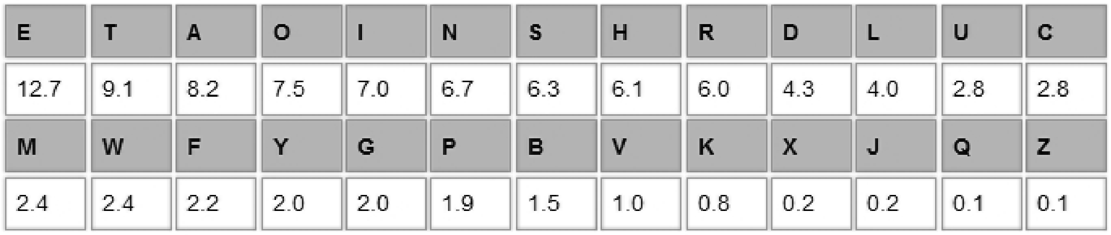
EXTRA CREDIT I

To break a Caesar cipher, we can use frequency analysis, a brute-force approach that tests all possible shifts and determines the most likely decryption by comparing letter distributions.

Frequency analysis relies on the idea that certain letters appear more frequently than others in a given language. For example, in English, the letters E, T, A, and O are among the most common, while Z and Q are the least common. The distribution of all the characters in English is depicted in the figure below:



E	T	A	O	I	N	S	H	R	D	L	U	C
12.7	9.1	8.2	7.5	7.0	6.7	6.3	6.1	6.0	4.3	4.0	2.8	2.8
M	W	F	Y	G	P	B	V	K	X	J	Q	Z
2.4	2.4	2.2	2.0	2.0	1.9	1.5	1.0	0.8	0.2	0.2	0.1	0.1

Figure 1. *Distribution letters in English (image source: https://ibmathsresources.com/tag/vigenere-cipher/)*

No matter how a message is encrypted using the Caesar cipher, the overall letter frequency remains unchanged—only shifted. By comparing the letter frequencies of an encrypted text with known English letter distributions, we can determine the correct shift.

To achieve this, we will analyze the similarity between the letter frequency distribution of the encrypted text (C) and the expected distribution of letters in English (E). The closer the two distributions, the more likely we have found the correct shift. Three different distance metrics will be used for comparison:

1. Chi-Squared distance

$$\chi^2(C, E) = \sum_{i='a'}^{i='z'} \frac{(C_i - E_i)^2}{E_i}$$

, where C_i represents the occurrence of the ith character, and E_i is the expected count of the ith character of the alphabet. For each possible character (*i* goes from ‘a’ to ‘z’), we measure the discrepancy between how often it appeared in the encrypted text (C_i) and how often it is expected to appear in English texts (E_i); the difference $C_i - E_i$ is squared such that we remove negative signs. The division by E_i is simply a normalization factor. The lower the Chi-square distance $\chi^2(C, E)$, the more similar the histograms C and E are.

2. Cosine Distance

Cosine Distance treats the frequency distributions as vectors in a multi-dimensional space, computing the angle between them:

$$D_{cos}(C, E) = 1 - \frac{\sum_{i=a}^{i='z'} C_i E_i}{\sqrt{\sum_{i=a}^{i='z'} (C_i)^2} \cdot \sqrt{\sum_{i=a}^{i='z'} (E_i)^2}}$$

This metric measures how well the two distributions align regardless of their magnitude. A cosine distance of 0 means the distributions are identical, while a value closer to 1 indicates that they are completely different.

3. Euclidian Distance

Euclidean distance, also known as the L2 distance, measures the straight-line distance between two frequency distributions, treating them as points in a multi-dimensional space (one dimension for each letter in the alphabet):

$$L_2 = \sqrt{\sum_{i='a'}^{i='z'} (C_i - E_i)^2}$$

This method treats the two distributions as points in space and measures their distance. A smaller value means the distributions are more similar, indicating the correct shift.

Breaking Ceasar cipher

As this algorithm is a brute force method to break the cipher, you should compute the histogram for all possible shifts, and compute the distance between these histograms (using one of the three methods) and the average distribution of the characters in English. The shift with the lowest distance is the solution.

The implementation consists of the following functions:

- Write a function that reads the distribution of the letters from a file (*distribution.txt*) and stores it into an array. The frequency of the letter 'a' is stored on the first line of the file (as a floating point number), the frequency of the letter 'b' is stored on the second line of the file, etc.

*void read_distribution(const char *filename, double distribution[ALPHABET_SIZE]);*

- Write a function that computes the normalized frequency of each character (a histogram) in a text.

*void compute_histogram(const char *text, double histogram[ALPHABET_SIZE]);*

- Write a function that computes the Chi-square distance between two histograms.

double chi_squared_distance(const double hist1[ALPHABET_SIZE], const double hist2[ALPHABET_SIZE]);

- Write a function that computes the Euclidian distance between two histograms.

double euclidean_distance(const double hist1[ALPHABET_SIZE], const double hist2[ALPHABET_SIZE]);

- Write a function that computes the Cosine distance between two histograms.

double cosine_distance(const double hist1[ALPHABET_SIZE], const double hist2[ALPHABET_SIZE]);

- Write a function that breaks the Caesar's cipher using frequency analysis. The function should try all possible shifts, compute the letter frequency histogram for each shifted text, compare it to the standard English letter distribution using a distance function (passed as a parameter), and return the top 3 shifts with the lowest distances.

void break_caesar_cipher(const char text, int top_shifts[TOP_N], double top_distances[TOP_N], double (*distance_function)(const double[], const double[])*

You can choose another function signature if you want. In the function above: *text* is the encrypted message to decrypt; *top_shifts* is an array storing the three most likely shifts based on the lowest distance scores; *top_distances* holds the corresponding distance values for each shift. *distance_function* is the comparison method (Chi-Squared, Cosine, or Euclidean).

- Finally, create a user-friendly menu-based application for Caesar's cipher-related tasks. The main features should be:
  - **Reading a text from the keyboard:** The user will type a text and the program will display it.
  - **Reading a text from a file:** The user can input a filename, and the program will read and display the text from the file.
  - **Encrypting a text with a specified shift:** The user provides a shift value, and the program applies the Caesar cipher encryption, shifting each letter accordingly while preserving case and ignoring non-alphabet characters.
  - **Decrypting a text with a known shift:** The user enters an encrypted text along with the original shift value, and the program reverses the shift to recover the original message.
  - **Computing and displaying the frequency distribution of a text:** The program analyzes the text and computes the letter frequency histogram, displaying the percentage of each letter's occurrence in the text.
  - **Breaking the encrypted text using frequency analysis:** The user inputs an encrypted message and selects a **distance metric** (Chi-squared, Euclidian distance, cosine distance) to compare its frequency distribution with standard English letter frequencies. The program systematically tests all possible shifts, computes the distance for each, and determines the most likely shift based on the lowest computed distance.

Analysis

Once you have successfully implemented the system, it is time to put it to the test. Your goal is to analyze how well each decryption method - Chi-Squared Distance, Cosine Distance, and Euclidean Distance - performs under different conditions. Generate a variety of encrypted texts using the Caesar cipher with different shifts. Start with short texts (10-20 characters) and compare them to longer texts(200+ characters). By applying random shifts (e.g., shift = 5, shift = 13), you will observe whether text length plays a role in the accuracy of decryption. Encrypt these texts and then attempt to automatically decrypt them using all three distance metrics.

Analyze the results: Which method consistently finds the correct shift? Are there instances where one metric outperforms the others? Do different methods excel depending on text length, or do they all perform similarly? Consider whether the structure of the text - such as letter distribution- affects decryption accuracy. If a text happens to contain unusual letter frequencies (for example, very few occurrences of 'E' or 'T', which are common in English), does that influence the success rate?

Other important things to investigate are errors and anomalies. Are there cases where the program selects the wrong shift? If so, why does this happen? Look for instances where certain letters appear disproportionately in the text - do these cases confuse the frequency analysis?

Once you have performed these tests, summarize your observations. Describe how the three decryption methods compare, explaining which one you believe is most effective in the majority of cases. Are there situations where one method works better than the others? Discuss any edge cases where frequency analysis seems to fail—such as very short texts, highly repetitive text patterns, or texts with limited letter variety.

Finally, document your results in a short and clear manner. Your final submission should not only include the working code but also a very short written report explaining your testing process, results, and conclusions. If applicable, provide examples of texts that were decrypted successfully and those that caused errors. Use graphs, tables, or visualizations if necessary to support your findings.

Useful resources:

https://ibmathsresources.com/2014/06/15/using-chi-squared-to-crack-codes/ .