CMPUT 299, Winter 2018, Assignment 8

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: "vigenereIMC.py" for problem 1, "a8.pdf" or "a8.txt" for problems 2 and 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.**

Problem 1 In Assignment 7, 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, it 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 f_i be the relative frequency of the letter i in a text, (that is, the count of i divided by the length of the text), and let e_i be its relative frequency in English. For example, $e_A = 0.0817$, because roughly 8.17% of English letters are 'A'. IMC is computed as follows:

$$\mathbf{IMC}(t, E) = \sum_{i=A}^{i=Z} f_i \cdot e_i$$

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

It is possible to figure out what letter was used to encipher a subsequence of the ciphertext by computing IMC for the subsequence, deciphered with each letter. There are, of course, only 26 possibilities, which is a small enough number that we can try them all. Let t be one of the subsequences of the ciphertext, and let t_i be the same subsequence deciphered using the letter i. First assume the letter is 'A', and compute $\mathbf{IMC}(t_A = t, E)$. Then assume the letter is 'B', apply a shift of -1 to t to produce t_B (remember we are decrypting, so we need to shift the cipher letters backwards), and compute $\mathbf{IMC}(t_B, E)$. Then try 'C', 'D', and so on, for all possible shifts of t. Whichever letter produces the highest $\mathbf{IMC}(t_i, E)$ is most likely the letter used to encipher t. Repeating this process for each letter in the key gives us the key.

Your task is to create a module "vigenereIMC.py", containing a function *vigenereSolver*, 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 *vigenereSolver* function should guess the key using the method described above, and return the key as a string. For example:

```
>>> ciphertext  = "QPWKALVRXCQZIKGRBPFAEOMFLJMSDZVDHXCXJYEBIMTRQWNMEAIZRVKCVKVL
XNEICFZPZCZZHKMLVZVZIZRRQWDKECHOSNYXXLSPMYKVQXJTDCIOMEEXDQVSRXLRLKZHOV"
>>> vigenereIMC.vigenereSolver(ciphertext, 5)
'EVERY'
```

Problem 2 Short answer: Up to this point, the cipher and plaintext alphabets have always been the English alphabet, and so encipherment and decipherment with the Vigenere cipher have used addition and subtraction modulo 26 (e.g. 'Y' + 'D' = 'B'). However, there is no need for this to be the case: we could apply the Vigenere cipher to any set of symbols, given an ordering (like the alphabet), and encrypt and decrypt modulo the size of that set. In fact, real-life spy communications are often first converted from letters into numbers, and only then encrypted using a strong cipher, such as the one-time pad.

For example, suppose our alphabet is the digits 0-9, with the typical integer ordering of '0123456789'. Then we can encrypt and decrypt sequences of digits with addition and subtraction modulo 10, given a key which is also a sequence of digits. For example, the sequence '748596', enciphered with the key '235', gives the ciphertext '973721' (take a moment to verify this for yourself). Similarly, encrypting "238" with key "479" gives the ciphertext "607".

Below is a table containing three plaintexts and corresponding ciphertexts, with encipherment being performed with the Vigenere cipher using the "digital alphabet" introduced above. Deduce the keys used to perform the encipherment, and report them in your "a8.pdf" or "a8.txt", in order.

#	PLAINTEXT	CIPHERTEXT
1	59194600180407032	12942396536377982
2	34361210582254144	98414815415165653
3	40424600740021112	85631116708328304

Problem 3 Short answer: The goal of this problem is to illustrate how using pseudorandom number generators (instead of truly random number sequences) can compromise the security of a cipher. In assignment 3, you investigated linear congruential generators, or LCGs. To recap, an LCG is a pseudorandom number generator which takes as input parameters a, b, m, and R_0, and computes $R_{i+1} = (aR_i + b) \pmod{m}$ for $i \geq 0$ The three keys from Problem 2 were generated using an LCG, with $m = 2^{56}$. R_0 was the key to cipher 1, R_1 was the key to cipher 2, and R_2 was the key to cipher 3.

Below are three more ciphertexts, numbered 4, 5, and 6; continuing the pattern, R_3 is the key to cipher 4, R_4 is the key to cipher 5, and R_5 is the key to cipher 6.

#	CIPHERTEXT	KEY	PLAINTEXT
4	92546927083553659	R_3	?
5	33534183920131324	R_4	?
6	86922160288892598	R_5	?

Find the following values, and report them in your "a8.pdf" or "a8.txt", in order:

- The LCG values a and b.
- The keys to ciphers 4-6.
- The plaintexts for ciphers 4-6.