КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3

Гоголєва Поліна ФБ-12

**Варіант 5**

**Криптоаналіз афінної біграмної підстановки**

**Мета роботи**: Набуття навичок частотного аналізу на прикладі розкриття моноалфавітної підстановки; опанування прийомами роботи в модулярній арифметиці.

**Хід роботи**

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

1. Реалізувати підпрограми із необхідними математичними операціями: обчисленням оберненого елементу за модулем із використанням розширеного алгоритму Евкліда, розв’язуванням лінійних порівнянь. При розв’язуванні порівнянь потрібно коректно обробляти випадок із декількома розв’язками, повертаючи їх усі.

def gcd(b, n):

    x0, x1 = 1, 0

    while n:

        q, b, n = b // n, n, b % n

        x0, x1 = x1, x0 - q \* x1

    return b, x0

def modinv(a, m):

    # Знаходження оберненого елементу за модулем m

    d, x = gcd(a, m)

    if d == 1:

        return d, x % m

    return d, None

def solve\_linear\_congruence(a, b, m):

    d, a\_inv = modinv(a, m)

    if a\_inv:

        return [(a\_inv \* b) % m]

    if not b % d:

        a, b, m = a / d, b / d, m / d

        x0 = (b \* modinv(a, m)[1]) % m

        return [int(x0 + (i - 1) \* m) for i in range(1, d + 1)]

2. За допомогою програми обчислення частот біграм, яка написана в ході виконання комп’ютерного практикуму №1, знайти 5 найчастіших біграм запропонованого шифртексту (за варіантом)

def count\_bigrams(file\_path):

    bigram\_with\_overlap\_counts = {}

    bigram\_without\_overlap\_counts = {}

    with open(file\_path, 'r', encoding='windows-1251') as file:

        text = file.read()

    cleaned\_text = text.lower()

    # Підрахунок частоти біграм

    for i in range(len(cleaned\_text) - 1):

        # Підрахунок частоти біграм з перетином

        bigram\_with\_overlap = cleaned\_text[i:i + 2]

        if bigram\_with\_overlap in bigram\_with\_overlap\_counts:

            bigram\_with\_overlap\_counts[bigram\_with\_overlap] += 1

        else:

            bigram\_with\_overlap\_counts[bigram\_with\_overlap] = 1

        if i % 2 == 0:

            # Підрахунок частоти біграм без перетину

            bigram\_without\_overlap = cleaned\_text[i:i + 2]

            if bigram\_without\_overlap in bigram\_without\_overlap\_counts:

                bigram\_without\_overlap\_counts[bigram\_without\_overlap] += 1

            else:

                bigram\_without\_overlap\_counts[bigram\_without\_overlap] = 1

    # Сортування біграм за частотою

    sorted\_bigram\_with\_overlap = sorted(bigram\_with\_overlap\_counts.items(), key=lambda x: x[1], reverse=True)

    sorted\_bigram\_without\_overlap = sorted(bigram\_without\_overlap\_counts.items(), key=lambda x: x[1], reverse=True)

    # Виведення п'ятьох найчастіших біграм

    print("П'ять найчастіших біграм з перетином:")

    for i in range(5):

        print(f"{sorted\_bigram\_with\_overlap[i][0]}: {sorted\_bigram\_with\_overlap[i][1]} разів")

    print("\nП'ять найчастіших біграм без перетину:")

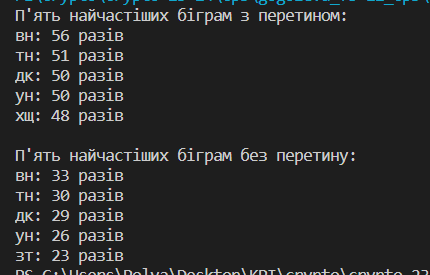
    for i in range(5):

        print(f"{sorted\_bigram\_without\_overlap[i][0]}: {sorted\_bigram\_without\_overlap[i][1]} разів")

# Шлях до файлу

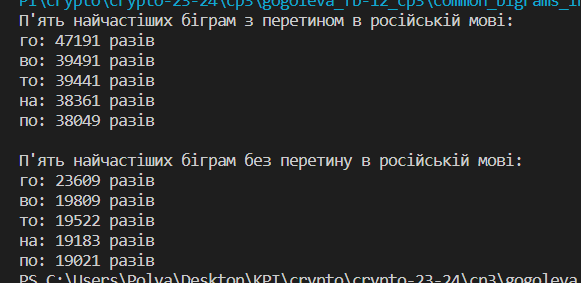
file\_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\tasks\cp3\variants\05.txt"

count\_bigrams(file\_path)

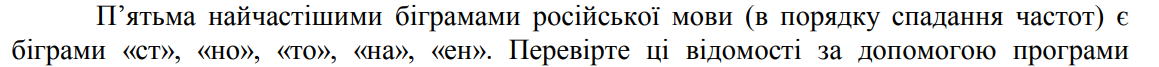


3. Перебрати можливі варіанти співставлення частих біграм мови та частих біграм шифртексту (розглядаючи пари біграм із п’яти найчастіших). Для кожного співставлення знайти можливі кандидати на ключ (a,b) шляхом розв’язання системи (1).

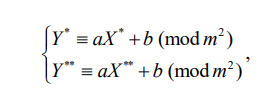
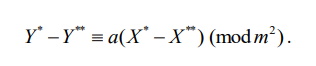
Ну тоді нам треба дізнатися п’ять найчастіших біграм російського тексту, використаєм той же код із першої лаби і перевіримо це на тому ж великому тексті російською. Отримали значення:



(потім я прибрала частину для розрахунку біграм з перетином, бо поняла що це not the case)

Хоча теоретично найчастішими біграмами російської мови є СТ, НО, ТО, НА, ЕН 

Можливо, наш текст був недостатньо довгим, щоб отримати більш точні значення, але в цілому вони +- співпадають, тому використовуватимемо «офіційні» теоретичні дані, бо їм я наче трохи більше довіряю :)

Тобто за даними частотності біграм виходить, що біграми вн тн дк ун зт у нашому шифртексті відповідають біграмам ст но то на ен у відповідному порядку по спаданню частотності. Це дає нам можливість розрахувати ймовірні пари ключів для афінного шифру. Для цього використаємо формули вказані у методичці:  
  

Код для розрахунку пари ключів виглядатиме так

alphabet = 'абвгдежзийклмнопрстуфхцчшщьыэюя'

theoretical\_bigrams = ('ст', 'но', 'то', 'на', 'ен')

def count\_a(x1, x2, y1, y2):

    x1 = alphabet.index(x1[0]) \* len(alphabet) + alphabet.index(x1[1])

    x2 = alphabet.index(x2[0]) \* len(alphabet) + alphabet.index(x2[1])

    y1 = alphabet.index(y1[0]) \* len(alphabet) + alphabet.index(y1[1])

    y2 = alphabet.index(y2[0]) \* len(alphabet) + alphabet.index(y2[1])

    results = solve\_linear\_congruence(y1 - y2, x1 - x2, len(alphabet) \*\* 2)

    if results is not None:

        return [x for x in [modinv(i, len(alphabet) \*\* 2)[1] for i in results] if x is not None]

def count\_b(x1, y1, a):

    x1 = alphabet.index(x1[0]) \* len(alphabet) + alphabet.index(x1[1])

    y1 = alphabet.index(y1[0]) \* len(alphabet) + alphabet.index(y1[1])

    return (y1 - a \* x1) % len(alphabet) \*\* 2

Але це загальні формули, для знаходження всіх можливих кандидатів пар ключів треба перевірити всі можливі перестановки (співставлення) біграм відкритого тексту і шифртексту, бо далеко не обов’язково найчастіша біграма російської мови буде переходити у найчастішу біграму шифртексту.

Для розрахунку всіх можливих пар ключів напишемо таку функцію, використовуючи ф-ю permutations з itertools, щоб не прописувати алгоритм знаходження всіх перестановок вручну.

def all\_possible\_keys(string, key\_size=5):

    f\_list = count\_bigrams(string)[:key\_size]

    possible\_keys = []

    for i in permutations(theoretical\_bigrams, 2):

        for j in range(len(f\_list) - 1):

            key\_1 = count\_a(i[0], i[1], f\_list[j][0], f\_list[j + 1][0])

            if key\_1 is None:

                continue

            for solution in key\_1:

                key = solution, count\_b(i[0], f\_list[j][0], solution)

                possible\_keys.append(key)

    return possible\_keys

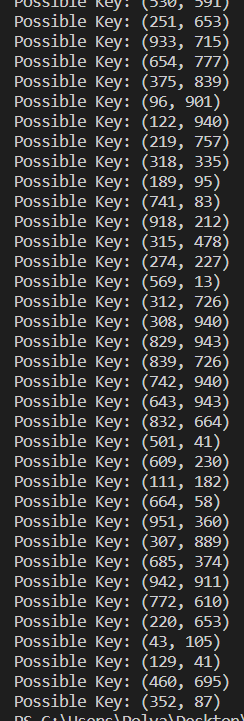
file\_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\tasks\cp3\variants\05.txt"

keys = all\_possible\_keys(file\_path)

for key in keys:

    print('Possible Key:', key)

Отримали довжелезний список всіх можливих ключів



Але ж вручну ми не будемо їх всіх перевіряти, правда?)

Маючи ключі дуже легко розшифрувати афінний шифр за допомогою однієї функції.

def decrypt(string, key):

    new\_str = ''

    for i in range(0, len(string), 2):

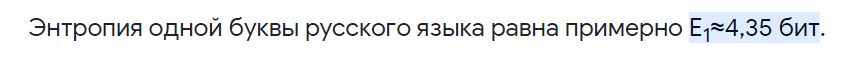
        y = alphabet.index(string[i]) \* len(alphabet) + alphabet.index(string[i + 1])

        x = (modinv(key[0], len(alphabet) \*\* 2)[1] \* (y - key[1])) % len(alphabet) \*\* 2

        new\_str += alphabet[x // len(alphabet)] + alphabet[x % len(alphabet)]

Але нам треба, щоб текст був змістовний. Тут ми плавно переходимо до наступного завдання.

4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.  
Я вирішила використовувати ентропійний фактор порівняння (бо він мені здався найлегшим not gonna lie)  
Як сказав великий інтернет:



Що навіть сходиться з нашими практично знайденими значеннями при виконанні першої лаби



Тож використовуватимемо це значення як істинне.

Думала, які саме параметри порівняння задати для ентропії, вирішила спробувати, щоб ентропія розшифрованого тексту була у межах 4.2<H<4.4, подивимся, що вийде. Для цього створимо кілька функцій. Перша буде стирена з першої лаби і буде розраховувати ентропію однієї літери, а друга буде приймати на ввід текст, використовувати попередню функцію і порівнювати значення ентропії з дозволеними межами.

def find\_entropy(decrypted\_text):

    entropy\_letter = 0.0

    total\_letters = len(decrypted\_text)

    # Підрахунок ентропії для літер

    sorted\_letter\_counts = {}

    for letter in decrypted\_text:

        if letter in sorted\_letter\_counts:

            sorted\_letter\_counts[letter] += 1

        else:

            sorted\_letter\_counts[letter] = 1

    for count in sorted\_letter\_counts.values():

        probability = count / total\_letters

        entropy\_letter -= probability \* math.log2(probability)

    return entropy\_letter

def check\_the\_text(decrypted\_text):

    if find\_entropy(decrypted\_text)>4.2 and find\_entropy(decrypted\_text)<4.4:

        return print(decrypted\_text)

Well, щоб перевірити, що це спрацює напишемо ще одну функцію, котра одразу і дешифруватиме і перевірятиме і виводитиме текст.

def decrypt\_and\_check(file\_path):

    try:

        with open(file\_path, 'r', encoding='windows-1251') as file:

            text = file.read()

    except FileNotFoundError:

        print(f"File '{file\_path}' not found.")

        return

    cleaned\_text = text.lower()

    keys = all\_possible\_keys(cleaned\_text)

    for key in keys:

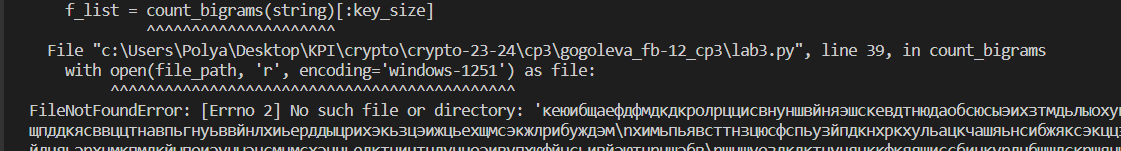
        decrypted\_text = decrypt(cleaned\_text, key)

        if check\_the\_text(decrypted\_text):

            print(f"Decrypted Text with Key {key}:")

            print(decrypted\_text)

і тут у мене все поломалось)))))))



Тому new commit coming soon when i`ll find an energy to deal with it.