Purpose

The goal of this lab is to give you some experience with dealing with large amounts of data, and in choosing appropriate algorithms and data structures to deal with it.

Preparation

We're going to use "War and Peace" from Project Gutenberg (Links to an external site.). You can grab a slightly cleaned-up version here: war_and_peace.txt.

Assignment

1. Use the `wc` command in Linux to see how many words are in "War and Peace". There are a lot. More than half a million. Read some of the book. It's a classic.

2. Write a class, called Book, that reads in all of the words in War and Peace and stores them in a list. It should be initialized like this:

   ```
   b = Book('war_and_peace.txt')
   ```

3. Implement a class function called `number_of_words()` that returns the number of words in the book.

4. Write a class function that checks how many words in one book are not in another book. Use this function to check how many words in the text of "War and Peace" are not in the dictionary on your computer (such as `/usr/share/dict/words`). **You should use operators to do this**, and the **returned list** should have no duplicates in it. For example, if `a` and `b` are both `Book()`s, then `a - b` should return a list of the words in `a` but not in `b`.

5. Think about three reasons why the previous part of the assignment is unbearably slow: one to do with the data, one to do with the data structure, and one to do with your algorithm. Grab a TA, and tell them why. Suggest one easy way to speed up your code.

6. Implement the change you described to the TA. Run your code, and verify that it goes faster.

7. Print out some of the words not in the dictionary. What do you notice about them? Implement code that gets rid of the extra stuff so that they are correctly identified as dictionary words. You probably want to do this in the class constructor `__init__`.

8. Write code to determine the most common *n* words in "War and Peace". Put the code in a class function called `common_words(self, n)`, which returns an ordered list of the *n* most common words in the Book.

9. **Graduate Students and Extra Credit for Undergraduates:** Write code to determine the frequency of letters used in "War and Peace". Print out the letters in frequency order (most frequent first), along with their counts, nicely formatted, using a class function `print_letter_frequencies`.

Grading

There are 8 points on offer for ME 499 students, and 12 points for ME 599 students. ME 499 students can earn extra credit for doing the ME 599 parts of the lab.

1. 1 point for reading in all the words in the text, and getting the same word count as the Linux `wc` command. [1 point total]

2. 1 point for code that compares the Book to the dictionary (also stored as a book), and determines how many words in the text are not in the dictionary. 1 point for showing this on some dictionary file and some text. [2 points total]

3. 2 points for successfully implementing a change to dramatically speed up your code. [3 points total]

4. 2 points for fixing two problems that stopped words from being correctly identified as dictionary words. [2 points total]

5. 2 points for determining the most common word.

6. **Graduate Students and Extra Credit for Undergraduates:** 2 points for determining the correct letter frequency.

Thoughts

1. The word "dictionary" in this lab may be ambiguous. It could mean either the data structure or the dictionary file on your computer. Please ask a TA if you're confused at any point.

2. **DO NOT USE** Counter from `collections` for this lab.

3. If you don't know about the `wc` command, I'm willing to bet that Google does. So does linux: type `man wc` to find out more. What's the `man` command? Type `man man` to find out.

4. The sample code will return the words from the novel line-by-line, which is not what you want. You can split this into a list of words using the `split()` function. I'd suggest you use something like `line.split()` to do the work. We'll talk about strings in class soon, but the documentation will fill you in on what this is doing.

5. Checking the words in the text against the dictionary will take a while. You should make sure your code works with a somewhat smaller data set than the entire text and the entire dictionary.

6. Think about where to put the code in your class, so that it works most efficiently. For example, it would make little sense to read in the dictionary every time you need to use it, so that code should go in the constructor.

7. Think of your dictionary file as another Book.

8. Call your python file words.py. Don't hardcode your username in the file, since we'll be running it from elsewhere. Make sure you include all of the files you need in your Canvas submission.

The Rules

Everything you do for this lab should be your own work. Don't look up the answers on the web, or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.