Trabalho 1 - Cifra de Vigenère
Segurança Computacional - Turma 01
Gabriel Borges - 202006401
Maycon Fabio - 200059742

1. Introdução

O trabalho é dividido em duas partes, onde a primeira contém o cifrador/decifrador e a segunda o ataque. O usuário poderá digitar uma mensagem e escolher uma chave para gerar um criptograma segundo a cifra de Vigenère. Sabendo a chave, é possível decifrar o criptograma e obter a mensagem original. Sem a chave, é preciso realizar um ataque ao criptograma para tentar descobrir a mensagem original, utilizando a técnica da análise de frequências da língua correspondente. O programa é capaz de realizar ataques à criptogramas originados de mensagens escritas em português ou inglês.

2. Implementação

2.1 Cifrador

Antes de cifrar a mensagem, é necessário formatá-la. Para isso, os acentos foram desconsiderados, todas as letras foram passadas para minúsculo, os números e espaços foram retirados. A mensagem formatada consiste apenas de caracteres pertencentes a variável alfabeto.

```python
def remover_acentos(input_str):
    acentos = {'á':'a', 'é':'e', 'í':'i', 'ó':'o', 'ú':'u', 'ã':'a', 'õ':'o', 'â':'a',
    'ê':'e', 'ô':'o', 'à':'a', 'ç':'c'}
    return ''.join(acentos[i] if i in acentos else i for i in input_str)

def format_str(msg):
    format_str = remover_acentos(msg.lower())
    return re.sub(r'[^a-z]', '', format_str)
```

Também foi feita uma função que coloca de volta os caracteres e números que foram retirados. Apenas acentos e maiúsculos não podem ser recuperados da mensagem original.

```python
def deformat_str(original, formated):
    index = 0
    result = ''
    for l in original.lower():
        if l.isalpha():
            result += formated[index]
            index += 1
        else:
            result += l
    return result
```

Para que a cifra possa ser aplicada, é preciso concatenar a chave até que seu tamanho seja maior ou igual ao da mensagem. Depois, o índice (0 a 25) de cada caractere da mensagem e chave são recuperados para formar o criptograma. Aquele obtido através da mensagem (c_index) é somado ao da chave (k_index). O resto da divisão dessa soma pelo tamanho do alfabeto irá resultar no índice correspondente ao caractere cifrado. Aplicando essa matemática em toda a mensagem, o criptograma é gerado.

```python
def cifrador(msg: str, key):
    format_msg = format_str(msg)
    key_ext = key
    while (len(format_msg) > len(key_ext)):
        key_ext += key

    criptograma = ''
    for i, c in enumerate(format_msg):
        c_index = alfabeto.find(c)
        k_index = alfabeto.find(key_ext[i])
        criptograma += alfabeto[(c_index + k_index) % len(alfabeto)]

    return deformat_str(msg, criptograma)
```

2.2 Decifrador

A decifração consiste no processo contrário. São recuperados os índices de cada caractere do criptograma (c_index) e da chave (k_index). Ao invés de somá-los como ocorre na cifração, basta subtrair esses índices. O resto da divisão dessa subtração pelo tamanho do alfabeto irá resultar no índice correspondente ao caractere da mensagem original. Aplicando essa matemática em todo o criptograma, a mensagem é recuperada.

```python
def decifrador(criptograma, key):
    format_cript = format_str(criptograma)
    key_ext = key
    while (len(format_cript) > len(key_ext)):
        key_ext += key

    msg = ''
    for i, c in enumerate(format_cript):
        c_index = alfabeto.find(c)
        k_index = alfabeto.find(key_ext[i])
        msg += alfabeto[(c_index - k_index) % len(alfabeto)]

    return deformat_str(criptograma, msg)
```

2.3 Ataque

A primeira parte do ataque consiste em estimar o tamanho da chave. A função find_key_size é responsável por fazer essa estimativa. Seu funcionamento é o seguinte, primeiro cada letra do criptograma é transformada em índices do alfabeto (0 a 25) e são armazenadas na variável cript_num. Depois, é criada uma matriz (shift_matrix) com todos os cript_num deslocados n vezes à direita, com n variando de 1 ao tamanho do criptograma. Cada linha da matriz possui uma frequência de coincidências em relação a cript_num. Toda vez que um caractere da linha bate com o caractere do criptograma, é acrescido 1 à frequência. Isso pode ser usado para definir o tamanho da mensagem, avaliando o espaçamento em que ocorrem os maiores números. Após isso, usamos a média e o desvio padrão dessa frequência para definir um fator de corte (lsup) que usamos para identificar onde aparecem as maiores coincidências. Medimos as distâncias entre cada ponto alto subsequente porque essa distância pode significar o tamanho da chave usada para criptografar essa mensagem. Verificamos qual é a distância mais frequente e assumimos ela como o tamanho da chave.

```python
def find_key_size(criptograma):
    format_cript = format_str(criptograma)
    cript_num = [alfatonum[c] for c in format_cript]
    shift_matrix = []
    for i in range(1, len(format_cript)):
        shift_matrix.append([-1]*i + cript_num[:-i])

    rows_freq = [0]
    for cript in shift_matrix:
        freq = 0
        for n, letter in enumerate(cript):
            if letter == cript_num[n]:
                freq += 1
        rows_freq.append(freq)

    rows_freq = np.array(rows_freq)
    desvio = np.std(rows_freq)
    media = np.mean(rows_freq)
    cutoff = desvio * cutoff_factor
    lsup = media + cutoff
    lsup_outliers = np.where(rows_freq > lsup)[0]

    len_freqs = np.diff(lsup_outliers).tolist() # diff between each subsequent lsup_outliers
    most_freq_len = max(set(len_freqs), key=len_freqs.count)

    return most_freq_len
```

Agora com o tamanho da chave encontrado iremos definir cada letra correspondente àquela posição da chave. Montamos uma frequência das letras com base numa parte do criptograma, que é definida pegando cada letra daquela posição da chave espaçadas pelo tamanho da chave. Então, se estamos tentando descobrir a primeira letra de uma chave de tamanho 7, olhamos as posições 0, 7, 14, 21… Com essa frequência em mãos, tentamos descobrir quantos “shifts” temos que dar no alfabeto para que a frequência de letras do idioma original seja a mais similar em relação a frequência que acabamos de calcular. Após encontrarmos esse valor, pegamos a quantidade de “shifts” que executamos para encontrar a letra da respectiva posição da chave (com “a” sendo 0 “shifts”, e assim por diante). Feito isso, encontramos uma chave provável.

```python
def find_key(criptograma, idioma = 'i'):
    format_cript = format_str(criptograma)
    stats_lang = stats_port if idioma == 'p' else stats_ing
    stats_lang = [stats_lang[l] for l in alfabeto]

    key_size = find_key_size(format_cript)
    key = ''
    for i in range(0, key_size):
        subcript = format_cript[i::key_size]
        stats_sub = calc_stats(subcript)

        max_accuracy, max_j = 0, 0
        for j in range(len(alfabeto)):
            shifted_alf = alfabeto[j:] + alfabeto[:j]
            shifted_stats = [stats_sub[l] for l in shifted_alf]
            result = sum([l * s for l, s in zip(stats_lang, shifted_stats)])

            if (result > max_accuracy):
                max_accuracy, max_j = result, j
        key += alfabeto[max_j]

    return key
```

3. Referências

https://pt.wikipedia.org/wiki/Frequência_de_letras
https://youtu.be/LaWp_Kq0cKs?si=_xYYFUHJ3lVvTVqE
https://medium.com/@lucapqg/como-detectar-e-tratar-outliers-com-python-ca2cf088c160