CS 182 FALL 2022, PROBLEM SET 0

Due: September 12, 2022 11:59pm

This problem set is a review of concepts in probability and algorithms. Topics include conditional probability, Bayes rule, union bound, linearity of expectation, big O notation, and Python programming (algorithms, recursion, OOP).

1. *Probability.* (20 points) From all his years running the chocolate factory, Charlie knows that classic chocolate bars that crack before reaching store shelves have a 2 in 300 chance of having air bubbles on the surface. He also knows that surface bubbles appear on 1 in 50 classic bars that won't crack as well. Looking through his files, Charlie's most recent Food Standards Agency report states that 1 in 100 classic bars from his factory crack before reaching the shelves.

(1) (5 points) Charlie just produced a test bar using his classic chocolate bar recipe, and it comes out with surface bubbles. Assuming that Charlie decides to ship and sell this bar, what is the probability that the bar cracks before reaching store shelves?

(2) (5 points) Charlie's factory makes 10,000 classic chocolate bars: 4,470 come out with surface bubbles and the rest do not. What is the expected number of cracked bars in this batch?

(3) (10 points) There was a mistake at the chocolate factory and peanuts were accidentally added to the liquid chocolate being used to create plain chocolate bars. Luckily, not that many peanuts were added so the probability of each chocolate bar produced in the next hour (until this batch of liquid chocolate is used up) having any peanuts in it is distributed *IID*[1] with the probability of containing peanuts being 1/2 for each bar.

If we examine chocolate bars until we find one with peanuts, is the expected number of regular plain bars observed more than the expected number of peanut bars observed within the batch of total bars examined? What if our stopping condition changes to

[1]In practice, the occurrence of peanuts in chocolate bars under this setting may not be IID since there may be parts of the liquid chocolate where peanuts are more concentrated than others e.g. if peanuts are more dense than chocolate, they would tend to fall to the bottom of the chocolate vat or if less dense, would tend to float to the surface.

seeing a total of 2 bars with added peanuts, how would your previous answer change (if at all)? Explain your reasoning and show your work.

Hint: Denoting the random variable of interest by X, define a suitable event, A, and use the law of total expectation: $\mathbb{E}[X] = \mathbb{E}[X|A]\Pr[A] + \mathbb{E}\left[X|\,\overline{A}\,\right]\Pr\left[\,\overline{A}\,\right]$.

2. *Algorithms.* (20 points) Assume we are working with an array of n floating point numbers that are sampled from the uniform distribution over the interval $[0, M]$. Consider the following sorting algorithm:

(1) Create $k = \Theta(n)$ empty arrays, e.g. $n/2$
(2) For each element X in the input array of length n, insert X into the array indexed at $\text{int}(\frac{X}{M} * k)$, where the int() operator performs truncation and the arrays are indexed starting at 0.
(3) Sort each of these k arrays using insertion sort
(4) Create an output sorted array by concatenating each of the k arrays in order to arrive at the overall sorted output array of numbers

Show that the average-case runtime — i.e., the expectation, over the randomly sampled elements — of the algorithm used in this setting is $O(n)$.

Hint: Reason about each step separately; the challenge is to analyze Step (3). For that, note that if array i contains n_i elements, the running time of insertion sort is $O((n_i)^2)$. But what is n_i, given the assumption of uniformly distributed elements? Use a Bernoulli random variable X_{ij} to model the inclusion of the jth element of the input array in the ith array for each element. You may find the following equality helpful given that $n_i = \sum_{j=1}^{n} X_{ij}$:

$$\mathbb{E}\left[n_i^2\right] = \mathbb{E}\left[\sum_{j=1}^{n} X_{ij}^2\right] + \mathbb{E}\left[\sum_{j=1}^{n} \sum_{k \neq j} X_{ij} X_{ik}\right].$$

3. *Programming Warm-Up.* (20 points)

(1) *Pascal's Triangle.* (10 points)

Write an iterative algorithm to generate the 0th to nth rows of Pascal's triangle where indexing starts at 0. E.g. the triangle corresponding to $n = 5$ will have 6 rows in total, one for the 0th row, the 1st row, etc. up until the 5th row. Complete the function in the `pset0.py` file and test your function against the provided test cases to make sure that it is working. Do not use any external libraries or online solutions, please use only base python functions and data types.

More info about Pascal's Triangle can be found here.

(2) *Flatten Nested Lists.* (10 points)

Write a recursive algorithm that outputs a list of integers which are extracted from a nested set of iterable lists. Your function should flatten a nested structure of integers contained in various lists into a simple list of integers. For example if the input is [1,2,[3,4],[[5],6,7]] then return [1,2,3,4,5,6,7].

Please be sure to use recursion for this exercise. Complete the function in the `pset0.py` file and test your function against the provided test cases to make sure that it is working. Do not use any external libraries or online solutions, please use only base python functions and data types.

4. *Encryption Algorithms.* (35 points)

(1) *Caesar Cipher.* (10 points)

(a) (8 points) Complete the class declaration in `pset0.py` to create a custom, user-defined data structure that can save a cipherkey, an alphabet, and also perform encryption and decryption of a message using a Caesar Cipher. A Caesar Cipher is a simple encryption method that shifts all letters by a fixed offset specified by the cipherkey's location in the alphabet. E.g. if our alphabet is the standard lower-case English letters abc...xyz and our message is "x" and our cipherkey is "b" then the resulting ciphertext would be "y" since the cipherkey "b" corresponds to the index 1 letter of the alphabet so we offset the plaintext message "x" by +1 letter to get "y". If our cipherkey was "d", we would offset by 3 since "d" is at index 3 in the alphabet (indexing from 0) which would result in our message being encrypted to "a". We wrap around to the start of the alphabet if our offset exceeds the last letter. Likewise, for decryption, instead of offsetting forward, we offset backwards to decrypt a ciphertext into plaintext. The diagram below illustrates how a Caesar cipher as described above would decrypt ciphertext to readable plaintext for a cipherkey of "d" corresponding to a 3 letter shift.

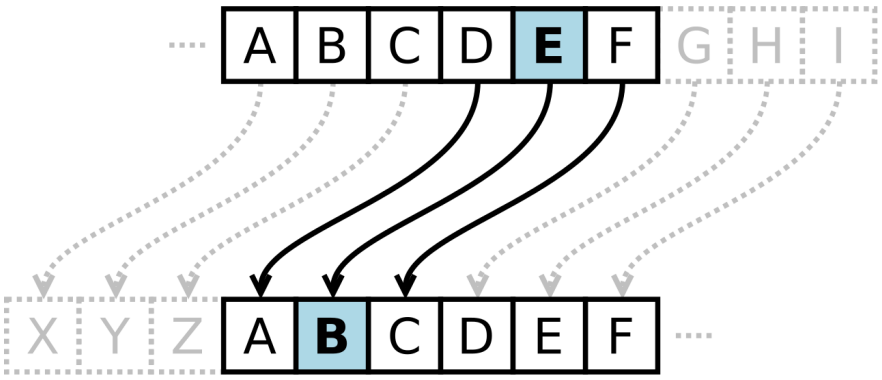


FIGURE 1. Source: https://en.wikipedia.org/wiki/Caesar_cipher

(b) (2 points) Bob thinks he's clever by encrypting his secret message not once, but twice using cipherkeys "h" and "t" respectively. What is the issue with Bob's approach? Would it make a difference if he reversed the order of the cipherkeys applied? (i.e. "t" then "h") [2]

[2] *Note:* In-depth knowledge of ciphers is not required to answer the logical reasoning questions in this section, just critical thinking. Short answers are fine, you can answer with just a sentence or two.

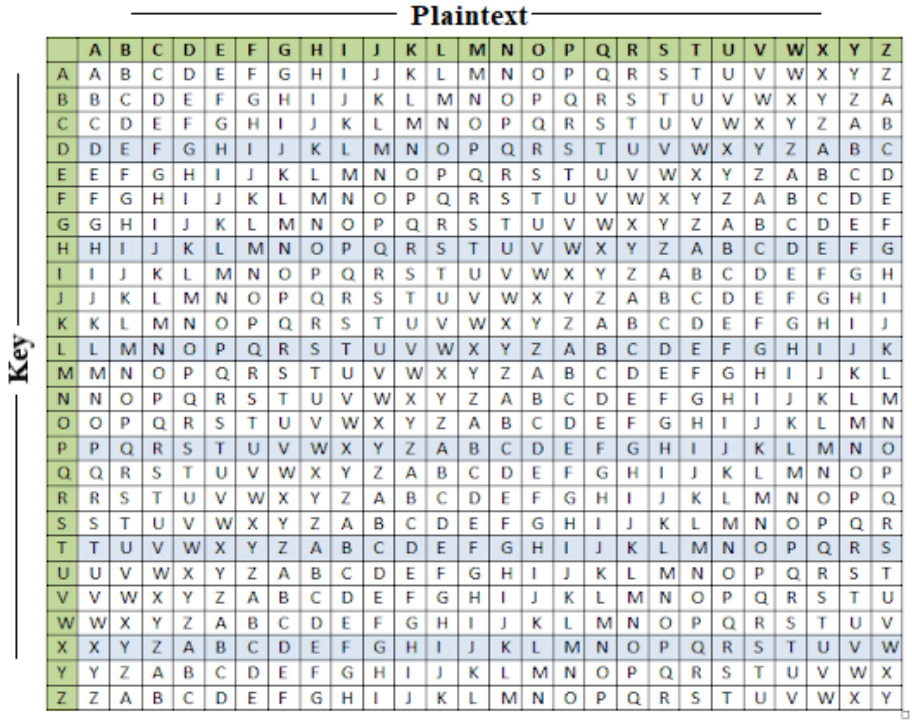
(2) *Caesar Cipher Auto-Decryption.* (10 points)

(a) (10 points) Add to the `CaesarCipher` class a new method called `auto_decrypt()` that will automatically try to guess the correct cipherkey of the message by trying each character in the saved alphabet and computing the % of words recognized in the English language contained in the resulting plaintext. Return a tuple containing the decrypted plaintext and auto-detected cipherkey. You may add early stopping and stop if you find a cipherkey that produces a plaintext output with at least 85% words recognized in the English word dictionary. If no key produces at least 85% recognizable words, then return the key with the highest % of recognized words. The `english_word_list` has been read in for you, pass in this object by reference into your method call. Add an optional argument called `verbose` with a default of False that will print out the percentage of words recognized as English words for each character tried as the cipherkey to track the progress of your auto-detection method.

(b) *Bonus problem.* (5 bonus points) Can you think of a faster way to auto-decrypt a secret message (in English) that was encrypted using a Caesar cipher? Briefly describe your approach and add an additional method to `CaesarCipher` in `pset0.py` called `auto_decrypt_bonus()` if you're up for the challenge. The top 5 fastest submissions by average runtime on a set of hidden test cases that return the correct output for all test cases will earn extra credit on this assignment. The class object is instantiated once at the start and all test cases are run on it. **Hint:** A table containing the average frequency of use for each letter in the alphabet in written English text has also been provided that you can utilize in your approach. Source: https://en.wikipedia.org/wiki/Letter_frequency.

(3) *Vigenère Cipher.* (15 points)

(a) (13 points) Implement a class-based Vigenère Cipher by completing the code for `VigenereCipher` in `pset0.py`. Your data structure should have a method for encryption and decryption. A Vigenère cipher can be thought of as a generalization of a Caesar cipher. Vigenère ciphers convert plaintext to ciphertext using a cipherkey that may be of length 1 or greater where each subsequent character in the cipherkey specifies the alphabetical offset for each subsequent plaintext letter being encrypted. We visualize the possible alphabetical offests using the following table:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

FIGURE 2. Source: https://i.ibb.co/GfDR26w/vigenere-cipher.png

The cipherkey is resued from start to end as many times as needed to encrypt or decrypt the entire message. A Caesar cipher is a special case of Vigenère Ciphers where the cipherkey is of length 1.

For example, consider the plaintext "helloworld" and a cipherkey "neat". We need to create a key with 10 characters, while neat only has 4. We therefore repeat our cipherkey to produce our final key: "neatneatne". We then encrypt using the column for our first letter, "h", using the row of our first letter of the cipher, "n", to get the first letter of our ciphertext, "u". Repeat the process to get the full ciphertext, "uilebaokyh". We can then decrypt the first letter of our ciphertext, "u", using the row of first letter of our cipherkey, "n", and locating the "u" in that row. "u" in this row is located in column "h", giving us our first plaintext letter. We repeat this process to then generate our full plaintext string "helloworld".

(b) (2 points) A Vernam cipher is a special case of a Vigenère cipher where the cipherkey must be at least as long as the plaintext input. Why might this be a stronger method of encryption then a regular Vigenère cipher? Assuming the cipherkey is a series of random characters, what would be the distribution of letters in the ciphertext would look like using this cipher?

5. Collaboration, Calibration and References (5 points).

(1) With whom did you work on this problem set? What (if any) references and/or resources did you use beyond the course lecture slides and textbook?

(2) (5 points) Approximately how long did it take you to complete this problem set? Please complete this brief survey worth 5 points, graded for completion.