

Vigenere

1) Implementação

Descrição da Cifração e Decifração

→ Tanto a Cifração quanto a Decifração Vigenere de mensagens estão implementadas na classe `VigenereCypher` do arquivo `encrypt_decrypt.py`

→ O código apresentado no arquivo `encrypt_decrypt.py` é uma implementação da Cifra de Vigenere. A classe `VigenereCypher` possui três métodos principais: `mapping`, `encrypt` e `decrypt`.

1. Método `mapping`:

Este método cria e retorna uma matriz de mapeamento, onde cada linha representa um alfabeto rotacionado em uma posição.

```
def mapping(self, alfabeto) -> list:
    mapping_matrix = [[chr((linha + i) % len(alfabeto) + ord("a"))
                       for i in range(len(alfabeto))]
                      for linha in range(len(alfabeto))]

    return mapping_matrix
```

2. Método `encrypt`:

O método `encrypt` recebe como parâmetros a mensagem original, a chave, a matriz de mapeamento e o alfabeto. Ele cria uma variável chamada `mensagem_cifrada`, que armazenará a mensagem criptografada. Usando um loop `while`, o método percorre a mensagem original e verifica se cada caractere está no alfabeto. Se estiver, ele encontra a linha e a coluna correspondentes na matriz de mapeamento usando os valores ordinais dos caracteres e adiciona o caractere mapeado à `mensagem_cifrada`. Se não estiver no alfabeto, o caractere é adicionado diretamente à `mensagem_cifrada`. Ao final, a `mensagem_cifrada` e seu comprimento são retornados como uma tupla.

```
def encrypt(self, mensagem_original, chave, mapping_matrix, alfabeto):
    mensagem_cifrada = ""

    indice = 0
    indice_alfabeto = 0
    while indice < mensagem_original[1]:
        if mensagem_original[0][indice] in alfabeto:
            coluna = ord(mensagem_original[0][indice]) - ord("a")
            linha = ord(chave[indice_alfabeto % len(chave)]) - ord("a")

            mensagem_cifrada += mapping_matrix[linha][coluna]
            indice_alfabeto += 1
        else:
            mensagem_cifrada += mensagem_original[0][indice]
            indice += 1

    mensagem_cifrada = (mensagem_cifrada, len(mensagem_cifrada))
    return mensagem_cifrada
```

3. Método `decrypt`:

O método `decrypt` é usado para decifrar a mensagem criptografada, revertendo o processo de criptografia. Ele recebe como parâmetros a `mensagem_cifrada`, a chave e o alfabeto. A variável `mensagem_decifrada` armazena a mensagem decifrada durante o processo. O método usa um loop `while` para percorrer a `mensagem_cifrada` e verifica se cada caractere está no alfabeto. Se estiver, ele calcula a coluna do caractere decifrado usando os valores ordinais dos caracteres e, em seguida, usa a função `chr()` para converter o valor ordinal na letra correspondente, que é adicionada à `mensagem_decifrada`. Se o caractere não estiver no alfabeto, ele é adicionado diretamente à `mensagem_decifrada`. Ao final, a `mensagem_decifrada` e seu comprimento são retornados como uma tupla.

```
def decrypt(self, mensagem_cifrada, chave, alfabeto):
    mensagem_decifrada = ""
```

```

indice = 0
indice_alfabeto = 0
while indice < mensagem_cifrada[1]:
    if mensagem_cifrada[0][indice] in alfabeto:
        if ord(mensagem_cifrada[0][indice]) >= ord(chave[0][indice_alfabeto % chave[1]]):
            coluna = ord(mensagem_cifrada[0][indice]) - ord(chave[0][indice_alfabeto % chave[1]])
        else:
            antes_da_rotacao = ord("z") - ord(chave[0][indice_alfabeto % chave[1]]) + 1
            depois_da_rotacao = ord(mensagem_cifrada[0][indice]) - ord("a")

            coluna = antes_da_rotacao + depois_da_rotacao

        mensagem_decifrada += chr(coluna + ord("a"))
        indice_alfabeto += 1
    else:
        mensagem_decifrada += mensagem_cifrada[0][indice]
        indice += 1

mensagem_decifrada = (mensagem_decifrada, len(mensagem_decifrada))
return mensagem_decifrada

```

Tamanho da Chave

→ O tamanho da chave é essencial para conseguirmos quebrar a criptografia Vigenere.

→ O processo para se encontrar possíveis tamanhos de chave está implementado no arquivo [key_size.py](#) e se tem como base o método de Kasiski, um método que visa encontrar possíveis tamanhos de chave utilizados em uma mensagem cifrada com a cifra de Vigenere. É baseado na análise de frequência de trigramas (conjuntos de três caracteres) ou n-gramas e na identificação de padrões repetidos.

1. A função `trigrama_busca` recebe a mensagem cifrada tratada e retorna uma lista de trigramas e suas respectivas posições na mensagem.

```

def trigrama_busca(mensagem_cifrada_tratada: str):
    trigramas = []

    for i in range(len(mensagem_cifrada_tratada)-2):
        trigrama = mensagem_cifrada_tratada[i:i+3]
        trigramas.append((trigrama, i))

    return trigramas

```

2. A função `histograma_trigrama` recebe a lista de trigramas e retorna um dicionário contendo a frequência e os índices das ocorrências de cada trigrama na mensagem cifrada.

```

def histograma_trigrama(trigramas: list):
    histograma = {}

    for trigrama, occur in trigramas:
        if trigrama in histograma:
            histograma[trigrama]["freq"] += 1
            histograma[trigrama]["ocurr"].append(occur)
        else:
            histograma[trigrama] = {"freq": 1, "ocurr": [occur]}

    return histograma

```

3. A função `tratamento` remove todos os caracteres não alfabéticos da mensagem cifrada e retorna a mensagem tratada, já que nesse trabalho em específico não estão sendo cifrados caracteres não alfabéticos.

```

def tratamento(mensagem_cifrada):
    mensagem_cifrada_tratada = re.sub(r"[^a-z]", "", mensagem_cifrada)

    return mensagem_cifrada_tratada

```

4. A função `dvalue_calculate` recebe um dicionário de trigramas filtrados pelo número de ocorrências e calcula a diferença entre as ocorrências consecutivas de cada trigrama. As diferenças são armazenadas em um novo dicionário.

```
def dvalue_calculate(histograma_trigrama_filtrado):
    dvalues = {}
    for trigrama in histograma_trigrama_filtrado:
        dvalues[trigrama] = []
        for i in range(len(histograma_trigrama_filtrado[trigrama]["ocurr"]) - 1):
            dvalues[trigrama].append(histograma_trigrama_filtrado[trigrama]["ocurr"][i+1] - histograma_trigrama_filtrado[trigrama]["ocurr"][i])
    return dvalues
```

5. As funções `mdc` e `mdc_lista` são utilizadas para calcular o máximo divisor comum (MDC) entre dois números e uma lista de números, respectivamente.

```
def mdc(a, b):
    while b:
        a, b = b, a % b
    return a

def mdc_lista(lista):
    resultado = lista[0]
    for elemento in lista[1:]:
        resultado = mdc(resultado, elemento)
    return resultado
```

6. A função `key_size_probabilities` é a função principal que reúne todos os recursos acima e realiza as seguintes etapas:
- Trata a mensagem cifrada, removendo caracteres não alfabéticos.
 - Obtém o histograma de trigramas
 - Descarta os trigramas com frequência menor que o valor especificado.
 - Calcula os dvalues (diferenças entre ocorrências consecutivas) para os trigramas restantes.
 - Calcula o MDC para cada lista de dvalues e armazena a contagem de cada MDC em um dicionário chamado `size_score`.
 - Ordena o dicionário `size_score` em ordem decrescente de frequência e salva os resultados em um arquivo chamado "KeySize/SizeScore.txt".

```
def key_size_probabilities(mensagem_cifrada, filtro_ocorrencias):
    mensagem_cifrada_tratada = tratamento(mensagem_cifrada)

    filtro_ocorrencias = 2

    histograma_trig = histograma_trigrama(trigrama_busca(mensagem_cifrada_tratada))
    histograma_trig = sorted(histograma_trig.items(), key=lambda x: x[1]["freq"], reverse=True)
    histograma_trigrama_filtrado = {trigrama: {"freq": dicionario["freq"], "ocurr": dicionario["ocurr"]}} for
        trigrama, dicionario in histograma_trig if
            dicionario["freq"] > filtro_ocorrencias

    dvalues = dvalue_calculate(histograma_trigrama_filtrado)

    size_score = {}
    divisor_list = [mdc_lista(dvalues[lista]) for lista in dvalues]

    for divisor in divisor_list:
        if divisor in size_score:
            size_score[divisor] += 1
        else:
            size_score[divisor] = 1

    size_score = sorted(size_score.items(), key=lambda x: x[1], reverse=True)

    with open("KeySize/SizeScore.txt", "w", encoding="utf-8") as file:
        for divisor in size_score:
            file.write(f"{divisor[0]}: {divisor[1] * 100 / len(divisor_list):.3f}%\n")
```

O resultado final é um arquivo chamado "KeySize/SizeScore.txt" contendo os possíveis tamanhos de chave e suas respectivas probabilidades, com base no teste de Kasiski.

Descoberta da Chave

→ Mas por que realizar todo esse trabalho para identificar possíveis tamanhos de chave?

Descobrimos o tamanho n da chave, podemos separar a mensagem cifrada em grupos de caracteres que foram cifrados pelo mesmo caractere da chave. E o que isso significa? Significa que os caracteres de mesmo grupo terão uma frequência relativa de letras bem parecida com a frequência relativa de letras do idioma em que a mensagem foi cifrada. Dessa forma, comparando essa distribuição com as de vários idiomas, não só podemos descobrir o idioma da mensagem original como também a chave da mensagem cifrada, assim conseguindo quebrar essa criptografia.

→ Assim chegamos na implementação da última parte, que é realizar de forma automática uma parte dos processos citados acima

→ O código apresentado no arquivo `key_by_size.py` implementa a geração de distribuições de cada grupo de caracteres cifrados pela mesma letra da chave, utilizando o tamanho da chave como parâmetro. No final o código gera um arquivo contendo as distribuições de frequência das letras de cada grupo. Esse arquivo se encontra integrado com o arquivo Power BI [Dashboards/dashboard de distribuicao.pbix](#) para que seja possível realizar a análise de distribuição de uma forma mais visualmente amigável.

→ A implementação pode ser descrita nos seguintes passos:

1. Função `batches`: Esta função recebe a mensagem cifrada e o tamanho da chave como argumentos e divide a mensagem em grupos/batches, onde cada grupo/batch contém caracteres cifrados pela mesma letra da chave. Os grupos são armazenados em uma lista chamada `batch_list`.

```
def batches(mensagem_cifrada, key_size):
    batch_list = []
    for i in range(key_size):
        aux_group = []
        for k in range(i, len(mensagem_cifrada), key_size):
            aux_group.append(mensagem_cifrada[k])

        batch_list.append(aux_group)

    return batch_list
```

2. Função `distribution`: Esta função calcula a distribuição de frequência dos caracteres na mensagem cifrada. Inicialmente, ela cria um dicionário chamado `dist` com chaves para cada letra do alfabeto e valores iniciados com 0. A função então percorre a mensagem cifrada e incrementa a contagem de cada caractere. Após isso, a função divide a contagem de cada caractere pelo tamanho da mensagem para obter a distribuição de frequência.

```
def distribution(mensagem_cifrada):
    dist = {x: [0] for x in "abcdefghijklmnopqrstuvwxyz"}
    for i in mensagem_cifrada:
        dist[i][0] += 1

    tam = len(mensagem_cifrada)
    for i in dist:
        dist[i][0] = dist[i][0]/tam

    return dist
```

3. Função `write_csv`: Esta função recebe um DataFrame do Pandas chamado `df_final` e o tamanho da chave como argumentos. Ela escreve o conteúdo do DataFrame em um arquivo CSV chamado "distribution.csv" na pasta "Distributions".

```
def write_csv(df_final, key_size):
    df_final.to_csv(r"Distributions/distribution.csv", sep=",", header=True, index=False, decimal=".")
```

4. Função `pbi_distributions`: Esta é a função principal que executa todos os passos acima mencionados. Primeiro, ela trata a mensagem cifrada removendo caracteres não alfabéticos. Em seguida, ela cria grupos de caracteres usando a função `batches`. Depois disso, ela calcula a distribuição de frequência para cada grupo usando a função `distribution`. A função então cria DataFrames do Pandas para cada distribuição e concatena todos os DataFrames em um único DataFrame chamado `df_final`. Por fim, a função `write_csv` é chamada para escrever o DataFrame final em um arquivo CSV. Esse DataFrame possui as colunas rf (Relative Frequency), letter (Letra), Distribution (Que é basicamente a numeração do grupo), para facilitar o processo de ETL no Power BI

```
def pbi_distributions(mensagem_cifrada, key_size):
    mensagem_cifrada_tratada = re.sub(r"^[a-z]", r"", mensagem_cifrada)

    groups = batches(mensagem_cifrada_tratada, key_size)

    dis_list = [distribution("".join(group)) for group in groups]

    dis_df_list = [pd.DataFrame(dis).T for dis in dis_list]
    for k, df in enumerate(dis_df_list):
        df.columns = ["rf"]
        df["letter"] = df.index
        df["distribution"] = k+1
    df_final = pd.concat(dis_df_list)

    write_csv(df_final, key_size)
```

app.py

- Esse código é basicamente o script que integra todas as funcionalidades em um programa único de linha de comando
- É recomendado a leitura do readme do [repositório git](#) para compreender melhor as instruções de utilização, mas de certa forma a execução é bem intuitiva e explicada pelo próprio programa

2) Desafios

Desafio 1

- O primeiro passo é descobrirmos possíveis tamanhos para a chave utilizada nessa cifragem utilizando a opção 2 do programa principal app.py:

```
_____ Vigenere Devthumos Tool _____
1) Encriptar
2) Decriptar
3) Probabilidades de Tamanhos de Chave
4) Distribuicao dos Agrupamentos em Power BI
5) Sair

Escolha uma Opcao: 3
```

- Filtramos quais trigramas serão utilizados para a o método de Kasisk através da escolha do parâmetro "maior que". Nesse caso escolhi 2 a fim de eliminar algumas ocorrências de trigramas que talvez sejam coincidências.

```
_____ Vigenere Devthumos Tool _____
_____ Menu 03 _____

Nao Esqueca de Inserir a Mensagem Cifrada a Ser Analisada Pelo Algoritmo no Arquivo "Mensagens/Mensagem_Cifrada.txt"
Numero de Ocorrencias de Trigramas Maior Que: 2
Probabilidades Encontradas Com Sucesso!
Encontre as Probabilidades de Tamanhos de Chave no Arquivo "KeySize/SizeScore.txt"
```

```

app.py x SizeScore.txt x encrypt_decrypt.py x
1 1: 66.667%
2 5: 20.000%
3 10: 6.667%
4 50: 6.667%
5

```

→ O segundo é escolhermos um possível tamanho de chave para começarmos a análise de distribuição. Nesse caso presumi que o Professor não escolheria uma chave de tamanho 1 e fui direto para o tamanho 5.

→ O terceiro passo é gerarmos distribuições de frequência de letras para compararmos com os dos idiomas inglês e português, assim como também descobriremos a chave.

```

_____ Vigenere Devthumos Tool _____
_____ Menu 04 _____

Nao Esqueca de Inserir a Mensagem Cifrada a Ser Analisada Pelo Algoritmo no Arquivo "Mensagens/Mensagem_Cifrada.txt"
    Possivel Tamanho de Chave: 5
Distribuicoes Encontradas Com Sucesso!
Analise as Distribuicoes Visualmente Atraves do PowerBI no Arquivo "Dashboards/dashboard_de_distribuicao.pbix"

```

→ O Quarto passo é descobrir qual é o idioma. A criptografia Vigenere não está alterando a ordem ou tamanho das palavras, bem como não está afetando caracteres não alfabéticos. Dessa forma é bem intuitivo identificar o idioma de cada mensagem.

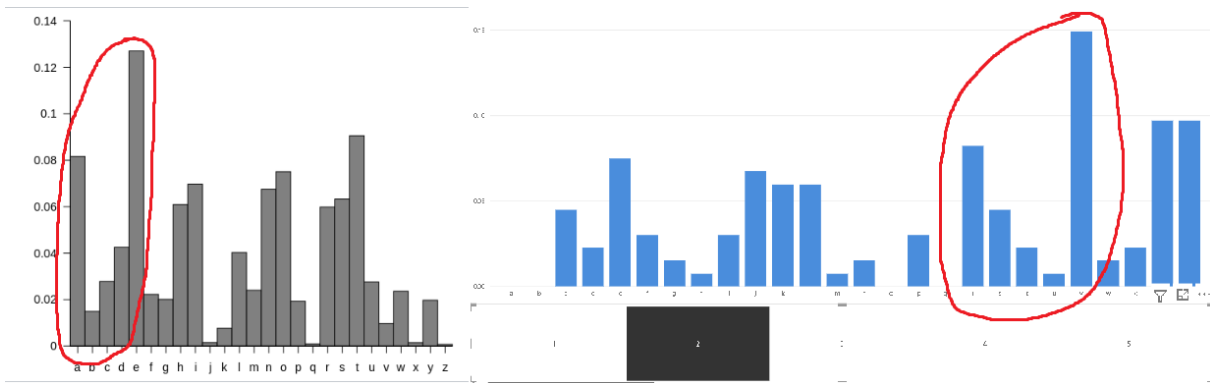
A primeira mensagem é em inglês e a segunda mensagem é em português.

→ O Quinto passo e último para decifrar a mensagem desafio 1 é comparar a distribuição de frequência de letras em inglês e descobriremos a chave numérica de César, já que podemos trabalhar com cada grupo como se fosse esse tipo de Cifra.

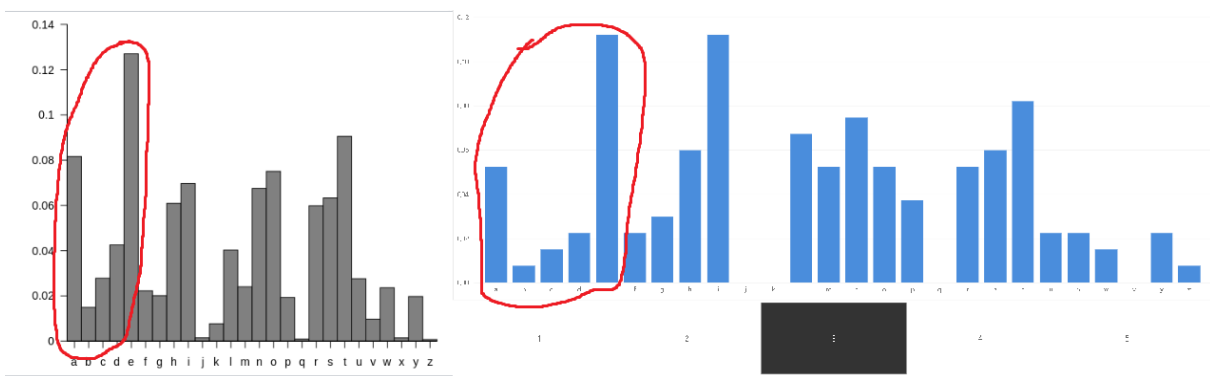
- O primeiro caractere da Chave é "a", é só analisarmos os picos de distribuição e encaixar na distribuição das letras em inglês



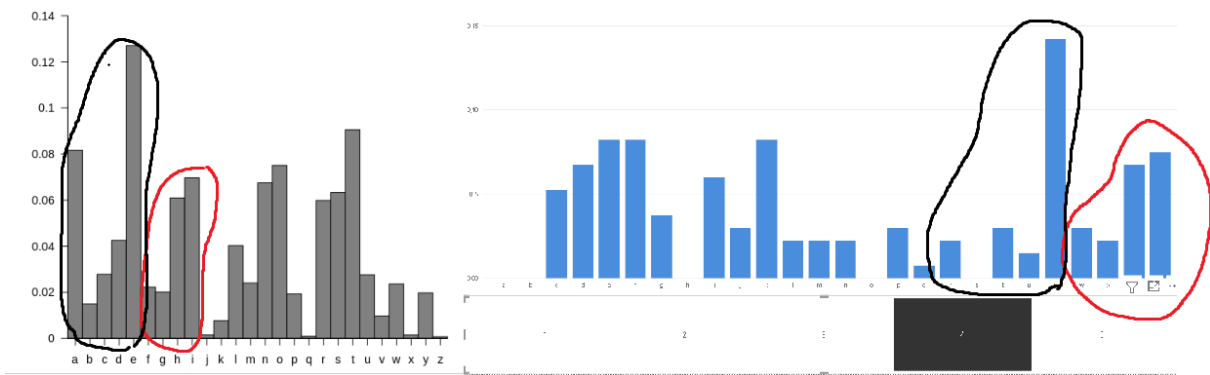
- O segundo caractere da Chave é "r"



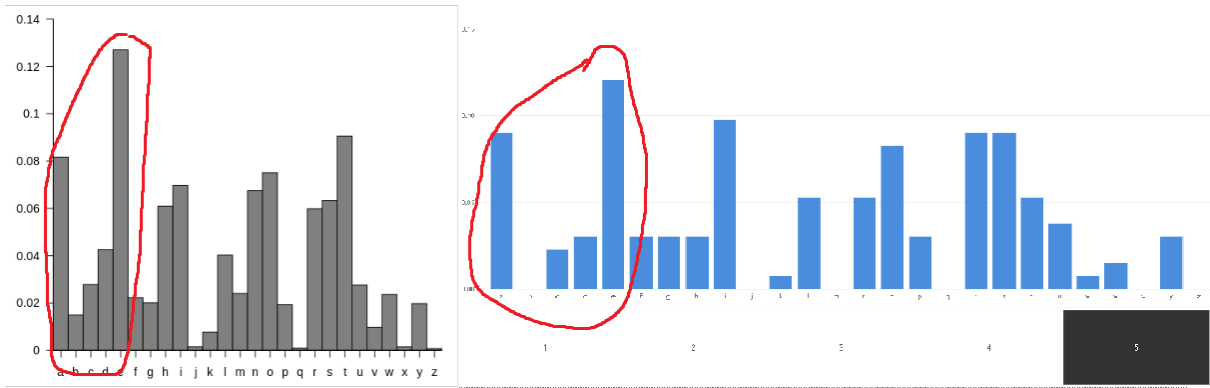
- O terceiro caractere da Chave é "a"



- O quarto caractere da Chave é "r"



- O quinto caractere da Chave é "a"



→ O sexto passo é decifrar a mensagem cifrada com a chave que encontramos

```

Vigenere Devthumos Tool
1) Encriptar
2) Decriptar
3) Probabilidades de Tamanhos de Chave
4) Distribuicao dos Agrupamentos em Power BI
5) Sair

Escolha uma Opcao: 2

Vigenere Devthumos Tool
Menu 02

Nao Esqueca de Inserir a Mensagem Cifrada a Ser Decifrada no Arquivo "Mensagens/Mensagem_Cifrada.txt"
Chave: arara
Mensagem Decifrada Com Sucesso!
Encontre a Mensagem Decifrada no Arquivo "Mensagens/Mensagem_Decifrada.txt"

```

regulating the circulation. whenever i
find myself growing grim about the mouth; whenever it is a damp,
drizzly november in my soul; whenever i find myself involuntarily
pausing before coffin warehouses, and bringing up the rear of every
funeral i meet; and especially whenever my hypos get such an upper
hand of me, that it requires a strong moral principle to prevent me
from deliberately stepping into the street, and methodically knocking
people's hats off--then, i account it high time to get to sea as soon
as i can. this is my substitute for pistol and ball. with a
philosophical flourish cato throws himself upon his sword; i quietly
take to the ship. there is nothing surprising in this. if they but
knew it, almost all men in their degree, some time or other, cherish
very nearly the same feelings towards the ocean with me.

Desafio 2

→ O primeiro passo é descobrirmos possíveis tamanhos para a chave utilizada nessa cifraem utilizando a opção 2 do programa principal app.py:

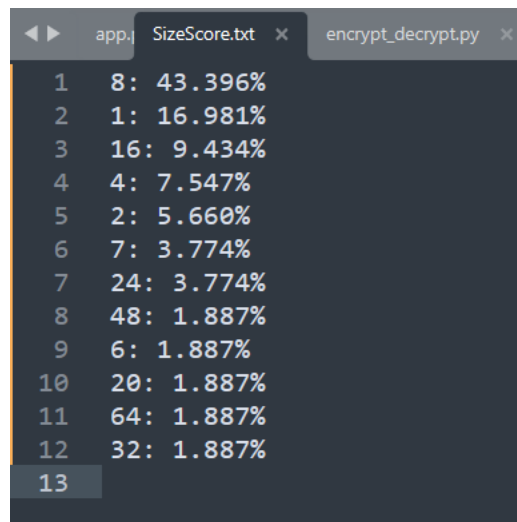

```
_____ Vigenere Devthumos Tool _____
1) Encriptar
2) Decriptar
3) Probabilidades de Tamanhos de Chave
4) Distribuicao dos Agrupamentos em Power BI
5) Sair

Escolha uma Opcao: 3
```

→ Filtramos quais trigramas serão utilizados para a o método de Kasisk através da escolha do parâmetro "maior que". Nesse caso escolhi 2 a fim de eliminar algumas ocorrências de trigramas que talvez sejam coincidências.

```
_____ Vigenere Devthumos Tool _____
_____ Menu 03 _____

Nao Esqueca de Inserir a Mensagem Cifrada a Ser Analisada Pelo Algoritmo no Arquivo "Mensagens/Mensagem_Cifrada.txt"
Numero de Ocorrencias de Trigramas Maior Que: 2
Probabilidades Encontradas Com Sucesso!
Encontre as Probabilidades de Tamanhos de Chave no Arquivo "KeySize/SizeScore.txt"
```



1	8: 43.396%
2	1: 16.981%
3	16: 9.434%
4	4: 7.547%
5	2: 5.660%
6	7: 3.774%
7	24: 3.774%
8	48: 1.887%
9	6: 1.887%
10	20: 1.887%
11	64: 1.887%
12	32: 1.887%
13	

→ O segundo é escolhermos um possível tamanho de chave para começarmos a análise de distribuição. Diferentemente do desafio anterior, considerei o tamanho da chave como 8.

→ O terceiro passo é gerarmos distribuições de frequência de letras para compararmos com os dos idiomas inglês e português, assim como também descobriremos a chave.

```
_____ Vigenere Devthumos Tool _____
_____ Menu 04 _____

Nao Esqueca de Inserir a Mensagem Cifrada a Ser Analisada Pelo Algoritmo no Arquivo "Mensagens/Mensagem_Cifrada.txt"
Possivel Tamanho de Chave: 8
Distribuicoes Encontradas Com Sucesso!
Analise as Distribuicoes Visualmente Atraves do PowerBI no Arquivo "Dashboards/dashboard_de_distribuicao.pbix"
```

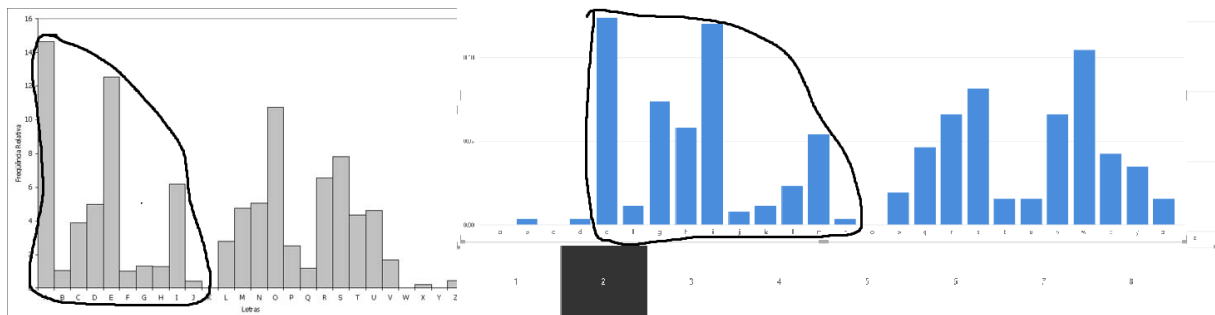
→ O Quarto passo é descobrir qual é o idioma. Já descobrimos anteriormente que a mensagem desafio 2 é em português, então basta compararmos as distribuições.

→ O Quinto passo e último para decifrar a mensagem desafio 1 é comparar a distribuição de frequência de letras em inglês e descobriremos a chave numérica de César, já que podemos trabalhar com cada grupo como se fosse esse tipo de Cifra.

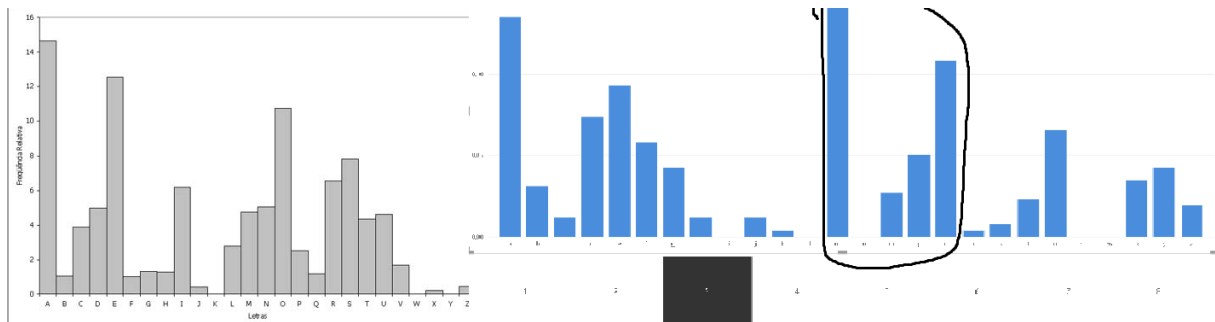
- O primeiro caractere da Chave é “t”



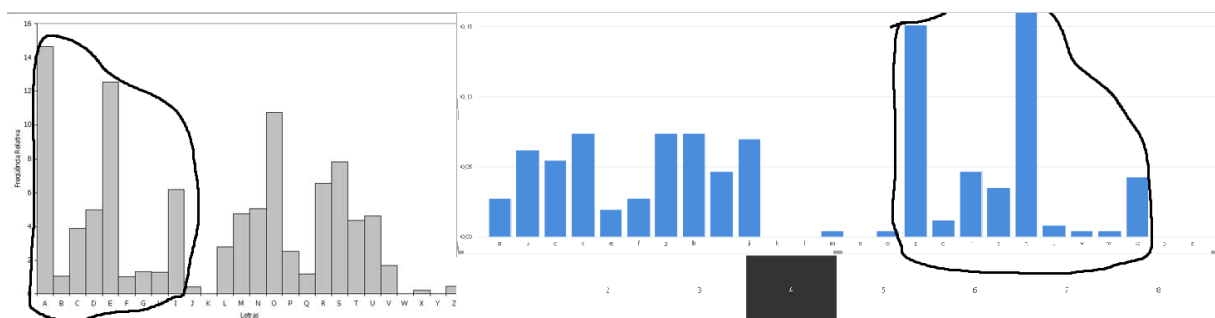
- O segundo caractere da Chave é “e”



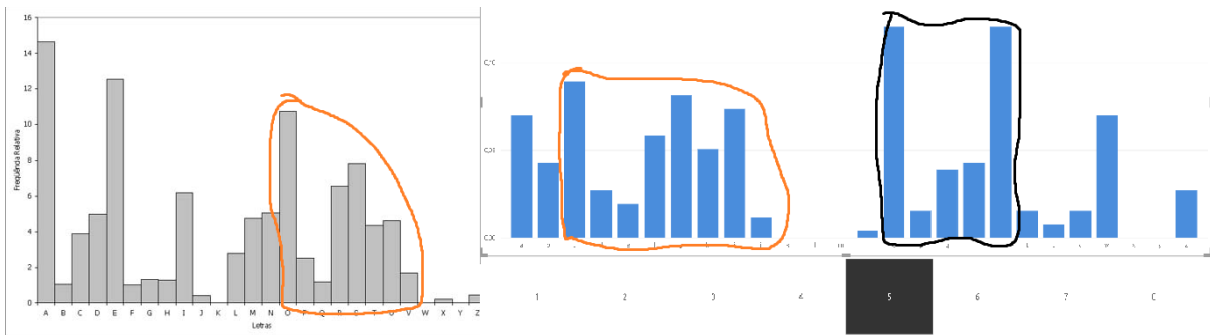
- O terceiro caractere da Chave é “m”



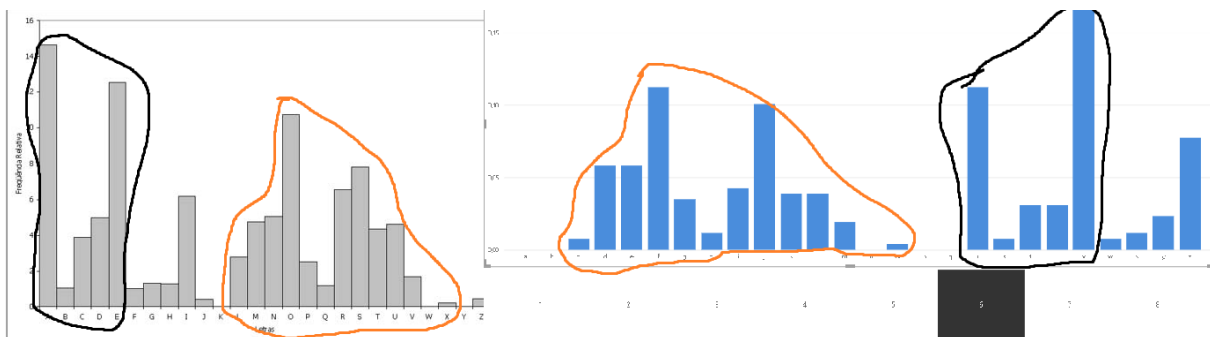
- O quarto caractere da Chave é “p”



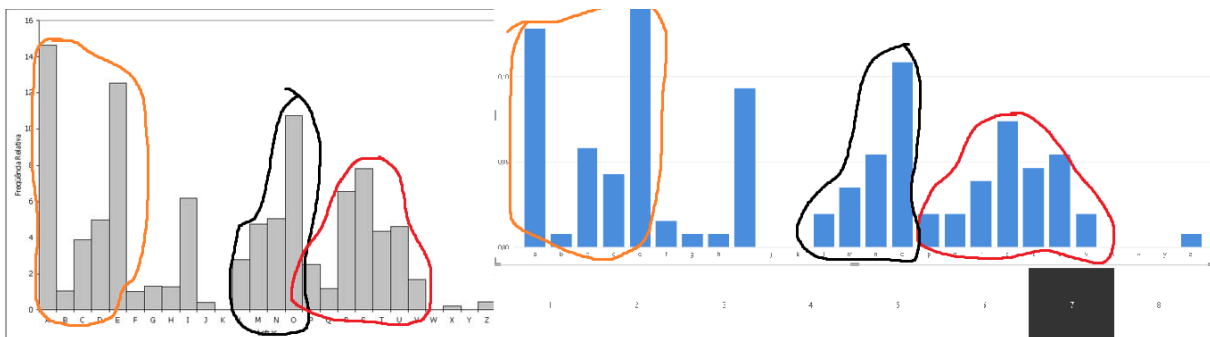
- O quinto caractere da Chave é “o”



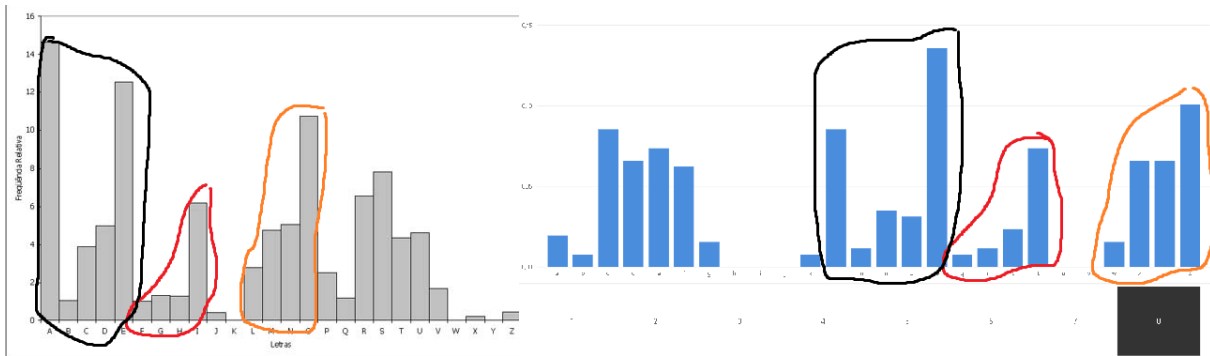
- O sexto caractere da Chave é “r”



- O sétimo caractere da Chave é “a”



- O oitavo caractere da Chave é “l”. Mesmo a distribuição estando mais difícil de identificar do que no desafio anterior, ainda dá para identificar a partir de outros picos de referência.



→ O sexto passo é decifrar a mensagem cifrada com a chave que encontramos

```
Nao Esqueca de Inserir a Mensagem Cifrada a Ser Decifrada no Arquivo "Mensagens/Mensagem_Cifrada.txt"
Chave: temporal
Mensagem Decifrada Com Sucesso!
Encontre a Mensagem Decifrada no Arquivo "Mensagens/Mensagem_Decifrada.txt"
```

algun tempo hesitei se devia abrir estas memorias pelo principio ou pelo fim, isto e, se poria em primeiro lugar o meu nascimento ou a minha morte. suposto o uso vulgar seja começar pelo nascimento, duas consideracoes me levaram a adotar diferente metodo: a primeira e que eu nao sou propriamente um autor defunto, mas um defunto autor, para quem a campa foi outro berco; a segunda e que o escrito ficaria assim mais galante e mais novo. moises, que tambem contou a sua morte, nao a pos no introito, mas no cabo: diferenca radical entre este livro e o pentateuco. dito isto, expirei as duas horas da tarde de uma sexta-feira do mes de agosto de 1869, na minha bela chacara de catumbi. tinha uns sessenta e quatro anos, rijos e prosperos, era solteiro, possuia cerca de trezentos contos e fui acompanhado ao cemiterio por onze amigos. onze amigos! verdade e que nao houve cartas nem anuncios. acresce que chovia — peneirava uma chuvinha miuda, triste e constante, tao constante e tao triste, que levou um daqueles fieis da ultima hora a intercalar esta engenhosa ideia no discurso que proferiu a beira de minha cova: — “vos, que o conhecestes, meus senhores, vos podeis dizer comigo que a natureza parece estar chorando a perda irreparavel de um dos mais belos caracteres que tem honrado a humanidade. este ar sombrio, estas gotas do ceu, aquelas nuvens escuras que cobrem o azul como um crepe funereo, tudo isso e a dor crua e ma que lhe roi a natureza as mais intimas entranhas; tudo isso e um sublime louvor ao nosso ilustre finado.” bom e fiel amigo! nao, nao me arrependo das vinte apolices que lhe deixei. e foi assim que cheguei a clausula dos meus dias; foi assim que me encaminhei para o undiscovered country de hamlet, sem as ânsias nem as duvidas do moco principe, mas pausado e tropego como quem se retira tarde do espetaculo. tarde e aborrecido. viramme ir umas nove ou dez pessoas, entre elas tres senhoras, minha irma sabina, casada com o cotrim, a filha, — um lirio do vale, — e... tenham paciencia! daqui a pouco lhes direi quem era a terceira senhora. contentem-se de saber que essa anonima, ainda que nao parenta, padeceu mais do que as parentas. e verdade, padeceu mais. nao digo que se carpisse, nao digo que se deixasse rolar pelo chao, convulsa. nem o meu obito era coisa altamente dramatica... um solteiroao que expira aos sessenta e quatro anos, nao parece que reuna em si todos os elementos de uma tragedia. e dado que sim, o que menos convinha a essa anonima era aparenta-lo. de pe, a cabeceira da cama, com os olhos estupidos, a boca entreaberta, a triste senhora mal podia crer na minha extincao.