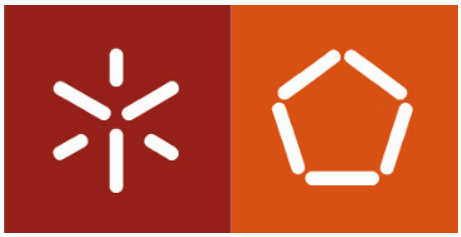
# Tecnologia Criptográfica

29 de Novembro de 2020

**Trabalho Prático 1**


| | |
|---|---|
| a83899 | André Morais |
| a84485 | Tiago Magalhães |


## Decifragem de Criptogramas com Cifras Clássicas




Mestrado Integrado em Engenharia Informática
Universidade do Minho

# Conteúdo

# 1 Introdução

No âmbito da Unidade Curricular de Tecnologia Criptográfica, foi nos proposto que decifrasse-mos três criptogramas cifrados com as cifras *affine*, de substituição, e Vigenère sem se saber a qual dos criptogramas correspondia cada cifra.

## 2 Estratégia de resolução

Através da informação fornecida pelo enunciado, sabemos que os textos cifrados correspondem a textos limpos escritos no idioma inglês, deste modo o criptograma cifrado com a cifra de Vigenère, através de análise de frequências será o que terá menos semelhanças com a distribuição mais provável das letras de um texto em inglês, já que neste tipo de cifra uma palavra pode ocorrer cifrada de diferentes maneiras, devido à periodicidade da chave. Já para as cifras de *affine* e de substituição a maneira de as atacar é através de análise de frequências e a cifra de *affine* corresponde a uma cifra de substituição através de uma função afim, logo irão ter uma distribuição parecida à da língua inglesa uma vez que cada letra irá aparecer no texto sempre na forma da mesma letra com a qual foi substituída ocorrendo desta maneira padrões.

| | |
|---|---|
| E | 12.02 |
| T | 9.10 |
| A | 8.12 |
| O | 7.68 |
| I | 7.31 |
| N | 6.95 |
| S | 6.28 |
| R | 6.02 |
| H | 5.92 |
| D | 4.32 |
| L | 3.98 |
| U | 2.88 |
| C | 2.71 |
| M | 2.61 |
| F | 2.30 |
| Y | 2.11 |
| W | 2.09 |
| G | 2.03 |
| P | 1.82 |
| B | 1.49 |
| V | 1.11 |
| K | 0.69 |
| X | 0.17 |
| Q | 0.11 |
| J | 0.10 |
| Z | 0.07 |

(a) Frequência de letras em inglês

```
Número de Caracteres:

('Z', 128, 5.069306930693069)
('J', 120, 4.752475247524752)
('Y', 120, 4.752475247524752)
('D', 114, 4.514851485148514)
('P', 103, 4.079207920792079)
('C', 101, 4.0)
('Q', 96, 3.801980198019802)
('W', 93, 3.683168316831683)
('R', 89, 3.5247524752475243)
('H', 88, 3.4851485148514856)
('M', 85, 3.3663366336633667)
('L', 83, 3.2871287128712874)
('N', 77, 3.0495049504950495)
('G', 75, 2.9702970297029703)
('T', 74, 2.9306930693069306)
('F', 73, 2.8910891089108914)
('O', 72, 2.851485148514852)
('X', 67, 2.6534653465346536)
('V', 59, 2.3366336633663365)
('I', 54, 2.1386138613861387)
('S', 54, 2.1386138613861387)
('B', 47, 1.8613861386138613)
('E', 47, 1.8613861386138613)
('K', 42, 1.6633663366336635)
('U', 41, 1.6237623762376239)
('A', 39, 1.5445544554455446)
```

(b) Frequência de letras criptograma 1

Figura 1: Resultados aplicação frequência de letras a cada criptograma

```
('J', 144, 9.856262833675565)
('A', 137, 9.3771389459274475)
('I', 101, 6.913073237508556)
('V', 91, 6.2286105407255315)
('S', 90, 6.160164271047227)
('G', 88, 6.023271731690623)
('H', 74, 5.065023956194388)
('Y', 65, 4.4490075290896645)
('Z', 48, 3.2854209445585225)
('N', 45, 3.0800821355236137)
('P', 43, 2.943189596167009)
('E', 37, 2.532511978097194)
('U', 37, 2.532511978097194)
('K', 29, 1.9849418206707734)
('L', 29, 1.9849418206707734)
('Q', 26, 1.7796030116358659)
('R', 20, 1.3689253935660506)
('T', 20, 1.3689253935660506)
('W', 17, 1.1635865845311435)
('M', 16, 1.0951403148528405)
('F', 15, 1.0266940451745379)
('O', 4, 0.2737850787132101)
('C', 2, 0.13689253935660506)
('B', 1, 0.06844626967830253)
('D', 1, 0.06844626967830253)
('X', 0, 0.0)
```

(a) Frequência de letras criptograma 2

```
Número de Caracteres:

('X', 128, 9.349890430971513)
('G', 118, 8.619430241051864)
('H', 95, 6.939371804236669)
('U', 90, 6.574141709276844)
('I', 89, 6.501095690284879)
('V', 83, 6.062819576333093)
('C', 76, 5.551497443389335)
('T', 67, 4.894083272461651)
('Q', 54, 3.9444850255661064)
('M', 38, 2.7757487216946677)
('L', 35, 2.5566106647187773)
('O', 33, 2.410518626734843)
('B', 27, 1.9722425127830532)
('N', 27, 1.9722425127830532)
('R', 25, 1.8261504747991233)
('J', 23, 1.6800584368151936)
('Z', 22, 1.6070124178232286)
('A', 16, 1.1687363038714391)
('K', 15, 1.095690284879474)
('S', 14, 1.0226442658875092)
('F', 13, 0.9495982468955442)
('Y', 4, 0.2921840759678598)
('P', 3, 0.2191380569758948)
('D', 0, 0.0)
('E', 0, 0.0)
('W', 0, 0.0)
```

(b) Frequência de letras criptograma 3

Figura 2: Resultados aplicação frequência de letras a cada criptograma

Com os resultados demonstrados nas Figuras 1 e 2, podemos observar que o primeiro criptograma apresenta uma maior diferença para uma distribuição normal de letras em inglês por isso corresponderá à ao criptograma cifrado com a cifra de Vigenère. Com isto, resta assim distinguir dois criptogramas, para distinguir as outras cifras aplicamos a nossa estratégia para o criptograma correspondente à cifra *affine* e via-mos qual retornava texto legível uma vez que não podíamos fazer isto usando a cifra de substituição,pois a de *affine* corresponde a uma de substituição e o contrário não.

## 2.1 Cifra Afim

A primeira abordagem para decifrar que pensamos foi *brute-force*, uma vez, que esta cifra apresenta um conjunto de chaves pequena, dado que, a função de cifragem corresponde a $E(x) = (ax + b) \mod m$, em que m é o tamanho do alfabeto no caso destes criptogramas é 26 (apenas as letras se encontram cifradas) e $a$ tem de ser um co-primo de $m(26)$, desta forma $a \in 1, 3, 5, 7, 9, 11, 15, 17, 19, 21, 23, 25$ e $0 \leqslant b \leqslant 25$.

Portanto, existem 12 números que são co-primos com 26 e que são menores que este. Para cada valor de $a$ existem 26 possíveis *shift's* (valor de b), portanto existem $12 * 26$ ou 312 possíveis chaves.

De maneira a otimizar-mos este processo, sabendo que os criptogramas apresentam espaços e que a palavra *the* é das mais frequentes em inglês, consequentemente aplicamos uma heurística que verifica a palavra mais frequente com três letras nos criptogramas e com base neste palpite associa-mo-la a *the* e com isto tiramos os valores de $a$ e $b$.

**Resultado da chave obtida** ($D(x) = (11x + 8) \mod 26$)

| **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** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 8 | 13 | 18 | 23 | 2 | 7 | 12 | 17 | 22 | 1 | 6 | 11 | 16 | 21 | 0 | 5 | 10 | 15 | 20 | 25 | 4 | 9 | 14 | 19 | 24 | 3 |
| **I** | **N** | **S** | **X** | **C** | **H** | **M** | **R** | **W** | **B** | **G** | **L** | **Q** | **V** | **A** | **F** | **K** | **P** | **U** | **Z** | **E** | **J** | **O** | **T** | **Y** | **D** |

Figura 3: Tabela de substituição

Com isto sabemos que o criptograma associado a esta cifra é:

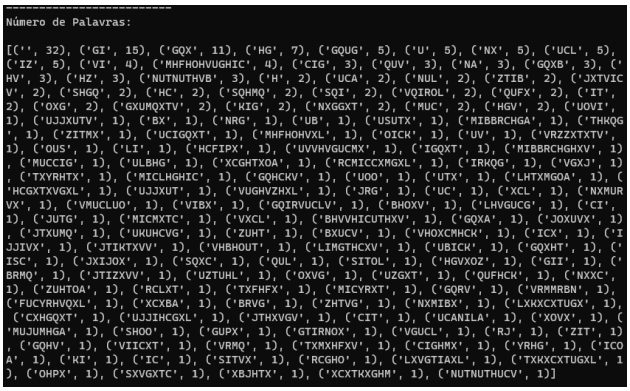
SJ SY VGJ JGG KUEH JG NACUSNA JHIJ QHIJ JHA QSYAYJ GL KIVOSVP, JHGYA QHG INA TAYJ AVJSJZAP JG JNUYJ JHASN GQV DUPWKAVJ, LSVP VAEAYYINM JG QINNIVJ JHASN NAZMSVW GV SJ, YHGUZP TA YUTKSJJAP JG TM JHIJ KSYEAZZIVAGUY EGZZAEJSGV GL I LAQ QSYA IVP KIVM LGGZSYH SVPSFSPUIZY, EIZZAP JHA RUTZSE. JHA KGYJ SVJGZANIVJ GL EHUNEHAY, JHA NGKIV EIJHGZSE EHUNEH, AFAV IJ JHA EIVGVSYIJSGV GL I YISVJ, IPKSJY, IVP ZSYJAVY RIJSAVJZM JG, I "PAFSZ'Y IPFGEIJA."JHA HGZSAYJ GL KAV, SJ IRRAINY, EIVVGJ TA IPKSJJAP JG RGYJHUKGUY HGVGUNY, UVJSZ IZZ JHIJ JHA PAFSZ EGUZP YIM IWISVYJ HSK SY OVGQV IVP QASWHAP. SL AFAV JHA VAQJGVSIV RHSZGYGRHM QANA VGJ RANKSJJAP JG TA CUAYJSGVAP, KIVOSVP EGUZP VGJ LAAZ IY EGKRZAJA IYYUNIVEA GL SJY JNUJH IY JHAM VGQ PG. JHA TAZSALY QHSEH QA HIFA KGYJ QINNIVJ LGN, HIFA VG YILAWUINP JG NAYJ GV, TUJ I YJIVPSVW SVFSJIJSGV JG JHA QHGZA QGNZP JG RNGFA JHAK UVLGUVPAP. SL JHA EHIZZAVWA SY VGJ IEEARJAP, GN SY IEEARJAP IVP JHA IJJAKRJ LISZY, QA INA LIN AVGUWH LNGK EANJISVJM YJSZZ; TUJ QA HIFA PGVA JHA TAYJ JHIJ JHA ABSYJSVW YJIJA GL HUKIV NAIYGV IPKSJY GL; QA HIFA VAWZAEJAP VGJHSVW JHIJ EGUZP WSFA JHA JNUJH I EHIVEA GL NAIEHSVW UY: SL JHA ZSYJY INA OARJ GRAV, QA KIM HGRA JHIJ SL JHANA TA I TAJJAN JNUJH, SJ QSZZ TA LGUVP QHAV JHA HUKIV KSVP SY EIRITZA GL NAEASFSVW SJ; IVP SV JHA KAIVJSKA QA KIM NAZM GV HIFSVW IJJISVAP YUEH IRRNGIEH JG JNUJH, IY SY RGYYSTZA SV GUN GQV PIM. JHSY SY JHA IKGUVJ GL EANJISVJM IJJISVITZA TM I LIZZSTZA TASVW, IVP JHSY JHA YGZA QIM GL IJJISVSVW SJ.

E o texto decifrado corresponde a:

it is not too much to require that what the wisest of mankind, those who are best entitled to trust their own judgment, find necessary to warrant their relying on it, should be submitted to by that miscellaneous collection of a few wise and many foolish individuals,called the public. the most intolerant of churches, the roman catholic church, even at the canonisation of a saint, admits, and listens patiently to, a "devil's advocate."the holiest of men, it appears,cannot be admitted to post humous honours, until all that the devil could say against him is known and weighed. if even the newtonian philosophy were not permitted to be questioned, man kind could not feelas complete assurance of its truth as they now do. the beliefs which we have most warrant for, have no safeguard to rest on, but a standing invitation to the whole world to prove them unfounded. if the challenge is not accepted, or is accepted and the attempt fails, weare far enough from certainty still; but we have done the best thatthe existing state of human reason admits of; we have neglected nothing that could give the truth a chance of reaching us: if the lists are kept open, we may hope that if there be a better truth, it will be found when the human mind is capable of receiving it; and in the meantime we may rely on having attained such approach to truth, as is possible in our own day. this is the amount of certainty attainable by a fallible being, and this the sole way of attaining it.

## 2.2 Método de Substituição

Como o o resultado obtido para cifra *affine* foi o criptograma 2, resta apenas o criptograma 3 que irá corresponder à cifra de substituição. Esta cifragem era a mais direta, isto é, apenas tínhamos de descobrir a que letra correspondia cada letra. A nossa intuição foi contabilizar, a partir do criptograma 3, o número de vezes que cada palavra aparecia.




| Word | Parts of speech | OEC rank |
|---|---|---|
| the | Article | 1 |
| be | Verb | 2 |
| to | Preposition | 3 |
| of | Preposition | 4 |
| and | Conjunction | 5 |
| a | Article | 6 |
| in | Preposition | 7 |
| that | Conjunction et al. | 8 |
| have | Verb | 9 |
| I | Pronoun | 10 |
| it | Pronoun | 11 |
| for | Preposition | 12 |

Figura 4: Frequência das palavras no criptograma

Pelas imagens apresentadas em cima, podemos perceber que a palavra mais frequente em inglês é a palavra **THE**, e por isso deduzimos que a palavra com 3 letras iria corresponder ao **THE**, que neste caso era a palavra **GQX**. O próximo passo foi encontrar palavras com as letras **GQX** como por exemplo **GQXB** e ir substituindo de acordo com a lista[1].

Sendo assim seguindo um processo iterativo, depois de algumas trocas concluímos que a tabela apresentada na Figura 5 representa a nossa "chave"

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

Figura 5: Tabela de substituição

Com isto sabemos que o criptograma associado a esta cifra é:

HG UOVI UJJXUTV VI GI BX, NRG H UB CIG USUTX GQUG UCA MIBBRCHGA QUV U THKQG GI ZITMX UCIGQXT GI NX MHFHOHVXL. VI OICK UV GQX VRZZXTXTV NA GQX NUL OUS LI CIG HCFIPX UVVHVGUCMX ZTIB IGQXT MIBBRCHGHXV, H MUCCIG ULBHG GQUG JXTVICV XCGHTXOA RCMICCXMGXL SHGQ GQXB IRKQG GI VGXJ HC UCL TXYRHTX GQUG U MICLHGHIC IZ GQHCKV SHGQ SQHMQ UOO SQI UTX LHTXMGOA HCGXTXVGXL UJJXUT GI NX VUGHVZHXL, VQIROL NX JRG UC XCL GI NXMURVX HG HV U VMUCLUO GI JXTVICV VIBX GQIRVUCLV IZ BHOXV LHVGUCG, SQI QUFX CI JUTG IT MICMXTC HC HG. OXG GQXB VXCL BHVVHICUTHXV, HZ GQXA JOXUVX, GI JTXUMQ UKUHCVG HG; UCL OXG GQXB, NA UCA ZUHT BXUCV (IZ SQHMQ VHOXCMHCK GQX GXUMQXTV HV CIG ICX), IJJIVX GQX JTIKTXVV IZ VHBHOUT LIMGTHCXV UBICK GQXHT ISC JXIJOX. HZ MHFHOHVUGHIC QUV KIG GQX NXGGXT IZ NUTNUTHVB SQXC NUTNUTHVB QUL GQX SITOL GI HGVXOZ, HG HV GII BRMQ GI JTIZXVV GI NX UZTUHL OXVG NUTNUTHVB, UZGXT QUFHCK NXXC ZUHTOA KIG RCLXT, VQIROL TXFHFX UCL MICYRXT MHFHOHVUGHIC. U MHFHOHVUGHIC GQUG MUC GQRV VRMMRBN GI HGV FUCYRHVQXL XCXBA, BRVG ZHTVG QUFX NXMIBX VI LXKXCXTUGX, GQUG CXHGQXT HGV UJJIHCGXL JTHXVGV UCL GXUMQXTV, CIT UCANILA XOVX, QUV GQX MUJUMHGA, IT SHOO GUPX GQX GTIRNOX, GI VGUCL RJ ZIT HG. HZ GQHV NX VI, GQX VIICXT VRMQ U MHFHOHVUGHIC TXMXHFXV CIGHMX GI YRHG, GQX NXGGXT. HG MUC ICOA KI IC ZTIB NUL GI SITVX, RCGHO LXVGTIAXL UCL TXKXCXTUGXL (OHPX GQX SXVGXTC XBJHTX) NA XCXTKXGHM NUTNUTHUCV.

E o texto decifrado corresponde a:

IT ALSO APPEARS SO TO ME, BUT I AM NOT AWARE THAT ANY COMMUNITY HAS A RIGHT TO FORCE ANOTHER TO BE CIVILISED. SO LONG AS THE SUFFERERS BY THE BAD LAW DO NOT INVOKE ASSISTANCE FROM OTHER COMMUNITIES, I CANNOT ADMIT THAT PERSONS ENTIRELY UNCONNECTED WITH THEM OUGHT TO STEP IN AND REQUIRE THAT A CONDITION OF THINGS WITH WHICH ALL WHO ARE DIRECTLY INTERESTED APPEAR TO BE SATISFIED, SHOULD BE PUT AN END TO BECAUSE IT IS A SCANDAL TO PERSONS SOME THOUSANDS OF MILES DISTANT, WHO HAVE NO PART OR CONCERN IN IT. LET THEM SEND MISSIONARIES, IF THEY PLEASE, TO PREACH AGAINST IT; AND LET THEM, BY ANY FAIR MEANS (OF WHICH SILENCING THE TEACHERS IS NOT ONE), OPPOSE THE PROGRESS OF SIMILAR DOCTRINES AMONG THEIR OWN PEOPLE. IF CIVILISATION HAS GOT THE BETTER OF BARBARISM WHEN BARBARISM HAD THE WORLD TO ITSELF, IT IS TOO MUCH TO PROFESS TO BE AFRAID LEST BARBARISM, AFTER HAVING BEEN FAIRLY GOT UNDER, SHOULD REVIVE AND CONQUER CIVILISATION. A CIVILISATION THAT CAN THUS SUCCUMB TO ITS VANQUISHED ENEMY, MUST FIRST HAVE BECOME SO DEGENERATE, THAT NEITHER ITS APPOINTED PRIESTS AND TEACHERS, NOR ANYBODY ELSE, HAS THE CAPACITY, OR WILL TAKE THE TROUBLE, TO STAND UP FOR IT. IF THIS BE SO, THE SOONER SUCH A CIVILISATION RECEIVES NOTICE TO QUIT, THE BETTER. IT CAN ONLY GO ON FROM BAD TO WORSE, UNTIL DESTROYED AND REGENERATED (LIKE THE WESTERN EMPIRE) BY ENERGETIC BARBARIANS.

## 2.3 Cifra de Vigenère

Assumindo que o criptograma 1 corresponde a esta cifra com base na explicação dada anteriormente, a estratégia seguida de maneira a decifrar este texto foi a seguinte: primeiro descobrimos o tamanho da chave, para isso obtivemos as palavras com três letras que apareciam no texto (**Figura 5**) e via-mos a diferença entre os offset's em que se encontravam no texto de modo a calcular a sua periodicidade e com isto vimos qual era o máximo divisor comum entre as várias diferenças obtidas, e como resultado deu-nos 5.

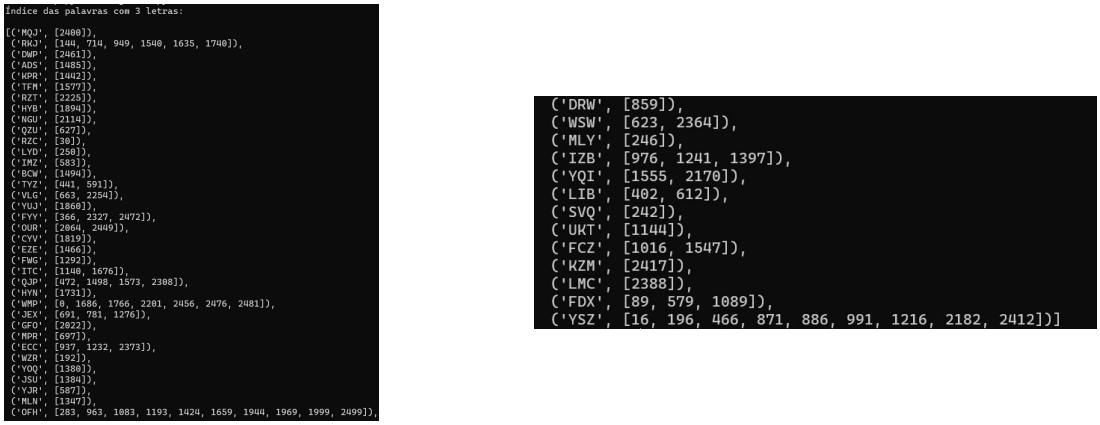



Figura 6: Frequência de palavras de 3 no criptograma

Após descoberta do tamanho da chave, dividimos o texto em blocos de 5 (**Figura 6**), uma vez, que pela definição de cifra de Viginère cada bloco será cifrado com a mesma chave, estes irão comportar-se como se fossem aplicados uma cifra de césar, por isso podemos aplicar como anteriormente a frequência de letras/palavras. Como WMP aparece algumas vezes assumimos que correspondia à palavra *the*, obtendo as três primeiras letras da chave **DFL** (chave parcial).

A palavra **YSZ** também aparece isolada num bloco de 5(_YSZ_). Com a nossa chave parcial obtida até então, o seu resultado originava TH, e como YSZ é dos trigramas mais frequentes assumimos que correspondia à palavra *the* e com isto obtive-mos **DFLV**.

Aplicando a nova chave parcial com uma letra aleatória no fim, como por exemplo, **X**, obtivemos um texto onde existia a palavra FRPM (**Figura 7**) que advinha da sequência de letras **QMMP**, ou seja esta teria de corresponder à palavra *FROM*, de maneira a fazer sentido. Consequentemente, fomos ao bloco da palavra QMMP e o **M** encontrava-se na quinta posição, vimos qual era a letra que teria de ser para dar a letra **O** e obtivemos o **Y**, concluindo assim, que a chave era **DFLVY** .


Figura 7: Criptograma em blocos de 5

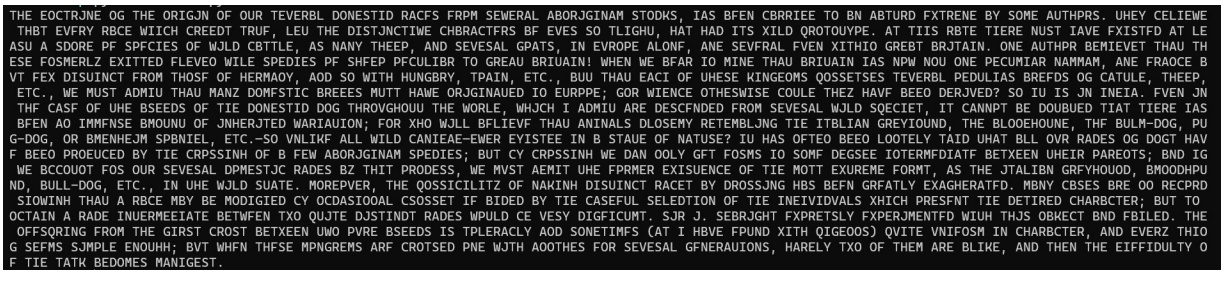



Figura 8: Decifragem com chave parcial

Com a chave DFLVY obtivemos o seguinte texto limpo:

THE DOCTRINE OF THE ORIGIN OF OUR SEVERAL DOMESTIC RACES FROM SEVERAL ABORIGINAL STOCKS, HAS BEEN CARRIED TO AN ABSURD EXTREME BY SOME AUTHORS. THEY BELIEVE THAT EVERY RACE WHICH BREEDS TRUE, LET THE DISTINCTIVE CHARACTERS BE EVER SO SLIGHT, HAS HAD ITS WILD PROTOTYPE. AT THIS RATE THERE MUST HAVE EXISTED AT LEAST A SCORE OF SPECIES OF WILD CATTLE, AS MANY SHEEP, AND SEVERAL GOATS, IN EUROPE ALONE, AND SEVERAL EVEN WITHIN GREAT BRITAIN. ONE AUTHOR BELIEVES THAT THERE FORMERLY EXISTED ELEVEN WILD SPECIES OF SHEEP PECULIAR TO GREAT BRITAIN! WHEN WE BEAR IN MIND THAT BRITAIN HAS NOW NOT ONE PECULIAR MAMMAL, AND FRANCE BUT FEW DISTINCT FROM THOSE OF GERMANY, AND SO WITH HUNGARY, SPAIN, ETC., BUT THAT EACH OF THESE KINGDOMS POSSESSES SEVERAL PECULIAR BREEDS OF CATTLE, SHEEP, ETC., WE MUST ADMIT THAT MANY DOMESTIC BREEDS MUST HAVE ORIGINATED IN EUROPE; FOR WHENCE OTHERWISE COULD THEY HAVE BEEN DERIVED? SO IT IS IN INDIA. EVEN IN THE CASE OF THE BREEDS OF THE DOMESTIC DOG THROUGHOUT THE WORLD, WHICH I ADMIT ARE DESCENDED FROM SEVERAL WILD SPECIES, IT CANNOT BE DOUBTED THAT THERE HAS BEEN AN IMMENSE AMOUNT OF INHERITED VARIATION; FOR WHO WILL BELIEVE THAT ANIMALS CLOSELY RESEMBLING THE ITALIAN GREYHOUND, THE BLOODHOUND, THE BULL-DOG, PUG-DOG, OR BLENHEIM SPANIEL, ETC.—SO UNLIKE ALL WILD CANIDAE—EVER EXISTED IN A STATE OF NATURE? IT HAS OFTEN BEEN LOOSELY SAID THAT ALL OUR RACES OF DOGS HAVE BEEN PRODUCED BY THE CROSSING OF A FEW ABORIGINAL SPECIES; BUT BY CROSSING WE CAN ONLY GET FORMS IN SOME DEGREE INTERMEDIATE BETWEEN THEIR PARENTS; AND IF WE ACCOUNT FOR OUR SEVERAL DOMESTIC RACES BY THIS PROCESS, WE MUST ADMIT THE FORMER EXISTENCE OF THE MOST EXTREME FORMS, AS THE ITALIAN GREYHOUND, BLOODHOUND, BULL-DOG, ETC., IN THE WILD STATE. MOREOVER, THE POSSIBILITY OF MAKING DISTINCT RACES BY CROSSING HAS BEEN GREATLY EXAGGERATED. MANY CASES ARE ON RECORD SHOWING THAT A RACE MAY BE MODIFIED BY OCCASIONAL CROSSES IF AIDED BY THE CAREFUL SELECTION OF THE INDIVIDUALS WHICH PRESENT THE DESIRED CHARACTER; BUT TO OBTAIN A RACE INTERMEDIATE BETWEEN TWO QUITE DISTINCT RACES WOULD BE VERY DIFFICULT. SIR J. SEBRIGHT EXPRESSLY EXPERIMENTED WITH THIS OBJECT AND FAILED. THE OFFSPRING FROM THE FIRST CROSS BETWEEN TWO PURE BREEDS IS TOLERABLY AND SOMETIMES (AS I HAVE FOUND WITH PIGEONS) QUITE UNIFORM IN CHARACTER, AND EVERY THING SEEMS SIMPLE ENOUGH; BUT WHEN THESE MONGRELS ARE CROSSED ONE WITH ANOTHER FOR SEVERAL GENERATIONS, HARDLY TWO OF THEM ARE ALIKE, AND THEN THE DIFFICULTY OF THE TASK BECOMES MANIFEST.

# 3 Conclusão

Dado como terminado este trabalho, concluímos que este permitiu-nos aprofundar os conhecimentos obtidos nas aulas teóricas, percebendo melhor na prática as diferenças entre os métodos lecionados de cifras clássicas.

Concluímos que as cifras de criptografia clássica apresentam limitações uma vez que revelam padrões nos criptogramas gerados nos casos da cifra *affine* e de substituição, enquanto que na cifra de Viginère só é segura se a chave for do tamanho da mensagem, uma vez que se determinar-mos o tamanho da chave esta vai ter uma periodicidade e gerar padrões no criptograma já que cada bloco do tamanho da chave se irá comportar como uma cifra de césar e assim adquirir as vulnerabilidades desta, por isso a sua segurança reside na chave, o que na prática não é muito eficiente, uma vez que as mensagens são grandes e com isto a chaves também terão de ser. Na resolução de todas as cifras com os "palpites"baseados nas análises de frequências um atacante neste tipo de cifra tem uma probabilidade alta de acertar, o que demonstra o quão importante é para a criptografia que um atacante tenha a mesma probabilidade de adivinhar ou não.

Onde surgiram mais dificuldades foi na decifragem do criptograma 1 com a cifra de Viginère, pois nós não estávamos a contar com os espaços e ficamos um bocado perdidos, pois nada estava a dar certo. Mas os obstáculos foram ultrapassados e conseguimos perceber onde estava a ratoeira.

# 4 Anexos

## 4.1 freqletras.py

```
import re
txt1 = "WMP BRHEMGQJ JD YSZ RWTBGQ ZA RZC QHAPMYO OJKHXEDA WLXCV QMMP DZTHWLG
 DGZMGJNYVJ XEJANX, FDX WCHS XYUWTZB YZ YQ LWQXWO CAYCZKH MT VTXZ DZECMUX. RKJJ
 ZHQTZTH ECYW PQCUD MYFJ RFLHS ZUJPYQ YCPC, WZR YSZ GNDOGQHEDTH NCYUFNOCUX WC JGZP
 XZ QONRCR, SVQ MLY LYD ULQO NUTEJRBUP. DY OFLX MYWJ OFHWP KXXE FDAP CANDOCG LO OJLNR
 F NARWP MI DKCFNPN RK RGOI XYWYWZ, FD KDSJ QKJPK, FYY VJGZPDQ BMDYD, LS ZSUTAZ DQZIC,
 LIB XPQCUFW CYJY ULYSDL LCZYW MMGWFTI. TYZ DZECMU MZJLJGZQ YSVR YSZPH QJPPJCGW
 JIDQWJO COJGZL BTGB XAZALJD MI DCCHU KCFZWDYU EJ JWPVR GCDRDNY! ZMPI ZJ WCDW DL RTIB
 YSVR GCDRDNY FDX IMZ YJR TYZ SJNPJLFC KDRXVJ, LIB KCVLFJ WSW QZU ITNRLSNO IWZH WMZNC
 TQ EHWXVLB, VLG DJ ZNEC KZYBYUD, QSFTI, JEX., MPR YSVR JLXF TQ RKJDZ NNYBBRRD
 NRXDZQVJD QHAPMYO AZAXQTVP GCZCGX JD HLOROJ, QKJPK, JEX., HZ PZDO DIXDR YSVR RLIW
 IZHCVYTX EWPZBV XPQW SVTH ZMGJNYVRHI DL JFMMSJ; DRW RFHSNZ RYSZPZNDZ FTFGB YSZW MLQC
 GPZL IPMGYJO? VT DR ND GQ TIBLF. CYJY GQ ECC HLNC TQ RKJ WPHJON RK OFH OJKHXEDA IZB
 WMCJSJMZPR YSZ ZTCGB, HCGFM D DIXDR FCZ GJDXCQIPY IWZH VJGZPDQ RGOI NNHHTZQ, TO
 FFYIMW MZ GTFWRHI OFDY OFHWP FDX WCHS VL NXHCQXP YPTFIR TQ GQMPMGWJO TDWTVRLTY; ITC
 UKT RGOQ WCONPQC YSVR FYDKDQD AOTDZJB CZQHRMGGQL OFH TOYONLI JWPTFRZYY, YSZ
 EQZJBKTFIB, ECC GFGJ-IZB, UFB-GTR, RW WJHSSZGP DKYQNPG, JEX.|XZ SQQTFC FWG ZNWY
 FFYDBDJ|ZTHW ZVLXEZB NY Y XEVRH ZA QFEPPH? DR MLN RKEZL GPZL QZJQHQJ QDNO RKFE YOQ
 JSU CVAHX JD IZBQ MLQC GPZL UCJBXHPY ED OFH NMMVXTIE TQ Y KPR DGZMGJNYVJ XAZALJD;
 EZE ZB NMMVXTIE BP ADS JLOD BCW QJPPX DL XZHC IPBPHJ DLWJCHCGNLOC GPOUHJY RKJTM
 SFCZLWX; YQI DD BP YFHZPLW QJP TFM VJGZPDQ YMPJDOGF CVAHX WW YSDQ UCJAHXD, ZJ HSVY
 VBPNE RKJ AMURPM HCTNRHSNZ RK OFH XJQW PSRUJXZ ITCHQ, LN WMP GWFWDYQ RMCBMZPLG,
 WJRTOCMXSO, EZWG-GTR, HYN., NY RKJ RGOI NRDYP. PTCZMYJC, WMP NRXDDZLQTOW TQ KDPTIE
 ITNRLSNO UFNZQ GJ AUTDNGQL CYV MZCQ RMCDYWT HCLBEHWLOCG. HYQD XYVJD YUJ JL WPXMUI
 NFRBTIE YSVR F MYFJ HYB MZ PTODDLJO ZB ZXADXTJLDQ XPRXDZQ NQ YLIPY ED OFH NVPHKFG
 VJWZAWNZI RK OFH TIBLATYSDQD UKNNC SWPNCQY OFH OZQLWPY FMLMYFYPM; GFO WT JZWFTI D
 CVAH TIRHWXZBLFEZ EJERCHS OUR BPGWJ YGVYTIAW CVAHX RMXQO ZH GZPB ODDINNPJW. NGU U.
 VJMMGJME CAUCZQVQJ CAUPMGPJYOCG HDRK ECGV ZWHHHE YQI AYLQPY. YSZ RKQNNUNYB IWZH
 WMP DLWDO FWZNQ GPOUHJY RZT KSUJ WPHJON LX OMOJCVZOD VLG DJKHYTHCV (VQ N CYYJ
 AMXSO ULYS NLLPJLV) LSLYP SQNQJPP TI FMLMYFYPM, FYY HAPMW YSDLJ DZCPX NGPUWZ HSZPEK;
 WSW HCCQ ECCVJ HMQLCZJV LMC HCJQVJO MQJ RGWM VLRYSZP KZM VJGZPDQ BCQJCVRLTYN, MLMBOD
 OUR ZA WMPH DWP YONVZ, FYY WMPI WMP BLKQDAXQET RK OFH EVQN MZARRPN PFYDDHXE."

txt2 = "HG UOVI UJJXUTV VI GI BX, NRG H UB CIG USUTX GQUG UCA MIBBRCHGA QUV U THKQG GI
ZITMX UCIGQXT GI NX MHFHOHVXL. VI OICK UV GQX VRZZXTXTV NA GQX NUL OUS LI CIG HCFIPX
UVVHVGUCMX ZTIB IGQXT MIBBRCHGHXV, H MUCCIG ULBHG GQUG JXTVICV XCGHTXOA RCMICCXMGXL
SHGQ GQXB IRKQG GI VGXJ HC UCL TXYRHTX GQUG U MICLHGHIC IZ GQHCKV SHGQ SQHMQ UOO SQI
UTX LHTXMGOA HCGXTXVGXL UJJXUT GI NX VUGHVZHXL, VQIROL NX JRG UC XCL GI NXMURVX HG HV
U VMUCLUO GI JXTVICV VIBX GQIRVUCLV IZ BHOXV LHVGUCG, SQI QUFX CI JUTG IT MICMXTC HC
HG. OXG GQXB VXCL BHVVHICUTHXV, HZ GQXA JOXUVX, GI JTXUMQ UKUHCVG HG; UCL OXG GQXB,
NA UCA ZUHT BXUCV (IZ SQHMQ VHOXCMHCK GQX GXUMQXTV HV CIG ICX), IJJIVX GQX JTIKTXVV
IZ VHBHOUT LIMGTHCXV UBICK GQXHT ISC JXIJOX. HZ MHFHOHVUGHIC QUV KIG GQX NXGGXT IZ
NUTNUTHVB SQXC NUTNUTHVB QUL GQX SITOL GI HGVXOZ, HG HV GII BRMQ GI JTIZXVV GI NX
UZTUHL OXVG NUTNUTHVB, UZGXT QUFHCK NXXC ZUHTOA KIG RCLXT, VQIROL TXFHFX UCL MICYRXT
MHFHOHVUGHIC. U MHFHOHVUGHIC GQUG MUC GQRV VRMMRBN GI HGV FUCYRHVQXL XCXBA, BRVG ZHTVG
QUFX NXMIBX VI LXKXCXTUGX, GQUG CXHGQXT HGV UJJIHCGXL JTHXVGV UCL GXUMQXTV, CIT
UCANILA XOVX, QUV GQX MUJUMHGA, IT SHOO GUPX GQX GTIRNOX, GI VGUCL RJ ZIT HG. HZ
GQHV NX VI, GQX VIICXT VRMQ U MHFHOHVUGHIC TXMXHFXV CIGHMX GI YRHG, GQX NXGGXT.
HG MUC ICOA KI IC ZTIB NUL GI SITVX, RCGHO LXVGTIAXL UCL TXKXCXTUGXL (OHPX
GQX SXVGXTC XBJHTX) NA XCXTKXGHM NUTNUTHUCV."
```

```
txt3="""SJ SY VGJ JGG KUEH JG NACUSNA JHIJ QHIJ JHA QSYAYJ GL KIVOSVP, JHGYAQHG
INA TAYJ AVJSJZAP JG JNUYJ JHASN GQV DUPWKAVJ, LSVP VAEAYYINM JGQINNIVJ JHASN
NAZMSVW GV SJ, YHGUZP TA YUTKSJJAP JG TM JHIJKSYEAZZIVAGUY EGZZAEJSGV GL I LAQ
QSYA IVP KIVM LGGZSYH SVPSFSPUIZY,EIZZAP JHA RUTZSE. JHA KGYJ SVJGZANIVJ GL
EHUNEHAY, JHA NGKIV EIJHGZSEEHUNEH, AFAV IJ JHA EIVGVSYIJSGV GL I YISVJ, IPKSJY,
IVP ZSYJAVYRIJSAVJZM JG, I "PAFSZ'Y IPFGEIJA." JHA HGZSAYJ GL KAV, SJ IRRAINY,
EIVVGJ TA IPKSJJAP JG RGYJHUKGUY HGVGUNY, UVJSZ IZZ JHIJ JHA PAFSZEGUZP YIM
IWISVYJ HSK SY OVGQV IVP QASWHAP. SL AFAV JHA VAQJGVSIVRHSZGYGRHM QANA VGJ
RANKSJJAP JG TA CUAYJSGVAP, KIVOSVP EGUZP VGJ LAAZIY EGKRZAJA IYYUNIVEA GL SJY
JNUJH IY JHAM VGQ PG. JHA TAZSALY QHSEHQA HIFA KGYJ QINNIVJ LGN, HIFA VG
YILAWUINP JG NAYJ GV, TUJ I YJIVPSVWSVFSJIJSGV JG JHA QHGZA QGNZP JG RNGFA JHAK
UVLGUVPAP. SL JHAEHIZZAVWA SY VGJ IEEARJAP, GN SY IEEARJAP IVP JHA IJJAKRJ
LISZY, QAINA LIN AVGUWH LNGK EANJISVJM YJSZZ; TUJ QA HIFA PGVA JHA TAYJ JHIJJHA
ABSYJSVW YJIJA GL HUKIV NAIYGV IPKSJY GL; QA HIFA VAWZAEJAPVGJHSVW JHIJ EGUZP
WSFA JHA JNUJH I EHIVEA GL NAIEHSVW UY: SL JHAZSYJY INA OARJ GRAV, QA KIM HGRA
JHIJ SL JHANA TA I TAJJAN JNUJH, SJQSZZ TA LGUVP QHAV JHA HUKIV KSVP SY
EIRITZA GL NAEASFSVW SJ; IVP SVJHA KAIVJSKA QA KIM NAZM GV HIFSVW IJJISVAP
YUEH IRRNGIEH JG JNUJH, IYSY RGYYSTZA SV GUN GQV PIM. JHSY SY JHA IKGUVJ GL
EANJISVJM IJJISVITZATM I LIZZSTZA TASVW, IVP JHSY JHA YGZA QIM GL IJJISVSVW SJ."""

abc = ["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"]

# Recebe um criptograma e calcula com frequência aparece cada letra
def contaPalavras(txt):
    count = 0
    listaPal = []
    searched = []
    txt_limpo = re.sub("[,|.|;|)|(]", " ", txt)
    words = txt_limpo.split(" ")
    for x in words:
        if(x not in searched):
            count = words.count(x)
            t=(x,count)
            listaPal.append(t)
        searched.append(x)
    Sort_Tuple(listaPal)
    print(listaPal)

    def main():
        contaPalavras(txt1)


if __name__ == "__main__":
    main()
```

### 4.2 afim.py

```
import sys
import math
import re
from string import ascii_lowercase

# co-primos com 26 sendo estes menores que 26
coprimes26=[1,3,5,7,9,11,15,17,19,21,23,25]

# Calcula inversa da multiplicação modular necessária
# para decifragem de uma cifra affine
def modInverse(a, m):
    a = a % m
    for x in range(1, m):
        if ((a * x) % m == 1):
            return x
    return 1

#Heuristica que com base no trigrama mais frequente tentamos encontar a função
#afim de forma
#a corresponder a THE
def generateKey(trigram,alphabet) :
    x = 0
    y = 0
    flag=0
    for a in coprimes26:
        for b in range(26):
            if ((modInverse(a,26)*(alphabet[trigram[0].lower()]-b))%26)==19 and
            ((modInverse(a,26)*(alphabet[trigram[1].lower()]-b))%26) == 7 and
            ((modInverse(a,26)*(alphabet[trigram[2].lower()]-b))%26) == 4:
                x=a
                y=b
                flag=1
                break;
    if not flag :
        print("No key")
    return (x,y)

#Ordena os tuplos pelo segundo argumento
def Sort_Tuple(tup):
    lst = len(tup)
    for i in range(0, lst):
        for j in range(0, lst-i-1):
            if (tup[j][1] < tup[j + 1][1]):
                temp = tup[j]
                tup[j]= tup[j + 1]
                tup[j + 1]= temp
    return tup
```

```
#Palavra com três letras mais frequente no criptograma
def mostFrequentTrigram(txt):
    count = 0
    listaPal = []
    searched = []
    txt_limpo = re.sub("[,|.|;|)|(]", "", txt)
    words = txt_limpo.split(" ")
    for x in words:
        if(x not in searched and len(x)==3):
            count = words.count(x)
            t=(x,count)
            listaPal.append(t)
        searched.append(x)
    Sort_Tuple(listaPal)
    return(listaPal[0][0][0],listaPal[0][0][1],listaPal[0][0][2])

#Decode para o texto final
def decode(key,ciphertext,alphabet) :
    clean_text = ""
    i=0
    for letter in ciphertext :
        isLetra=re.match("[A-Z]",letter)
        if(isLetra):
            value_letter=(modInverse(key[0],26)*(alphabet[letter.lower()]-key[1]))%26
            for letra, value_letra in alphabet.items():
                if value_letra == value_letter:
                    encrypted_letter= letra
            clean_text += encrypted_letter
        else:
            clean_text += letter
    print(clean_text)
```

```
def main():
    alphabet={}
    i=0
    # Associa por exemplo A->0 B->1 ... Z->26
    for c in ascii_lowercase:
        alphabet[c]=i
        i+=1

    txt="""SJ SY VGJ JGG KUEH JG NACUSNA JHIJ QHIJ JHA QSYAYJ GL KIVOSVP, JHGYAQHG
    INA TAYJ AVJSJZAP JG JNUYJ JHASN GQV DUPWKAVJ, LSVP VAEAYYINM JGQINNIVJ JHASN
    NAZMSVW GV SJ, YHGUZP TA YUTKSJJAP JG TM JHIJKSYEAZZIVAGUY EGZZAEJSGV GL I LAQ
    QSYA IVP KIVM LGGZSYH SVPSFSPUIZY,EIZZAP JHA RUTZSE. JHA KGYJ SVJGZANIVJ GL
    EHUNEHAY, JHA NGKIV EIJHGZSEEHUNEH, AFAV IJ JHA EIVGVSYIJSGV GL I YISVJ, IPKSJY,
    IVP ZSYJAVYRIJSAVJZM JG, I "PAFSZ'Y IPFGEIJA." JHA HGZSAYJ GL KAV, SJ IRRAINY,
    EIVVGJ TA IPKSJJAP JG RGYJHUKGUY HGVGUNY, UVJSZ IZZ JHIJ JHA PAFSZEGUZP YIM
    IWISVYJ HSK SY OVGQV IVP QASWHAP. SL AFAV JHA VAQJGVSIVRHSZGYGRHM QANA VGJ
    RANKSJJAP JG TA CUAYJSGVAP, KIVOSVP EGUZP VGJ LAAZIY EGKRZAJA IYYUNIVEA GL SJY
    JNUJH IY JHAM VGQ PG. JHA TAZSALY QHSEHQA HIFA KGYJ QINNIVJ LGN, HIFA VG
    YILAWUINP JG NAYJ GV, TUJ I YJIVPSVWSVFSJIJSGV JG JHA QHGZA QGNZP JG RNGFA JHAK
    UVLGUVPAP. SL JHAEHIZZAVWA SY VGJ IEEARJAP, GN SY IEEARJAP IVP JHA IJJAKRJ
    LISZY, QAINA LIN AVGUWH LNGK EANJISVJM YJSZZ; TUJ QA HIFA PGVA JHA TAYJ JHIJJHA
    ABSYJSVW YJIJA GL HUKIV NAIYGV IPKSJY GL; QA HIFA VAWZAEJAPVGJHSVW JHIJ EGUZP
    WSFA JHA JNUJH I EHIVEA GL NAIEHSVW UY: SL JHAZSYJY INA OARJ GRAV, QA KIM HGRA
    JHIJ SL JHANA TA I TAJJAN JNUJH, SJQSZZ TA LGUVP QHAV JHA HUKIV KSVP SY
    EIRITZA GL NAEASFSVW SJ; IVP SVJHA KAIVJSKA QA KIM NAZM GV HIFSVW IJJISVAP
    YUEH IRRNGIEH JG JNUJH, IYSY RGYYSTZA SV GUN GQV PIM. JHSY SY JHA IKGUVJ GL
    EANJISVJM IJJISVITZATM I LIZZSTZA TASVW, IVP JHSY JHA YGZA QIM GL IJJISVSVW SJ."""


    triagram=mostFrequentTrigram(txt)
    print("-------------------------")
    print("Palavra mais frequente com 3 letras:\n")
    print(mostFrequentTrigram(txt))
    print("-------------------------")
    print("Valor de (a,b):\n")
    print(generateKey(triagram,alphabet))
    key=generateKey(triagram,alphabet)
    print("-------------------------")
    print("Texto final:\n")
    decode(key,txt,alphabet)


if __name__ == "__main__":
    main()
```

### 4.3 subs.py

```
import re

txt = "HG UOVI UJJXUTV VI GI BX, NRG H UB CIG USUTX GQUG UCA MIBBRCHGA QUV U THKQG GI
ZITMX UCIGQXT GI NX MHFHOHVXL. VI OICK UV GQX VRZZXTXTV NA GQX NUL OUS LI CIG HCFIPX
UVVHVGUCMX ZTIB IGQXT MIBBRCHGHXV, H MUCCIG ULBHG GQUG JXTVICV XCGHTXOA RCMICCXMGXL
SHGQ GQXB IRKQG GI VGXJ HC UCL TXYRHTX GQUG U MICLHGHIC IZ GQHCKV SHGQ SQHMQ UOO SQI
UTX LHTXMGOA HCGXTXVGXL UJJXUT GI NX VUGHVZHXL, VQIROL NX JRG UC XCL GI NXMURVX HG HV
U VMUCLUO GI JXTVICV VIBX GQIRVUCLV IZ BHOXV LHVGUCG, SQI QUFX CI JUTG IT MICMXTC HC
HG. OXG GQXB VXCL BHVVHICUTHXV, HZ GQXA JOXUVX, GI JTXUMQ UKUHCVG HG; UCL OXG GQXB,
NA UCA ZUHT BXUCV (IZ SQHMQ VHOXCMHCK GQX GXUMQXTV HV CIG ICX), IJJIVX GQX JTIKTXVV
IZ VHBHOUT LIMGTHCXV UBICK GQXHT ISC JXIJOX. HZ MHFHOHVUGHIC QUV KIG GQX NXGGXT IZ
NUTNUTHVB SQXC NUTNUTHVB QUL GQX SITOL GI HGVXOZ, HG HV GII BRMQ GI JTIZXVV GI NX
UZTUHL OXVG NUTNUTHVB, UZGXT QUFHCK NXXC ZUHTOA KIG RCLXT, VQIROL TXFHFX UCL MICYRXT
MHFHOHVUGHIC. U MHFHOHVUGHIC GQUG MUC GQRV VRMMRBN GI HGV FUCYRHVQXL XCXBA, BRVG ZHTVG
QUFX NXMIBX VI LXKXCXTUGX, GQUG CXHGQXT HGV UJJIHCGXL JTHXVGV UCL GXUMQXTV, CIT
UCANILA XOVX, QUV GQX MUJUMHGA, IT SHOO GUPX GQX GTIRNOX, GI VGUCL RJ ZIT HG. HZ
GQHV NX VI, GQX VIICXT VRMQ U MHFHOHVUGHIC TXMXHFXV CIGHMX GI YRHG, GQX NXGGXT.
HG MUC ICOA KI IC ZTIB NUL GI SITVX, RCGHO LXVGTIAXL UCL TXKXCXTUGXL (OHPX
GQX SXVGXTC XBJHTX) NA XCXTKXGHM NUTNUTHUCV."
abc = ["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"]

#Ordena os tuplos pelo segundo argumento
def Sort_Tuple(tup):
    lst = len(tup)
    for i in range(0, lst):
        for j in range(0, lst-i-1):
            if (tup[j][1] < tup[j + 1][1]):
                temp = tup[j]
                tup[j]= tup[j + 1]
                tup[j + 1]= temp
    return tup


#Conta o número de caracteres
def conta(txt):
    count = 0
    lista = []
    for x in abc:
        y = txt.count(x)
        t = (x,y)
        lista.append(t)
    Sort_Tuple(lista)
    print(lista)
```

```
#Conta o número de vezes que uma palavra aparece retornando (palavra, int)
def contaPalavras(txt):
    count = 0
    listaPal = []
    searched = []
    txt_limpo = re.sub("[,|.|;|)|(]", " ", txt)
    words = txt_limpo.split(" ")
    for x in words:
        if(x not in searched):
            count = words.count(x)
            t=(x,count)
            listaPal.append(t)
        searched.append(x)
    Sort_Tuple(listaPal)
    print(listaPal)

#Substitui a letra x por a letra y
def subs(txt):
    dic = {}
    dic["U"]="A"
    dic["N"]="B"
    dic["M"]="C"
    dic["L"]="D"
    dic["X"]="E"
    dic["Z"]="F"
    dic["K"]="G"
    dic["Q"]="H"
    dic["H"]="I"
    dic["P"]="K"
    dic["O"]="L"
    dic["B"]="M"
    dic["C"]="N"
    dic["I"]="O"
    dic["J"]="P"
    dic["Y"]="Q"
    dic["T"]="R"
    dic["V"]="S"
    dic["G"]="T"
    dic["R"]="U"
    dic["F"]="V"
    dic["S"]="W"
    dic["A"]="Y"
    txt_limpo = ""
    for x in txt:
        isLetra=re.match("[A-Z]",x)
        if (x not in dic and isLetra):
            txt_limpo += "_"
        elif(not isLetra):
            txt_limpo += x
        else:
            txt_limpo += dic[x]
    return txt_limpo
```

```
def main():
    print("Número de Caracteres:\n")
    conta(txt)
    print("-------------------------")
    print("Número de Palavras:\n")
    contaPalavras(txt)
    print("-------------------------")
    print("Texto final:\n")
    print(subs(txt))

if __name__ == "__main__":
    main()
```

### 4.4 viginere.py

```
import re
from pprint import pprint

#Ter cuidado com os espaços

txt = "WMP BRHEMGQJ JD YSZ RWTBGQ ZA RZC QHAPMYO OJKHXEDA WLXCV QMMP DZTHWLG
 DGZMGJNYVJ XEJANX, FDX WCHS XYUWTZB YZ YQ LWQXWO CAYCZKH MT VTXZ DZECMUX. RKJJ
 ZHQTZTH ECYW PQCUD MYFJ RFLHS ZUJPYQ YCPC, WZR YSZ GNDOGQHEDTH NCYUFNOCUX WC JGZP
 XZ QONRCR, SVQ MLY LYD ULQO NUTEJRBUP. DY OFLX MYWJ OFHWP KXXE FDAP CANDOCG LO OJLNR
 F NARWP MI DKCFNPN RK RGOI XYWYWZ, FD KDSJ QKJPK, FYY VJGZPDQ BMDYD, LS ZSUTAZ DQZIC,
 LIB XPQCUFW CYJY ULYSDL LCZYW MMGWFTI. TYZ DZECMU MZJLJGZQ YSVR YSZPH QJPPJCGW
 JIDQWJO COJGZL BTGB XAZALJD MI DCCHU KCFZWDYU EJ JWPVR GCDRDNY! ZMPI ZJ WCDW DL RTIB
 YSVR GCDRDNY FDX IMZ YJR TYZ SJNPJLFC KDRXVJ, LIB KCVLFJ WSW QZU ITNRLSNO IWZH WMZNC
 TQ EHWXVLB, VLG DJ ZNEC KZYBYUD, QSFTI, JEX., MPR YSVR JLXF TQ RKJDZ NNYBBRRD
 NRXDZQVJD QHAPMYO AZAXQTVP GCZCGX JD HLOROJ, QKJPK, JEX., HZ PZDO DIXDR YSVR RLIW
 IZHCVYTX EWPZBV XPQW SVTH ZMGJNYVRHI DL JFMMSJ; DRW RFHSNZ RYSZPZNDZ FTFGB YSZW MLQC
 GPZL IPMGYJO? VT DR ND GQ TIBLF. CYJY GQ ECC HLNC TQ RKJ WPHJON RK OFH OJKHXEDA IZB
 WMCJSJMZPR YSZ ZTCGB, HCGFM D DIXDR FCZ GJDXCQIPY IWZH VJGZPDQ RGOI NNHHTZQ, TO
 FFYIMW MZ GTFWRHI OFDY OFHWP FDX WCHS VL NXHCQXP YPTFIR TQ GQMPMGWJO TDWTVRLTY; ITC
 UKT RGOQ WCONPQC YSVR FYDKDQD AOTDZJB CZQHRMGGQL OFH TOYONLI JWPTFRZYY, YSZ
 EQZJBKTFIB, ECC GFGJ-IZB, UFB-GTR, RW WJHSSZGP DKYQNPG, JEX.|XZ SQQTFC FWG ZNWY
 FFYDBDJ|ZTHW ZVLXEZB NY Y XEVRH ZA QFEPPH? DR MLN RKEZL GPZL QZJQHQJ QDNO RKFE YOQ
 JSU CVAHX JD IZBQ MLQC GPZL UCJBXHPY ED OFH NMMVXTIE TQ Y KPR DGZMGJNYVJ XAZALJD;
 EZE ZB NMMVXTIE BP ADS JLOD BCW QJPPX DL XZHC IPBPHJ DLWJCHCGNLOC GPOUHJY RKJTM
 SFCZLWX; YQI DD BP YFHZPLW QJP TFM VJGZPDQ YMPJDOGF CVAHX WW YSDQ UCJAHXD, ZJ HSVY
 VBPNE RKJ AMURPM HCTNRHSNZ RK OFH XJQW PSRUJXZ ITCHQ, LN WMP GWFWDYQ RMCBMZPLG,
 WJRTOCMXSO, EZWG-GTR, HYN., NY RKJ RGOI NRDYP. PTCZMYJC, WMP NRXDDZLQTOW TQ KDPTIE
 ITNRLSNO UFNZQ GJ AUTDNGQL CYV MZCQ RMCDYWT HCLBEHWLOCG. HYQD XYVJD YUJ JL WPXMUI
 NFRBTIE YSVR F MYFJ HYB MZ PTODDLJO ZB ZXADXTJLDQ XPRXDZQ NQ YLIPY ED OFH NVPHKFG
 VJWZAWNZI RK OFH TIBLATYSDQD UKNNC SWPNCQY OFH OZQLWPY FMLMYFYPM; GFO WT JZWFTI D
 CVAH TIRHWXZBLFEZ EJERCHS OUR BPGWJ YGVYTIAW CVAHX RMXQO ZH GZPB ODDINNPJW. NGU U.
 VJMMGJME CAUCZQVQJ CAUPMGPJYOCG HDRK ECGV ZWHHHE YQI AYLQPY. YSZ RKQNNUNYB IWZH
 WMP DLWDO FWZNQ GPOUHJY RZT KSUJ WPHJON LX OMOJCVZOD VLG DJKHYTHCV (VQ N CYYJ
 AMXSO ULYS NLLPJLV) LSLYP SQNQJPP TI FMLMYFYPM, FYY HAPMW YSDLJ DZCPX NGPUWZ HSZPEK;
 WSW HCCQ ECCVJ HMQLCZJV LMC HCJQVJO MQJ RGWM VLRYSZP KZM VJGZPDQ BCQJCVRLTYN, MLMBOD
 OUR ZA WMPH DWP YONVZ, FYY WMPI WMP BLKQDAXQET RK OFH EVQN MZARRPN PFYDDHXE."


#Dá print de todas as palavras com 3 letras e o índice destas
def contaPalavras(txt):
    dic = {}
    i = 0
    count = 0
    lista = []
    txt_limpo = re.sub("[!|?|||-|,|.|;|)|(]", " ", txt)
    words = txt_limpo.split(" ")
    parts = list(set(words))
    for x in parts:
        i += 1
        if (len(x) == 3):
            l= [m.start() for m in re.finditer(x, txt)]
            par = (x,l)
            lista.append(par)
    return lista
```

```
#Substitui o texto cifrado através da chave que lhe é dada
def decode_viginere(txt):
    chave="DFLVY"
    alphabet="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    data={}
    i=0
    for x in alphabet :
        data[x]=i # insert in map associa alfabeto a número
        i+=1
    index_key=0   # key vai ser repetida ao longo do texto com periodo len(key)
    cif=0
    letra=""
    texto_decifrado=""
    res = txt.split(' ')
    texto_size=len(res)
    for word in res :
        for x in word :
            special=re.match("[();,-.|?!\'‘´]",x) # carateres que nao sao cifrados
            if not special :
                #if x not in data : # ve se pertence ao alfabeto
                    #return text
                cif=(data[x]-data[chave[index_key]]) % len(alphabet)
                for key, value in data.items(): #letra correspondente a letra cifrada
                     if cif == value:
                        letra=key
                texto_decifrado+=letra
            else :
                texto_decifrado+=x # se for especial fica sem estar cifrado
            index_key+=1 # aumenta key
            if(index_key==len(chave)) : # periodo da key
                index_key=0
        texto_size-=1
        if texto_size>=1 :
            index_key+=1 # aumenta key
            if(index_key==len(chave)) : # perido da key
                index_key=0
            texto_decifrado+=' ' #adiciona espaço entre palavras
    print(texto_decifrado)


def main():
    print("-------------------------")
    print("Índice das palavras com 3 letras:\n")
    pprint(contaPalavras(txt))
    print("-------------------------")
    print("Texto final:\n")
    print(decode_viginere(txt))

if __name__ == "__main__":
    main()
```

# Referências

[1] Most common words in English:
https://en.wikipedia.org/wiki/Most$_c ommon_w ords_i n_E nglish$