



**Universidade do Minho**  
Escola de Engenharia

# Criptografia

**TP1**



João Miguel da Silva Alves (83624)

Paulo Jorge Alves (84480)

**MESTRADO INTEGRADO EM ENGENHARIA BIOMÉDICA**  
**INFORMÁTICA MÉDICA 2020/2021**

# Cifra de Substituição

É o tipo de cifra na qual cada letra da mensagem é substituída por outra completamente aleatória. O código Python para fazer um ataque a esta cifra encontra-se nas figuras abaixo.

```
from re import *
def substituiçao(ciphertext, letra_ingles, Lista_ingles):
    dic1= {}
    dic1['A'] = 8.2
    dic1['B'] = 1.5
    dic1['C'] = 2.8
    dic1['D'] = 4.3
    dic1['E'] = 12.7
    dic1['F'] = 2.2
    dic1['G'] = 2.0
    dic1['H'] = 6.1
    dic1['I'] = 7.0
    dic1['J'] = 0.2
    dic1['K'] = 0.8
    dic1['L'] = 4.0
    dic1['M'] = 2.4
    dic1['N'] = 6.7
    dic1['O'] = 1.5
    dic1['P'] = 1.9
    dic1['Q'] = 0.1
    dic1['R'] = 6.0
    dic1['S'] = 6.3
    dic1['T'] = 9.1
    dic1['U'] = 2.8
    dic1['V'] = 1.0
    dic1['W'] = 2.4
    dic1['X'] = 0.2
    dic1['Y'] = 2.0
    dic1['Z'] = 0.1
    lista_char = [" ", ",", ";", ".", ":", "(", ")", "-", "\n"]
    maximo = 0
    lista = lista_ingles
    lista_com_posicoes = []
    letra_maisco = ""
    for caractere in ciphertext:
        if caractere not in lista and caractere not in lista_char:
            co = ciphertext.count(caractere)
            if co > maximo:
                letra_maisco = caractere
    print(letra_maisco)
    lista_com_posicoes = posicoes(letra_maisco, ciphertext)
    subciphertext = ciphertext.replace(letra_maisco, letra_ingles)
    lista.append(letra_ingles)
    print(subciphertext)
    variavel = int(input(("Deseja continuar. Se sim prima 2 se souber a proxima correspondencia. Se nao souber prima 1. Se quiser acabar senao prima 0")))
    if variavel == 2:
        a = True
        while a == True:
            letra_cifra = input("Introduza a letra da cifra que quer decifrar: ")
            letra_ing = input("Introduza a letra inglesa para decifrar: ")
            lista.append(letra_ing)
            cripto = subs(subciphertext, letra_cifra, letra_ing, lista_com_posicoes)
            lista_com_posicoes = lista_com_posicoes + (posicoes(letra_cifra, subciphertext))
            print(cripto)
            if letra_cifra == "":
                a = False
            if letra_cifra != "":
                subciphertext = cripto
    return subciphertext
```

Figura 1 – Código para a cifra de substituição

```
def posicoes(letra, cifra):
    lista_posicoes = []
    for i in range(len(cifra)):
        if cifra[i] == letra:
            lista_posicoes.append(i)
    return lista_posicoes

def subs(ciphertext, letra_cifra, letra_inglesa, lista):
    ciphertext = list(ciphertext)
    for i in range(len(ciphertext)):
        if (i not in lista) and (ciphertext[i] == letra_cifra):
            ciphertext[i] = letra_inglesa
    ciphertext = "".join(ciphertext)
    return ciphertext

def file(f):
    with open(f, 'r') as f1:
        arquivo1 = f1.read()
    return arquivo1

def main():
    ciphertext = file('cripto3.txt')
    substituiçao(ciphertext, 's', [])
    main()
```

Figura 2 - Código para a cifra de substituição (continuação)

Na cifra de substituição, é feito um *shift* de uma letra na cifra para a correspondente letra no texto limpo. Por exemplo, se a letra V na cifra corresponder à letra S, então qualquer letra V na cifra corresponderá à letra S no texto limpo e, assim, sucessivamente.

Para decifrar esta cifra necessitamos de construir um dicionário, sendo as chaves as letras do alfabeto e os valores correspondem às suas frequências relativas na língua inglesa. De seguida, construímos uma lista com os caracteres da cifra que não iram ser cifrados. Depois fomos ao nosso criptograma, identificar a letra que mais aparecia. A partir daí, fomos testar se, de facto, a letra que mais aparecia no criptograma correspondia à letra “E”, que é a letra mais comum em textos de língua inglesa. Contudo, ao realizar essa correspondência e assumindo a partir daí outras correspondências, não obtínhamos palavras em inglês. Por esse motivo, tentamos que a letra com mais ocorrências correspondesse à letra “T”, mas usando o mesmo raciocínio anterior, não obtivemos resultados. De seguida, tentamos para a letra “A” e letra “I”, mas sem sucesso. No momento que fizemos a correspondência da letra que mais aparecia no criptograma com a letra “S”, algumas palavras, principalmente de duas letras começavam a fazer sentido no texto.

Depois, entramos num ciclo *while* em que só terminava quando o “*user*” não soubesse mais nenhuma correspondência, o que correspondia ao momento em que terminava e já tinha obtido o texto limpo. Em cada iteração fomos fazendo correspondências, uma vez que palavras começavam a formar-se. Em cada iteração é pedido ao utilizador que assuma uma correspondência de uma letra na cifra para uma letra do alfabeto inglês e essa substituição é feita com recurso à função `subs()`, que retorna um criptograma com as alterações feitas. A função `subs()` irá para cada caractere dado como input presente na cifra e que não esteja no conjunto dos caracteres especiais que não são cifrados, substituir pela letra do alfabeto inglês também dado como input. Com recurso, também, à função “*posições*”, é possível obter uma lista das posições em que foram feitas a substituição, de forma a não ocorrer mais nenhuma substituição nas próximas iterações. No fim de encontrar correspondência para todas as letras na cifra, o utilizador não dá nenhum caractere como input e o ciclo *while* acaba e é retornado o texto limpo. É também dado como output na tela um dicionário que contém como chaves as várias letras na cifra e como valores as letras do alfabeto inglês que cada uma corresponde.

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.

{'V': 'S', 'I': 'O', 'G': 'T', 'B': 'M', 'X': 'E', 'H': 'I', 'U': 'A', 'O': 'L', 'J': 'P', 'T': 'R', 'S': 'W', 'N': 'B', 'R': 'U', 'C': 'N', 'K': 'G', 'Q': 'H', 'Z': 'F', 'M': 'C', 'F': 'V', 'L': 'D', 'A': 'Y', 'P': 'K', 'Y': 'Q', 'D': 'D', 'E': 'E'}

Figura 3 -Texto limpo obtido para a cifra de substituição, e dicionário com todas as correspondências.

## Cifra de Affine

A cifra de affine é um tipo de cifra de substituição, em que cada letra de um alfabeto é mapeada para seu equivalente numérico, criptografada usando uma função matemática simples  $(y=ax+b)\bmod 26$  e convertida de volta para uma letra.

O código Python para o ataque a esta cifra está representado nas fotos seguintes.

```
def file(f):
    with open(f, 'r') as f1:
        arquivo1=f1.read()
    return arquivo1

#função que calcula a inversa de um número que, no final de contas, será a inversa do valor de a dado como argumento
def inversa(numero):
    for i in range(0,27):
        if ((numero*i) % 26)==1:# para um numero ser o inverso de outra: o produto dos dois mod 26 deverá ser igual a 1
            return i

    sys.exit("Nao existe inversa para %d" %numero)

#função que fará o brute force attack. Para cada valor de a(todos os números impares de 0 a 26 excluindo o 13) poderá haver 26 valores de b
def affine(ciphertext):
    ncaracteres = [" ", ",", ";", ".", ":", "(", "-", "\n", "!"]
    for a in range(0,26):
        if (a%2 != 0) and (a != 13):
            for b in range(0,26):
                continuar = int(input("Deseja continuar prima 1 senao prima 0: "))
                plaintext=""
                if continuar == 0:
                    exit();
                if continuar==1:
                    inversa1 = inversa(a)
                    for caractere in ciphertext:
                        if caractere not in ncaracteres:
                            num = ord(caractere)
                            car_plaintext = chr((((inversa1*(num -65 - b)) % 26)+65) #usar a expressão que está associada à cifra de
                            plaintext += car_plaintext
                        else:
                            plaintext += caractere

                    print(plaintext)

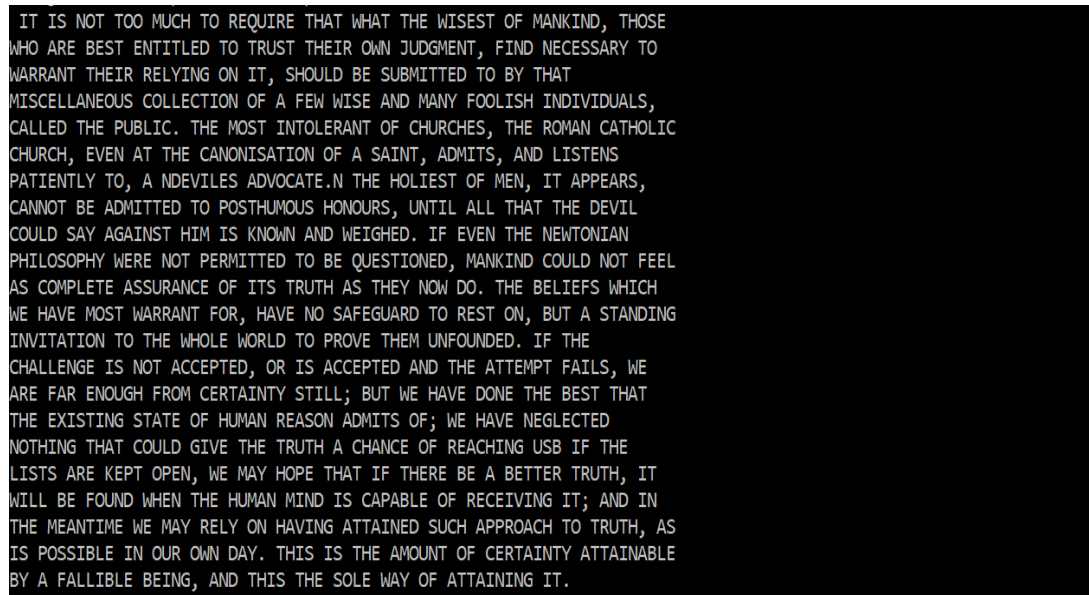
def main():
    ciphertext = file('cripto2.txt')
    affine(ciphertext)
main()
```

Figura 4 – Código para a cifra de affine

Para a cifra de affine, realizamos um “*brute force attack*”. Na expressão matemática usada na cifra de affine ( $y=a^{-1}(x-b) \bmod 26$ ) existem 12 valores possíveis para a variável “a”, uma vez que existem 12 números possíveis que sejam co-primos com o número 26 e existem 26 números possíveis para a variável “b”, desde o número 0 ao 25. Portanto, há 312 ( $12 \cdot 26$ ) chaves possíveis para decifrar um criptograma que foi cifrado pela cifra de affine.

Para efetuar um ataque deste tipo, implementamos a função “*affine*” que recebe como argumento o criptograma cifrado pela cifra de affine. De seguida, através de dois ciclos “for”, iteramos para cada valor de “a”, de forma a determinar a sua inversa. Por fim, com o “b” e a inversa de “a” obtivemos o texto limpo através da expressão escrita acima.

O “*brute force attack*” realizado, numa primeira perspetiva, poderá parecer que demorará imenso tempo, no entanto, assumindo que uma pessoa carrega no *enter* de dois em dois segundos, isso significa que durante um minuto terá carregado 30 vezes e, por isso, terá obtido 30 textos limpos. Em pouco mais de dez minutos, qualquer pessoa é capaz de ter carregado as 312 vezes no *enter* e obtido os 312 textos limpos possíveis. Após 138 iterações, a qual corresponde a um valor 11 para a variável “a” e um valor “8” para a variável b, verificou-se que o texto mostrado era o pretendido (figura 5).



```
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 NUN'S ADVOCATE. IN THE HOLIEST OF MEN, IT APPEARS,
CANNOT BE ADMITTED TO POSTHUMOUS 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, MANKIND COULD NOT FEEL
AS 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, WE
ARE FAR ENOUGH FROM CERTAINTY STILL; BUT WE HAVE DONE THE BEST THAT
THE 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.
```

Figura 5 – Texto limpo para a cifra de affine

## Cifra de Vigenère

A cifra de Vigenère é um método de criptografia que utiliza uma série de diferentes "cifras de César" com base nas letras de uma palavra-chave.

O código Python para o ataque a esta cifra está representado nas fotos abaixo.

```
#função que descobre os fatores de um número qu é passado como argumento
def print_factors(x):
    list_factors = []
    for i in range(1, x + 1):
        if x % i == 0:
            list_factors.append(i)
    return list_factors

#função que determina o tamanho da chave da cifra
def key_lenght(ciphertext):
    ncaracteres = [" ", ",", ";", ".", "-", "(", ")", "\n", "?", "!", ""]
    dic = {}
    for caractere in ciphertext: #com este ciclo substitui-se na cifra, todos os caracteres presentes na lista ncaracteres por espaço
        if caractere in ncaracteres:
            ciphertext = ciphertext.replace(caractere, " ")
    for i in range(0, len(ciphertext)): #com este ciclo calcular e colocar num dicionário as posições de palavras formadas por três l
        subseq = ciphertext[i:i+3]
        if subseq in dic:
            continue
        for j in range(i+3, len(ciphertext)-3):
            if subseq == ciphertext[j:j+3]:
                if subseq not in dic:
                    dic[subseq] = [i, j]
                else:
                    dic[subseq].append(j)
    dic_fatores = {}
    for element in dic: #com este ciclo colocar num dicionário onde as chaves são os fatores das posições do dicionário anterior e o
        for i in range(len(dic[element])):
            x = print_factors(dic[element][i])
            for j in x:
                if j not in dic_fatores:
                    dic_fatores[j] = 1
                else:
                    dic_fatores[j] += 1
    key_lenght = 5 # o tamanho da chave foi obtido através da leitura e análise do dicionário em que o fator 5 aparecia muitas vezes
    return key_lenght
```

Figura 6 – Código para a cifra de vigenère

Para o criptograma cifrado com a cifra de vigenère, implementamos a função designada por “key\_lenght”. Primeiramente, substitui no criptograma todos os caracteres que se encontram na lista “ncaracteres” por espaços, uma vez que no ciclo “for” a seguir tentar-se-á encontrar palavras formadas por três letras que se repetem ao longo do criptograma e essas mesmas palavras não podem conter letras dessa lista. De seguida, é colocado num dicionário onde as chaves são as palavras de três letras e os valores correspondem a uma lista que contém todas posições onde essa palavra se encontra no criptograma.

De seguida é implementado um dicionário onde as chaves serão cada fator das posições encontradas antes e o valor será o número de vez que esse fator ocorre. Para calcular os fatores recorre-se à função “print\_factors”. Após uma análise ao dicionário obtido, verificou-se que o tamanho da chave seria 5, visto que o fator 5 aparecia várias vezes sendo um possível valor para o tamanho da chave. Mais tarde, após tentativas e erro e por análise do output das funções seguintes que retornavam um texto limpo, verificou-se que o tamanho da chave teria de ser 5, pois com esse tamanho conseguíamos encontrar várias palavras em inglês a serem formadas.

```

def viginere(key_lenght,ciphertext):
    ncaracteres = [" ", ".", ":", ";", ",", "(", ")", "-", "\n", "?", "!", ""]
    for i in range(0, key_lenght):#obter a letra que mais aparece no subcriptograma, o qual corresponde ao criptograma original, co
        subseq = ciphertext[i:len(ciphertext):key_lenght]
        letra_maisco = ""
        maximo = 0
        for caractere in subseq:
            if caractere not in ncaracteres:
                co = ciphertext.count(caractere)
                if co > maximo:
                    letra_maisco = caractere
        print(letra_maisco)
        shift = []#lista que contém as letras correspondentes à chave
        letra_shift = chr((((ord(letra_maisco)-65) - (ord("A")-65))*26) +65)#a letra que mais aparece por análise do ciclo anterior
        shift.append(letra_shift)
        print(shift)
        index = 0
        subciphertext = ver_cifra(ciphertext,index,shift,key_lenght)
        print(subciphertext)
        variavel = int(input("Se souber uma correspondencia prima 2. Senao prima 1 e continue: "))
        if variavel == 1:
            continue
        if variavel == 2:#assumimos que o utilizador sabe a correspondência
            a = True
            while a ==True:
                correspondencia = input("Diga qual é o shift e que posicao: ")
                shift.append(correspondencia)
                index = index + 1
                subciphertext = ver_cifra(subciphertext,index,shift,key_lenght)#função a qual retorna um criptograma onde as letras
                print(subciphertext)
                print(shift)
                if correspondencia == "":
                    a = false
                if correspondencia != "":
                    continue
            print(shift)
            return shift

```

Figura 7 – Código para a cifra de vigenère

Com a implementação da função “viginère”, obtínhamos um subcriptograma que correspondia ao criptograma original, contudo com as letras nas posições 5 em 5.

De seguida, fomos descobrir qual a letra que mais ocorria nesse subcriptograma e fizemo-la corresponder à letra “A” do alfabeto inglês. Esta correspondência resulta no facto, de suspeitarmos que a primeira palavra formada por “WMP” do criptograma original correspondia à palavra “THE”. Como para obter a letra T a partir da letra W, é necessário que a primeira letra da chave seja um “D”. Como a letra que mais aparecia no subcriptograma era um “D”, logo para que a primeira letra da chave fosse um “D”, teríamos de associá-la à letra “A” no texto limpo.

A seguir, é pedido ao utilizador que ao olhar para o criptograma agora alterado possa saber uma correspondência possível. Essa correspondência irá constituir a letra seguinte da chave da cifra e com ajuda da função “ver\_cifra”, todas as letras nessas posições irão sofrer esse Shift, obtendo, assim, um novo criptograma com essas alterações. Após algumas tentativas, verificou-se que a chave da cifra era [D, F, L, V, Y].



```

def ver_cifra(ciphertext,index,lista,key_lenght):
    ncaracteres = [" ",".",",",";",":","'","(",")","-","\\n","?","!"]
    shift = lista
    ciphertext = list(ciphertext)
    for i in range(index,len(ciphertext),key_lenght):
        if ciphertext[i] not in ncaracteres:
            ciphertext[i] = chr(((ord(ciphertext[i])-65)-(ord(shift[index])-65))%26)+65)
    ciphertext = "".join(ciphertext)
    return ciphertext

'''esta função foi implementada, uma vez que quando foi descoberta a chave, esta estava a ser aplicada ao primeiro parágrafo,
contudo não estava a ser aplicada ao segundo e, por isso, obtíamos um texto em inglês para o primeiro mas não para o segundo
parágrafo. Portanto, aplicamos a chave agora na forma ['Y','D','F','L','V'], uma vez que à última letra do primeiro parágrafo
foi aplicado o shift correspondente à letra V. Com isto obtemos o texto inglês para o segundo parágrafo.'''
def decode2(ciphertext):
    ncaracteres = [" ",".",",",";",":","'","(",")","-","\\n","?","!"]
    shift = ['Y','D','F','L','V']
    ciphertext = list(ciphertext)
    for i in range(len(shift)):
        for j in range(i,len(ciphertext),5):
            if ciphertext[j] not in ncaracteres:
                ciphertext[j] = chr((((ord(ciphertext[j])-65)-(ord(shift[i])-65))%26)+65)
    ciphertext = "".join(ciphertext)
    return ciphertext

def main():
    ciphertext = file('cripto1.txt')
    ciphertext2 = file('cripto4.txt')
    key_lenght(ciphertext)
    texto_limpo= viginere(key_lenght(ciphertext),ciphertext)
    print(texto_limpo)
    texto_limpo2 = decode2(ciphertext2)
    print(texto_limpo2)
main()

```

Figura 8 – Código para a cifra de vigenère

A função “decode2” foi implementada, uma vez que quando foi descoberta a chave, esta estava a ser aplicada ao primeiro parágrafo, contudo não estava a ser aplicada ao segundo e, por isso, obtíamos um texto em inglês para o primeiro, mas não para o segundo parágrafo. Provavelmente, não estava a reconhecer a passagem de linha através do caractere “\\n”. Portanto, aplicamos a chave agora na forma ['Y','D','F','L','V'], uma vez que à última letra do primeiro parágrafo foi aplicado o *shift* correspondente à letra V. Com isto obtemos o texto inglês para o segundo parágrafo.

```

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.-
ME

```

Figura 9 – Primeiro parágrafo do texto limpo



UNLIKE ALL WILD CANIDA EYEVER 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.

*Figura 10 - Segundo parágrafo do texto limpo*