22C:16 (CS:1210) Homework 4

Due via ICON on Friday, April 18th, 4:59 pm

What to submit: Your submission for this homework will consist of three text files, named `hw4a.py`, `hw4b.py`, and `hw4c.py`. These should contain Python programs for Problems (a), (b), and (c) respectively. The file `hw4a.py` should contain the definition of the function `CaeserCipher` and the main program that encrypts a plain text file. The file `hw4b.py` should contain the functions `parseSentences`, `replaceNonLetters`, `parseWords`, `computeFrequenciesFast`, and `mergeDictionaries` followed by a main program. In all cases, feel free to create an use helper functions. All files should each start with with a comment block containing your name, section number, and student ID. You will get no credit for this homework if your files are named differently, have a different format (e.g., docx), and if your files are missing your information. For this homework (and all future homeworks), make sure that your program is carefully documented and variables names are chosen with care.

(a) To solve this problem, look up *Caeser Cipher* on Wikipedia.

(i) Write a function called `CaeserCipher` that takes as parameters a string `s` and an integer `k` and replaces every letter (upper case or lower case) in `s` by the letter obtained by shifting the original letter by the amount `k`. The function should return the encrypted text as string. For example, if the function is called as

```
CaeserCipher("healthy", 2)
```

then the function should return `"jgcnvja"`. Notice that shifting `"y"` by 2 causes a wrap around the sequence of lower case letters and yields `"a"`.

Notes: (i) The parameter `k` may be positive, 0, or negative causing a shift to the right, no shift, and a shift to the left respectively. (ii) Characters in `s` that are not letters should remain as they are. (iii) Lower case letters should be replaced by other lower case letters and upper case letters should be replaced by other upper case letters.

(ii) Write a program that reads some plain text from a file called `plain.txt`, asks the user for a Caeser Cipher key `k`, and produces as output the corresponding encrypted text in a file called `encrypted.txt`. The plain text should be encrypted using the given Caeser Cipher key `k`. For example, the file `plain.txt` might be the following:

```
Now we are engaged in a great civil war,
testing whether that nation, or any nation, so conceived and so dedicated,
can long endure.
                                        -- Abraham Lincoln, Nov 19, 1863
```

Suppose that when your program asks for a Caeser Cipher key, the user responds by entering `4`. Given below is the file `encrypted.txt` obtained from `plain.txt` by using 4 as the Caeser Cipher key. Notice how the output produced by the program does not disturb the punctuation or the whitespaces.

```
Rsa ai evi irkekih mr e kviex gmzmp aev,
xiwxmrk alixliv xlex rexmsr, sv erc rexmsr, ws gsrgimzih erh ws hihmgexih,
ger psrk irhyvi.
                                        -- Efveleq Pmrgspr, Rsz 19, 1863
```

(b) For various purposes, it would be nice to have a sizeable "dictionary" of English words. In this problem, you are asked to build such a dictionary by processing the texts `war.txt`, `hyde.txt`, and `illiad.txt` and extracting all words from these texts that are not proper nouns. Your program should finally write output to a file called `dictionary.txt`. When

your program has completed execution, this file should contain a sequence of distinct words, one-per-line, in alphabetical order. You should use the definition of "word" that we used in Homework 3 and in the program `characters.py` that we implemented in class. Also, you should use the approach in `characters.py` to identify proper nouns. (Recall that in our implementation of `characters.py` we defined a proper noun as a word that starts with an upper case letter that does not start a sentence.)

(i) To start your program, copy the functions `parseSentences`, `replaceNonLetters`, and `parseWords` from the program `characters.py` that we wrote in class. Then implement a function called `computeFrequenciesFast` that takes the "nested" list of words produced by `parseWords` and returns a collection of distinct words that are not proper nouns and their frequencies. The words and their corresponding frequencies should be stored as key-value pairs in a Python dictionary. Also, the words should all be in lower case.

For example, suppose that `L` is the list:

```
[["The", "name", "is", "Bond", "James", "Bond"],
 ["Where", "is", "the", "treasure"],
 ["Bond", "cackled", "insanely"]]
```

Then the function call

```
computeFrequenciesFast(L)
```

should return the dictionary

```
{"the": 2, "name": 1, "is": 2, "where": 1, "treasure": 1,
 "cackled": 1, "insanely": 1}
```

Notice that the words `"Bond"` and `"James"` are eliminated because these are proper nouns. This is according to our rule that any word that occurs in a capitalized form not at the beginning of the sentence is a proper noun. Also, notice that all the remaining words appear in lower case. Finally, notice that the remaining words appear as keys of the dictionary with their frequencies appearing as corresponding values.

Note: In Homework 3, you wrote a function called `computeWordFrequencies`. The function you are supposed to write now, `computeFrequenciesFast` is quite similar, except that this function uses a Python dictionary as the data structure for maintaining word-frequency pairs. As a result, you should expect `computeFrequenciesFast` to be much faster than `computeWordFrequencies`.

(ii) Write a function called `mergeDictionaries` that takes two parameters, say `bigDict` and `smallDict`, each being a Python dictionary containing word-frequency pairs. The function should update `bigDict` in the manner shown in the following example. If `bigDict` is the dictionary {`"the":2, "name":1, "is":2`} and `smallDict` is the dictionary {`"skill":4, "name":1, "the":3`} then `bigDict` should be updated to {`"the":5, "name":2, "is":2, "skill":4`}. In the updated dictionary, the word `"the"` has frequency 5 because that is the sum of the frequencies of `"the"` in the two dictionaries.

(iii) Now write the main program that constructs three word-frequency Python dictionaries, one for `war.txt`, one for `hyde.txt`, and one for `illiad.txt`. Your main program needs to call `parseSentences`, `parseWords`, and `computeFrequenciesFast` in order to do this. It then "merges" these three dictionaries into one `masterDictionary` by calling the function `mergeDictionaries` a couple of times. Finally, it extracts from `masterDictionary` words with frequency at least 10 and writes these out into the file `dictionary.txt` in alphabetical order.

(c) In Problem 1 you started with a plain text document and converted it into an encrypted document using the Caeser Cipher. In this problem, your task is to *decrypt* a given encrypted text. Write a program that reads from a file called `encrypt.txt`. You can assume that `encrypt.txt` was produced from some English language plain text file using the program you wrote in Problem 1. The main challenge for decryption is that you don't know the Caeser Cipher key that was used for decryption. The approach I propose to discovering this key is to try all 26 possible key values and pick the key value that turns the encrypted text into understandable English text. To determine if the decrypted text is understandable you should consult the file `dictionary.txt` that your program in Problem 2 produced.

In more detail, the algorithm I am suggesting is as follows. Start by reading words from `dictionary.txt` and store these in a Python dictionary. Then read the file `encrypt.txt` and extract words from this file. Then consider each possible value of the Caeser Cipher key $k = 0, 1, \ldots, 25$ and for each value of k, decrypt each of the extracted words. Then check how many of these decrypted words appear among the words in your word dictionary. One would expect that for all except one value of k we would get gibberish when we decrypt and therefore almost none of the words (for that value of k) would be in your word dictionary. However, for one value of k many of the decrypted words would be in the word dictionary. Therefore, you should pick the value of k that maximizes the number of decrypted words that appear in the word dictionary and use this value of k to decrypt the text. Write your decrypted output to a file called `plain.txt`.
