# notebook

November 8, 2024

## 0.1 Criptografia por Deslocamento

A criptografia por deslocamento, é uma das técnicas de criptografia mais antigas já criadas. Ela consiste em substituir cada letra de uma mensagem por outra, deslocando-a no alfabeto por um número fixo de posições. Esse número, chamado de chave de deslocamento, deve ser um inteiro, geralmente entre 1 e 25 no caso do alfabeto latino, para evitar que o deslocamento seja redundante (chaves iguais a 0 ou 26 resultam no texto original).

No funcionamento da cifra, cada letra é convertida para um índice numérico correspondente à sua posição no alfabeto (A = 0, B = 1, ..., Z = 25). Em seguida, o valor da chave é somado a esse índice, e o resultado passa por uma operação de módulo 26, garantindo que o índice resultante permaneça dentro do intervalo válido para o alfabeto. A nova posição determina a letra substituta. Por exemplo, com uma chave de deslocamento de 3, a letra "A" se torna "D", "B" se torna "E", e assim por diante.

Embora historicamente usada para proteger mensagens confidenciais, a Cifra de César oferece pouca segurança no contexto atual, especialmente com a utilização dos computadores no auxlio desta quebra, pois pode ser facilmente quebrada por força bruta (testando todas as chaves possíveis) ou análise de frequência.

A seguir, iremos desenvolver um código que implementará uma Criptografia por Deslocamento. O programa será capaz de gerar uma chave de deslocamento que define o número de posições a serem alteradas no alfabeto, criptografar a mensagem original com base na chave gerada e também será capaz de descriptografar tanto utilizando a chave de deslocamento tanto quanto sem esta chave utilizando apenas análise de frequência das letras no português.

### 0.1.1 Código para Geração da Chave, Leitura do Plaint-Text e Geração da Cifra

Para o estudo da criptografia por deslocamento, utilizaremos a primeira página do livro:"A Revolução dos Bixos" por George Orwell. Da edição da Gaveta do Povo. A escolha foi motivada pela necessidade que para a quebra de um texto por análise de frequência, é importante ter um grande volume de palavras escritas de forma coerente e ortograficamente corretas. Assim, nada melhor que um livro para nos fornercer esses dados.

Mas para isso é necessário uma série de filtragens para tornar o texto livre de acentuação, caracteres especiais e letras como "ç".

Assim, vamos desenvolver a seguinte função para a leitura e tratamento do nosso texto a ser cifrado:

[518]:
```
import math
import random
```

```
import itertools
import unicodedata
import numpy as np
import pandas as pd
from data.letras import frequencia_letras

def read_plain_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        plain_text = file.read()

    plain_text = plain_text.replace(" ", "")

    plain_text = unicodedata.normalize('NFD', plain_text)
    plain_text = ''.join(char for char in plain_text if unicodedata.
 ↪category(char) != 'Mn')

    plain_text = plain_text.lower().replace("ç", "")
    plain_text = plain_text.replace(".", "")
    plain_text = plain_text.replace(",", "")
    plain_text = plain_text.replace("-", "")
    plain_text = plain_text.replace(":", "")
    plain_text = plain_text.replace(";", "")
    plain_text = plain_text.replace(")", "")
    plain_text = plain_text.replace("(", "")
    plain_text = plain_text.replace("{", "")
    plain_text = plain_text.replace("}", "")
    plain_text = plain_text.replace("\n", "")

    return plain_text
```

Já com nosso texto a ser cifrado, partiremos para a escolha da Chave de Deslocamento. A escolha da chave será aleatória e será um número de 1 a 25, fora isso iremos desenvolver uma função auxiliar para nos ajudar futuramente, basicamente será uma lista das letras do alfabeto com o número de sua posição:

[519]:
```
def alphabet_mapping():
    alphabet_map = {}
    for i in range(ord('a'), ord('z') + 1):
        alphabet_map[chr(i)] = i - ord('a')
    return alphabet_map

def generate_key():
    return random.randint(1, 25)
```

Com o texto pronto, a chave e nossa função auxiliar, iremos partir para a função de geração da cifra, ela será bem simples devendo apenas receber a chave e deslocar todos os caracteres do texto a partir do número da chave.

[520]:
```python
def encrypt(alphabet_map, plain_text, key):
    encrypted_text = []
    for char in plain_text:
        if char in alphabet_map:
            new_index = (alphabet_map[char] + key) % len(alphabet_map)
            encrypted_text.append(list(alphabet_map.keys())[new_index])

    return ''.join(encrypted_text)
```

Pronto. Agora, devemos ser capazes de ler o texto a ser criptografado, gerar uma chave aleatória e, por fim, gerar a cifra correspondente a chave fornecida.

[521]:
```python
alphabet_map = alphabet_mapping()

plain_text = plain_text = read_plain_text("../src/data/revo_bichos.txt")
print(f"Texto original: {plain_text}")

key = generate_key()
print(f"Chave de encriptação gerada: {key}")

cypher_text = encrypt(alphabet_map, plain_text, key)
print(f"Texto cifrado gerado:\n {cypher_text}")
```

```
Texto original: srjonesdafazendasolartinhatrancadoosgalinheiroscomotodaanoitemas
estavabebadodemaisparaselembrardefecharasportinholaslateraiscomocirculodeluzdesu
alanternadancandodeumladoparaooutroelesearrastoupelopatiotirouasbotasnaportadosf
undosdacasaencheuumultimocopodecervejanobarrildacopaesubiuparaacamaondeasrajones
jaestavaroncandoassimquealuzdoquartoseapagouhouveumaagitacaoeumrebulicoemtodasas
instalacoesdafazendaduranteodiahaviaseespalhadoanoticiadequeovelhomajorumporcoda
racamiddlewhitemagnificotiveraumsonhoestranhonanoiteanterioredesejavacontaloaoso
utrosanimaistodoscombinaramquedeveriamseencontraremsegurancanograndeceleiroassim
queosrjonesestivesseforadocaminhoovelhomajorassimochamavamemboraseunomedeexibica
ofossebelezadewillingdongozavadetantorespeitonafazendaquetodosestavamprontospara
perderumahoradesonoafimdeouviroqueeletinhaadizeroi
Chave de encriptação gerada: 8
Texto cifrado gerado:
 azrwvmalinihmvliawtizbqvpibzivkilwwaoitqvpmqzwakwuwbwliivwqbmuiamabidijmjilwlmu
iqaxiziamtmujzizlmnmkpiziaxwzbqvpwtiatibmziqakwuwkqzkctwlmtchlmacitivbmzvilivkiv
lwlmcutilwxiziwwcbzwmtmamizziabwcxmtwxibqwbqzwciajwbiavixwzbilwancvlwalikiaimvkp
mccuctbquwkwxwlmkmzdmrivwjizzqtlikwximacjqcxiziikiuiwvlmiazirwvmarimabidizwvkivl
wiaaquycmitchlwycizbwamixiowcpwcdmcuiioqbikiwmcuzmjctqkwmubwliaiaqvabitikwmalini
hmvlilczivbmwlqipidqiammaxitpilwivwbqkqilmycmwdmtpwuirwzcuxwzkwlizikiuqlltmepqbm
uiovqnqkwbqdmzicuawvpwmabzivpwvivwqbmivbmzqwzmlmamridikwvbitwiwawcbzwaivquiqabwl
wakwujqviziuycmlmdmzqiuammvkwvbzizmuamoczivkivwozivlmkmtmqzwiaaquycmwazrwvmamabq
dmaamnwzilwkiuqvpwwdmtpwuirwziaaquwkpiuidiumujwziamcvwumlmmfqjqkiwnwaamjmtmhilme
qttqvolwvowhidilmbivbwzmaxmqbwvinihmvliycmbwlwamabidiuxzwvbwaxizixmzlmzcuipwzilm
awvwinqulmwcdqzwycmmtmbqvpiilqhmzwq
```

Caso o código acima não tenha sido executado corretamente, execute do início todos os blocos de

código em ordem e execute novamente este último bloco.

### 0.1.2 Código para Descriptografia com a Chave

Agora, iniciaremos a descriptografia do nosso texto cifrado com a chave.

Iremos novamente criar uma função com um comportamento bem simples, a função irá deslocar inversamente o nosso texto cifrado a partir da chave fornecida:

[522]:
```
def decrypt(alphabet_map, cypher_text, key):
    decrypted_text = []
    for char in cypher_text:
        if char in alphabet_map:
            new_index = (alphabet_map[char] - key) % len(alphabet_map)
            decrypted_text.append(list(alphabet_map.keys())[new_index])
        else:
            decrypted_text.append(char)
    return ''.join(decrypted_text)

decrypted_text_with_key = decrypt(alphabet_map, cypher_text, key)
print(f"Texto resultante da desencriptação com a chave:\n␣
 ↪{decrypted_text_with_key}")
```

```
Texto resultante da desencriptação com a chave:
 srjonesdafazendasolartinhatrancadoosgalinheiroscomotodaanoitemasestavabebadodem
aisparaselembrardefecharasportinholaslateraiscomocirculodeluzdesualanternadancan
dodeumladoparaooutroelesearrastoupelopatiotirouasbotasnaportadosfundosdacasaench
euumultimocopodecervejanobarrildacopaesubiuparaacamaondeasrajonesjaestavaroncand
oassimquealuzdoquartoseapagouhouveumaagitacaoeumrebulicoemtodasasinstalacoesdafa
zendaduranteodiahaviaseespalhadoanoticiadequeovelhomajorumporcodaracamiddlewhite
magnificotiveraumsonhoestranhonanoiteanterioredesejavacontaloaosoutrosanimaistod
oscombinaramquedeveriamseencontraremsegurancanograndeceleiroassimqueosrjonesesti
vesseforadocaminhoovelhomajorassimochamavamemboraseunomedeexibicaofossebelezadew
illingdongozavadetantorespeitonafazendaquetodosestavamprontosparaperderumahorade
sonoafimdeouviroqueeletinhaadizeroi
```

Agora, testaremos se o resultado da nossa desiscritografia está correto:

[523]:
```
if decrypted_text_with_key == plain_text:
    print("Criptografia gerada e revertida com sucesso!!")
```

```
Criptografia gerada e revertida com sucesso!!
```

Caso a menssagem acima não tenha sido apresentada, execute do início todos os blocos de código em ordem e execute novamente este último bloco.

### 0.1.3 Código para Descriptografia sem chave - Análise de Frequência

Agora, tentaremos quebrar a cifra sem utilizar a chave de deslocamento. A única informação disponível será o texto cifrado em si. Essa função será mais complexa do que as anteriores, pois

envolverá uma heurística que determinará qual é a chave mais provável para o texto com base na frequência das letras no português.

[524]:
```
def break_shift_cipher(alphabet_map, cypher_text):
    alphabet = list(alphabet_map.keys())
    cypher_frequencies = {letter: 0 for letter in alphabet}

    for char in cypher_text:
        if char in cypher_frequencies:
            cypher_frequencies[char] += 1

    total_letters = sum(cypher_frequencies.values())
    cypher_frequencies = {k: (v / total_letters) * 100 for k, v in␣
 ↪cypher_frequencies.items()}

    best_key = None
    best_correlation = float('-inf')

    for key in range(len(alphabet)):
        shifted_frequencies = [0] * len(alphabet)

        for i, letter in enumerate(alphabet):
            shifted_index = (i - key) % len(alphabet)
            shifted_frequencies[shifted_index] = cypher_frequencies.get(letter,␣
 ↪0)

        correlation = 0
        for i in range(len(alphabet)):
            correlation += shifted_frequencies[i] * frequencia_letras[i]

        if correlation > best_correlation:
            best_correlation = correlation
            best_key = key

    decrypted_text = decrypt(alphabet_map, cypher_text, best_key)

    return decrypted_text

decrypted_text_without_key = break_shift_cipher(alphabet_map, cypher_text)
print(f"Texto originado da desencriptação sem a chave:␣
 ↪{decrypted_text_without_key}")
```

```
Texto originado da desencriptação sem a chave: srjonesdafazendasolartinhatrancad
oosgalinheiroscomotodaanoitemasestavabebadodemaisparaselembrardefecharasportinho
laslateraiscomocirculodeluzdesualanternadancandodeumladoparaooutroelesearrastoup
elopatiotirouasbotasnaportadosfundosdacasaencheuumultimocopodecervejanobarrildac
opaesubiuparaacamaondeasrajonesjaestavaroncandoassimquealuzdoquartoseapagouhouve
umaagitacaoeumrebulicoemtodasasinstalacoesdafazendaduranteodiahaviaseespalhadoan
```

```
oticiadequeovelhomajorumporcodaracamiddlewhitemagnificotiveraumsonhoestranhonano
iteanterioredesejavacontaloaosoutrosanimaistodoscombinaramquedeveriamseencontrar
emsegurancanograndeceleiroassimqueosrjonesestivesseforadocaminhoovelhomajorassim
ochamavamemboraseunomedeexibicaofossebelezadewillingdongozavadetantorespeitonafa
zendaquetodosestavamprontosparaperderumahoradesonoafimdeouviroqueeletinhaadizero
i
```

Assim, vamos testar se o nosso resultado de fato está correto:

[525]:
```
if decrypted_text_without_key == plain_text:
    print("Criptografia quebrada com sucesso!!")
```

```
Criptografia quebrada com sucesso!!
```

Caso a menssagem acima não tenha sido apresentada, execute do início todos os blocos de código em ordem e execute novamente este último bloco.

#### 0.1.4 Análise de Complexidade

A função de geração de chave: generate_key() possui complexidade O(1)

Já as funções de: encrypt() decrypt() break_shift_cipher() possuem complexidade O(n)

Sendo a break_shift_cipher() a mais pesada, pois fará sempre n +26*26 computações, mas, para uma análise assintótica, ainda possui a mesma complexidade, O(n), das demais funções.

#### 0.1.5 Análise da Estratégia

As estratégias utilizadas são suficientes boas, podendo criptografar, descriptografar e quebrar a cifra para qualquer cenário, mesmo para N realmente grandes.

### 0.2 Criptografia por Transposição

A criptografia por transposição é uma técnica antiga de criptografia que, ao contrário da cifra de deslocamento, não altera as letras da mensagem, mas sim a sua ordem. Ela consiste em reorganizar as letras de uma mensagem de acordo com uma chave, que define como as letras serão rearranjadas. A chave, nesse caso, será uma palavra de até 26 caracteres, sem caracteres repetidos, que indicará a ordem das colunas na matriz de transposição, com base na posição das letras ordenadas.

No funcionamento dessa cifra, o texto é disposto em uma matriz com um número fixo de colunas, determinado pela chave. O número de linhas é calculado de forma a preencher completamente o texto. Caso o texto não preencha completamente a matriz, letras adicionais são inseridas em ordem alfabética, até completar o espaço necessário. Após a disposição do texto na matriz, as letras são lidas coluna por coluna, seguindo a ordem definida pela chave, formando assim o texto cifrado.

Embora a cifra de transposição tenha sido amplamente utilizada ao longo da história para proteger informações sensíveis, ela oferece pouca segurança no contexto moderno, pois pode ser quebrada através da análise de padrões, como a busca por repetições de caracteres ou pela reconstrução do padrão de transposição.

A seguir, desenvolveremos um código para implementar a Criptografia por Transposição. O programa será capaz de gerar uma chave de transposição aleatória que definirá a ordem das colunas na matriz, criptografar a mensagem original com base na chave gerada e, também, descriptografar a

mensagem tanto utilizando a chave de transposição quanto sem ela para chaves de até 7 caracteres, empregando a análise de frequência de dígrafos no português.

### 0.2.1 Código para Geração da Chave, Leitura do Plaint-Text e Geração da Cifra

Aqui ainda utilizaremos o mesmo texto do George Orwell, pois também será necessário uma grande quantidade de linhas na nossa matriz de transposição para a quebra por análise de frequência. Assim, a mesma função utilizada anteriormente será aplicada novamente:

[526]:
```
import math
import random
import itertools
import numpy as np
import pandas as pd
import unicodedata
from data.digrafos import frequencia_digrafos

def read_plain_text(file_path):
    with open(file_path, 'r', encoding='utf-8') as file:
        plain_text = file.read()

    plain_text = plain_text.replace(" ", "")

    plain_text = unicodedata.normalize('NFD', plain_text)
    plain_text = ''.join(char for char in plain_text if unicodedata.
 ↪category(char) != 'Mn')

    plain_text = plain_text.lower().replace("ç", "")
    plain_text = plain_text.replace(".", "")
    plain_text = plain_text.replace(",", "")
    plain_text = plain_text.replace("-", "")
    plain_text = plain_text.replace(":", "")
    plain_text = plain_text.replace(";", "")
    plain_text = plain_text.replace(")", "")
    plain_text = plain_text.replace("(", "")
    plain_text = plain_text.replace("{", "")
    plain_text = plain_text.replace("}", "")
    plain_text = plain_text.replace("\n", "")

    return plain_text
```

Já com nosso texto a ser cifrado, partiremos para a escolha da Chave de Transposição. A escolha da chave será aleatória e será uma palavra de até 26 caracteres sem repetição de caracteres, fora isso iremos usar a mesma função auxiliar do problema anterior.

[527]:
```
def alphabet_mapping():
    alphabet_map = {}
    for i in range(ord('a'), ord('z') + 1):
```

```
        alphabet_map[chr(i)] = i - ord('a')
    return alphabet_map

def generate_key(alphabet_map, characters_length = 10):

    if characters_length < 1 or characters_length > 26:
        characters_length = 26

    shuffled_map = list(alphabet_map.items()).copy()
    random.shuffle(shuffled_map)

    key = []
    for i in range(characters_length):
        key.append(shuffled_map[i][0])

    return key
```

Com o texto pronto, a chave e nossa função auxiliar, iremos partir para a função de geração da cifra. Diferente da função de geração da cifra de deslocamento, esta terá um processamento mais pesado, gerando uma matriz de transposição e depois realizará a leitura ordenada pela posição das letras da chave percorrendo as letras em ordem alfabética.

[528]:
```
def encrypt_transposition(alphabet_map, plain_text, key):
    key_length = len(key)
    sorted_key = sorted(key)

    num_rows = len(plain_text) // key_length
    if (len(plain_text) % key_length) != 0:
        num_rows = num_rows+1

    matrix = []
    for _ in range(num_rows):
        matrix.append([''] * key_length)

    index = 0
    for row in range(len(matrix)):
        for c in range(len(matrix[row])):
            if(index < len(plain_text)):
                matrix[row][c] = plain_text[index]
            else:
                adjusted_index = (index - len(plain_text)) % len(alphabet_map)
                matrix[row][c] = chr(adjusted_index+ord('a'))
            index += 1

    cypher_text = ''

    for i in range(len(key)):
```

```
        col = key.index(sorted_key[i])
        for row in matrix:
            cypher_text = cypher_text + row[col]

    return cypher_text
```

Pronto. Agora, devemos ser capazes de ler o texto a ser criptografado, gerar uma chave aleatória e, por fim, gerar a cifra correspondente a chave fornecida. A chave de transposição foi prefixada com o intuito de tornar processo de quebra da cifra mais rápido, evitando as n! permutações que um algoritmo de transposição gera como possibilidade.

[529]:
```
alphabet_map = alphabet_mapping()

plain_text = plain_text = read_plain_text("../src/data/revo_bichos.txt")
print(f"Texto original: {plain_text}")

key = ['a', 'h', 'j', 'v', 't', 'n']
print(f"Chave de encriptação gerada: {key}")

cypher_text = encrypt_transposition(alphabet_map, plain_text, key)
print(f"Texto cifrado gerado: {cypher_text}")
```

```
Texto original: srjonesdafazendasolartinhatrancadoosgalinheiroscomotodaanoitemas
estavabebadodemaisparaselembrardefecharasportinholaslateraiscomocirculodeluzdesu
alanternadancandodeumladoparaooutroelesearrastoupelopatiotirouasbotasnaportadosf
undosdacasaencheuumultimocopodecervejanobarrildacopaesubiuparaacamaondeasrajones
jaestavaroncandoassimquealuzdoquartoseapagouhouveumaagitacaoeumrebulicoemtodasas
instalacoesdafazendaduranteodiahaviaseespalhadoanoticiadequeovelhomajorumporcoda
racamiddlewhitemagnificotiveraumsonhoestranhonanoiteanterioredesejavacontaloaoso
utrosanimaistodoscombinaramquedeveriamseencontraremsegurancanograndeceleiroassim
queosrjonesestivesseforadocaminhoovelhomajorassimochamavamemboraseunomedeexibica
ofossebelezadewillingdongozavadetantorespeitonafazendaquetodosestavamprontospara
perderumahoradesonoafimdeouviroqueeletinhaadizeroi
Chave de encriptação gerada: ['a', 'h', 'j', 'v', 't', 'n']
Texto cifrado gerado: sselhcgeoaetbmrmertarmuuarceoolrptotossauiovbdepaeoerdmuaa
hmamioiaadnaslnaommdmwacrnraaoeoarmdbmvsnmagcoqjssdnlomabuecszlovnpadotrpraefuui
irdnaaaaimamaaaabfaisaolzlnaupueaeiuarfdeumdeaasamansooqzrpoacrcdncfathehodvapai
hgoahannrjnooaoiqeetsnreauoteohhrovoneaealnatefadaoadhsivenzjadrtdlronavdisresnl
icodaanmatssloastuanmoejrcurasetnaudtauaaeoasoadeaeateejordintuonoteatssisnurere
calsenifcooacaroxobdigdoiaqovnreoomieheezonnshcdtseeaedarleocluenddoeruaropooset
prolaucdjaanilueuutultsldnaiaaaieouoaemieotneiscotiomaemoeroermresaiejimmediseid
aasnntspsemdaoqtdinasiaonsoiebdplrhootcrestaoaaoaopibaddahloenipianajvasaqsoeieu
maaserdipocuhrcclefvssotrealuntordacrundiissermvasaesebolwgzteoeeemopuaoeoeaoofa
troiotosaoseacphasidenddlrretotsnanccuccarobaorsacseoogvgobestezuovsdiqloradtiim
ehiedvaoascaeinaganesoevoaomshmamifeenoertzusatarrndrlar
```

Caso o código acima não tenha sido executado corretamente, execute do início todos os blocos de código em ordem e execute novamente este último bloco.

### 0.2.2 Código para Descriptografia com chave

Agora iremos desenvolver o processo de descriptografar utilizando a chave de transposição fornecida. Esse algoritmo irá percorrer os passos inversos do algoritmo de criptografia, gerando de forma ordenada uma matriz de transposição com base na chave e, por fim, retornará o texto original.

[530]:
```
def decrypt_transposition(alphabet_map, cypher_text, key):
    key_length = len(key)
    sorted_key = sorted(key)

    num_rows = len(cypher_text) // key_length
    if (len(cypher_text) % key_length) != 0:
        num_rows += 1

    matrix = []
    for _ in range(num_rows):
        matrix.append([''] * key_length)

    col = 0
    for i in range(len(key)):
        index = key.index(sorted_key[i])
        for row in range(num_rows):
            matrix[row][index] = cypher_text[col]
            col += 1

    plain_text = ''
    for row in matrix:
        plain_text += ''.join(row)

    return plain_text

decrypted_text_with_key = decrypt_transposition(alphabet_map, cypher_text, key)
print(f"Texto resultante da desencriptação com a chave:␣
 ↪{decrypted_text_with_key}")
```

```
Texto resultante da desencriptação com a chave: srjonesdafazendasolartinhatranca
doosgalinheiroscomotodaanoitemasestavabebadodemaisparaselembrardefecharasportinh
olaslateraiscomocirculodeluzdesualanternadancandodeumladoparaooutroelesearrastou
pelopatiotirouasbotasnaportadosfundosdacasaencheuumultimocopodecervejanobarrilda
copaesubiuparaacamaondeasrajonesjaestavaroncandoassimquealuzdoquartoseapagouhouv
eumaagitacaoeumrebulicoemtodasasinstalacoesdafazendaduranteodiahaviaseespalhadoa
noticiadequeovelhomajorumporcodaracamiddlewhitemagnificotiveraumsonhoestranhonan
oiteanterioredesejavacontaloaosoutrosanimaistodoscombinaramquedeveriamseencontra
remsegurancanograndeceleiroassimqueosrjonesestivesseforadocaminhoovelhomajorassi
mochamavamemboraseunomedeexibicaofossebelezadewillingdongozavadetantorespeitonaf
azendaquetodosestavamprontosparaperderumahoradesonoafimdeouviroqueeletinhaadizer
oi
```

Assim, vamos testar se o nosso resultado de fato está correto:

[531]:
```
if decrypted_text_with_key == plain_text:
    print("Criptografia gerada e revertida com sucesso!!")
```

```
Criptografia gerada e revertida com sucesso!!
```

Caso a menssagem acima não tenha sido apresentada, execute do início todos os blocos de código em ordem e execute novamente este último bloco.

### 0.2.3 Código para Descriptografia sem chave - Força Bruta

Agora, tentaremos quebrar a cifra sem utilizar a chave de transposição. A única informação disponível será o texto cifrado em si e o tamanho da chave. O tamanho da chave foi fixado para tornar o código mais simples, mas bastaria percorrer as possibilidades de tamanhos em que fosse permitido ter um modulo com resto zero e pegar a possibilidade com maior score, mas, novamente, seria uma complicação a mais que não agrega ao tema em questão. Essa função será mais complexa do que as anteriores, pois envolverá uma heurística pesada para escolhar qual será a ordem da palavra chave que fará mais sentido com base na frequência assimétrica dos digrafos no português.

A estratégia adotada para quebrar a cifra de transposição envolve testar todas as permutações possíveis da ordem das colunas da matriz gerada a partir do texto cifrado. A matriz é construída considerando o comprimento da chave, e cada permutação representa uma tentativa de reorganizar as colunas. O objetivo é encontrar a permutação que maximize o "score", que é calculado com base na frequência de digrafos presentes na matriz e em uma tabela de frequências pré-estabelecida.

A função calcula o score para cada permutação das colunas e seleciona a que resulta no maior valor, o que indica a maior probabilidade de que a sequência de colunas corresponda à disposição original do texto. A melhor permutação é então usada para reorganizar o texto cifrado e decifrá-lo, recuperando o conteúdo original.

[532]:
```
def calculate_score(df, column_sequence, cypher_text, key_length):
    num_rows = len(cypher_text) // key_length
    if len(cypher_text) % key_length != 0:
        num_rows += 1

    matrix = [[''] * key_length for _ in range(num_rows)]
    col = 0
    for i in range(len(column_sequence)):
        index = column_sequence[i]
        for row in range(num_rows):
            if col < len(cypher_text):
                matrix[row][index] = cypher_text[col]
                col += 1

    # Calcula o score baseado na frequência dos dígrafos
    score = 0
    for row in matrix:
        for i in range(len(row) - 1):
            if row[i] and row[i + 1]:
                digraph = row[i] + row[i + 1]
```

```
                if digraph[0] in df.index and digraph[1] in df.columns:
                    score += df.at[digraph[0], digraph[1]]

    return score

def break_transposition_cipher(df, alphabet_map, cypher_text):
    key_length = 6
    best_sequence = None
    best_score = -1

    # Gerar todas as combinações possíveis de colunas
    for perm in itertools.permutations(range(key_length)):
        score = calculate_score(df, perm, cypher_text, key_length)

        if score > best_score:
            best_score = score
            best_sequence = perm

    # Decifrar o texto com a melhor sequência encontrada
    plain_text = decrypt_transposition(alphabet_map, cypher_text, best_sequence)
    return plain_text

#Preparando os dados para o Pandas
tamanho_grupo = 26
data= [frequencia_digrafos[i:i + tamanho_grupo] for i in range(0,␣
 ↪len(frequencia_digrafos), tamanho_grupo)]
#Criando um DataFrame com Pandas
df = pd.DataFrame(data)
# Ajusta o índice e colunas conforme necessário
df.index = [chr(i) for i in range(ord('a'), ord('z') + 1)]
df.columns = [chr(i) for i in range(ord('a'), ord('z') + 1)]

decrypted_text_without_key = break_transposition_cipher(df, alphabet_map,␣
 ↪cypher_text)
print(f"Texto originado da desencriptação sem a chave:␣
 ↪{decrypted_text_without_key}")
```

```
Texto originado da desencriptação sem a chave: srjonesdafazendasolartinhatrancad
oosgalinheiroscomotodaanoitemasestavabebadodemaisparaselembrardefecharasportinho
laslateraiscomocirculodeluzdesualanternadancandodeumladoparaooutroelesearrastoup
elopatiotirouasbotasnaportadosfundosdacasaencheuumultimocopodecervejanobarrildac
opaesubiuparaacamaondeasrajonesjaestavaroncandoassimquealuzdoquartoseapagouhouve
umaagitacaoeumrebulicoemtodasasinstalacoesdafazendaduranteodiahaviaseespalhadoan
oticiadequeovelhomajorumporcodaracamiddlewhitemagnificotiveraumsonhoestranhonano
iteanterioredesejavacontaloaosoutrosanimaistodoscombinaramquedeveriamseencontrar
emsegurancanograndeceleiroassimqueosrjonesestivesseforadocaminhoovelhomajorassim
ochamavamemboraseunomedeexibicaofossebelezadewillingdongozavadetantorespeitonafa
```

```
zendaquetodosestavamprontosparaperderumahoradesonoafimdeouviroqueeletinhaadizero
i
```

Assim, vamos testar se o nosso resultado de fato está correto:

[533]:
```
if decrypted_text_without_key == plain_text:
    print("Criptografia quebrada com sucesso!!")
```

```
Criptografia quebrada com sucesso!!
```

Caso a menssagem acima não tenha sido apresentada, execute do início todos os blocos de código em ordem e execute novamente este último bloco.

### 0.2.4 Análise de Complexidade

A função de geração de chave: generate_key() possui complexidade O(1)

As funções de: encrypt() decrypt() possuem complexidade O(n)

Já a função de quebra de cifra: break_shift_cipher() possui complexidade O(n!)

### 0.2.5 Análise da Estratégia

As estratégias utilizadas são suficientes boas para os cenários de: criptografar e descriptografar, mesmo para N realmente grandes.

A estratégia de quebra da cifra não é boa o suficente para tamanho de chave muito grandes visto que sua complexidade é n!.