Trabalho 01 de Segurança Computacional

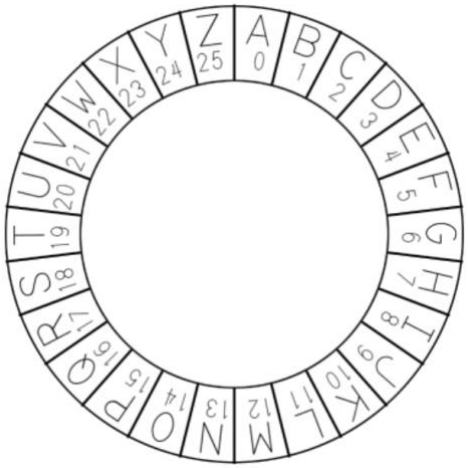
Cifra de Vigenère

Fernando de Alcantara - 190125586

André Ivan – 190084197

Como funciona a Cifra de Vigenère?

Pense em um círculo com as letras do alfabeto e seus respectivos índices.



Com isso em mente, o acesso a cada letra do círculo pode ser feito por meio de seus índices. E, melhorando um pouco, podemos dizer que o acesso a qualquer letra do círculo é feito utilizando-se do módulo de 26, isto é:

Índice = (valor inteiro) % 26, assim garantimos que nunca sairemos do círculo.

Agora, iremos criptografar a frase: “O rato roeu a roupa do rei de Roma” utilizando como chave a palavra: ‘Travesso’. Para facilitar a compreensão, desconsideraremos os espaços e utilizaremos letras maiúsculas.

Utilizando o círculo, faremos o seguinte processo:

Repetindo a chave até o tamanho da frase, e depois somando os respectivos índices, teremos a nossa frase criptografada:

Mensagem: ORATOROEUAROUPADOREIDEROMA
Chave: TRAVESSOTRAVESSOTRAVESSOTR
Cifrada: HIAOSJGSNRRJYHSRHIEDHWJCFR

Onde a cifra de cada letra da mensagem se deu somando os índices da respectiva letra da mensagem com o da letra da chave. Exemplo:

A primeira letra da mensagem ‘O’ de índice 14 somada com a primeira letra da chave ‘T’ de índice 19 é: (14 + 19)%26 -> 33%26 = 7 que é o índice da letra H que é a primeira letra da nossa mensagem cifrada. E repetindo o processo para as letras seguintes chegamos na mensagem cifrada resultante. Também podemos ver que, com a chave sendo conhecida, o processo para decifrar também é tranquilo. Sendo necessário apenas fazer a subtração do índice da chave com a da mensagem cifrada. Exemplo:

Subtraindo o índice da letra ‘H’ que é a primeira letra da mensagem cifrada com o índice de ‘T’ que é a primeira letra da chave, teremos: (7 - 19)%26 -> (-12)%26 que em módulo resulta em 14 que é o índice da letra ‘O’ que é a primeira letra da mensagem original. Basta repetir o processo para as próximas letras que voltaremos para a mensagem original.

1 – Parte I: Cifrador/Decifrador

No código que foi feito em Python, mostramos a aplicação dessa lógica para criarmos um cifrador e um decifrador da cifra de Vigenère. Note que o alfabeto utilizado contém apenas letras minúsculas. `alfabeto = 'abcdefghijklmnopqrstuvwxyz'`

Para cifrar a mensagem utilizamos a função encriptar:

```
29  def encriptar(mensagem, chave):
30      mensagem_em_lista = [mensagem[i: i + len(chave)] for i in range(0, len(mensagem), len(chave))]
31
32      encriptado = ''
33      for pedaço in mensagem_em_lista:
34          i = 0
35          for letra in pedaço:
36              indice = (alfabeto.index(letra.lower()) + alfabeto.index(chave[i].lower())) % len(alfabeto)
37              encriptado += alfabeto[indice] if letra.islower() else alfabeto[indice].upper()
38              i += 1
39
40      return encriptado
```

Na linha 29, recebemos a mensagem que será cifrada e a chave usada para cifrar. Na linha 30 'quebramos' a mensagem em pedaços do tamanho da chave (equivalente a repetir a chave até o tamanho da mensagem). A mensagem cifrada será armazenada na variável 'encriptado'.

No for-loop mais externo é onde iteramos por cada pedaço da mensagem e inicializamos a variável 'i' para iterar sobre a chave (note que ela é zerada sempre que o loop recomeça).

No for-loop mais interno é onde iteramos pelas letras da mensagem cifrando-as, somando o índice da letra da mensagem com o índice da letra da chave e concatenando a letra do índice resultante na variável 'encriptado'.

Note que algumas condições foram adicionadas para preservar o estado de maiúscula ou minúscula da letra (linha 37).

Agora para decifrar a mensagem utilizamos a função decriptar:

```
43  def decriptar(mensagem, chave):
44      mensagem_em_lista = [mensagem[i: i + len(chave)] for i in range(0, len(mensagem), len(chave))]
45
46      decriptado = ''
47      for pedaço in mensagem_em_lista:
48          i = 0
49          for letra in pedaço:
50              indice = (alfabeto.index(letra.lower()) - alfabeto.index(chave[i].lower())) % len(alfabeto)
51              decriptado += alfabeto[indice] if letra.islower() else alfabeto[indice].upper()
52              i += 1
53
54      return decriptado
```

Note que a única diferença entre a função de encriptar e decriptar ocorre na linha 50, onde o operador de soma é trocado pelo operador de subtração não sendo necessário nenhuma outra modificação para a função de decifrar a mensagem. O porque disso ser possível já foi discutido na primeira pagina deste trabalho.

Parte II: ataque de recuperação de senha por análise de frequência

O processo de recuperar a chave utilizada para cifrar a mensagem foi dividido em duas etapas onde a primeira se da em achar um provável tamanho da chave e a segunda se da em achar a possível chave utilizando análise de frequência.

```
150 def atacar(mensagem):
151     tamanho_chave = pegar_tamanho_chave(mensagem)
152     if tamanho_chave == 0:
153         return None
154     else:
155         print('Chaves encontradas (em ordem de possivel melhor chave): ', *tamanho_chave)
156         tamanho_escolhido = int(input('Escolha uma começando do índice 0: ', ))%len(tamanho_chave)
157     chave = pegar_chave(mensagem, tamanho_chave[tamanho_escolhido])
158
159     return chave
```

ETAPA 1

ETAPA 2

Note que o que é retornado da função pegar_tamanho_chave(mensagem) é uma lista de possíveis tamanho da chave ordenada da mais provável para a menos provável, dando a possibilidade para o usuário escolher qual o tamanho ele quer utilizar para prosseguir.

```
82  def pegar_tamanho_chave(mensagem, tamanho: int = 20):
83      tabela_ic = []
84      tamanho_maximo_chave = tamanho
85
86      for tamanho_suposto in range(tamanho_maximo_chave):
87          soma_ic = 0.0
88
89          for i in range(tamanho_suposto):
90              sequencia = ''
91
92              for j in range(0, len(mensagem[i:]), tamanho_suposto):
93                  sequencia += mensagem[i+j]
94
95              if (len(sequencia) > 1):
96                  soma_ic += pegar_indice_coincidencia(sequencia)
97
98          ic_medio = soma_ic / tamanho_suposto if not tamanho_suposto == 0 else 0.0
99          tabela_ic.append(ic_medio)
100
101     tabela_ic_ordenada = sorted(tabela_ic, reverse=True)
102
103     melhores_suposições = list(map(lambda valor: tabela_ic.index(valor), tabela_ic_ordenada))
104     melhores_suposições = [suposição for suposição in list(dict.fromkeys(melhores_suposições)) if suposição ≠ 0]
105
106     return melhores_suposições
```

Para a primeira etapa (achar o provável tamanho da chave) foi utilizado um conceito conhecido como índice de coincidência. O índice de coincidência pode ser usado para estimar o tamanho da chave. O que ocorre aqui é o seguinte, o índice de coincidência com a maior média é um bom candidato a ser o tamanho da chave. Como isso foi feito nesse código:

Nas duas primeiras linhas (83 e 84), inicializamos a tabela que armazenará todos índices de coincidência e também inicializamos uma variável com o tamanho máximo da chave que iremos supor, nesse caso 20.

Para exemplificar o que está ocorrendo no for-loop, tomemos o texto cifrado “RSTCS JLSLR SLFEL GWLFI ISIKR MGL” e o tamanho_suposto = 3. O que ocorrerá é o seguinte:

Para cada valor de ‘i’ na iteração do tamanho_suposto, o texto será dividido em 3 conjuntos (sequências):

Conjunto 1 : RCLRFGFSRL Conjunto 2: SSSSEWIIM Conjunto 3: TJLLLLIKG

Para cada conjunto desse, calcularemos o índice de coincidência:

```
70  def pegar_indice_coincidencia(sequencia):
71      N = float(len(sequencia))
72      soma_frequencia = 0.0
73
74      for letra in alfabeto:
75          soma_frequencia += sequencia.count(letra) * (sequencia.count(letra) - 1)
76
77      indice_coincidencia = soma_frequencia/ (N*(N-1))
78
79      return indice_coincidencia
```

Após calcularmos o índice de coincidência de cada conjunto note que o seu resultado é somado na variável soma_ic e obtemos o seu valor médio dividindo a soma de todos os índices de coincidência obtidos pelo valor do tamanho_suposto. Note que esse processo discutido para tamanho_suposto = 3, é feito para cada valor de 0 a tamanho_maximo_chave (nesse caso 20). Ou seja, no final obteremos 20 índices de coincidência médio, como sua posição na tabela_ic corresponde ao tamanho da chave,

poderemos usar isso para efetuar o restante da lógica, apenas ordenando a tabela do maior índice médio para o menor e pegando os possíveis tamanhos de chave respectivos mantendo a ordem. Depois retornamos esses valores e deixamos o usuário escolher o valor que achar melhor, note que também poderíamos simplesmente retornar a posição do maior índice de coincidência médio, pois em tese ele seria o tamanho da chave mais provável mas optamos por deixar para o usuário escolher.

Com isso, podemos prosseguir para a etapa 2:

```python
def pegar_chave(mensagem, tamanho_chave):
    chave = ''

    for i in range(tamanho_chave):
        sequencia = ''

        for j in range(0, len(mensagem[i:]), tamanho_chave):
            sequencia += mensagem[i+j]

        chave += analise_frequencia(sequencia)

    return chave
```

Na etapa 2, o processo de divisão da mensagem no for-loop é semelhante com o da etapa anterior em que dividimos a mensagem em 3 conjuntos, a diferença é que, supondo que o tamanho_suposto = 3 fosse o melhor candidato, logo tamanho_chave = 3. O que temos é que para cada iteração de tamanho_chave, utilizando analise de frequência decidimos um bom candidato(letra) para preencher aquele espaço da chave onde o usuário também pode interagir para selecionar a letra. Veja:

```python
def analise_frequencia(sequencia):
    todos_qui_quadrados = [0] * 26

    for i in range(26):
        sequencia_deslocada = [alfabeto[(alfabeto.index(letra.lower())-i)%26] for letra in sequencia]

        frequencias_ocorrencias_letras = [float(sequencia_deslocada.count(letra))/float(len(sequencia)) for letra in alfabeto]

        soma_qui_quadrado = 0.0
        frequencia = frequencias_portugues if utilizar_frequencias_portugues else frequencias_ingles
        for j in range(26):
            soma_qui_quadrado+=((frequencias_ocorrencias_letras[j] - float(frequencia[j]))**2)/float(frequencia[j])

        todos_qui_quadrados[i] = soma_qui_quadrado

    letra_com_menor_qui_quadrado = todos_qui_quadrados.index(min(todos_qui_quadrados))

    for qui_quadrado in sorted(todos_qui_quadrados)[:5]:
        print(f'({alfabeto[todos_qui_quadrados.index(qui_quadrado)]}:{qui_quadrado:.2f})', end = ' ')
    letra = input('Escolha uma letra ou aperte enter: ')
    return letra if len(letra) == 1 and letra in alfabeto else alfabeto[letra_com_menor_qui_quadrado]
```

O que ocorre na função de analise de frequência é que tendo uma sequência de caracteres, deslocamos essa sequência uma vez mais para a esquerda. Ex:

A sequência RCLRFGFSRL deslocada uma vez para a esquerda: QBKQEFRERQK
Duas vezes: PAJPDEDQPJ Três vezes: OZIOCDCPOI e assim por diante.
Após deslocar a sequência calculamos a frequência de cada letra (linha 116) para utilizarmos para o calculo do qui quadrado. Depois de calcularmos o qui-quadrado guardamos no array para após deslocar por todas as letras do alfabeto e ter todos os 26 qui quadrados, pegarmos o de menor valor e mostrar os 5 menores para o usuário e deixa-lo escolher qual usar. Note que o menor qui quadrado corresponde a letra mais provável para aquela posição da chave. Por fim concatenamos todos as letras da chave e tentamos decriptar com a função decriptar(mensagem, chave).