CMPUT 331, Fall 2023, Assignment 8 (Version 1.1)

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: "a8.py" for problem 1 and problem 2, "plaintext.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. Attribute any code you use, even if it is from the textbook.**

Problem 1

In Assignment 7, you implemented a function which took as input a string, and computed its *Index of Coincidence* (IC), the probability that two randomly selected characters from this string would be identical. You then used this function to automatically deduce the length of the key used to create a Vigenere ciphertext.

In this assignment, you will implement the remaining steps in the efficient automated mathematical cryptanalysis of Vigenere ciphertexts. In particular, you will write code which can deduce the key used to create a ciphertext, given the length of the key. You will begin by implementing a combinatorial function known as the *Index of Mutual Coincidence* (IMC).

IMC considers *two* frequency distributions over the same set of characters, and measures the probability that two randomly sample characters, one from each distribution, will be identical. For the purposes of this problem, one of these distributions will be the standard frequency distribution of letters in a typical English text, as given in the textbook:

```
"E": 0.1270, "T": 0.0906, "A": 0.0817, "O": 0.0751, "I": 0.0697, "N": 0.0675,
"S": 0.0633, "H": 0.0609, "R": 0.0599, "D": 0.0425, "L": 0.0403, "C": 0.0278,
"U": 0.0276, "M": 0.0241, "W": 0.0236, "F": 0.0223, "G": 0.0202, "Y": 0.0197,
"P": 0.0193, "B": 0.0129, "V": 0.0098, "K": 0.0077, "J": 0.0015, "X": 0.0015,
"Q": 0.0010, "Z": 0.0007
```

Call this distribution e. So, $e(A)$ is the probability of a randomly sampled character from this distribution being 'A', and $e(A) = 0.0817$.

The second distribution, call it t, will be the frequency distribution of the letters in a given string, call it s. The IMC of t and e is defined as follows:

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

For example, if the input string $s =$"THAT", then $t(T) = 2/4 = 0.5$, and $t(H) = t(A) = 1/4 = 0.25$. Then:

$$\mathbf{IMC}(s) = t(T)e(T) + t(H)e(H) + t(A)e(A)$$

$$= 0.5 * 0.0906 + 0.25 * 0.0609 + 0.25 * 0.0817$$

$$= 0.08095$$

The key insight is that **IMC**(s) will be higher if the character frequency distribution of a text is closer to the frequency distribution of English.

Create a module named "a8.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 for the encipherment. The function should *return a list* containing the *ten keys with the highest total IMC*, in order, with the key having the highest total IMC being first. The keys should be simple ASCII strings.

The total IMC of a key is computed as follows: Let k be the keylength; then the ciphertext consists of k interleaved subsequences, each enciphered by one of the k key letters. For each subsequence, decipher it using the corresponding letter in the key. Then, compute the IMC of the deciphered subsequence (comparing, as always, to the standard English frequency distribution). The sum of the k IMC values is the total IMC of the key.

Your code should not be case-sensitive, and it should be able to handle input strings containing any ASCII characters. Non-alphabetical characters, such as whitespace or punctuation, should be skipped for the purposes of encipherment and decipherment, as in 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. Here is a test case:

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

ciphertext = "Vyc fnweb zghkp wmm ciogq dost kft 13 eobp bdzg uf uwxb jv\
dxgoncw rtag ymbx vg ucrbrgu rwth gemjzv yrq tgcwxf"
best_keys = vigenereKeySolver(ciphertext, 6)
assert best_keys[0] == "CRYPTO"
```

Problem 2

Add the function "hackVigenere" to your "a8.py" module. This function should take as input a Vigenere ciphertext, and should *return a string* containing the key. It should try multiple likely key lengths (your code from Assignment 7 should be useful for this), and multiple possible keys for each length (your vigenereKeySolver function from Problem 1 should be useful). To compare different keys, you should adapt the n-gram score you implemented in Assignment 6, Problem 2 and compute the score of the decipherment obtained using that key. You will need to modify the code to take a string as input, rather than a mapping and a ciphertext. You will again be provided with a text file to use as a source of English n-gram statistics; you can hard-code the name of this file (you don't need to take it as an argument).

Note that some design decisions, such as the value(s) of n to use in your key scoring function, and how many keys are up to you. Your code will be evaluated on its overall ability to crack Vigenere ciphertexts, given no additional information (you may, however, assume all plaintexts are English). To reiterate: You may find that the first key or key length your program tries is not correct, an outcome which may be indicated by a low n-gram score. If this is the case, then your program can continue trying more keys and different key lengths to try and find a better scoring key.

Your program will 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 be able to handle input strings containing any ASCII characters. Non-alphabetical characters, such as whitespace or punctuation, should be skipped for the purposes of encipherment and decipherment, as in 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.

```
ciphertext = "XUOD QK H WRTEMFJI JOEP EBPGOATW JSZSZV OVVQY JWMY JHTNBAVR GU
OMLLGG KYODPWU YSWMSH OK ZSSF AVZS BZPW"
key = hackVigenere(ciphertext)
assert key == "ECGLISH"

ciphertext = "A'q nrxx xst nskc epu qr uet zwg'l aqiobfk, uf M gwif ks yarf
jsfwspv xh lemv qx ls yfvd. Vmpfwtmvu sivsqg vbmarek e owva csgy xkdi tys.
K teg linc mm'k lkd fr llg ner zi ugitcw Jv ghmpfe'x ldigg fxuewji hx xjv
rhawg fymkmfv lbk akehho."
key = hackVigenere(ciphertext)
assert key == "SECRET"

ciphertext = "JDMJBQQHSEZNYAGVHDUJKCBQXPIOMUYPLEHQFWGVLRXWXZTKHWRUHKBUXPIG
DCKFHBZKFZYWEQAVKCQXPVMMIKPMXRXEWFGCJDIIXQJKJKAGIPIOMRXWXZTKJUTZGEYOKFBLWP
SSXLEJWVGQUOSUHLEPFFMFUNVVTBYJKZMUXARNBJBUSLZCJXETDFEIIJTGTPLVFMJDIIPFUJWT
AMEHWKTPJOEXTGDSMCEUUOXZEJXWZVXLEQKYMGCAXFPYJYLKACIPEILKOLIKWMWXSLZFJWRVPR
UHIMBQYKRUNPYJKTAPYOXDTQ"
key = hackVigenere(ciphertext)
assert key == "QWERTY"
```

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 a8.py module, "crackPassword()" that hacks this file using the function from the previous problem and outputs decrypted plaintext in **"plaintext.txt"** file. Your program must complete its hacking in under one minute when run on a lab machine.