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

0. 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.

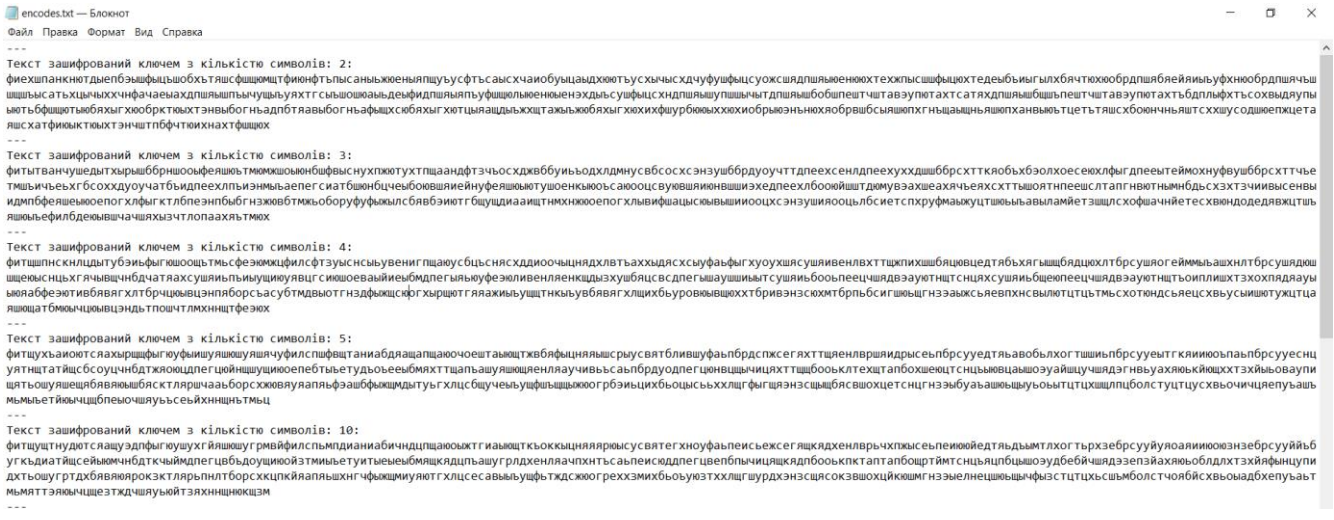
$$y_i = (x_i + k_{i \bmod r}) \bmod m,\ i = \overline{0,n}.$$

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.



2. 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.

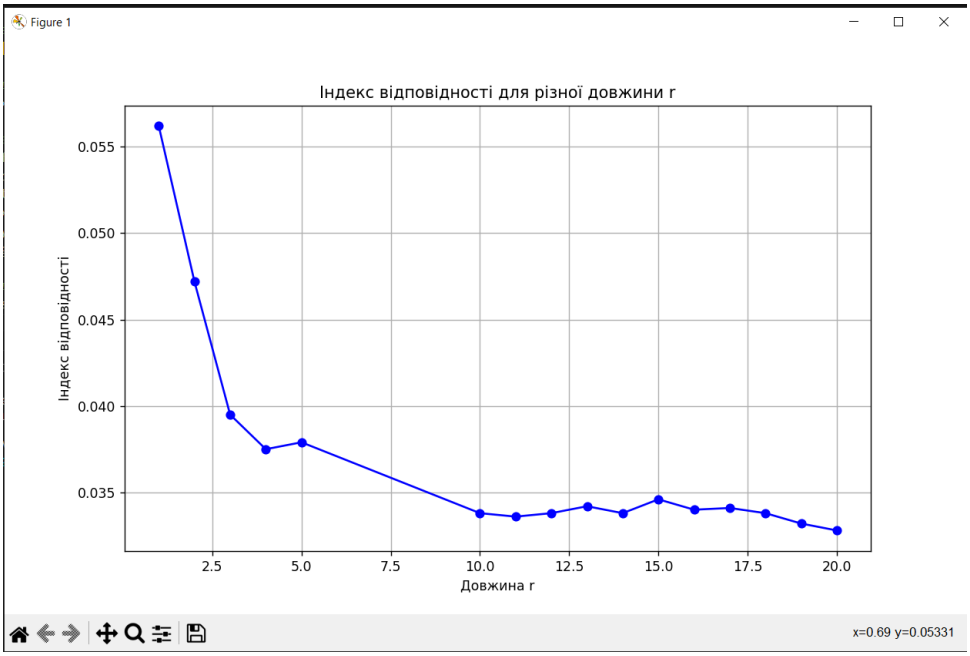
```
def calculate_index_of_coincidence(text):
    text = ''.join(filter(str.isalpha, text))
    text = text.lower()

    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 :)



3. 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: 0.037310868874659 35

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?
```

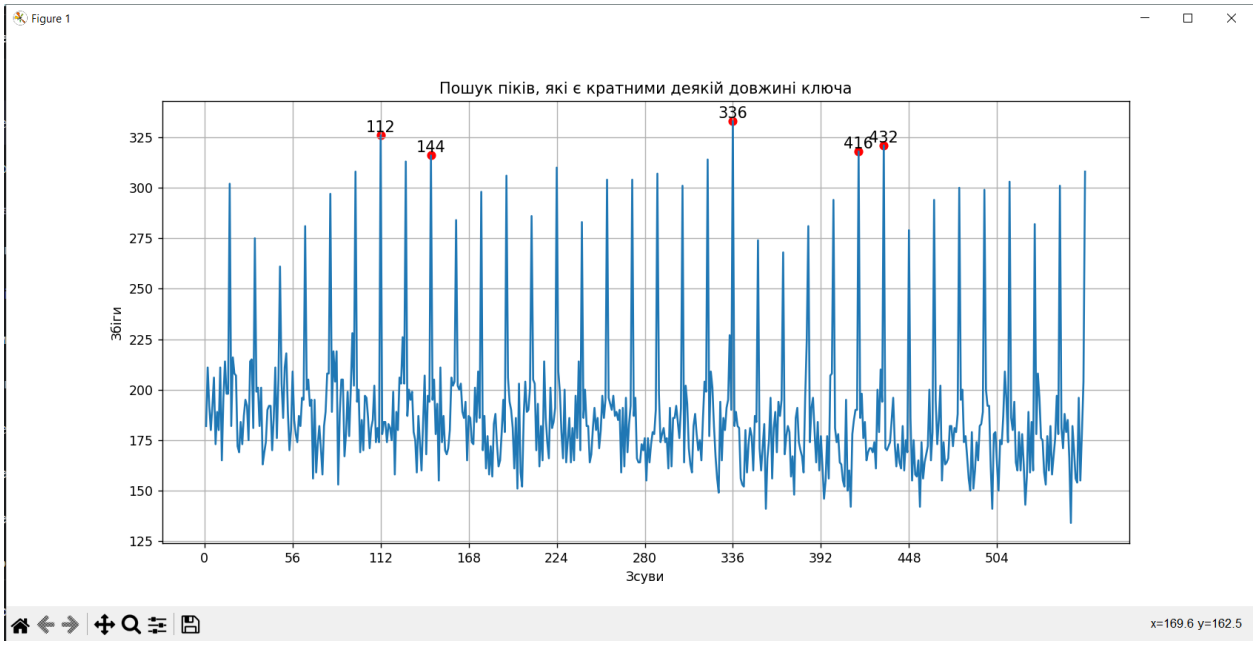
```
if offsets:
    stepsize = int(max(offsets) / 10)
    plt.xticks(np.arange(0, max(offsets), step=stepsize))
# Підписи для вісей та заголовок
plt.xlabel('Зсуви')
plt.ylabel('Збіги')
plt.title('Пошук піків, які є кратними деякій довжині ключа')

# 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:
фиьмлдццфждмцххфгтогхйийжпхнпцгапьтсжпмлперсстпддхцелййміучфхтс
фййірзтсхпойижйцфлпйтеищщтзтьтпфтмрйагщжйтпхлрйрмпмстпчцйрвпмы
жрйзмтжтпдтфгсжтжтпйсдбяйполтпщжссцлхгэй

Найчастіше літера: т

Блок 1:
ушзхиеьзбнрбйхуекцксштдкчънксйьнртнатнургейсчхклзцйнахуцтубкукзлхем
утктгуфпшлшурбзфхэккеубкшчхтзццзррпнъуушхфдкууеуткурмтйечтбтйатгп
псйхнчндунйнцнчлрзрубеттрепншйхфухбэшеуусцрикуесснсхжз

нруттцкшеуръбнбеппуцчхтхуъунштртптйуйеоткауаамнэрпкттзхдшуцукфнкй
тэнеюенпнчкхикеенроахфуцуптьучхдмржчзэфпшчефрфчкктжуззйьчнмеуефе
пуъздззецунсаозтутчззецдзчткаунеррнь

Найчастіше літера: у

Блок 2:
шырпыцрчързвлротыээээльщнънчъпушлжхфуьшзхршжкшуурэлэмнчцлфэорц
ыпжцлпщьупылытрэущьтщюуэхэкршпэкуиэурщьшэщшхмлытмээнсроъулща
шжщшнлчюныэърврщшппкюъэшуррьлъьдушюцюэшщуыьрэюпуэъррщщнпь
урцнущр

мумлууышьиьсьцюшьлшлрющрцщмпвжууржьннншлшмнффштрщщыщуыш
щдъшпьыктлушбузкчлфьблщыьнщрэцщщээплррвыжчлщжзиуцшслоылыршщ
жжниьрстсньушыглмлзшущщчъэрвхшшфшупрнхшлщюычсъцжьрш

Найчастіше літера: ш

Блок 3:
нбшббцъцюцяуфпьтуьнубнщтцюяоьоыаруцэхацыьацупьтаырыкйоэьеыбьяоь йыьпщякъоьщоьаяхрнушшакэуысуюоэмацщзшнокеэюочотышюутыьбхрсьцю рййщощтйцьщуяюоьйаъюрьцаыурьшцнйфцауьпаоутуэыыьюепсцэмобыьгыщ т

йрюъшчуьоанцщояуахерыфэбйтюоцуяюыъеуоуумцуьэъоузпщняцщьэцоьыац роннубряшэшцшэзцоэуэыьцптыющбчрщучнэрщячяоууыоьъцщцюгщщчуыоо ыуйгцояъююнбщчыуькеыщуйхьяуощцъающцгцьйжняй

Найчастіше літера: у

Блок 4:
эшлььшоыррэцршврчцчашцющрщэцхтлршнащруожушчожрщзшлшацыщнлр ытэцъррнщуэнлэозьышэшщъчщлжшщыущцщхщтуъурлупсрыюощнрщлкнуц лэхрщпррыакщшьщннышнцоъмщшлрлнупучхьщааруьжцншрьэьцщрэчлэкщь щспьшфллзр

эщлплмщнпуъэлчържллгжзыцъурсцсзлрбуцпмтшкьцылворизцэьроуфцрщщв лоъюфьтэщлиьущлъьщпущвьшпщлкъщурэънырулмзэъмщмэуьзкпщщрмшуш ъуьънкнщулвмцщъцьэтлщумфцощцшрфщзлшцлъгэшьюф

Найчастіше літера: щ

Блок 5:
хихръимноокцкгруцдцтгноуфщшцихфшцгшрфхцнэириящйнццгырихикцгхцц нккръхрлънрюхцикнршусксцкнртднркнмынктхцръроххрщстхитнпщцшмрцп нцкфрзэаиннхрцриъфъмццхирриншуххъчлнкгуиръцщснкщитцшфрпъцкшмах

цчбужгйрйфцнлрнчяуърмнзтцъпнцрцщщкърюншрттдихнцрътрдимцыикклшр хицчцъзикъъухщнггуныщхцгиахпшэмхяцуитпфркдшхкгрхъхцыччхрнзцштъц уннщтпнихтннъзихфхгзццйнцхфаулнщуцныгтфм

Найчастіше літера: ц

Блок 6:
явцоюуцыхцсуюэвцюывгающщщюгцьысгццвцвююьтувягщябвцбжгуэщщйбссь вьвцуяютьцьэцмиащвьщпюсавшьхсшюьцэдгюсщщхшыгсцжьнювсптфсгэйць зсэаьаьующэьврыугуяянааъйчыжэьясцяццхъмцнхгущьабясжююсяжсбэюиьц щ

ыясркьщюцуаьсюжяывцвухэсвняатгазяцмысвряяяювхюбюяягщэвдэутцяяюсв абжсгбсщяцсгггвяюэюявцюццхсбзяцуруяямюфццящуумцмхвщбююжсэсяювс чхуяюуфсдафсюацэягхдымяуюсдщяшысхьхэуця

Найчастіше літера: я

Блок 7:
уцрабупбущыубьщыцьььььаъоьемцуояыьщшььууапцяяьаттебьцъяутяеккмнт ьаьюощмэьчкюцацщэьтщуюрбшнуцюыгкуыъяоыььмьбьщоуэзошюрбущьуру цуьцоъъцшшъэщшцэфьююыоуьоцшрсърщдуцящыэуььуьужтбнххзъпубуууьч

ащаюцьаухьйужуьтонщкбуььщэъююуцрьарцауятаыьооцбуъеюыьаэооьтхыбц ооооююьжеуасцуящшфцпояаяыыыьроуящящцыпгьюхясщобыегбябуьйьпхррь щтуыьюьыуеюушцбьпушцьъьаэййяьжтоюагшожьюф

Найчастіше літера: ь

Блок 8:
емпфясьмгйжлуужйкгвоутжьосбенулойуйткжгишйъоууепмфуотйбтдфлжзпел биткпиэшжещжйзйгяпепбзпущрсвйпьпигйвууэонъшупйедсйдпупйобсмжофо трмгйгжпйсйбуйосййжмпоссбрбожййлдужьсмсшуттмвзибфэпьнгегзцу

ббэжцобшитукжфнднтэптеружпипжмфжвпбкэелопжреуалргжфужгпоомпеои гнмгббджзкслграйожкьйубрьййсмогвпжтмбмтгббупйжршжтслнтносббйлгжж епйпнжъбпмпнмаьтвжгоийпккувэбньпггдпиле

Найчастіше літера: п

Блок 9:
укатцыакяьябюаынцыуоьаъъуоууашцйяйсъэцьтыщуоььущняьйыыыаьяьыупц ьмрбряоыуюууъъупйтяьюауяоцюоьыцъроъуйокауыцубшжьььчпсорбцшоьты ъыуюущяюьщццяъкбьоыьтщпаеооюжьюьобьюфууноцюуьцйругыыщщцхюы ору

шсяжюуауоэшяъроьццыыаоьэегоххцъщюъщбрцоьюеюьъбгьящтухцаоткъьуо яарщыыьсоыьоыуррыысщркрцгумьуйяьръньюцээфаопсаюуыцъцьубоьстщоуь ыычъаршуяшкэояящаоюцьмнщъгууепуаъойтюкцк

Найчастіше літера: ь

Блок 10:
ыэъюбюювхэфбхюэягшэбьмгяюевщюрэьтфшлшбъэюрягхпвмсбуэюхшрыбаш

флящйлвюрфюыбвыехэабэыбююыыъшрчтрбтэъхчэыхрзхьээрмсфвелютрфоъ бщхююлыръхпрэмысяшхьсбшстргахчвшгуэсыббюряфьббгвттыюьюрхююып пштр

ьбрпхртхббюшюбхбвюыюбтшбшхгятраюшрысзвбчуюэшьхцюфъююафхабхэ башсъюяхлшшюэрчхлэбюушфююоыбшъютэбюахтбщгзюхфаэффяртшыреьзб ъибхгъзсшхфшрхгалтэюипююхурвуыюъхрайхрвлсвехюуся

Найчастіше літера: ю

Блок 11:
ььырауыбсьурысйотъыубщэюпцяшщшуыубчяашюйэмюшэаууьшюоррцюмш уцоеьшцжяяуоъьайшоюзоъьуасшцуьхррюеэщйояоцеъшырэкьхэыыцощэяроь ьяяыэшгшяяыэхбшойцбзьбжъюцпяьрбоыхьйькщацыющыяатцоьфхюаыятсяр ауже

рэщшыхгщаащхррыэьакюыьэющсаоцсошрхооцьбоьуйяыяуаььряуьеуьрцаьхь ьсьыхнъропьзчырсьууьхаушъюэкцьяышуацжуяюоьыууьруцбыэюкцюцнюцяу гчропщяыоцпцтцэъыюояьькэбеысоюсыуъцьягбуь

Найчастіше літера: ь

Блок 12:
ьфчаъэъвагэняачэчтаэшрачмюсаъъгъдзъэтееявлатвчэгэъчгяапъцачьфяйеччдб дшефяыаэфъяабйтаъйэююъчтчатчщдбечеьаваэфнаасфяъочкййяэчъаефчттяче шъфэвцчьцътчшндэяцюттйчвяечэьътаэъадэфтатчсьиэскоюъч

нвэфъятчячоцэайъфдятсэвчъаещлтгебяхвдвшэбуюфчдесщвчбгввуясиффеэваэ ътюфпуэфчюачацфьбютяъчттсчгднтвтдъяачячаяагяйщбъачбхнчьшчдваъяяэф чъщввияяаютаьдыбьъъбутэчфабчэцчаггв

Найчастіше літера: ч

Блок 13:
адешъаоъаиндтяьншъсмысдьмьтяещядынпнлюунтнвъышйяйввтыояытцпыъи яшэъпхютъттппньъъээтяцатхютшдфюпюынъыэшьнттклхпююъъсюмртътгтям сээгсъиэстщчмянхатэяяхянммныспхтаирцххорпняптхчсщфшпыынчхыатъ

еннкмыэциъфтнхнсхытюшйхсшюъатщяэыынитысмэихыоъьшэтчычытшряаы йдттътмюхьцнлыьыьсщтюаэычиппююьпшнцчыэнясоыююрхопитъэпсеыъпф ыхрнъспыххъшюпнытмньтьснпюыхюрэнъыффьшпфасщячн

Найчастіше літера: т

Блок 14:
ркббшьауцмртфуучднккжцчдчкюубфечмзклюцнпждунцбткэферсаужопччусн
бнтнчдтепсфачхаупдибхчрмутучшскчжтумуухпцйчжперчкахфтйиухшйнкфне
нзйоцезнестцйхчккксбттсрьркцийкеъцнуеужбарцшктшсцрззтцфтьу

кзиозиеъоьуэйсрмрикфцтрнцчеъфккнцфээроктерцккуректфцлзмкхаштднзукм
зкжужийюурхшеркхкллфцэфтутузфудэпкктйпшудхусхееейкшеееззурнеморл
кнаммтфррхьхутурйтеуурбчфйхуаунзцку

Найчастіше літера: у

Блок 15:
ефлцрбфъсхчдчхндзиыыдусыцщощкщфчйоассыышдцнюыцчъсчыоламчсьччъ
лаыыдойрокъцщтхсшойхчсйочйъсьчхчнифцйннлччфсчсйфьсътьчийчхошчан
чфрэощчшффюхчдычошцфцчхчйсчоонфччрхьеущфоснюсеъцснцоешцсесоор

шойкчфуьшслцйссйъчччччзшыйхччъщфодшсемцолчйцъмдмлоччсехцитфйлф
йклщцнйхсъйончовъйюснцочосщчщолсчыысафъдочыъьоснчалцйцъфзлчйкчо
ъсцйцъсцмнрчъейцсльщзнйцщъдяучччыоцрйозц

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
зустрічаюча 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

```
PI\crypto\crypto-23-24\cp2\gogoleva_fb-12_cp2\decoder.py'
еёкёлисоворойеёй
```

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.

код	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
исх. текст	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П

код	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
исх. текст	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я

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

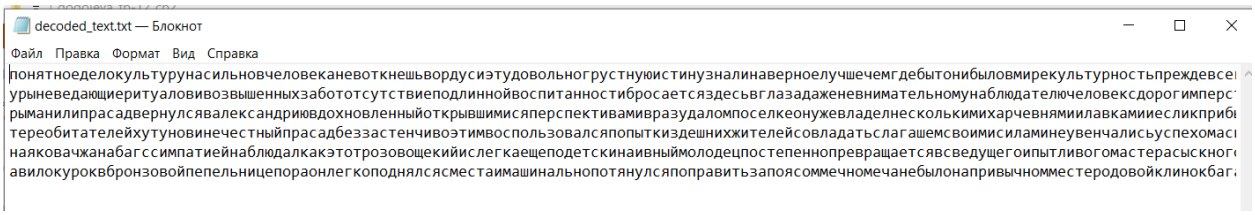
(це було боляче not gonna lie)

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('a')
            if char.islower():
                decrypted_char = chr(((ord(char) - ord('a') - shift) % 32) +
ord('a'))
            else:
                decrypted_char = chr(((ord(char) - ord('A') - shift) % 32) +
ord('A'))
        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)
```

```python
# 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")
```

```
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.