Information Retrieval

Assignment 1 - Text Processing Functions

Due date: 1/20 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 Message Board 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 either Java or Python, so you may want to use this homework to brush up your knowledge of one of those languages.
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 method signatures in this specification are informal; their purpose is to explain the inputs and outputs of the methods. You should think carefully about which data structures to use within the programming language you chose to use. Points will be taken out for using inefficient data structures.
4. At points, the assignment may be underspecified or slightly wrong. In those cases, make your own assumptions and be prepared to defend them.

Part A: Utilities (20 points)

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: `List<Token> tokenizeFile(TextFile)`

Write another method that prints out a list of tokens onto the screen.
Method: `void print(List<Token>)`

The Reader will provide a test text file during the f2f presentation. For this part, it is expected that your program will read this text file and print out the corresponding list of tokens.

Part B: Word Frequencies (20 points)

Write a method/function that counts the total number of words and their frequencies in a token list.

Method: `Pairs<Token,Count> computeWordFrequencies(List<Token>)`

Write another 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)

Method: `void print(Pairs<Token,Count>)`

The Reader will provide a test text file during the f2f presentation. 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.

Part C: 3-grams (30 points)

A 3-gram is three words that occur consecutively in a file. For example, *three words that,* and *words that occur* are all 3-grams from the previous sentence.

Write a method/function that counts the total number of 3-grams and their frequencies in a token list.

Method: `Pairs<3Gram,Count> computeThreeGramFrequencies(List<Token>)`

Write another method that prints out the 3-gram frequency counts onto the screen. The print out should be ordered by decreasing frequency. (so, highest frequency 3-grams first)

Method: `void print(Pairs<3Gram,Count>)`

The Reader will provide a test text file during the f2f presentation. For this part, it is expected that your program will read this text file, tokenize it, count the 3-grams, and print out the 3-gram frequencies.

Part D: Anagrams (30 points)

An anagram is the result of rearranging the letters of a word [or phrase] to produce a new word [or phrase]. For example: *silent* is an anagram of *listen*.

Write a method/function that detects all tokens that have anagrams in a list of tokens, and lists those anagrams.

Method: `Pairs<Token,List<Anagram>> detectAnagrams(List<Token>)`

Use WordNet. http://wordnet.princeton.edu/wordnet/download/

Once you have implemented your anagram detection algorithm, please perform a short analysis of its runtime complexity (does it run in linear time relative to the size of the input? Polynomial time? Exponential time?)

Write another method that prints out the tokens and their anagrams onto the screen. The print out should be ordered alphabetically by token.

Method: `void print(Pairs<Token,List<Anagram>>)`

The Reader will provide a test text file during the f2f presentation. For this part, it is expected that your program will read this text file, tokenize it, detect the anagrams, and print them out. For this part, the reader will also analyze your anagram detection algorithm; algorithms that perform better will be given more credit than those that perform poorly.

Submitting Your Assignment

Your submission should be a single zip file submitted to EEE containing all your source code.

Late Submission Policy

-1% for every hour past the deadline for the first 24 hours. Flat -25% until 7 days late. No submission accepted past one week after due date.

Grading Process

You will meet with the grader for about 10 minutes during the week starting on 1/21. During that meeting, be ready for the following:

1. Show your program running on your own computer, on an input file provided by the grader on a USB stick
2. Answer questions about your program (as submitted to EEE) and its behavior (during the meeting)

You should **not** continue to work on your program once you submit it to EEE, as the reader will be looking at your submitted code.

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 (30%)
   a) Do you demonstrate understanding of the code?
3. Efficiency (30%)
   a) How quickly does the program work on large inputs?