CMPUT 396, Fall 2019, Assignment 7

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 three files for this assignment: "vigenereIMCHacker.py" for problem 1 and problem 2, "a7.pdf" or "a7.txt" for problem 3, and a README file (also in either PDF or plain text). **If your code requires any external modules or other files to run, include them in your submission as well. Your submission should be self-contained. Modules from the textbook must not be edited – if you wish to modify code from the textbook, put it in a module with a different name.**

Problem 1

In Assignment 6, you used a mathematical concept known as the *Index of Coincidence* (IC) to deduce the length of the key used to create a Vigenere ciphertext. The next step is, given the key length, to deduce the key itself. Since there are 26^n possible keys of length n, a brute force approach is generally not feasible. Instead, we want to find each of the n letters independently, so that the running time of the decipherment process will be linear in the key length, rather than exponential. So, the plan is to come up with a function that takes one of the n interleaved subsequences that make up the ciphertext – the letters enciphered by the same symbol of the key – and identifies the letter that was used to encipher them. We can then apply this function to each key letter in turn to figure out the key one letter at a time.

Chapter 20 of the textbook gives a simple method of doing this, but in practice, this method is not very effective, nor is it mathematically principled. In this exercise, you will use a mathematical concept known as the *Index of Mutual Coincidence* (IMC). Conceptually, IMC is similar to IC: while IC, given a single string, measures the probability of two randomly selected characters from the string being identical, IMC takes two strings, and measures the probability of two randomly selected characters, one from each string, being identical.

Let $t(i)$ be a function that takes a letter and gives its relative frequency in a text, (that is, the number of occurrences of i divided by the length of the text), and let $e(i)$ be a similar function that gives the relative frequency of the letter i in most English texts, that is, the probability of randomly choosing a particular character from a typical English string.

For example, $e(A) = 0.0817$, because roughly 8.17% of English letters are 'A'. The IMC is computed as follows:

$$\mathbf{IMC}(t) = \sum_{i=A}^{i=Z} t(i) \cdot e(i)$$

The key insight is that $\mathbf{IMC}(t)$ will be higher if the character frequency distribution of a text is closer to the frequency distribution of English.

It is possible to figure out what letter was used to encipher a subsequence of the ciphertext by computing the IMC for the subsequence, deciphered with each possible letter. There are, of course, only 26 possibilities, which is a small enough number for us to try them all. The letter with the highest IMC is likely the letter that was used to encipher the subsequence, however, this may not always be true.

Your task is to create a module named "vigenereIMCHacker.py", containing a function

named *vigenereKeySolver*, that takes two arguments: *ciphertext*, which is a string containing a ciphertext created using the Vigenere cipher, and *keylength*, which is an integer giving the length of the key used to do so. Your *vigenereKeySolver* function should return a list of the ten most likely keys (i.e. the ten keys with the highest total IMC, as described above), represented as ASCII strings.

Your code should not be case-sensitive, and it should work on the full set of ASCII characters. Non-alphabetical characters, such as whitespace or punctuation, should be skipped, but included in your program's output (this is the same behaviour as previous assignments). You may assume that the key will only contain alphabetical characters and that it will be of a length less than or equal to 10. Your function will only be evaluated based on the first key in the list returned by your function, but its ability to give multiple guesses will become useful for the next problem.

```
ciphertext = "QPWKALVRXCQZIKGRBPFAEOMFLJMSDZVDHXCXJYEBIMTRQWNMEAIZRVKCV\
KVLXNEICFZPZCZZHKMLVZVZIZRRQWDKECHOSNYXXLSPMYKVQXJTDCIOMEEXDQVSRXLRLKZHOV"
a = vigenereKeySolver(ciphertext, 5)
assert a[0] == "EVERY"
```

Problem 2

Now you will combine the method you implemented in the previous assignment for determining the length of the key used to create a Vigenere ciphertext, with the method you implemented in the previous problem for finding the key, given the key length. This will provide a complete Vigenere cipher cracking program.

Add the function "hackVigenere" to your "vigenereIMCHacker.py" module which uses your function from Problem 1 to find the key used to generate a given Vigenere ciphertext. (Hint: Your function from Assignment 6 Problem 4 ("keylengthIC") may be useful for determining the possible lengths of cipertext.) The "hackVigenere" function should take one argument, a string containing a Vigenere ciphertext, and should return a string containing the key.

It may be necessary to try multiple keys and key lengths; we suggest using the textbook's "detectEnglish" library to select the best decipherment from multiple likely keylengths and keys. The overall ability of your code to break Vigenere ciphers will be considered when we evaluate your code. Your program may be tested with keys that are not words. For example, "zxcvbnm" is a valid key of length 7.

Your code should not be case-sensitive, and it should work on the full set of ASCII characters. Non-alphabetical characters, such as whitespace or punctuation, should be skipped, but included in your program's output (this is the same behaviour as previous assignments). You may assume that the key will only contain alphabetical characters and that it will be of a length less than or equal to 10. To reiterate: You may find that the first key your program tries is not the solution, as indicated by detectEnglish returning false. If this is the case, then your program should continue trying more keys different key lengths until a correct key is obtained.

Problem 3

Provided with this assignment is a file, "password_protected.txt", that has been encrypted using the vigenere cipher. Add a function to your module, "crackPassword()" that hacks this file and prints out decrypted plaintext using the function from the previous problem. Your program must complete its hacking in under one minute when run on a lab machine.