CSC3621 Cryptography **Exercise 1** **Georgi Nikolov**

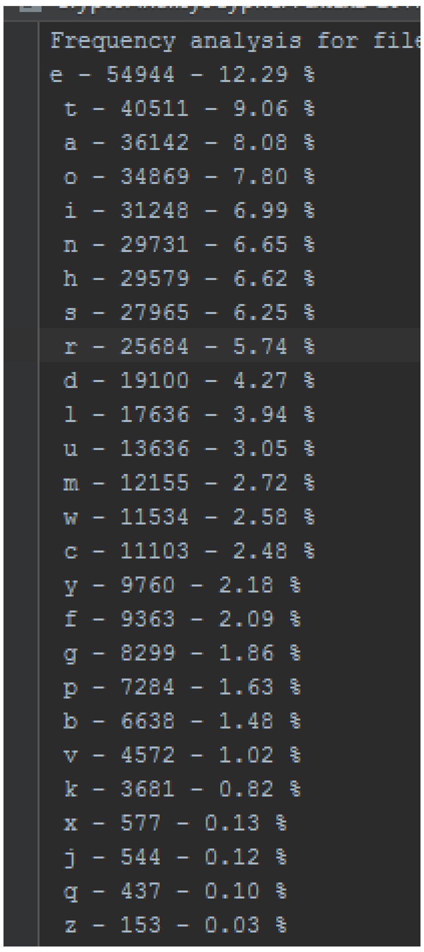
B4036790

The first part of the coursework, was quite enjoyable, just to familiarize us with what the basic approaches in cryptography are, I personally found it enjoyable.

The first step I took to solve it was to write a little Utils class which was supposed to be reused for the second part of the coursework, along the other methods in this class I wrote a private class (which later on I realised it was a mistake) to deal with the Frequency Analysis itself. The idea was to have this class hidden in the CryptoUtils class and then manipulate objects of FrequencyAnaliser type, this was achieved at some extend, but my implementation was not a particularly good, I should have put some more thought in it.

However, all required functionalities are present. I had an issue with the Frequency Analysis, my code was skipping about 30 000 valid characters (A-Zz-A) from the file, later on I realised that I have forgotten to convert the text to lower case. For the implementation of this I decided to use mainly streams, because they make the code more readable and are easy to write, but not all the time of course. The results from frequency analysis were correct with corresponding sources on the web.

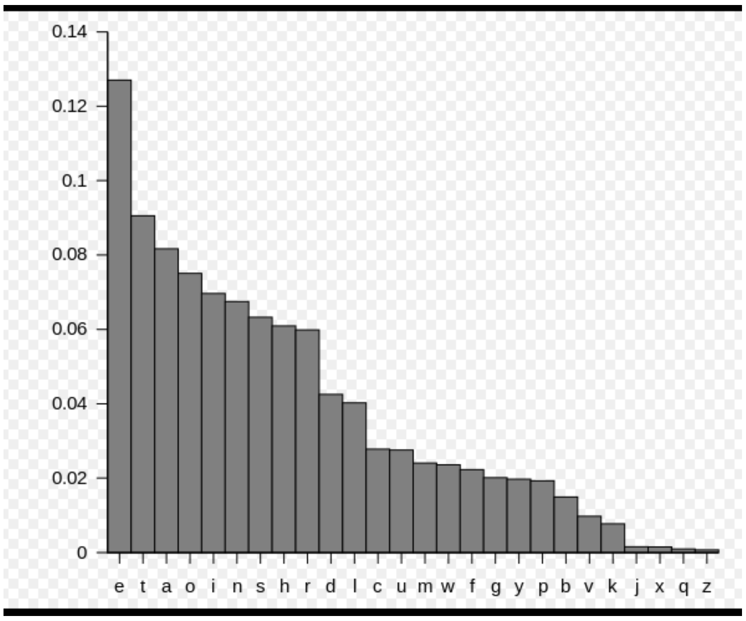
Here are my findings :



```
Frequency analysis for file
e - 54944 - 12.29 %
 t - 40511 - 9.06 %
 a - 36142 - 8.08 %
 o - 34869 - 7.80 %
 i - 31248 - 6.99 %
 n - 29731 - 6.65 %
 h - 29579 - 6.62 %
 s - 27965 - 6.25 %
 r - 25684 - 5.74 %
 d - 19100 - 4.27 %
 l - 17636 - 3.94 %
 u - 13636 - 3.05 %
 m - 12155 - 2.72 %
 w - 11534 - 2.58 %
 c - 11103 - 2.48 %
 y - 9760 - 2.18 %
 f - 9363 - 2.09 %
 g - 8299 - 1.86 %
 p - 7284 - 1.63 %
 b - 6638 - 1.48 %
 v - 4572 - 1.02 %
 k - 3681 - 0.82 %
 x - 577 - 0.13 %
 j - 544 - 0.12 %
 q - 437 - 0.10 %
 z - 153 - 0.03 %
```

(Right picture from Wikipedia)

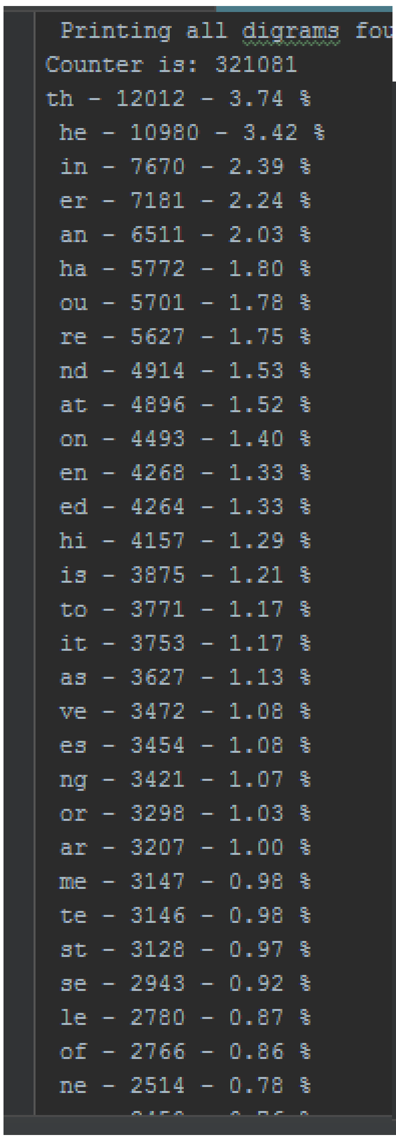
As we can observe, the letter distributions are almost identical.



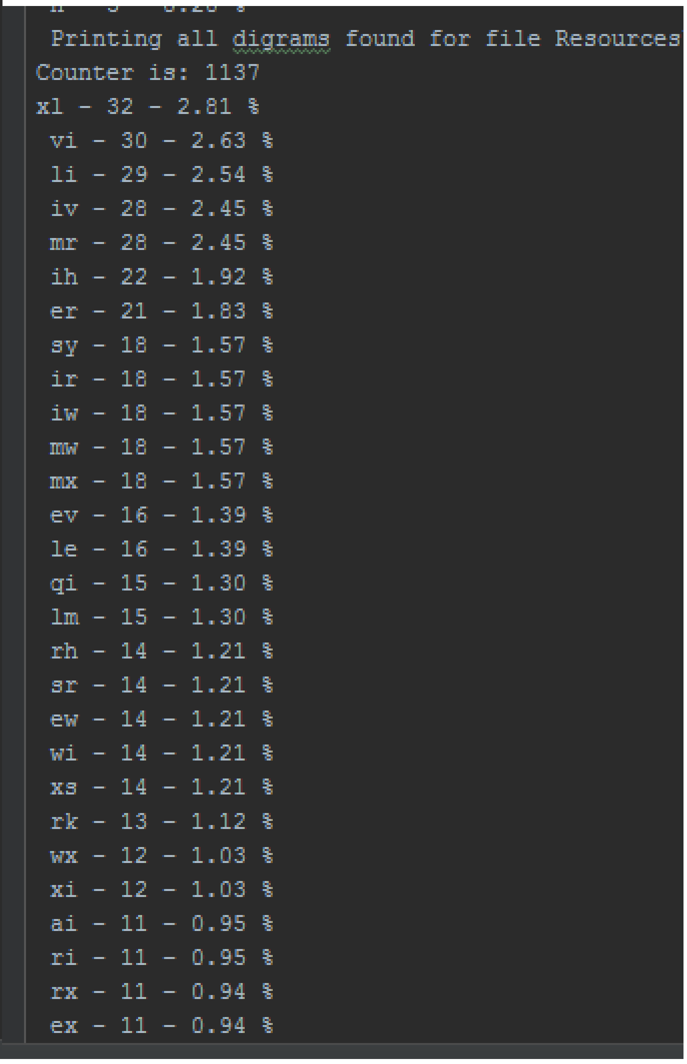
The second step, was to implement methods which would find bigrams and trigrams in the texts, once I had those, it was made clear that we probably had to deal with a shift cipher. In the cyphered text, we observe a lot of appearances of the string “xli” which in a way implied that it could be corresponding to “the” in the original text, by looking at the bigrams and trigrams for both of the texts, I was able to work out that the cipher used is not a substitution one. Afterwards I worked out the shift values and deciphered the text. Another approach would have been to perform exhaustive search, since the key space is not significant.

An issue that I have encountered was breaking the shift mechanism while I tried to optimize it for the second part of the coursework, which forced me to introduce two functions for character rotation, because it was simply easier than trying to debug it and make it work for both of them. I think it was particularly bad idea to keep using integers for the char values, from the research I have made many people suggest using byte values or hexadecimal values. Nevertheless, I managed to present a working implementation. I lesson I have learned is that a single simple mistake will always result in completely different output from the program.

Here we can see the relations of letter distribution in the encrypted and plain text for exercise 1:



```
 Printing all digrams fou
Counter is: 321081
th - 12012 - 3.74 %
 he - 10980 - 3.42 %
 in - 7670 - 2.39 %
 er - 7181 - 2.24 %
 an - 6511 - 2.03 %
 ha - 5772 - 1.80 %
 ou - 5701 - 1.78 %
 re - 5627 - 1.75 %
 nd - 4914 - 1.53 %
 at - 4896 - 1.52 %
 on - 4493 - 1.40 %
 en - 4268 - 1.33 %
 ed - 4264 - 1.33 %
 hi - 4157 - 1.29 %
 is - 3875 - 1.21 %
 to - 3771 - 1.17 %
 it - 3753 - 1.17 %
 as - 3627 - 1.13 %
 ve - 3472 - 1.08 %
 es - 3454 - 1.08 %
 ng - 3421 - 1.07 %
 or - 3298 - 1.03 %
 ar - 3207 - 1.00 %
 me - 3147 - 0.98 %
 te - 3146 - 0.98 %
 st - 3128 - 0.97 %
 se - 2943 - 0.92 %
 le - 2780 - 0.87 %
 of - 2766 - 0.86 %
 ne - 2514 - 0.78 %
```

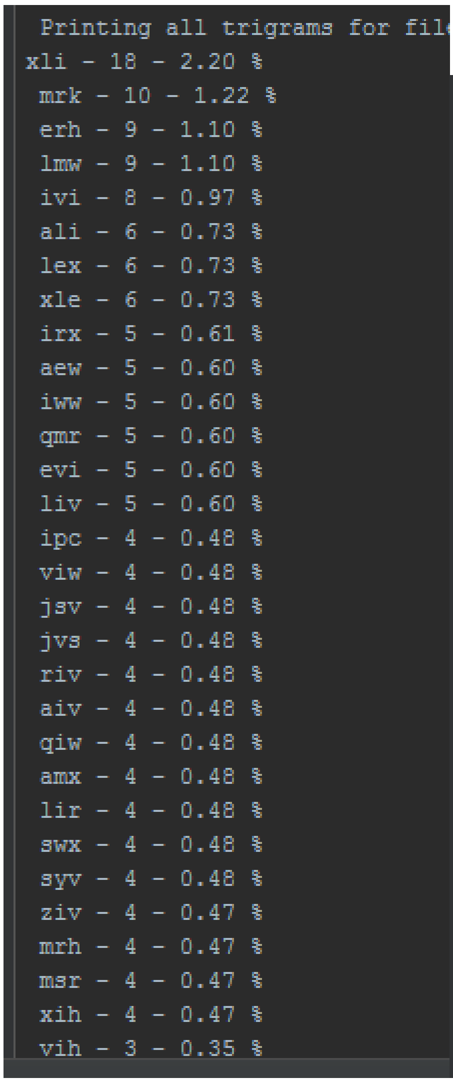


```
 Printing all digrams found for file Resources
Counter is: 1137
xl - 32 - 2.81 %
 vi - 30 - 2.63 %
 li - 29 - 2.54 %
 iv - 28 - 2.45 %
 mr - 28 - 2.45 %
 ih - 22 - 1.92 %
 er - 21 - 1.83 %
 sy - 18 - 1.57 %
 ir - 18 - 1.57 %
 iw - 18 - 1.57 %
 mw - 18 - 1.57 %
 mx - 18 - 1.57 %
 ev - 16 - 1.39 %
 le - 16 - 1.39 %
 qi - 15 - 1.30 %
 lm - 15 - 1.30 %
 rh - 14 - 1.21 %
 sr - 14 - 1.21 %
 ew - 14 - 1.21 %
 wi - 14 - 1.21 %
 xs - 14 - 1.21 %
 rk - 13 - 1.12 %
 wx - 12 - 1.03 %
 xi - 12 - 1.03 %
 ai - 11 - 0.95 %
 ri - 11 - 0.95 %
 rx - 11 - 0.94 %
 ex - 11 - 0.94 %
```

Plain text Ciphered text

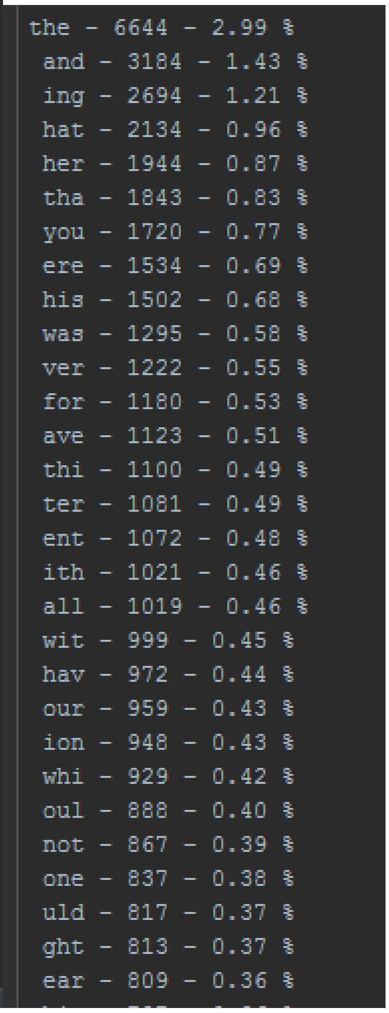
The pictures above, give us hints what could be the possible shift value for the characters. Then the trigrams, give us even further proof:

Ciphered text



```
 Printing all trigrams for fil
xli - 18 - 2.20 %
 mrk - 10 - 1.22 %
 erh - 9 - 1.10 %
 lmw - 9 - 1.10 %
 ivi - 8 - 0.97 %
 ali - 6 - 0.73 %
 lex - 6 - 0.73 %
 xle - 6 - 0.73 %
 irx - 5 - 0.61 %
 aew - 5 - 0.60 %
 iww - 5 - 0.60 %
 qmr - 5 - 0.60 %
 evi - 5 - 0.60 %
 liv - 5 - 0.60 %
 ipc - 4 - 0.48 %
 viw - 4 - 0.48 %
 jsv - 4 - 0.48 %
 jvs - 4 - 0.48 %
 riv - 4 - 0.48 %
 aiv - 4 - 0.48 %
 qiw - 4 - 0.48 %
 amx - 4 - 0.48 %
 lir - 4 - 0.48 %
 swx - 4 - 0.48 %
 syv - 4 - 0.48 %
 ziv - 4 - 0.47 %
 mrh - 4 - 0.47 %
 msr - 4 - 0.47 %
 xih - 4 - 0.47 %
 vih - 3 - 0.35 %
```

Plaintext



```
the - 6644 - 2.99 %
 and - 3184 - 1.43 %
 ing - 2694 - 1.21 %
 hat - 2134 - 0.96 %
 her - 1944 - 0.87 %
 tha - 1843 - 0.83 %
 you - 1720 - 0.77 %
 ere - 1534 - 0.69 %
 his - 1502 - 0.68 %
 was - 1295 - 0.58 %
 ver - 1222 - 0.55 %
 for - 1180 - 0.53 %
 ave - 1123 - 0.51 %
 thi - 1100 - 0.49 %
 ter - 1081 - 0.49 %
 ent - 1072 - 0.48 %
 ith - 1021 - 0.46 %
 all - 1019 - 0.46 %
 wit - 999 - 0.45 %
 hav - 972 - 0.44 %
 our - 959 - 0.43 %
 ion - 948 - 0.43 %
 whi - 929 - 0.42 %
 oul - 888 - 0.40 %
 not - 867 - 0.39 %
 one - 837 - 0.38 %
 uld - 817 - 0.37 %
 ght - 813 - 0.37 %
 ear - 809 - 0.36 %
```

Another thing that I observed was while working with the bigrams and trigrams, I had certain values, but colleagues of mine had totally different ones, so I used a couple of online tools, and there was no consistent result amongst them which was quite surprising.

To run the code just execute, the Runner class from the package containing exercise one, no user interaction is needed, a new file "analysis" followed by the current date and time will appear, where any crypto and frequency analysis results can be seen. A file with the deciphered text will appear the file name is "ceaserDecrypt" followed by the current time and date. All these are written under the resources folder, within the project. Comment are available in the code to make it easier to understand.