Analysis 2, 2020/2021
Assignment

Caesar solver

Formative assignment

Introduction

The Roman general Julius Caesar encrypted secret messages using a rotational shift through the alphabet. For example, if the shift is 3, then you apply the following transformation[1]: A→D, B→E, C→F, D→G, E→H, ..., V→Y, W→Z, X→A, Y→B, Z→C.

The sentence **"The die has been cast!"** becomes **"Wkh glh kdv ehhq fdvw!"** using this encryption method. And if we use a shift of 10, **"I came, I saw, I conquered."** is encrypted as **"S mkwo, S ckg, S myxaeobon."**.

In this formative assignment you will implement the encryption and decryption of the Caesar cipher with a given shift. You will also apply a cryptanalysis technique called "quadgram statistics" to implement an algorithm that finds the original plaintext given a Caesar-encrypted ciphertext where the shift is unknown.

Quadgrams

When cracking a cipher, it is useful to have a method that tests how closely a string of letters looks like English (or any other language, but we shall use English in this assignment). A common technique is to look at letter frequencies (E is by far the most common letter in the English language, for instance), but this has its limitations as it does not take letter *combinations* into account. Quadgram statistics provides a test that does look at letter combinations rather than individual letter frequencies.

A *quadgram* is a string of 4 letters, for instance "rhcp". Not all quadgrams are equally common in English. The most common quadgram is "tion", and the quadgram "qzwx" does not occur in English at all.

[1] In Caesar's time, only a subset of the current 26 letters of the alphabet was actually used, but in this assignment we shall not be bothered by such historical minutiae.

Given a text, we can consider all the quadgrams that occur in it. When doing so, you should ignore all spaces and punctuation, etc, and only look at letters. It is also convenient to convert everything to lowercase.

For example, take the plaintext **"The die has been cast!"**. This has the quadgrams "thed", "hedi", "edie", "dieh", "ieha", "ehas", "hasb", "asbe", "sbee", "been", "eenc", "enca", "ncas", "cast".
Its encryption **"Wkh glh kdv ehhq fdvw!"** consists of the quadgrams "wkhg", "khgl", "hglh", "glhk", "lhkd", "hkdv", "kdve", "dveh", "vehh", "ehhq", "hhqf", "hqfd", "qfdv", "fdvw".

Quadgram fitness

One can clearly see that the quadgrams in the plaintext look more English than the ones in the ciphertext, but how to measure this?

We have uploaded a file `english_quadgrams.py` that defines a large dictionary **`quadgram_score`** of quadgrams and associated values. The lower its value, the more common a quadgram is in English. Download this file, and save it to your Python working directory. When you launch a Python interpreter from this directory, you should then be able to import the module **`english_quadgrams`**. Try this!

According to the dictionary **`quadgram_score`**, the most common quadgram "tion" has value 5.770749985, the slightly less common quadgram "afte" has value 7.650136414, and the extremely uncommon quadgram "aepz" has value 22.16407866. The quadgram "qzwx" is not present in the dictionary at all; let's give those quadgrams a value of 23.

The *quadgram fitness* of a text is the sum of the values of the quadgrams in the text. Let us do this for the plaintext given above as well as its associated ciphertext:

The die has been cast!			**Wkh glh kdv ehhq fdvw!**	
Quadgram	Value		Quadgram	Value
`thed`	7.459737785		`wkhg`	21.06546638
`hedi`	8.489536227		`khgl`	19.52502134
`edie`	10.28174369		`hglh`	19.96685409
`dieh`	13.04541508		`glhk`	23 (not present)

`ieha`	11.96182074		`lhkd`	18.69834276
`ehas`	8.652643824		`hkdv`	21.06546638
`hasb`	8.738062922		`kdve`	18.90598213
`asbe`	8.619790388		`dveh`	12.51007864
`sbee`	8.749150157		`vehh`	18.25205566
`been`	7.75779258		`ehhq`	21.47093148
`eenc`	9.874032586		`hhqf`	21.47093148
`enca`	10.24719589		`hqfd`	21.47093148
`ncas`	10.72830426		`qfdv`	23 (not present)
`cast`	9.309401287		`fdvw`	20.55464075
Total	**133.9146274**		**Total**	**280.9567026**

We see that the quadgram fitness of the English plaintext is 133.9146274 and the quadgram fitness of the random-looking ciphertext is 280.9567026. So by quadgram statistics, the plaintext looks more English than the ciphertext indeed: lower means better.

Finding the shift

Let us now try to crack the message **"Lqdqlm ivl Kwvycmz!"**. We are given that it is a Caesar-encrypted text, but we don't know the shift that was used. Let us try them all and compute the quadgram fitness of the associated decryptions:

Shift	Decryption	Quadgram fitness
0	`Lqdqlm ivl Kwvycmz!`	271.219286
1	`Kpcpkl huk Jvuxbly!`	268.3975856
2	`Jobojk gtj Iutwakx!`	228.6981259
3	`Inanij fsi Htsvzjw!`	224.4766076
4	`Hmzmhi erh Gsruyiv!`	224.9589184
5	`Glylgh dqg Frqtxhu!`	268.6464795
6	`Fkxkfg cpf Eqpswgt!`	262.9757986

7	`Ejwjef boe Dporvfs!`	213.1799671
8	`Divide and Conquer!`	124.0962487
9	`Chuhcd zmc Bnmptdq!`	253.1059654
10	`Bgtgbc ylb Amloscp!`	222.2541446
11	`Afsfab xka Zlknrbo!`	242.3831518
12	`Zereza wjz Ykjmqan!`	237.7264577
13	`Ydqdyz viy Xjilpzm!`	276.2070949
14	`Xcpcxy uhx Wihkoyl!`	269.2731034
15	`Wbobwx tgw Vhgjnxk!`	270.0750497
16	`Vanavw sfv Ugfimwj!`	237.0004422
17	`Uzmzuv reu Tfehlvi!`	228.5108208
18	`Tylytu qdt Sedgkuh!`	229.0992738
19	`Sxkxst pcs Rdcfjtg!`	239.5252779
20	`Rwjwrs obr Qcbeisf!`	228.1831231
21	`Qvivqr naq Pbadhre!`	245.6774313
22	`Puhupq mzp Oazcgqd!`	271.6885074
23	`Otgtop lyo Nzybfpc!`	205.2989991
24	`Nsfsno kxn Myxaeob!`	239.9828558
25	`Mrermn jwm Lxwzdna!`	249.5153756

Even if we didn't know any English, we could still decipher the message by picking the shift that leads to the lowest and thus best quadgram fitness. The shift giving the best fit is **8** and the original message is **"Divide and Conquer!"**.

Requirements

You must submit a Python file (.py, no zip files this time) that satisfies the following requirements:

1. The file name must be of the form **`caesar_`*`studentnumber`*`.py`** (assuming you work alone). So if your student number is 7083170, the file name must be **`caesar_7083170.py`**.

   When working in a group, all student numbers must be added to the file name and separated by underscores, like so: **`caesar_`*`studentnumber1_studentnumber2`*`.py`**. So if you're working in a pair and your student numbers are 5738316 and 3617707, the file name should be **`caesar_5738316_3617707.py`**.

2. A function called **`group_info()`** must be implemented. It has no parameters, and must return a list of tuples, where each tuple consists of a student's number, their name, and their class code.

   Example: Invoking **`group_info()`** should return something that looks like **`[("7083170", "Suzette Edemann", "INF1X")]`** in case you work alone, or something like **`[("5738316", "Matilde Ree", "INF1Y"), ("3617707", "Elmer Feit", "INF1Y")]`** when working in a group.

3. A function called **`encrypt_caesar(plaintext, shift)`** that encrypts a plaintext using a Caesar cipher with a given shift must be implemented. The function must return the encrypted text as a string.

   Example: **`encrypt_caesar("The die has been cast!", 3)`** must return **`"Wkh glh kdv ehhq fdvw!"`**.

4. A function called **`decrypt_caesar(ciphertext, shift)`** that decrypts a ciphertext using a Caesar cipher with a given shift must be implemented. The function must return the decrypted text as a string.

   Example: **`decrypt_caesar("S mkwo, S ckg, S myxaeobon.", 10)`** must return **`"I came, I saw, I conquered."`**.

5. A function called **`quadgram_fitness(text)`** that returns the quadgram fitness of a string must be implemented. Definition and examples of quadgram fitness are given above. The function must make use of the dictionary **`quadgram_score`** in the provided file [english_quadgrams.py](), which must be imported (**not copied!**).

   Example: **`quadgram_fitness("Wkh glh kdv ehhq fdvw!")`** must return a number that coincides with **280.9567026** to at least 5 decimal places.

6. A function called **`solve_caesar(ciphertext)`** that returns the plaintext for a Caesar-encrypted ciphertext must be implemented. You can use quadgram analysis to find the shift used, but the shift should not be output (just the plaintext).

   Example: **`solve_caesar("Lqdqlm ivl Kwvycmz!")`** must return **`"Divide and Conquer!"`**.

7. The functions should not have any side effects. So they should have no print or input statements, not affect global variables, etc.

8. Apart from [english_quadgrams.py](), you may only import from modules that are provided by the Python standard library.

To summarize and test the above requirements: if your student number is 7083170 and you run a standard Python interpreter from a directory where both **`caesar_7083170.py`** and [english_quadgrams.py]() are present, the following should work out (return values are given in **red**):

```
>>> import caesar_7083170
>>> caesar_7083170.group_info()  # Of course, your result will differ.
[('7083170', 'Suzette Edemann', 'INF1X')]  # Student ID is a string!
>>> caesar_7083170.encrypt_caesar("The die has been cast!", 3)
'Wkh glh kdv ehhq fdvw!'
>>> caesar_7083170.decrypt_caesar("S mkwo, S ckg, S myxaeobon.", 10)
'I came, I saw, I conquered.'
>>> caesar_7083170.quadgram_fitness("Wkh glh kdv ehhq fdvw!")
280.95670257053285  # A few digits at the end may differ
>>> caesar_7083170.solve_caesar("Lqdqlm ivl Kwvycmz!")
'Divide and Conquer!'
```

The function names, signatures, and behaviors must match exactly what is in the requirements. Of course you may - and are encouraged to - define additional helper functions that help implement the required functions. Also, we will use more test cases than the ones given in this assignment description.

Submission

Submission goes via Teams; where exactly this has to be done will be announced on Teams itself.

When working in a group, one group member (the group leader) submits the Python file, and all group members submit a group-info message with names and student numbers of the entire group, clearly indicating who is the group leader. Feedback will be given to the group leader only, who will then communicate it to the other group members.

You are not allowed to work together with someone who has a different teacher! Here we mean the teacher in your schedule. Concretely, the following cross-group pairings are allowed:

- INF1A + INF1F
- INF1B + INF1E
- INF1C + INF1G
- INF1D + INF1J
- INF1H + INF1I

Deadline

The assignment must be submitted no later than Friday 18 December 2020.

Nvvk sbjr!