Analysis 2, 2020/2021
Assignment

Vigenère solver

Summative assignment

Introduction

The Vigenère cipher, invented by the Italian cryptographer Giovan Battista Bellaso in 1553, but wrongly attributed to the French cryptographer Blaise de Vigenère (who did invent a similar cipher), is one of the earliest serious attempts to create a cipher that is more secure than an easily breakable substitution cipher. And secure it was: it went unbroken for over 3 centuries until the German cryptographer Friedrich Kasiski found a way to crack it in 1863.

We call the Vigenère cipher a "hand cipher" because encryption and decryption can easily be performed using pencil and paper. Nowadays, almost all hand ciphers can be cracked using a computer, and this is exactly what you will do with the Vigenère cipher.

Description of the cipher

The Vigenère cipher uses a *key*, which can be any string of lowercase letters, for instance **"ape"**. Let us consider the following plaintext message we want to encrypt: **"Bokito and Einstein have the same birthday!"**. Now write down the plaintext message, and below it repeatedly write down the key:

```
Bokito and Einstein have the same birthday! (plaintext)
apeape ape apeapeap eape ape apea peapeape  (key)
```

The key letters are then used to rotationally shift the letters of the plaintext above them, using the scheme a=0, b=1, c=2, ..., z=25. So for our key we use a=0, p=15, e=4, resulting in the ciphertext **"Bdoiis ach Exrsiiic laki twi spqe qmrildpc!"**:

```
Bokito and Einstein have the same birthday! (plaintext)
apeape ape apeapeap eape ape apea peapeape  (key)
Bdoiis ach Exrsiiic laki twi spqe qmrildpc! (ciphertext)
```

Let's also give an example of a decryption using the Vigenère cipher. Suppose we receive the ciphertext message **"M abxock hnsp ge oycs rts akqc at n zpyzh."** and we know it was encrypted using the Vigenère cipher with **"monkey"** as key. We write down the ciphertext and the key repeatedly below it. And we decrypt it by shifting letters backwards through the alphabet using m=12, o=14, n=13, k=10, e=4, y=24:

```
M abxock hnsp ge oycs rts akqc at n zpyzh.  (ciphertext)
m onkeym onke ym onke ymo nkey mo n keymo   (key)
A monkey tail is also the name of a plant.  (plaintext)
```

In this summative assignment you will implement the encryption and decryption of the Vigenère cipher with a given key. You will also implement a probabilistic optimization algorithm that finds the original plaintext given a Vigenère-encrypted ciphertext where the key is unknown. This optimization will make use of quadgram statistics, which you learned about and implemented in the formative assignment on the Caesar cipher.

A probabilistic search algorithm

Suppose we are given an encrypted message and we want to decrypt it, for instance the message **"V id wueirl lk tb ml vvxk vn pweorndvkkdoaaeg wgirs."** which somehow we know to be a Vigenère-encrypted message with an key of length 7, but we don't know which particular length-7 key was used.

We could try all possible letter strings of length 7 to decrypt the message, but then we'd have to try out 26^7 = 8031810176 possible keys, which is quite a large number. For larger key lengths it quickly becomes unfeasible to perform such an exhaustive search. But there are many ways around this, and a randomized search is what we shall focus on in this assignment.

Start with a random key, for example **"xlgyntq"**, leading to the decryption **"Y xx yhlsua fm gi wo kpzx cx slyqeunyzefbhkhv qivyc."**, which has a quadgram fitness of 755.8838865.

The next step is to *mutate* the key: randomly select one letter of the key and change it randomly into another letter. This way we may get the key **"xlgyntn"**, giving the decryption **"Y xx yhlvua fm gi zo kpzx ca slyqeuqyzefbhnhv qivyf."** with a quadgram fitness of 774.9566723. We want to minimize the quadgram fitness, and this new key is worse than the starting key, so we reject it.

We now continue trying out random mutations of the starting key until we find one that gives a lower quadgram fitness. We then replace the starting key with the new key, keep trying out random mutations until we get a better key, which is the one we will continue with, etc, etc. To clarify, the first few steps could look like this:

Step	Key	Decryption	Fitness	Keep?
0	xlgyntq	Y xx yhlsua fm gi wo kpzx cx slyqeunyzefbhkhv qivyc.	**755.884**	**Start**
1	xlgynt**n**	Y xx yhlvua fm gi zo kpzx ca slyqeuqyzefbhnhv qivyf.	774.956	No
2	xlgy**e**tq	Y xx yqlsua fm pi wo kpzg cx slyqnunyzefkhkhv qieyc.	763.432	No
3	xlgy**f**tq	Y xx yplsua fm oi wo kpzf cx slyqmunyzefjhkhv qidyc.	**742.084**	**YES!**
4	xl**r**yftq	Y xm yplsua um oi wo kezf cx slnqmunyztfjhkhv fidyc.	**731.582**	**YES!**
5	xlryft**g**	Y xm yplcua um oi go kezf ch slnqmuxyztfjhuhv fidym.	**721.159**	**YES!**
6	xl**f**yftg	Y xy yplcua gm oi go kqzf ch slzqmuxyzffjhuhv ridym.	746.568	No
7	xlryft**a**	Y xm ypliua um oi mo kezf cn slnqmudyztfjhahv fidys.	**711.750**	**YES!**
8	xlr**d**fta	Y xm tpliua uh oi mo keuf cn slnlmudyztajhahv fddys.	**675.747**	**YES!**
9	xlr**x**fta	Y xm zpliua un oi mo keaf cn slnrmudyztgjhahv fjdys.	701.160	No
10	xlrd**p**ta	Y xm tfliua uh ei mo keuv cn slnlcudyztazhahv fdtys.	679.525	No
11	xlrd**l**ta	Y xm tjliua uh ii mo keuz cn slnlgudyztadhahv fdxys.	704.735	No
12	xlrd**r**ta	Y xm tdliua uh ci mo keut cn slnlaudyztaxhahv fdrys.	**657.340**	**YES!**
13	x**z**rdrta	Y jm tdlium uh ci mo weut cn sxnlaudyltaxhahh fdrys.	**644.800**	**YES!**
14	xzrd**g**ta	Y jm tolium uh ni mo weue cn sxnlludyltaihahh fdcys.	647.444	No
15	ezrdrta	R jm tdlinm uh ci mh weut cn lxnlaudrltaxhaah fdrys.	657.636	No

We see that after 15 steps the best key so far is **"xzrdrta"**. It has a quadgram fitness of 644.800, which is a big improvement over the starting key, but it certainly does not give a readable text yet.

Let us perform more steps, but this time we only list the steps where we actually do get an improvement:

Step	Key	Decryption	Fitness	Keep?
16	xzrdr**n**a	Y jm tdrium uh co mo weut in sxnlaadyltaxnahh fdres.	**608.291**	**YES!**
24	xz**e**drna	Y jz tdrium hh co mo wrut in sxalaadylgaxnahh sdres.	**606.525**	**YES!**
30	x**i**edrna	Y az tdriud hh co mo nrut in soalaadycgaxnahy sdres.	**578.522**	**YES!**
35	xi**t**drna	Y ak tdriud sh co mo ncut in sollaadycraxnahy ddres.	**537.313**	**YES!**
57	**r**itdrna	E ak tdriad sh co mu ncut in yollaadecraxnany ddres.	**510.840**	**YES!**
122	**d**itdrna	S ak tdriod sh co mi ncut in mollaadscraxnaby ddres.	**507.817**	**YES!**
141	ditd**a**na	S ak turiod sh to mi ncuk in mollradscraonaby ddies.	**494.474**	**YES!**
178	ditd**g**na	S ak toriod sh no mi ncue in mollladscrainaby ddces.	**481.628**	**YES!**
192	di**s**dgna	S al toriod th no mi ndue in momlladscsainaby edces.	**468.366**	**YES!**
225	**n**isdgna	I al toried th no my ndue in comlladicsainary edces.	**453.404**	**YES!**
263	nis**k**gna	I al moried ta no my ndne in comeladicstinary ewces.	**444.472**	**YES!**

Looking at step numbers, we see that improvements are getting scarcer and scarcer. After step 263, the fitness doesn't improve anymore.

However, the text starts looking more and more like real English, and with some effort one is probably able to decrypt the message by hand now. But this is not what we will settle for, as we want our randomized search to find the complete solution.

Improvement: allow bad mutations to survive

The problem with the key **"niskgna"** is that it is a *local* minimum: every direct mutation of it gives a worse key. What we want is a *global* minimum: a key that is better than any other key.

We can often reach global minima if we proceed with a bad mutation every once in a while. There are many possible strategies for this, from simple ones to very sophisticated one. For our purposes, a simple strategy is good enough.

We'll use the following scheme. First, choose a *survival rate* for bad mutations; we'll use 1%, i.e. 0.01, in our example (a higher rate than that is not recommended). If we are about to reject a key because it is worse than the currently kept one, we draw a random number between 0 and 1; if it is less than the survival rate, we proceed with the bad key anyway. This way we will regularly get kicked away from local minima and hopefully reach a global minimum.

Let us do this, starting with a random key, and see how it goes. Again, we only list those steps that change the currently kept key.

Step	Key	Decryption	Fitness	Keep?
0	kdeoalp	L fz iutthi hw tq xb srjk ky ftaarcolhgpoplud ssigd.	665.673	Start
4	kde**s**alp	L fz eutthi hs tq xb srfk ky ftawrcolhgloplud soigd.	650.591	YES!
7	kde**k**alp	L fz mutthi ha tq xb srnk ky ftaercolhgtoplud swigd.	626.718	YES!
8	kd**q**kalp	L fn mutthi va tq xb sfnk ky ftoercolhutoplud gwigd.	610.300	YES!
13	kd**y**kalp	L ff mutthi na tq xb sxnk ky ftgercolhmtoplud ywigd.	602.986	YES!
32	kd**r**kalp	L fm mutthi ua tq xb senk ky ftnercolhttoplud fwigd.	601.278	YES!
34	k**c**rkalp	L gm mutthj ua tq xb tenk ky funercolittoplue fwigd.	567.904	YES!
54	**j**crkalp	M gm muttij ua tq xc tenk ky gunercomittoplve fwigd.	567.197	YES!
58	jcrkal**m**	M gm mutwij ua tq ac tenk kb gunercrmittopove fwigg.	562.827	YES!
59	**r**crkalm	E gm mutwaj ua tq au tenk kb yunercreittopone fwigg.	546.746	YES!
66	rcrka**m**m	E gm muswaj ua tp au tenk jb yunerbreittooone fwifg.	542.544	YES!
85	rcrka**b**m	E gm mudwaj ua ta au tenk ub yunermreittozone fwiqg.	553.112	SURV!
89	rcr**y**abm	E gm yudwaj um ta au tezk ub yunqrmreitfozone fiiqg.	620.626	SURV!
91	rcr**e**abm	E gm sudwaj ug ta au tetk ub yunkrmreitzozone fciqg.	598.724	YES!
95	rcrea**a**m	E gm suewaj ug tb au tetk vb yunkrnreitzoaone fcirg.	598.495	YES!
99	rcrea**p**m	E gm supwaj ug tm au tetk gb yunkryreitzolone fcicg.	569.153	YES!
105	rcreap**u**	E gm supoaj ug tm su tetk gt yunkryjeitzolgne fcicy.	629.162	SURV!

108	rcr**v**apu	E gm bupoaj up tm su teck gt yuntryjeitiolgne flicy.	577.055	YES!
118	r**i**rvapu	E am bupoad up tm su neck gt yontryjectiolgny flicy.	539.639	YES!
122	rirvap**w**	E am bupmad up tm qu neck gr yontryhectioleny flicw.	510.538	YES!
161	rirvap**a**	E am bupiad up tm mu neck gn yontrydectiolany flics.	493.101	YES!
188	rirva**n**a	E am buriad up to mu neck in yontradectionany flies.	416.958	YES!
314	**n**irvana	I am buried up to my neck in contradictionary flies.	402.203	YES!
344	ni**l**vana	I as buried ap to my nkck in cottradiczionary llies.	472.514	SURV!

You can see that in steps 85, 89, 105, and 344, chance has decided to proceed with a worse key than before.

The key **"nirvana"** in step 314 gives an English text, and also the lowest quadgram fitness. But since we allow bad keys to survive, the search will go on with other keys as well. This means that in your implementation you do have to keep track of the best-so-far key!

Also, you must decide after how many steps to stop the search. We recommend you take at least $1000*n^2$, where n is the key length; so in this example that would be $1000*7^2 = 49000$, which is much less than the 8 billion steps an exhaustive search would take.

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 **`vigenere_`*`studentnumber`*`.py`** (assuming you work alone). So if your student number is 7083170, the file name must be **`vigenere_7083170.py`**.

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

be **`vigenere_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_vigenere(plaintext, key)`** that encrypts a plaintext using the Vigenère cipher with a given key must be implemented. The function must return the encrypted text as a string.

   Example: **`encrypt_vigenere("Bokito and Einstein have the same birthday!", "ape")`** must return **`"Bdoiis ach Exrsiiic laki twi spqe qmrildpc!"`**.

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

   Example: **`decrypt_vigenere("M abxock hnsp ge oycs rts akqc at n zpyzh.", "monkey")`** must return **`"A monkey tail is also the name of a plant."`**.

5. A function called **`quadgram_fitness(text)`** that returns the quadgram fitness of a string must be implemented, if you have not already done so. Definition and examples of quadgram fitness are given in the formative assignment. 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.

   Remark: if you've already implemented this correctly in your formative

assignment, you can simply copy the code from there.

6. A function called **`solve_vigenere(ciphertext, keylen)`** must be implemented that returns a tuple consisting of the key and plaintext for a Vigenère-encrypted ciphertext and given key length.

   Example: **`solve_vigenere("V id wueirl lk tb ml vvxk vn pweorndvkkdoaaeg wgirs.", 7)`** must return **`("nirvana", "I am buried up to my neck in contradictionary flies.")`**.

7. For keys of length up to 10, the execution time of the function **`solve_vigenere`** must not take more than 20 seconds on a reasonably up-to-date computer.

8. The functions should not have any side effects. So they should have no print or input statements, not affect global variables, etc. Nor should there be such statements in global scope (defining constants is allowed there).

9. 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 **`vigenere_7083170.py`** and **`english_quadgrams.py`** are present, the following should work out (return values are given in **red**):

```
>>> import vigenere_7083170
>>> vigenere_7083170.group_info()  # Your result differs of course.
[('7083170', 'Suzette Edemann', 'INF1X')]  # Student ID is a string!
>>> vigenere_7083170.encrypt_vigenere("Bokito and Einstein have the
same birthday!", "ape")
'Bdoiis ach Exrsiiic laki twi spqe qmrildpc!'
>>> vigenere_7083170.decrypt_vigenere("M abxock hnsp ge oycs rts akqc
at n zpyzh.", "monkey")
'A monkey tail is also the name of a plant.'
>>> vigenere_7083170.quadgram_fitness("Wkh glh kdv ehhq fdvw!")
280.95670257053285  # A few digits at the end may differ
>>> vigenere_7083170.solve_vigenere("V id wueirl lk tb ml vvxk vn
pweorndvkkdoaaeg wgirs.", 7)  # Must run in 20 seconds or less
('nirvana', 'I am buried up to my neck in contradictionary flies.')
```

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.

You are only allowed to work in a group if (1) you attended at least 10 of the 14 lessons given and (2) you have scored at least 7 out of a possible 10 points for your homework. All members of the group must match these conditions.

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 Sunday 7 February 2021.

Mccg woeu!