Trabalho de Segurança Computacional
Cifra de Vigenère

Douglas Vieira Alcantara – 18/0031333

Cifrador e De-Cifrador

Foi pedido a criação de um programa python, que realize a cifração e decifração utilizando o método da Cifra de Vigenère.

Cifrador

Para a criação do programa de cifração, foram criadas as seguintes funções de suporte:

- Função que remove acentos do texto a ser cifrado

```
3
4   #FUNÇÃO QUE TRATA A MENSAGEM. RETIRANDO ACENTOS
5   def tratar_mensagem(mensagem_bruta):
6       mensagem_tratada = unidecode(mensagem_bruta).upper() #transforma todos os char com acentos em char sem acentos
7       return(mensagem_tratada)
```

- Função que recebe a chave a ser utilizada e cria a sua versão cíclica, que terá o mesmo comprimento do texto a ser cifrado.

```
#FUNÇÃO QUE PEGA A CHAVE E GERA A CHAVE CÍCLICA, IGUALANDO AO COMPRIMENTO DA MENSAGEM
def gerar_chave_ciclica(string, chave):
    key = list(chave)
    if len(string) == len(chave):
        return(chave)
    else:
        tamanho_loop = ((len(string) // len(chave)) + 1) #define quantas vezes é necessario repetir a chave, até alcançar o comprimento
        chave_ciclica_cheia = ''
        for i in range(tamanho_loop):
            chave_ciclica_cheia = (chave_ciclica_cheia + chave) # repete a chave quantas vezes foram necessárias
        chave_ciclica_final = chave_ciclica_cheia[0:len(string)] # remove o excesso de char, para ficar do tamanho da mensagem. Substr
    return(chave_ciclica_final.upper()) #retorna a chave ciclica criada, maiuscula
```

- Função que cifra a mensagem

```
#FUNÇÃO QUE CIFRA A MENSAGEM
def cifrar(mensagem_para_cifrar, chave, letra_para_indice, indice_para_letra, alfabeto):
    mensagem_cifrada = ''
    i = 0
    number = 0
    for letra in mensagem_para_cifrar: # loop de letra em letra da mensagem
            number = (letra_para_indice[letra] + letra_para_indice[chave[i]]) % len(alfabeto) # busca numero do indice da letra corresp
            mensagem_cifrada += indice_para_letra[number] # appenda na mensagem cifrada, a letra correspontende ao numero do indice
            i += 1
    return mensagem_cifrada
```

E temos a função main, onde essas funções de suporte são utilizadas:

```python
#MAIN
def main():
    mensagem = "No final das contas, não são os anos de sua vida que contam. É a vida em seus anos."
    chave = "chave_poderosa".upper()
    alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 !@#$%¨&*()_+=-,.".upper()


    #Gera indices que serão usados como base, na hora de trocar os char. Com essa abordagem de índice,
    # consigo cifrar qualquer char, desde que esteja previsto na variavel alfabeto.
    letra_para_indice = dict(zip(alfabeto, range(len(alfabeto))))
    indice_para_letra = dict(zip(range(len(alfabeto)), alfabeto))

    #trata a mensagem, retirando acentos
    mensagem_tratada = tratar_mensagem(mensagem)
    print("\nmensagem tratada:", mensagem_tratada,"\n")

    #gera a chave ciclica que será usada, igualando ao tamanho da mensagem
    chave_ciclica = gerar_chave_ciclica(mensagem_tratada,chave)
    print("chave cilcica   :", chave_ciclica,"\n")

    #cifra a mensagem
    mensagem_cifrada = cifrar(mensagem_tratada, chave_ciclica, letra_para_indice, indice_para_letra, alfabeto)
    print("Mensagem Cifrada:", mensagem_cifrada,"\n")
main()
```

Vale ressaltar que se utilizou um método utilizado de índices flexíveis. Ao invés de usar a representação numérica dos caracteres como índice, foi utilizado a criação de um novo índice, que permite a utilização de qualquer caractere, desde que esteja declarado no alfabeto.

Decifrador

Segue a função de decifração:

```python
#FUNÇÃO QUE DE-CIFRA A MENSAGEM
def decifrar(mensagem_para_descifrar, chave, letra_para_indice, indice_para_letra, alfabeto):
    mensagem_decriptada = ""
    i = 0
    for letra in mensagem_para_descifrar:
        number = (letra_para_indice[letra] - letra_para_indice[chave[i]]) % len(alfabeto)
        mensagem_decriptada += indice_para_letra[number]
        i += 1
    return mensagem_decriptada
```

A função de decifração utiliza o mesmo método da função de cifração, simplesmente invertendo o sinal da operação dos índices.

O programa main de decifração utiliza as mesmas funções de suporte listadas no item anterior:

```python
#MAIN
def main():
    mensagem = "PV 1MHPZ#HR7BCQUTVW(,2DSA7SO@VSEEH47#HV- UC&V4H_,5XIAQ7NVHMU$,,O#ZZRS GT #IO8-DR67R"
    chave = "chave_poderosa".upper()
    alfabeto = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 !@#$%¨&*()_+=-,.".upper()

    #Gera indices que serão usados como base, na hora de trocar os char. Com essa abordagem de índice, consigo cifrar qualquer char
    letra_para_indice = dict(zip(alfabeto, range(len(alfabeto))))
    indice_para_letra = dict(zip(range(len(alfabeto)), alfabeto))

    #trata a mensagem, retirando acentos
    mensagem_tratada = tratar_mensagem(mensagem)
    print("\nmensagem tratada:", mensagem_tratada,"\n")

    #gera a chave ciclica que será usada, igualando ao tamanho da mensagem
    chave_ciclica = gerar_chave_ciclica(mensagem_tratada,chave)
    print("chave cilcica   :", chave_ciclica,"\n")

    #de-cifra a mensagem
    mensagem_descifrada = decifrar(mensagem_tratada, chave_ciclica, letra_para_indice, indice_para_letra, alfabeto)
    print("Mensagem De-Cifrada:", mensagem_descifrada,"\n")

main()
```

Ataque

Para o ataque a cifra, foi utilizado o método de comparação estatística entre as distribuições das letras, comparando com a distribuição padrão do idioma.

Para essa comparação, o método estatístico usado, foi o chi quadrado, para medir a proximidade entre duas distribuições.

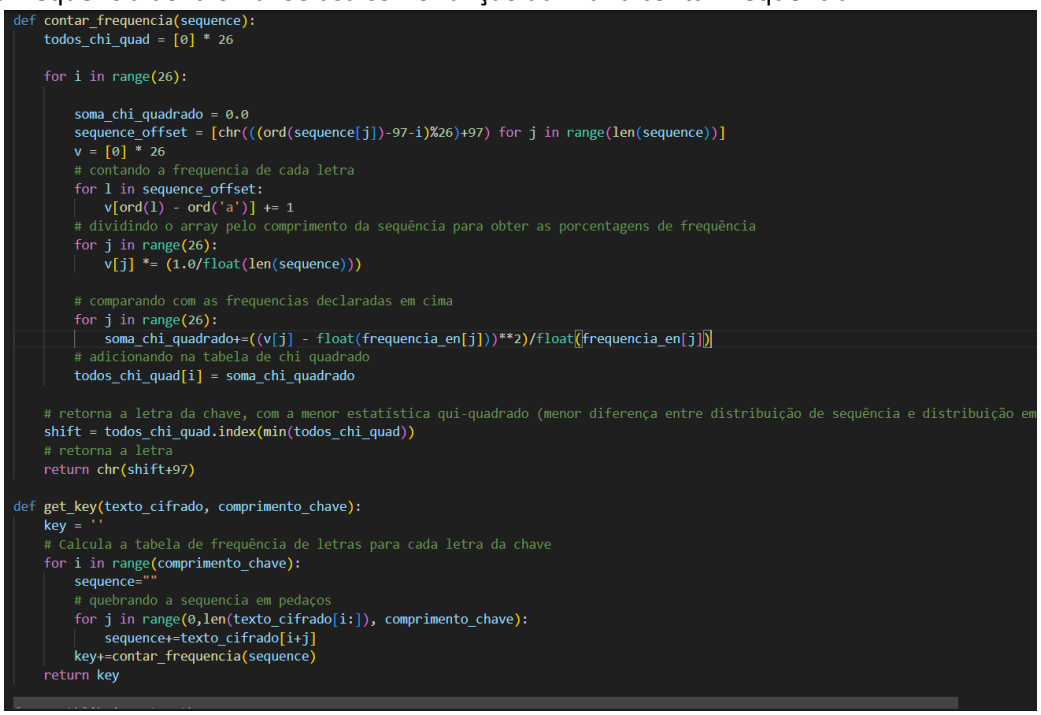
No ataque foi utilizado a seguinte main:

```
from pathlib import Path
def main():
    #lendo dados arquivo texto
    texto_cifrado_bruto = Path('desafio1.txt').read_text()
    print("texto bruto:\n", texto_cifrado_bruto,"\n")

    # Mnatendo somente char que sejam numeros ou letras
    texto_cifrado = ''
    for x in texto_cifrado_bruto:
        if x.isalpha():
            texto_cifrado += x.lower()
    print("texto tratado:\n", texto_cifrado,"\n")

    # Inferindo o tamanho da chave
    comprimento_chave=get_comprimento_chave(texto_cifrado)
    print("Comprimento provavel da chave {}".format(comprimento_chave))

    # Inferindo a chave em si
    key = get_key(texto_cifrado, comprimento_chave)
    print("chave: {}".format(key))

main()
```

Seguindo os seguintes passos:

- Primeiramente se utilizou de uma limpeza para remover char que não fossem alfa numéricos. Para ser compatível com o dicionário.
- Quebra a sequencia de várias formas formas diferentes, e faz o calculo do maior índice de coincidência entre o texto, que indica o comprimento da chave para aquele caso
    -

```
# Retorna o comprimento da chave com o índice médio de coincidência mais alto
def get_comprimento_chave(texto_cifrado):
    tabela_auxiliar=[]
    # Divide o texto cifrado em sequências com base no comprimento da chave adivinhada de 0 até o comprimento máximo da chave es
    for comprimento_tentativa in range(tamanho_tentativa):
        soma_auxiliar=0.0
        media_auxilia=0.0
        for i in range(comprimento_tentativa):
            sequence=""
            # quebra em sequencias
            for j in range(0, len(texto_cifrado[i:]), comprimento_tentativa):
                sequence += texto_cifrado[i+j]
            soma_auxiliar+=indice_auxiliar(sequence)
        # evitar divisão por zero
        if not comprimento_tentativa==0:
            media_auxilia=soma_auxiliar/comprimento_tentativa
        tabela_auxiliar.append(media_auxilia)

        # retorna o índice do índice de coincidência mais alto (comprimento de chave mais provável)
    melhor_tentativa = tabela_auxiliar.index(sorted(tabela_auxiliar, reverse = True)[0])
    segunda_melhor_tentativa = tabela_auxiliar.index(sorted(tabela_auxiliar, reverse = True)[1])
    # Como este programa pode retornar que a chave é  duas vezes ela mesma, ou três vezes ela mesma, manter valores menores
    # A distribuição de frequência para a chave "chave" vs "chavechave" seria muito parecida
    if melhor_tentativa % segunda_melhor_tentativa == 0:
        return segunda_melhor_tentativa
    else:
        return melhor_tentativa
```

- Após chegar no provável comprimento da chave, é calculado a tabela de afinidade entre as distribuições, onde se consegue mapear a frequência de cada letra e compará-la com a frequência do idioma. Se usa como função auxiliar a contar frequência.
    - 

```
def contar_frequencia(sequence):
    todos_chi_quad = [0] * 26

    for i in range(26):

        soma_chi_quadrado = 0.0
        sequence_offset = [chr(((ord(sequence[j])-97-i)%26)+97) for j in range(len(sequence))]
        v = [0] * 26
        # contando a frequencia de cada letra
        for l in sequence_offset:
            v[ord(l) - ord('a')] += 1
        # dividindo o array pelo comprimento da sequência para obter as porcentagens de frequência
        for j in range(26):
            v[j] *= (1.0/float(len(sequence)))

        # comparando com as frequencias declaradas em cima
        for j in range(26):
            soma_chi_quadrado+=((v[j] - float(frequencia_en[j]))**2)/float(frequencia_en[j])
        # adicionando na tabela de chi quadrado
        todos_chi_quad[i] = soma_chi_quadrado

    # retorna a letra da chave, com a menor estatística qui-quadrado (menor diferença entre distribuição de sequência e distribuição em
    shift = todos_chi_quad.index(min(todos_chi_quad))
    # retorna a letra
    return chr(shift+97)

def get_key(texto_cifrado, comprimento_chave):
    key = ''
    # Calcula a tabela de frequência de letras para cada letra da chave
    for i in range(comprimento_chave):
        sequence=""
        # quebrando a sequencia em pedaços
        for j in range(0,len(texto_cifrado[i:]), comprimento_chave):
            sequence+=texto_cifrado[i+j]
        key+=contar_frequencia(sequence)
    return key
```

Utilizando este método, após a execução, se obtém os seguintes resultados:

```
2 1 quebrar vigenere en.py"
texto bruto:
 rvgllakieg tye tirtucatzoe.  whvnvvei i
winu mpsecf xronieg giid abfuk thv mfuty; wyenvvvr ik ij a drmg,
drzzqly eomemsei in dy jouc; wyenvvvr i wied mpsvlf znmollnkarzlp
palszng seworv cfffzn narvhfusvs, rnd srzngznx up khv rerr ff emeiy
flnvrac i deek; aed ejpvcirlcy wyeeevvr dy hppfs gvt jucy ae upgei
haed ff mv, tyat zt ieqliies r skroeg dorrl grieczplv tf prvvvnt de
wrod dvliseiatvlp stvpginx ieto khv stievt, aed detyouicrlcy keotkieg
geoglv's hrtj ofw--tyen, z atcolnk it yixh tzmv to xek to jer as jofn
aj i tan.  khzs ij mp susskitltv foi pzstfl rnd sacl.  wzty a
pyicosfpyicrl wlolrzsh tako tyrfws yidsecf lpoe hzs snoid; i huzetcy
kakv tf thv syip.  khvre zs eotyieg slrgrijieg ie tyis.  zf khep blt
keen it, rldosk acl mvn zn tyezr dvgiee, jode tzmv or ftyer, thvrijh
merp nvarcy khe jade fvecinxs kowrrus tye fcern nity mv.

texto tratado:
 rvgllakiegtyetirtucatzoewhvnvveiiwinumpsecfxronieggiidabfukthvmfutywyenvvvrikijadrmgdrzzqlyeomemseiindyjoucwyenvvvriwiedmpsvlfznmollnkarzlppalszngs
eworvcfffznnarvhfusvsrndsrzngznxupkhvrerrffemeiyflnvracideekaedejpvcirlcywyeeevvrdyhppfsgvtjucyaeupgeihaedffmvtyatztieqliiesrskroegdorrlgrieczplvtfp
rvvvntdewroddvliseiatvlpstvpginxietokhvstievtaeddetyouicrlcykeotkieggeoglvshrtjofwtyenzatcolnkityixhtzmvtoxektojerasjofnajitankhzsijmpsusskitltvfoip
zstflrndsaclwztyapyicosfpyicrlwlolrzshtakotyrfwsyidsecflpoehzssnoidihuzetcykakvtfthvsyipkhvrezseotyiegslrgrijiegietyiszfkhepbltkeenitrldoskaclmvnznt
yezrdvgieejodetzmvorftyerthvrijhmerpnvarcykhejadefvecinxskowrrustyefcernnitymv

Comprimento provavel da chave 15
chave: araraararaarara
PS D:\One Drive\OneDrive\UNB\segurança computacional>
```

Com essa chave, podemos utilizar os programas de decifração, para tentar decifrar o texto:

```
v2_2_quebrar_vigenere_pt.py    v2_1_quebrar_vigenere_en.py ●    v2_3_vigenere_decifrar_desafio1.py ×
v2_3_vigenere_decifrar_desafio1.py > ...
from v0_0_vigenere_funcoes import *

#MAIN
def main():
    mensagem    = Path('desafio1.txt').read_text()
    chave       = "araraararaarara".upper()
    alfabeto    = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".upper()


    #Gera indices que serão usados como base, na hora de trocar os char. Com essa abordagem de índice, consigo cifrar qualquer
    letra_para_indice = dict(zip(alfabeto, range(len(alfabeto))))
    indice_para_letra = dict(zip(range(len(alfabeto)), alfabeto))

    #trata a mensagem, retirando acentos
    mensagem_tratada = tratar_mensagem(mensagem)
    mensagem_tratada = ''.join(x.upper() for x in mensagem_tratada if x.isalpha())
    print("\nmensagem tratada:", mensagem_tratada,"\n")

    #gera a chave ciclica que será usada, igualando ao tamanho da mensagem
    chave_ciclica = gerar_chave_ciclica(mensagem_tratada,chave)
    print("chave cilcica   :", chave_ciclica,"\n")

    #de-cifra a mensagem
    mensagem_descifrada = decifrar(mensagem_tratada, chave_ciclica, letra_para_indice, indice_para_letra, alfabeto)
    print("Mensagem De-Cifrada:", mensagem_descifrada,"\n")

main()
```

Onde obtemos o seguinte resultado:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

RARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARAR
AARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARARAARAR

Mensagem De-Cifrada: REGULATINGTHECIRCULATIONWHENEVERIFINDMYSELFGROWINGGRIMABOUTTHEMOUTHWHENEVERITISADAMPDRIZZLYNOVEMBERINMYSOULWHENEVERIFINDMYSELFI
NVOLUNTARILYPAUSINGBEFORECOFFINWAREHOUSESANDBRINGINGUPTHEREAROFEVERYFUNERALIMEETANDESPECIALLYWHENEVERMYHYPOSGETSUCHANUPPERHANDOFMETHATITREQUIRESASTR
ONGMORALPRINCIPLETOPREVENTMEFROMDELIBERATELYSTEPPINGINTOTHESTREETANDMETHODICALLYKNOCKINGPEOPLESHATSOFFTHENIACCOUNTITHIGHTIMETOGETTOSEAASSOONASICANTH
ISISMYSUBSTITUTEFORPISTOLANDBALLWITHAPHILOSOPHICALFLOURISHCATOTHROWSHIMSELFUPONHISSWORDIQUIETLYTAKETOTHESHIPTHEREISNOTHINGSURPRISINGINTHISIFTHEYBUTK
NEWITALMOSTALLMENINTHEIRDEGREESOMETIMEOROTHERCHERISHVERYNEARLYTHESAMEFEELINGSTOWARDSTHEOCEANWITHME
```

E podemos ver que agora o resultado faz sentido, se separarmos as palavras corretamente:

Mensagem De-Cifrada: REGULATING THE CIRCULATION WHENEVER I FIND MYSELF GROWING GRIM ABOUT THE MOUTH WHENEVER......