CMPUT 331, Fall 2020, Assignment 5
version f20.2

All assignment submissions must conform to the Assignment Submission Specifications *posted on eClass. Ensure that your submission follows these specifications before submitting your work.*

You will produce a total of four files for this assignment: "a5p1.py" for problem 1, "a5p2.py" for problem 2, your "a5.pdf" or "a5.txt" for problem 3, and a README file (also in either PDF or plain text).

All encipherment and decipherment in this assignment uses the simple substitution cipher. Lowercase and uppercase letters are to be treated as the same.

Problem 1

Create a Python module named "a5p1.py", containing functions named "freqDict(ciphertext)" and "freqDecrypt(mapping, ciphertext)". When invoked with a text enciphered using the substitution cipher, the "freqDict(text)" function should perform frequency analysis on the entire text, using the frequency statistics methods from the textbook, and return a dictionary whose keys are the cipher characters, with the value for each key being the plaintext character it is assigned to using frequency analysis. For English texts that have been deciphered with the correct key, the value of the most frequent cipher character should be 'E', and the value of the least frequent cipher character should be 'Z'. If a character does not appear in the ciphertext, it should have a value of 0 in the dictionary. If two or more letters occur the same number of times in the ciphertext, the letters that occur earlier in the alphabet should be considered to have a higher frequency. The "freqDecrypt(mapping, ciphertext)" function simply accepts a dictionary mapping of characters (in the format returned by "freqDict") and returns the resulting text after applying the mapping to each character in the ciphertext.

Note: This method of cracking the substitution cipher is far from perfect and it is unlikely to return the correct key.

We provide the letters in the alphabet sorted by the frequency in English (from Wikipedia's article on Letter Frequency):

```
ETAOIN = "ETAOINSHRDLCUMWFGYPBVKJXQZ"
```

Problem 2

There are two primary ways of evaluating a solution to a simple substitution cipher: *key accuracy* and *decipherment accuracy*. Key accuracy is the proportion of cipher character types in the alphabet which are mapped to their correct plaintext characters. Decipherment accuracy is the proportion of cipher character tokens in the ciphertext which are mapped to their correct plaintext characters.

Create a Python module named "a5p2.py" that contains a function named "evalDecipherment(text1, text2)" where text1 is a plaintext and text2 is a version of text1 that has had some substitution applied to it (via freqDecrypt). evalDecipherment should compare the two files, and return a list containing two fields that correspond to the key accuracy and decipherment accuracy of text2 w.r.t the plaintext, text1. For example, if $t1$ contains the plaintext "this is an example", $t2$ might contain "tsih ih an ezample" – the text in $t1$ was enciphered, and $t2$ contains an imperfect attempt to decipher it. This decipherment has a key accuracy of 8/11, since there are 11 character types, and three of them, 'h', 's', and 'x', were deciphered

incorrectly; it also has a decipherment accuracy of 11/15, since it is 15 characters long, but only 11 character tokens in the decipherment are correct. Thus, the function should return the list [0.7272727272727273, 0.7333333333333333]. Your program should only count alphabetical characters in its decipherment evaluation and it should be case-insensitive.

Problem 3

Short answer: You now have at least two different ways of hacking the simple substitution cipher: frequency analysis (deciphering the n^{th} most frequent cipher symbol into the n^{th} most frequent plaintext symbol), and the fully-automated dictionary-based method presented in the textbook file, simpleSubHacker.py. Included with this assignment is a zip file with seven plaintexts each with a copy that has been enciphered using the substitution cipher and a random key[1] (Some of these texts may not be in English). Using your code from problem 1, decipher each of these texts using frequency analysis, and report the key accuracy and decipherment accuracy that frequency analysis obtains for each of the seven texts. Repeat this procedure for all seven ciphertexts using the solver from the textbook ("simpleSubHacker.py") and a substitution solver of your choice that you have found on the internet (**You must provide the URL for this solver**). Inside the provided zip file we provide you with partial decipherments. Treat these as if they are outputs from a solver and evaluate their decipherment and key accuracies w.r.t. the first 100 characters of each plaintext. For each of the other solvers, evaluate the decipherment and key accuracies on **the full texts**. You should present your results in a table like the one below:

Name	Frequency		Textbook		Provided		Found	
	KA	DA	KA	DA	KA	DA	KA	DA
deer								
forest								
pangram								
tree								
woodm								
1984								
finnegan								

"KA" and "DA" correspond to "key accuracy" and "decipherment accuracy". The methods refer, respectively, to frequency analysis (problem 1), the pattern-based solver given in the textbook (NOT the improved version you created in Assignment 4), the provided decipherment files, and the solver you find online.

1. Does the choice of key used to encrypt the text files affect the results of your evaluation? Explain.
2. Based on the data you have recorded for the provided texts, what conclusions can you make about the method of decipherment used by your program?
3. For which texts do the solvers have the highest accuracy scores and what are the properties of these texts?
4. Can you think of any changes that could you make to your program to improve its performance?

Include your answers, your accuracy measurements, and the URL for the solver that you found on the internet in your "a5.pdf" or "a5.txt" file.

[1]Credit: All plaintexts adapted from Wikipedia articles and classical literature.