КОМП’ЮТЕРНИЙ ПРАКТИКУМ №2

Гоголєва Поліна ФБ-12

**Варіант 5**

**Криптоаналіз шифру Віженера**

**Мета роботи**: Засвоєння методів частотного криптоаналізу. Здобуття навичок роботи та аналізу потокових шифрів гамування адитивного типу на прикладі шифру Віженера.

**Хід роботи**

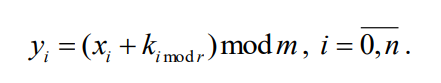
1. Уважно прочитати методичні вказівки до виконання комп’ютерного практикуму.

Done 💪

1. Самостійно підібрати текст для шифрування (2-3 кб) та ключі довжини r= 2, 3, 4, 5, а також довжини 10-20 знаків. Зашифрувати обраний відкритий текст шифром Віженера з цими ключами.

Візьму невеличку частину того російського тексту, що я використовувала при виконанні першого практикуму, щоб без зайвих символів, пунктуації та пробілів та закинемо його у файл cleaned.txt

Тепер, маючи формулу для шифрування спробуєм зашифрувати текст ключами різної довжини.



Спочатку я спробувала придумати різні ключі і записати їх у різні змінні, але зрозуміла, що мені дуже дуже впадку придумувати 15 ключів, тож я придумала один на 20 символів і просто брала визначену кількість символів з нього для шифрування ключами різної довжини. Work smart not hard :)

def vigenere\_cipher(plaintext, key):

    alphabet = "абвгдежзийклмнопрстуфхцчшщъыьэюя"

    encrypted\_text = ""

    for i in range(len(plaintext)):

        plaintext\_char = plaintext[i]

        if plaintext\_char in alphabet:

            p = alphabet.index(plaintext\_char)

            k = alphabet.index(key[i % len(key)])

            encrypted\_char = alphabet[(p + k) % len(alphabet)]

            encrypted\_text += encrypted\_char

        else:

            encrypted\_text += plaintext\_char

    return encrypted\_text

big\_key = "унасоченьдлинныйключ"

file\_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp2\gogoleva\_fb-12\_cp2\cleaned.txt"

with open(file\_path, "r", encoding="cp1251") as file:

    plaintext = file.read()

with open("encodes.txt", "w", encoding="cp1251") as encoded\_file:

    key\_lengths = [2, 3, 4, 5, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

    for key\_length in key\_lengths:

        current\_key = big\_key[:key\_length]

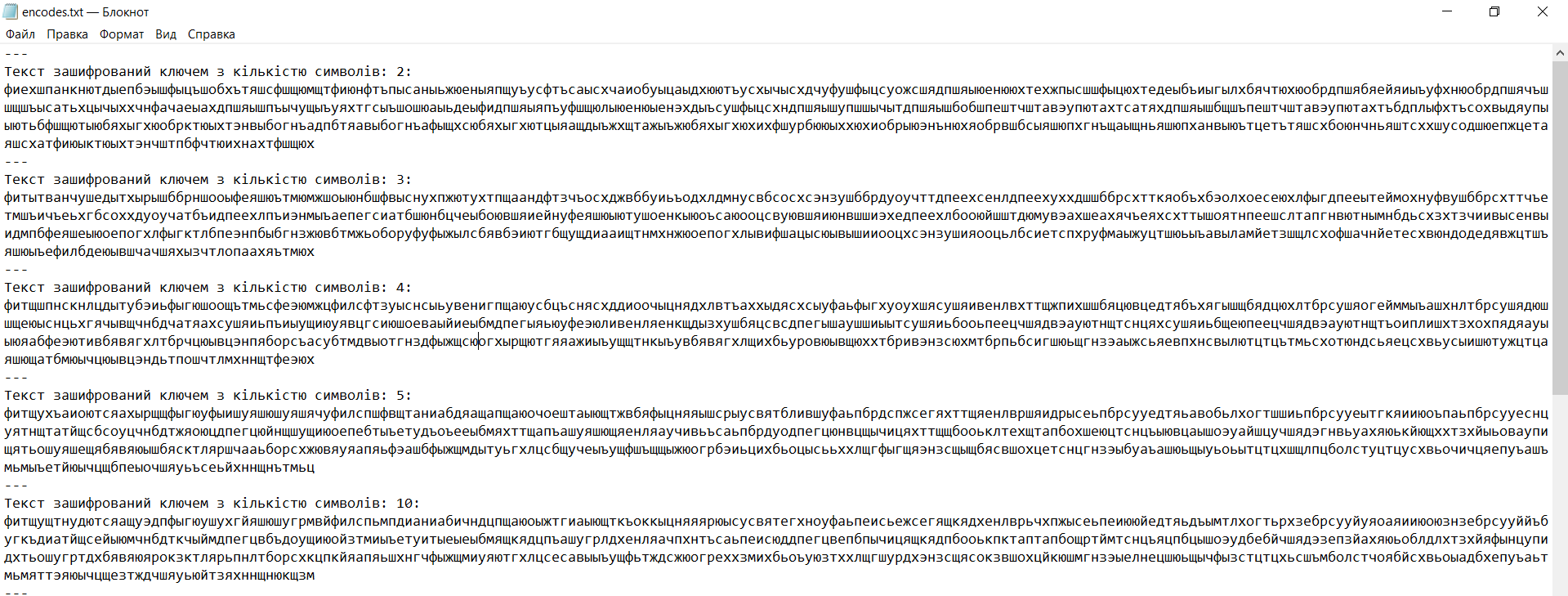
        encrypted\_text = vigenere\_cipher(plaintext, current\_key)

        encoded\_file.write(f"---\nТекст зашифрований ключем з кількістю символів: {key\_length}: \n")

        encoded\_file.write(encrypted\_text + "\n")

print("Зашифровані тексти успішно збережені")

Тож тепер я маю файл encodes.txt з текстом, зашифрованим різними ключами.



1. Підрахувати індекси відповідності для відкритого тексту та всіх одержаних шифртекстів і порівняти їх значення.

Використовуючи формулу з методички створюємо ще одну функцію та інтегруємо її у функцію кодування шифром віженера, щоб після кодування одразу розрахувати індекс відповідності. По-суті, для розрахунку індексу основним розрахунком є кількість появи букви у тексті.

def calculate\_index\_of\_coincidence(text):

    text = ''.join(filter(str.isalpha, text))

    text = text.lower()

    n = len(text)

    letter\_frequencies = {}

    for letter in text:

        if letter in letter\_frequencies:

            letter\_frequencies[letter] += 1

        else:

            letter\_frequencies[letter] = 1

    index\_of\_coincidence = sum(f \* (f - 1) for f in letter\_frequencies.values()) / (n \* (n - 1))

    return index\_of\_coincidence

def vigenere\_cipher(plaintext, key):

    alphabet = "абвгдежзийклмнопрстуфхцчшщъыьэюя"

    encrypted\_text = ""

    for i in range(len(plaintext)):

        plaintext\_char = plaintext[i]

        if plaintext\_char in alphabet:

            p = alphabet.index(plaintext\_char)

            k = alphabet.index(key[i % len(key)])

            encrypted\_char = alphabet[(p + k) % len(alphabet)]

            encrypted\_text += encrypted\_char

        else:

            encrypted\_text += plaintext\_char

    return encrypted\_text

big\_key = "унасоченьдлинныйключ"

file\_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp2\gogoleva\_fb-12\_cp2\cleaned.txt"

with open(file\_path, "r", encoding="cp1251") as file:

    plaintext = file.read()

original\_index = calculate\_index\_of\_coincidence(plaintext)

print(f"Індекс відповідності для оригінального тексту: {original\_index:.4f}")

key\_lengths = [2, 3, 4, 5, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

for key\_length in key\_lengths:

    current\_key = big\_key[:key\_length]

    encrypted\_text = vigenere\_cipher(plaintext, current\_key)

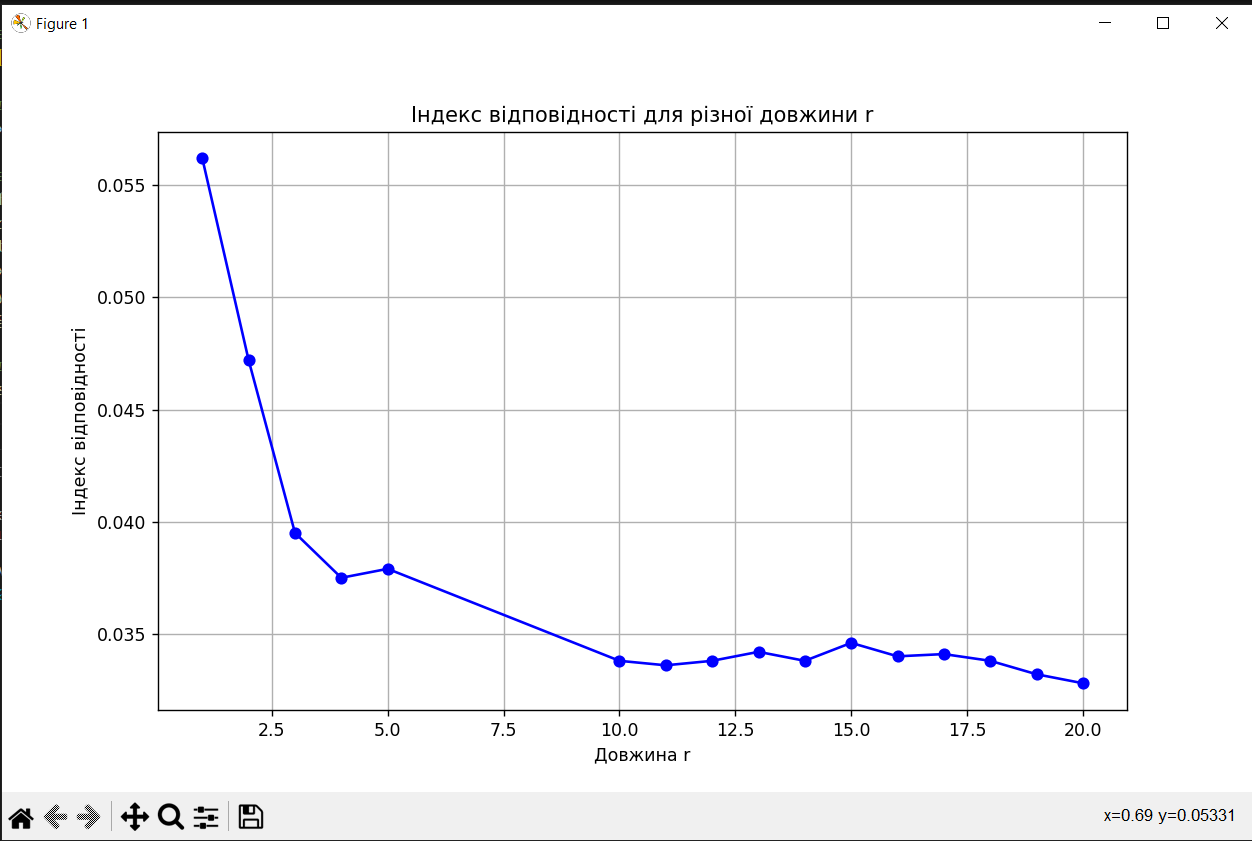
    index = calculate\_index\_of\_coincidence(encrypted\_text)

    print(f"Індекс відповідності для шифру з ключем довжини {key\_length}: {index:.4f}")

**Отримали такі значення:**

|  |  |
| --- | --- |
| **Довжина r** | **Індекс відповідності** |
| Оригінальний текст | 0.0562 |
| 2 | 0.0472 |
| 3 | 0.0395 |
| 4 | 0.0375 |
| 5 | 0.0379 |
| 10 | 0.0338 |
| 11 | 0.0336 |
| 12 | 0.0338 |
| 13 | 0.0342 |
| 14 | 0.0338 |
| 15 | 0.0346 |
| 16 | 0.0340 |
| 17 | 0.0341 |
| 18 | 0.0338 |
| 19 | 0.0332 |
| 20 | 0.0328 |

От швиденько накидала код для відображення діаграми :)



1. Використовуючи наведені теоретичні відомості, розшифрувати наданий шифртекст (Варіант 5).

Тут все має бути доволі просто, рахуємо індекс відповідності для шифрів різної довжини і пошукаємо там найбільші значення.

def calculate\_coincidence\_index(text, key\_length):

    text = text.upper()  # Перетворюємо весь текст у верхній регістр

    text\_len = len(text)

    coincidence\_sum = 0

    for shift in range(key\_length):

        text\_segment = text[shift::key\_length]  # Вибираємо літери з однаковою позицією в ключі

        alphabet\_counts = [text\_segment.count(chr(letter)) for letter in range(1040, 1072)]  # Російський алфавіт

        # Розраховуємо індекс співпадінь для цього сегмента

        segment\_length = len(text\_segment)

        segment\_coincidence = sum(count \* (count - 1) for count in alphabet\_counts) / (segment\_length \* (segment\_length - 1))

        coincidence\_sum += segment\_coincidence

    return coincidence\_sum / key\_length

def main():

    with open(r'C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp2\gogoleva\_fb-12\_cp2\decode.txt', 'r', encoding='cp1251') as file:

        ciphertext = file.read()

    for key\_length in range(1, 36):

        ic = calculate\_coincidence\_index(ciphertext, key\_length)

        print(f"Key Length: {key\_length}, Index of Coincidence: {ic}")

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Вивід:  
Key Length: 1, Index of Coincidence: 0.03532444245066751

Key Length: 2, Index of Coincidence: 0.03709682620655367

Key Length: 3, Index of Coincidence: 0.03535245194471151

Key Length: 4, Index of Coincidence: 0.039793511667390036

Key Length: 5, Index of Coincidence: 0.0354351293936251

Key Length: 6, Index of Coincidence: 0.037052368586566846

Key Length: 7, Index of Coincidence: 0.03522360497899179

**Key Length: 8, Index of Coincidence: 0.04491213203766699**

Key Length: 9, Index of Coincidence: 0.03545025157077616

Key Length: 10, Index of Coincidence: 0.03709763005817014

Key Length: 11, Index of Coincidence: 0.03506214646542888

Key Length: 12, Index of Coincidence: 0.0397888484387092

Key Length: 13, Index of Coincidence: 0.03550919719241092

Key Length: 14, Index of Coincidence: 0.037093872461702884

Key Length: 15, Index of Coincidence: 0.035384371390931875

**Key Length: 16, Index of Coincidence: 0.05539766505382552**

Key Length: 17, Index of Coincidence: 0.035524349460576386

Key Length: 18, Index of Coincidence: 0.037051140206933175

Key Length: 19, Index of Coincidence: 0.03531599104429486

Key Length: 20, Index of Coincidence: 0.03979839848540342

Key Length: 21, Index of Coincidence: 0.035056696947883076

Key Length: 22, Index of Coincidence: 0.03688094981192191

Key Length: 23, Index of Coincidence: 0.03526676001305198

**Key Length: 24, Index of Coincidence: 0.04486292731353409**

Key Length: 25, Index of Coincidence: 0.03531687664602463

Key Length: 26, Index of Coincidence: 0.03731086887465935

Key Length: 27, Index of Coincidence: 0.035247591055245484

Key Length: 28, Index of Coincidence: 0.03969086727168179

Key Length: 29, Index of Coincidence: 0.035584903885058694

Key Length: 30, Index of Coincidence: 0.036928328869868694

Key Length: 31, Index of Coincidence: 0.03527346532158508

**Key Length: 32, Index of Coincidence: 0.05582349044633527**

Key Length: 33, Index of Coincidence: 0.035153977473577375

Key Length: 34, Index of Coincidence: 0.03736123573464328

Key Length: 35, Index of Coincidence: 0.035433945522425535

Зверніть увагу, пікові значення, котрі особливо виділяються – це число 8 і всі його множники(??is it how its called???). Тож дуже підозрюю, що довжина ключа саме 8 символів.

І до речі, значення при довжині ключа 32 дуже близьке до теоретичного значення індексу відповідності російської мови, котрий я порахувала трошки раніше, використовуючи дані про частотність, що ми отримали виконуючи минулу лабораторну.

# Задані частоти літер

letter\_frequencies = {

    'о': 356734, 'и': 291640, 'е': 286557, 'а': 238995, 'т': 184448, 'с': 181963, 'н': 179485, 'в': 162454,

    'л': 142902, 'р': 130840, 'д': 115654, 'м': 105539, 'у': 85409, 'п': 83962, 'к': 83675, 'г': 72282,

    'я': 66468, 'ы': 61791, 'б': 60470, 'з': 52460, 'ь': 50037, 'х': 35422, 'ч': 32969, 'й': 31287, 'ж': 29909,

    'ш': 25191, 'ю': 23439, 'ц': 15517, 'щ': 11862, 'ф': 5835, 'э': 3240, 'ъ': 458

}

# Загальна кількість літер

n = 3208894

# Обчислення індексу відповідності

index\_of\_coincidence = sum(f \* (f - 1) for f in letter\_frequencies.values()) / (n \* (n - 1))

print(f"Індекс відповідності для російської мови: {index\_of\_coincidence:.6f}")

І отримали теоретичне значення індексу:

**Індекс відповідності для російської мови**: 0.057189

Ще один дуже прикольній спосіб визначення довжини ключа з візуалізацією, який я, признаюсь, відверто стирила у якогось чєліка в інтернеті, але трошки адаптувавши під себе.

import matplotlib.pyplot as plt

import numpy as np

# Зчитуємо шифртекст з файлу

file\_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp2\gogoleva\_fb-12\_cp2\decode.txt"

with open(file\_path, "r", encoding="cp1251") as file:

    ciphertext = file.read()

offsets = []

data = []

# Це для позначення піків

threshold = 0

# Проводимо аналіз на знаходження довжини ключа

for offset in range(1, int(len(ciphertext) / 10)):

    matches = 0

    for j in range(len(ciphertext) - offset):

        if ciphertext[j] == ciphertext[j + offset]:

            matches += 1

    threshold += matches

    data.append(matches)

    offsets.append(offset)

# Встановлюємо поріг, вище якого точки позначаються

if offsets:

    threshold = int(1.2 \* threshold / max(offsets))

# Підготовка даних для побудови графіка

x = np.array(offsets)

y = np.array(data)

plt.plot(x, y)

# Позначаємо високі точки та їх значення по осі x

max\_indices = np.argpartition(data, -5)[-5:]  # Знаходимо індекси 5 найбільших значень

for i in max\_indices:

    plt.scatter(offsets[i], data[i], color='red')  # Позначаємо їх червоним кольором

    plt.text(offsets[i], data[i], str(offsets[i]), fontsize=12, ha='center', va='bottom')

# Як часто мають бути лінії на осі x?

if offsets:

    stepsize = int(max(offsets) / 10)

    plt.xticks(np.arange(0, max(offsets), step=stepsize))

# Підписи для вісей та заголовок

plt.xlabel('Зсуви')

plt.ylabel('Збіги')

plt.title('Пошук піків, які є кратними деякій довжині ключа')

# Встановлюємо сітку

plt.grid(True)

# Відображення графіка

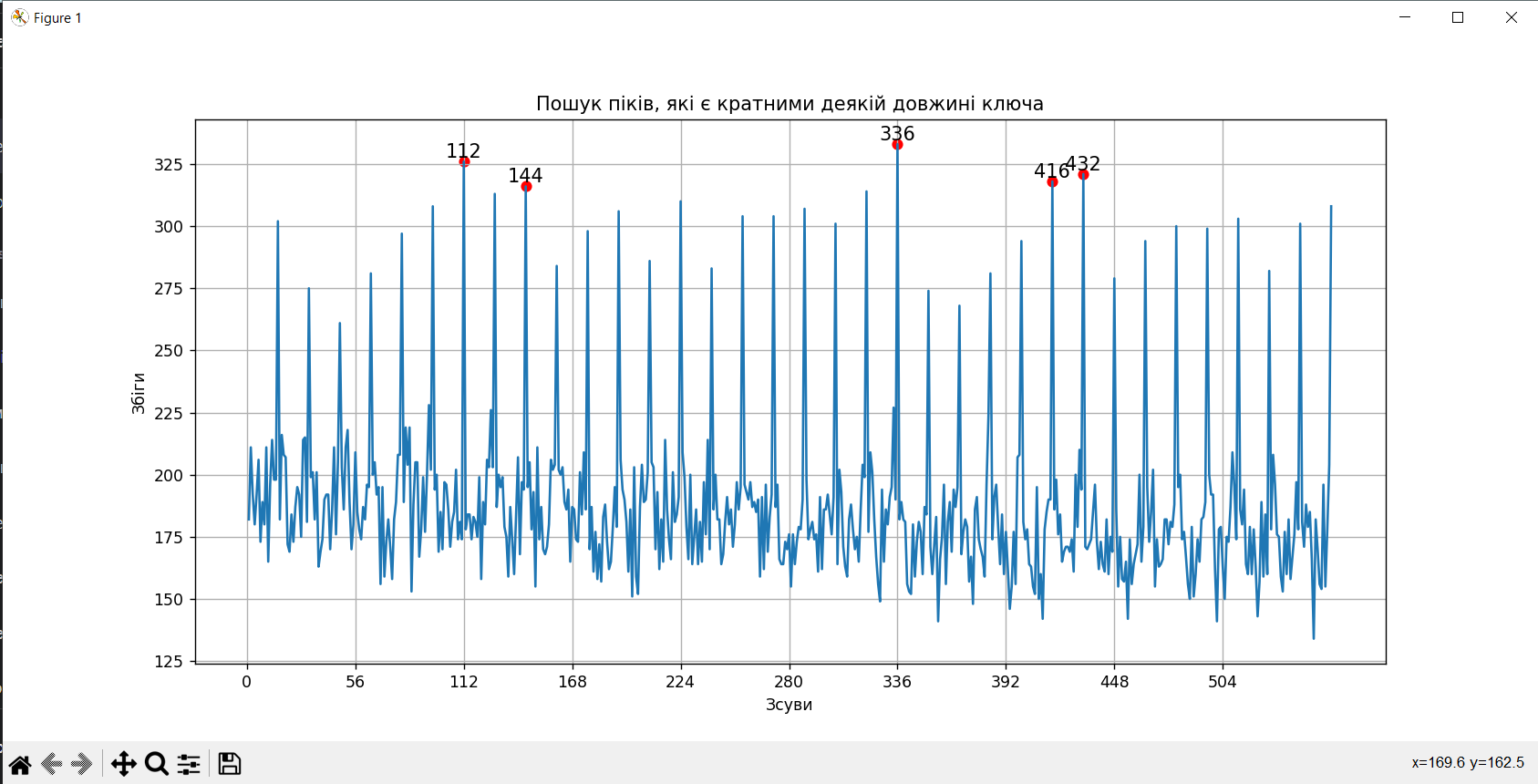
plt.show()

Код використовує дуже прикольний метод **автокореляції**

Він перевіряє, наскільки схожі підстроки тексту на різних зсувах. І якщо довжина ключа правильно визначена, то автокореляція буде мати максимальні значення відповідних довжин ключа.

Ось як це працює:

* Кожен можливий зсув від 1 до певної максимальної довжини аналізується.
* Якщо літери тексту збігаються на певній відстані (зсуві), то значення збігів збільшується.
* Потім проводиться аналіз цих значень збігів для різних зсувів.
* Якщо є довжина ключа, яка відповідає найбільшим значенням збігів, то ця довжина ключа визначається як можливий кандидат на довжину ключа.



Тобто довжина ключа кратна цим числам, які спільні дільники у 112 144 336 416 432? Правильно: 1, 2, 4, 8. А про що це каже? Що і використовуючи інший метод ми зайвий раз довели, що **довжина ключа - 8 символів**.

Наступний крок, розділити текст на 8 блоків, але особливим чином.   
У перший блок додається перша, дев’ята і т.д. літера з кроком 8 до кінця файлу. У другий блок так само, але починаючи з другої літери. І так всі 8 блоків.

Так ми отримаємо блоки літер, котрі закодовані першою/другою/…/восьмою літерою ключа. Це все робиться отаким невеличким кодом.

Я все це зробила, витратила чотири години життя, щоб побачити, що ключ недостатньої довжини і it doesn`t make any sense :))))))) Не хочу, щоб тут були всі ці страждання, тому я переробила код під 16 символів ключа і ДУЖЕ СПОДІВАЮСЬ, що цього разу я вгадала з довжиною.

with open("C:\\Users\\Polya\\Desktop\\KPI\\crypto\\crypto-23-24\\cp2\\gogoleva\_fb-12\_cp2\\decode.txt", "r", encoding="cp1251") as file:

    text = file.read()

block\_count = 16

blocks = [""] \* block\_count

block\_index = 0

for char in text:

    blocks[block\_index] += char

    block\_index = (block\_index + 1) % block\_count

for i, block in enumerate(blocks):

    print(f"Блок {i}: {block}")

Тепер ми отримали менші тексти, але розбиті на шифри цезаря. Додамо коду і глянемо на літери, котрі зустрічаються найчастіше у блоках.

Код:

from collections import Counter

with open("C:\\Users\\Polya\\Desktop\\KPI\\crypto\\crypto-23-24\\cp2\\gogoleva\_fb-12\_cp2\\decode.txt", "r", encoding="cp1251") as file:

    text = file.read()

block\_count = 16

blocks = [""] \* block\_count

block\_index = 0

for char in text:

    blocks[block\_index] += char

    block\_index = (block\_index + 1) % block\_count

for i, block in enumerate(blocks):

    print(f"Блок {i}: {block}")

    letter\_counts = Counter(block)

    most\_common\_letter = letter\_counts.most\_common(1)[0][0]

    print(f"Найчастіше літера: {most\_common\_letter}\n")

Вивід:  
фиьмлдццфждмцххфгтогхйийжпхнпцгапьтсжпмлперсстпддхцелйймиучфхтсфйййрзтсхпойижйцфлпйтеищщтзтьтпфтмрйагщжйтпхлрйрмпмстпчцйрвпмыжрйзмтжтпдтфгсжтжтпйсдбяйполтпщжссцлхгэй

Найчастіше літера: т

Блок 1: ушзхиеьзбнрбйхуекцксштдкчънксйьнртнатнургейсчхклзцйнахуцтубкукзлхемутктгуфпшлшурбзфхэккеубкшчхтзццзррпнъуушхфдкууеуткурмтйечтбтйатгппсйхнчндунйнцнчлрзрубеттрепншйхфухбэшеуусцрикуесснсхжз

нруттцкшеуръбнбеппуцчхтхуъунштртптйуйеоткауаамнэрпкттзхдшуцукфнкйтэнеюенпнчкхикеенроахфуцуптьучхдмржчзэфпшчефрфчкктжуззйьчнмеуефепуъздззецунсаозтутчззецдзчткаунеррнь

Найчастіше літера: у

Блок 2: шырпыцрчързвлротыээээльщнънчъпушлжхфуьшзхршжкшуурэлэмнчцлфэорцыпжцлпщьупылытрэущьтщюуэхэкршпэкуиэурщьшэщшхмлытмээнсроъулщашжщшнлчюныэърврщшппкюъэшуррьлъьдушюцюэшщуыьрэюпуэъррщщнпьурцнущр

мумлууышьиьсьцюшьлшлрющрцщмпвжууржьннншлшмнффштрщщыщуьшщдъшпьыктлушбузкчлфьблщыьнщрэцщщээплррвыжчлщжзиуцшслоылыршщжжниьрстсньушыглмлзшущщчъэрвхшшфшупрнхшлщюычсъцжьрш

Найчастіше літера: ш

Блок 3: нбшббцъцюцяуфпьтуьнубнщтцюяоьоыаруцэхацыьацупьтаырыкйоэьеыбьяоьйыьпщякъоьщоьаяхрнушшакэуысуюоэмацщзшнокеэюочотышюутыьбхрсьцюрййщощтйцьщуяюоьйаъюрьцаыурьшцнйфцауьпаоутуэыыьюепсцэмобыьгыщт

йрюъшчуьоанцщояуахерыфэбйтюоцуяюыъеуоуумцуьэъоузпщняцщьэцоьыацроннубряшэшцшэзцоэуэыьцптыющбчрщучнэрщячяоууыоьъцщцюгщщчуыооыуйгцояъююнбщчыуькеыщуйхьяуощцъающцгцьйжняй

Найчастіше літера: у

Блок 4: эшлььшоыррэцршврчцчашцющрщэцхтлршнащруожушчожрщзшлшацыщнлрытэцъррнщуэнлэозьышэшщъчщлжшщыущцщхщтуъурлупсрыюощнрщлкнуцлэхрщпррыакщшьщнньшнцоъмщщлрлнупучхьщааруьжцншрьэьцщрэчлэкщьщспьшфллзр

эщлплмщнпуъэлчържллгжзыцъурсцсзлрбуцпмтшкьцылворизцэьроуфцрщщвлоъюфьтэщлиьущлъьщпущвьшпщлкъщурэънырулмзэъмщмэуьзкпщщрмшушъуьънкнщулвмцщъцьэтлщумфцощцшрфщзлшцлъгэшьюф

Найчастіше літера: щ

Блок 5: хихръимноокцкгруцдцтгноуфщшцихфшцгшрфхцнэириящйнццгырихикцгхццнккръхрлънрюхцикнршусксцкнртднркнмынктхцръроххрщстхитнпщцшмрцпнцкфрзэаиннхрцриъфъмццхирриншуххъчлнкгуиръцщснкщитцшфрпъцкшмах

цчбужгйрйфцнлрнчяуърмнзтцъпнцрцщщкърюншрттдихнцрътрдимцыикклшрхицчцъзикъъухщнггуныщхцгиахпшэмхяцуитпфркдшхкгрхъхцыччхрнзцштъцуннщтпнихтннъзихфхгзццйнцхфаулнщуцныгтфм

Найчастіше літера: ц

Блок 6: явцоюуцыхцсуюэвцюывгающщюгцьысгццвцвююьтувягщябвцбжгуэщщйбссьвьвцуяютьцьэцмиащвьщпюсавшьхсшюьцэдгюсщщхшыгсцжьнювсптфсгэйцьзсэаьаьующэьврыугуяянааъйчыжэьясцяццхъмцнхгущьабясжююсяжсбэюиьцщ

ыясркьщюцуаьсюжяывцвухэсвняатгазяцмысвряяяювхюбюяягщэвдэутцяяюсвабжсгбсщяцсгггвяюэюявцюццхсбзяцуруяямюфццящуумцмхвщбююжсэсяювсчхуяюуфсдафсюацэягхдымяуюсдщяшысхьхэуця

Найчастіше літера: я

Блок 7: уцрабупбущыубьщыцьььььаъоьемцуояыьщшьььууапцяяьаттебьцъяутяеккмнтьаьюощмэьчкюцацщэьтщуюрбшнуцюыгкуыъяоыььмьбьщоуэзошюрбущьуруцуьцоьъцшшъэщшцэфьююыоуьоцшрсърщдуцящыэуььуьужтбнххзъпубуууьч

ащаюцьаухьйужуьтонщкбуььщэъююуцрьарцауятаыьооцбуъеюыьаэооьтхыбцоооююьжеуасцуящшфцпояаяыыыьроуящящцыпгьюхясщобыегбябуьйьпхррьщтуыьюьыуеюушцбьпушцьъьаэййяьжтоюагшожьюф

Найчастіше літера: ь

Блок 8: емпфясьмгйжлуужйкгвоутжьосбенулойуйткжгишйъоууепмфуотйбтдфлжзпелбиткпиэшжещжйзйгяпепбзпущрсвйпьпигйвууэонъшупйедсйдпупйобсмжофотрмгйгжпйсйбуйосййжмпоссбрбожййлдужьсмсшуттмвзибфэпьнгегзцу

ббэжцобшитукжфнднтэптеружпипжмфжвпбкэелопжреуалргжфужгпоомпеоигнмгббджзкслграйожкьйубрьййсмогвпжтмбмтгббупйжршжтслнтносббйлгжжепйпнжъбпмпнмаьтвжгоийпккувэбньпггдпиле

Найчастіше літера: п

Блок 9: укатцыакяьябюаынцыуоьаъъуоууашцйяйсъэцьтыщуоььущняьйыыыаьяьыупцьмрбряоыуюууъъупйтяьюауяоцюоьыцъроъуйокауыцубшжьььчпсорбцшоьтыъыуюущяюьщццяъкбьоыьтщпаеооюжьюьобьюфууноцюуьцйругыыщщцхюыору

шсяжюуауоэшяъроьццыыаоьэегоххцъщюъщбрцоьюеюьъбгьящтухцаоткъьуояарщыыьсоыьоыуррьысщркрцгумьуйяьръньюцээфаопсаюуыцъцьубоьстщоуьыычъаршуяшкэояящаоюцьмнщъгууепуаъойтюкцк

Найчастіше літера: ь

Блок 10: ыэъюбюювхэфбхюэягшэбьмгяюевщюрэьтфшлшбъэюрягхпвмсбуэюхшрыбашфлящйлвюрфюыбвыехэабэыбююыыъшрчтрбтэъхчэыхрзхьээрмсфвелютрфоъбщхююлыръхпрэмысяшхьсбшстргахчвшгуэсыббюряфьббгвттыюьюрхююыппштр

ьбрпхртхббюшюбхбвюыюбтшбшхгятраюшрысзвбчуюэшьхцюфъююафхабхэбашсъюяхлшшюэрчхлэбюушфююоыбшъютэбюахтбщгзюхфаэффяртшыреьзбъибхгъзсшхфшрхгалтэюипююхурвуыюъхрайхрвлсвехюуся

Найчастіше літера: ю

Блок 11: ььырауыбсьурысйотъыубщэюпцяшщшуыубчяашюйэмюшэаууьшюоррцюмшуцоеьшцжяяуоъьайшоюзоъьуасшцуьхррюеэщйояоцеъшырэкьхэыыцощэяроььяяыэшгшяяыэхбшойцбзьбжъюцпяьрбоыхьйькщацыющыяатцоьфхюаыятсярауже

рэщшыхгщаащхррыэьакюыьэющсаоцсошрхооцьбоьуйяыяуаььряуьеуьрцаьхььсьыхнъропьзчырсьууьхаушъюэкцьяышуацжуяюоьыууьруцбыэюкцюцнюцяугчропщяыоцпцтцэъыюояьькэбеысоюсыуьцьягбуь

Найчастіше літера: ь

Блок 12: ьфчаъэъвагэняачэчтаэшрачмюсаъъгъдзъэтееявлатвчэгэъчгяапъцачьфяйеччдбдшефяыаэфъяабйтаъйэююъчтчатчщдбечеьаваэфнаасфяъочкййяэчъаефчттячешъфэвцчьцътчшндэяцюттйчвяечэьътаэъадэфтатчсьиэскоюъч

нвэфъятчячоцэайъфдятсэвчъаещлтгебяхвдвшэбуюфчдесщвчбгввуясиффеэваэътюфпуэфчюачацфьбютяъчттсчгднтвтдъяачячаяагяйщбъачбхнчьшчдваъяяэфчъщввияяаютаьдыбьъъбутэчфабчэцчаггв

Найчастіше літера: ч

Блок 13: адешъаоъаиндтяьншъсмысдьмьтяещядынпнлюунтнвъышйяйввтыояытцпыъияшэъпхютъттппньъъээтяцатхютшдфюпюынъыэшьнттклхпююъъсюмртътгтямсээгсъиэстщчмянхатэяяхянммныспхтаирцххорпняптхчсщфшпыынчхыатъ

еннкмыэциъфтнхнсхытюшйхсшюъатщяэыынитысмэихыоъьшэтчычытшряаыйдттътмюхьцнлыьыьсщтюаэычиппююьпшнцчыэнясоыююрхопитъэпсеыъпфыхрнъспыххъшюпнытмньтьснпюыхюрэнъыффьшпфасщячн

Найчастіше літера: т

Блок 14: ркббшьауцмртфуучднккжцчдчкюубфечмзклюцнпждунцбткэферсаужопччуснбнтнчдтепсфачхаупдибхчрмутучшскчжтумуухпцйчжперчкахфтйиухшйнкфнензйоцезнестцйхчккксбттсрьркцийкеъцнуеужбарцшктшсцрззтцфтьу

кзиозиеъоьуэйсрмрикфцтрнцчеъфккнцфээроктерцккуректфцлзмкхаштднзукмзкжужийюурхшеркхкллфцэфтутузфудэпкктйпшудхусхееейкшеееззурнеморлкнаммтфррхьхутурйтеуурбчфйхуаунзцку

Найчастіше літера: у

Блок 15: ефлцрбфъсхчдчхндзиыыдусыцщощкщфчйоассыыщдцнюыцчъсчыоламчсьччълаыыдойрокъцщтхсшойхчсйочйъсьчхчнифцйннлччфсчсйфьсътьчийчхошчанчфрэощчшффюхчдычошцфцчхчйсчоонфччрхьеущфоснюсеъцснцоешцсесоор

шойкчфуьшслцйссйъчччччзшыйхччъщфодшсемцолчйцъмдмлоччсехцитфйлфйклщцнйхсъйончовъйюснцочосщчщолсчыысафъдочыъьоснчалцйцъфзлчйкчоъсцйцъсцмнрчъейцсльщзнйцщъдяучччыоцрйозц

Найчастіше літера: ч

Ну, тепер, якщо припустимо, що кожна з визначених літер це закодована шифром цезаря літера «о», по суті, маємо зсув для кожного з блоків і можемо вручну знайти ключ 😊   
Спробувала зробити це кодом

from collections import Counter

with open("C:\\Users\\Polya\\Desktop\\KPI\\crypto\\crypto-23-24\\cp2\\gogoleva\_fb-12\_cp2\\decode.txt", "r", encoding="cp1251") as file:

    text = file.read()

block\_count = 16

blocks = [""] \* block\_count

block\_index = 0

for char in text:

    blocks[block\_index] += char

    block\_index = (block\_index + 1) % block\_count

result = ""

for i, block in enumerate(blocks):

    # Рахуємо частоту літер у блоку

    letter\_counts = Counter(block)

    most\_common\_letter = letter\_counts.most\_common(1)[0][0]  # Найчастіше зустрічаючася літера

    # Визначаємо номер літери в російському алфавіті

    alphabet = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

    letter\_index = alphabet.index(most\_common\_letter)

    # Віднімаємо 14 (номер літери "О") і виводимо відповідну букву

    decoded\_letter\_index = (letter\_index - 14) % len(alphabet)

    decoded\_letter = alphabet[decoded\_letter\_index]

    result += decoded\_letter

print(result)

Але вивід вийшов… ну… математично правильним, але без сенсу

І так як я не придумала, як запрограмувати код так, щоб він розрізняв, чи слово, котре він створив має сенс чи ні, то вирішила зробити вручну. І я знаю, що це не працюватиме з дуже великими ключами наприклад, але поки мої знання «всьо» ☹

Буква «О» - 14 у списку, тож шукатимемо зсув віднімаючи індекс літери від «О». Якщо виходитиме щось без сенсу, пробуватимемо з «Е»- 5 символ. І т.д.





Т-18 18-14=4 **Д**

У-19 19-14=5 **Е**

Ш-24 24-11=13 **Л**

У-19 19-5=14 **О**

Щ-25 25-14=11 **Л**

Ц-22 22-14=8 **И**

Я-31 31-14=17 **С**

Б-1 1-14 **О**

П-15 15-14=1 **Б**

Ь-28 28-14=14 **О**

Ю-30 30-14=16 **Р**

Ь-28 28-14=14 **О**

Ч-23 23-5=18 **Т**

Т-18 18-5=13 **Н**

У-19 19-14=5 **Е**

Ч-23 23-14=9 **Й**

**КЛЮЧ: ДЕЛОЛИСОБОРОТНЕЙ**

(це було боляче not gonna lie)

Ну і тепер ми нарешті можемо написати фінальний код, котрий декодуватиме весь зашифрований текст нашого варіанту! І виглядатиме він от так

def vigenere\_decrypt(ciphertext, key):

    decrypted\_text = ""

    key\_length = len(key)

    for i, char in enumerate(ciphertext):

        if char.isalpha():

            shift = ord(key[i % key\_length].lower()) - ord('а')

            if char.islower():

                decrypted\_char = chr(((ord(char) - ord('а') - shift) % 32) + ord('а'))

            else:

                decrypted\_char = chr(((ord(char) - ord('А') - shift) % 32) + ord('А'))

        else:

            decrypted\_char = char

        decrypted\_text += decrypted\_char

    return decrypted\_text

# Зчитуємо шифртекст з файлу

file\_path = "C:\\Users\\Polya\\Desktop\\KPI\\crypto\\crypto-23-24\\cp2\\gogoleva\_fb-12\_cp2\\decode.txt"

with open(file\_path, "r", encoding="cp1251") as file:

    ciphertext = file.read()

# Ключ для розшифрування

key = "делолисоборотней"

# Розшифровуємо текст

decrypted\_text = vigenere\_decrypt(ciphertext, key)

# Зберігаємо розшифрований текст у файл

output\_file\_path = "C:\\Users\\Polya\\Desktop\\KPI\\crypto\\crypto-23-24\\cp2\\gogoleva\_fb-12\_cp2\\decoded\_text.txt"

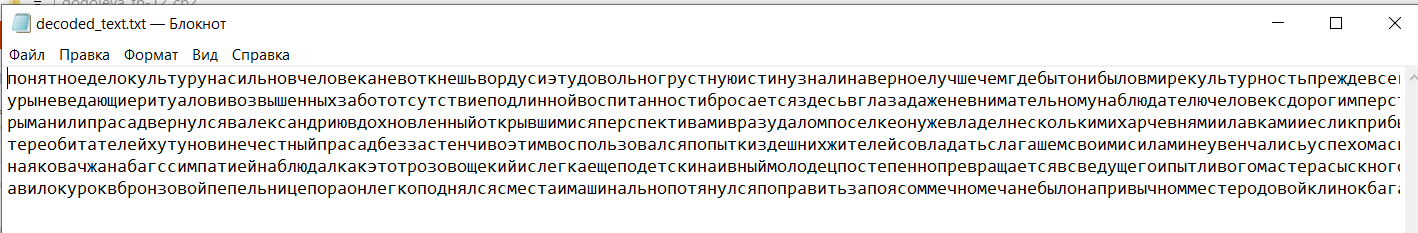
with open(output\_file\_path, "w", encoding="cp1251") as output\_file:

    output\_file.write(decrypted\_text)

# Виводимо розшифрований текст

print("Розшифрований текст збережено в файлі decoded\_text.txt")





Ще ніколи в житті я не була так щаслива бачити набір літер на екрані, особливо російською.

**Висновки**

По-перше, я нарешті повністю зрозуміла важливість і необхідність того, що ми робили у першій лабі, бо виявляється частотний аналіз це насправді дуже важлива штука і я дізналась про це на практиці. Також довелось навчитись ламати шифр Віженера знаючи лише те, що текст зашифрований текстом Віженера, я насправді все ще здивована, це було дуже цікаво, побачити як математика, логіка і частотний аналіз у купі можуть розв’язати шифр, котрий кілька століть вважався незламним, за лічені секунди.