performance of your program.**
- **You should get the file names from command line arguments.** Do not hard code the input file names in your code or read them from system standard input (stdin). As the assignment will be graded using an automatic grader, not doing this will result in losing the whole credit for the assignment.

- **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.

Submitting Your Assignment

Your submission should be a single zip file containing two programs, one for part A, the other for part B. Something like this:

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

Submit it to Canvas.

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 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%)
    1. How well does the behavior of the program match the specification?
    2. How does your program handle bad input?
2. Understanding (30%)
    1. Do you demonstrate an understanding of the code?
3. Efficiency (30%)
    1. How quickly does the program work on large inputs?