МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий фізико-технічний інститут

Кафедра інформаційної безпеки

Дисципліна «Криптографія»

Комп’ютерний практикум

Робота No 3

Виконав : студент групи ФБ-24 Луняка Артем

Київ – 2024

**Тема:**

Криптоаналіз афінної біграмної підстановки

**Мета:**

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

**Варіант 10**

***Завдання до виконання***

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.

2. За допомогою програми обчислення частот біграм, яка написана в ході виконання комп’ютерного практикуму No1, знайти 5 найчастіших біграм запропонованого шифртексту (за варіантом).

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).

4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.

5. Повторювати дії 3-4 доти, доки дешифрований текст не буде змістовним.

**1. Написання програм для виконання роботи.**

В якості мови програмування виберемо Python. Напишемо декілька службових модулів та основну програму для виконання лабораторної роботи. У службових модулях опишемо декілька класів та функцій, які ми зможемо використовувати і надалі для схожих задач. У цій роботі ми також будемо використовувати модулі, написані в рамках виконання лабораторних робіт №1 та №2: alphabet, text\_source, ngram, text\_source\_plus, а також функцію inverse з модуля affine\_cipher, який був написаний для виконання домашнього завдання.

**2. Модуль bigram\_alphabet.**

Модуль bigram\_alphabet призначено для роботи з алфавітом, що складається з біграм деякого звичайного алфавіту. Основне функціональне навантаження виконує клас BigramAlphabet. Цей клас за побудовою дуже схожий на раніше написаний клас Alphabet, який був реалізований для виконання домашньої роботи і використаний у попередніх лабораторних роботах. Відмінності класу BigramAlphabet полягають у тому, що елементами алфавіту є не символи, а біграми, та окрім кількості символів в алфавіті(m), нам треба також тримати у пам’яті значення m2.

from alphabet import Alphabet  
  
  
class BigramAlphabet:  
  
 def \_\_init\_\_(self, alphabet\_string=""):  
 self.\_alphabet = Alphabet(alphabet\_string)  
 self.\_m = len(alphabet\_string)  
 self.\_m2 = self.\_m \* self.\_m  
  
 self.bigrams\_dict = dict()  
 for i in range(self.\_m):  
 letter1 = self.\_alphabet.get\_letter(i)  
 for j in range(self.\_m):  
 bigram = letter1 + self.\_alphabet.get\_letter(j)  
 self.bigrams\_dict[bigram] = i \* self.\_m + j  
 self.numbers\_dict = dict(zip(self.bigrams\_dict.values(), self.bigrams\_dict.keys()))  
  
  
 @property  
 def m(self):  
 return self.\_m  
  
 @property  
 def m2(self):  
 return self.\_m2  
  
 @property  
 def alphabet(self):  
 return self.\_alphabet  
  
 def get\_number(self, bigram):  
 return self.bigrams\_dict.get(bigram)  
  
 def get\_bigram(self, number):  
 return self.numbers\_dict.get(number)  
  
 def get\_numbers\_list(self, bigrams\_string):  
 return [self.get\_number(bigrams\_string[i: i + 2])  
 for i in range(0, len(bigrams\_string), 2)]  
  
 def get\_bigrams\_list(self, numbers):  
 return [self.get\_bigram(n) for n in numbers]  
  
 def get\_all\_bigrams(self):  
 return list(self.bigrams\_dict.keys())

**3. Модуль bigram\_affine\_cipher**

Модуль bigram\_affine\_cipher реалізує дії над шифром афінної підстановки біграм, а також додаткові дії над арифметикою лишків по модулю.

Дії над шифром реалізує клас BigramAffineCipher. Цей клас схожий на раніше описаний клас AffineCipher. Основними методами класу BigramAffineCipher є методи cipher та decipher – шифрування та дешифрування тексту.

Крім класу BigramAffineCipher в модулі ще є функції: gcd – обчислення найбільшого спільного дільника за допомогою звичайного алгоритму Евкліда, congruence\_solver – розвязання порівняння вигляду ax=b mod n, get\_affine\_keys\_from\_congruence – отримання кандидатів ключів шифру на підставі розв’язку системи порівнянь. Останню функцію пояснимо докладніше. Згідно з методичними вказівками, для знаходження а можна використати таке порівняння:

(Y\*-Y\*\*)=a(X\*-X\*\*) mod m2

У цьому порівнянні можемо перенести а до лівої частини та отримаємо таке порівняння:

(Y\*-Y\*\*)а-1=(X\*-X\*\*) mod m2

У цьому порівнняні невідомим є а-1, а всі інші величини відомі. Розв’яжемо це порівняння відповідно методичних вказівок та отримаємо 0, 1 або декілька розв’язків. Ці розв’язки функція get\_affine\_keys\_from\_congruence повертає у місце виклику. Треба зазначити, що отримані розв’язки є значеннями а-1. Отже, для отримання значень а треба буде знайти обернені величини до розв’язків.

from affine\_cipher import inverse  
from bigram\_alphabet import BigramAlphabet  
  
  
def gcd(m, n):  
 if m == 0 and n == 0:  
 return 0  
  
 if n == 0:  
 return m  
  
 while m != 0:  
 n = n % m  
 n, m = m, n  
 return n  
  
  
def congruence\_solver(a, b, m):  
 *"""Solves ax = b mod m congruence"""* solutions = []  
 d = gcd(a, m)  
 if d != 1:  
 if d == 0 or b % d != 0:  
 return solutions  
  
 a //= d  
 b //= d  
 m //= d  
 a1 = inverse(a, m)  
 x = (a1 \* b) % m  
 if d == 1:  
 solutions.append(x)  
 else:  
 solutions = [x + k \* m for k in range(d)]  
 return solutions  
  
  
def get\_affine\_keys\_from\_congruence(y1, y2, x1, x2, m):  
 *"""Find a from (y1 - y2) = a(x1 - x2) (mod m)"""* a\_inversed\_list = congruence\_solver((y1 - y2 + m) % m, (x1 - x2 + m) % m, m)  
 a\_b\_list = list()  
 for a\_inversed in a\_inversed\_list:  
 a = inverse(a\_inversed, m)  
 if a is None:  
 continue  
  
 b = (y1 + m - a \* x1) % m  
 a\_b\_list.append((a, b))  
 return a\_b\_list  
  
  
class BigramAffineCipher:  
  
 def \_\_init\_\_(self, a, b, bigram\_alphabet: BigramAlphabet):  
 self.\_a = a  
 self.\_b = b  
 self.\_bigram\_alphabet = bigram\_alphabet  
 self.\_m = self.\_bigram\_alphabet.m2  
 self.\_a\_inversed = inverse(self.\_a, self.\_m)  
 if self.\_a\_inversed is None:  
 raise ValueError(f"Parameter a({self.\_a}) is not coprime to m({self.\_m})")  
  
 def cipher(self, bigrams\_string):  
 numbers\_list = [(self.\_a \* x + self.\_b) % self.\_m  
 for x in self.\_bigram\_alphabet.get\_numbers\_list(bigrams\_string)]  
 return self.\_bigram\_alphabet.get\_bigrams\_list(numbers\_list)  
  
 def decipher(self, ciphered\_bigrams\_string):  
 numbers\_list = [self.\_a\_inversed \* (x + self.\_m - self.\_b) % self.\_m  
 for x in self.\_bigram\_alphabet.get\_numbers\_list(ciphered\_bigrams\_string)]  
 return self.\_bigram\_alphabet.get\_bigrams\_list(numbers\_list)

**4.** **Функція inverse**

Функція inverse призначена для знаходження оберненого до заданого числа а у розумінні модульної арифметики, тобто розв’язку порівняння а=1 mod m. Обернене обчислюється з використанням розширеного алгоритму Евкліда, як було зазначено у методичних вказівках.

def inverse(a, m):  
 t = 0  
 newt = 1  
 r = m  
 newr = a  
 while newr != 0:  
 quotient = r // newr  
 t, newt = (newt, t - quotient \* newt)  
 r, newr = (newr, r - quotient \* newr)  
 if r > 1:  
 return None  
  
 if t < 0:  
 t = t + m  
 return t

**5. Модуль detect\_open\_text**

Модуль detect\_open\_text призначений для розробки аналізатора відкритого тексту. Цей модуль не містить класів, а лише функції, що перевіряють критерії, яким має задовольняти відкритий текст натуральною мовою. Усі функції повертають булівський результат(істина, якщо критерії задоволено та хибність, якщо не задоволено).

Функції та їх призначення:

check\_most\_frequent\_criteria – перевіряє, що сума частот трьох найбільш частих символів мови близька до суми частот цих символів у тексті;

check\_least\_frequent\_criteria - перевіряє, що сума частот трьох найменш частих символів мови близька до суми частот цих символів у тексті;

check\_most\_frequent\_bigrams\_criteria - перевіряє, що сума частот трьох найбільш частих біграм мови близька до суми частот цих біграм у тексті;

check\_index\_of\_coincidence\_criteria - перевіряє, що індекс відповідності мови близький до індексу відповідності тексту.

from alphabet import Alphabet  
from text\_source\_plus import TextSourcePlus  
from ngram import NGrams  
from vizhener\_decipher import index\_of\_coincidence  
  
  
def check\_most\_frequent\_criteria(source: TextSourcePlus, alphabet: Alphabet,  
 language\_frequencies\_sorted):  
 f\_quantity = 3  
 ngrams = NGrams(alphabet, 1, source)  
 ngrams.feed()  
 source\_frequencies = ngrams.get\_ngrams\_frequencies()  
 *# calculate sum of frequencies in text of 3 most frequent chars in language* source\_sum = 0  
 for char in list(language\_frequencies\_sorted.keys())[: f\_quantity]:  
 source\_sum += source\_frequencies[char]  
 *# calculate sum of frequencies of 3 most frequent chars in language* language\_sum = sum(list(language\_frequencies\_sorted.values())[: f\_quantity])  
 print("check\_most\_frequent\_criteria", language\_sum, source\_sum)  
 sigma = (max(language\_frequencies\_sorted.values())  
 - min(language\_frequencies\_sorted.values())) / 4 *#(alphabet.m // f\_quantity)* return abs(source\_sum - language\_sum) <= sigma  
  
  
def check\_least\_frequent\_criteria(source: TextSourcePlus, alphabet: Alphabet,  
 language\_frequencies\_sorted):  
 f\_quantity = 3  
 ngrams = NGrams(alphabet, 1, source)  
 ngrams.feed()  
 source\_frequencies = ngrams.get\_ngrams\_frequencies()  
 source\_sum = 0  
 for char in list(language\_frequencies\_sorted.keys())[-f\_quantity:]:  
 source\_sum += source\_frequencies[char]  
 language\_sum = sum(list(language\_frequencies\_sorted.values())[-f\_quantity:])  
 print("check\_least\_frequent\_criteria", language\_sum, source\_sum)  
 sigma = (max(language\_frequencies\_sorted.values())  
 - min(language\_frequencies\_sorted.values())) / 4 *#(alphabet.m // f\_quantity)* return abs(source\_sum - language\_sum) <= sigma  
  
  
def check\_most\_frequent\_bigrams\_criteria(source\_cut: TextSourcePlus, alphabet: Alphabet,  
 language\_bigrams\_frequencies\_sorted):  
 *# have to delete first char of source text to check bigrams at intersections* f\_quantity = 3  
 ngrams = NGrams(alphabet, 2, source\_cut)  
 ngrams.feed()  
 source\_frequencies = ngrams.get\_ngrams\_frequencies()  
 source\_sum = 0  
 for bigram in list(language\_bigrams\_frequencies\_sorted.keys())[: f\_quantity]:  
 source\_sum += source\_frequencies[bigram]  
 language\_sum = sum(list(language\_bigrams\_frequencies\_sorted.values())[: f\_quantity])  
 print("check\_most\_frequent\_bigrams\_criteria", language\_sum, source\_sum)  
 *# m = alphabet.m \*\* 2* sigma = (language\_sum) / 4 *#(m // f\_quantity)* return abs(source\_sum - language\_sum) <= sigma  
  
  
def check\_index\_of\_coincidence\_criteria(source: TextSourcePlus,  
 language\_index\_of\_coincidence):  
 ind = index\_of\_coincidence(source)  
 print("check\_index\_of\_coincidence\_criteria", language\_index\_of\_coincidence, ind)  
 sigma = language\_index\_of\_coincidence / 4  
 return abs(language\_index\_of\_coincidence - ind) <= sigma

**6. Основна програма**

Основна програма для лабораторної роботи міститься у модулі lab3. У цьому модулі, зокрема описано константи для шляхів та назв файлів, що використовуються у програмі, а також рядок символів використовуваного алфавіту. Функція get\_most\_frequent повертає задану кількість(5) найчастіших біграм текстового джерела. У якості текстового джерела виступає великий файл, який використовувався у лабораторних роботах №1, №2 та вважається представником російської мови, а також зашифрований файл згідно варіанту. Функція gen\_combinations генерує усі можливі комбінації найчастіших біграм мови та шифртексту.

Функція is\_likely\_open\_text перевіряє чи є текст-кандидат відкритим текстом. Будемо вважати, що текст-кандидат є відкритим текстом, якщо він задовольняє усім чотирьом раніше описаним критеріям.

Функція decipher пробує дешифрувати шифртекст з використанням раніше описаних функцій та класів. Якщо ця функція знаходить відкритий текст, то припиняє подальшу обробку та пошук кандидатів.

У самій головній програмі ми утворюємо текстове джерело для великого тексту російською мовою(файл з лабораторної роботи №1) та текстове джерело для невеликого шифртексту. Для цих текстових джерел обчислюємо індекси відповідності.

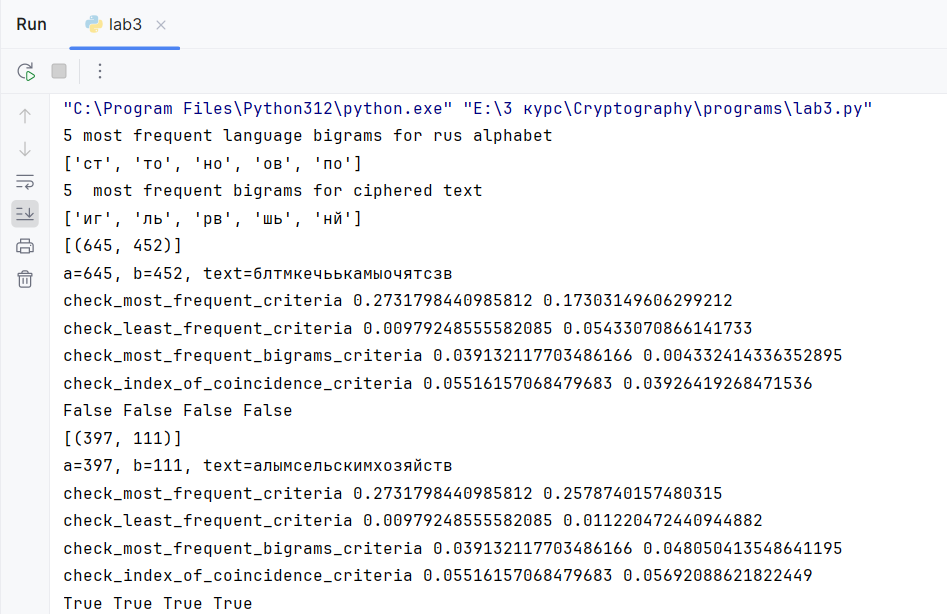
Окрім цього будуємо кількість та частоту входжень монограм та біграм у великий текст(мову) та шифртекст.

Після цього пробуємо дешифрувати текст з 10 варіанту за допомогою функції decipher. Якщо відкритий текст знайдено, то показуємо його по рядках з 60 символів.

from text\_source\_plus import TextSourcePlus  
from alphabet import Alphabet  
from ngram import NGrams  
from bigram\_alphabet import BigramAlphabet  
from bigram\_affine\_cipher import BigramAffineCipher, get\_affine\_keys\_from\_congruence  
from detect\_open\_text import check\_most\_frequent\_bigrams\_criteria, check\_most\_frequent\_criteria, \  
 check\_least\_frequent\_criteria, check\_index\_of\_coincidence\_criteria, index\_of\_coincidence  
  
PATH\_LAB1 = "..\\lab1\\"  
LONG\_FILE\_NAME = "rus\_text.txt"  
PATH\_LAB3 = "..\\lab3\\"  
CIPHERED\_FILE\_NAME = "V10.txt"  
  
RUS\_LOWERCASE = "абвгдежзийклмнопрстуфхцчшщыьэюя"  
MOST\_FREQUENT\_NUMBER = 5  
  
  
def get\_most\_frequent(ngrams: NGrams, bigram\_alphabet: BigramAlphabet, n):  
 frequencies = ngrams.get\_ngrams\_frequencies(to\_sort=True)  
 numbers\_list = [bigram\_alphabet.get\_number(bigram) for bigram in list(frequencies.keys())[: n]]  
 bigrams\_list = [bigram for bigram in list(frequencies.keys())[: n]]  
 print(bigrams\_list)  
 return numbers\_list  
  
  
def gen\_combinations(bigrams\_most\_frequent, language\_most\_frequent):  
 for i, bigram in enumerate(bigrams\_most\_frequent):  
 for j, language\_bigram in enumerate(language\_most\_frequent):  
 for i1 in range(i + 1, len(bigrams\_most\_frequent)):  
 for j1 in range(j + 1, len(language\_most\_frequent)):  
 yield bigram, language\_bigram, \  
 bigrams\_most\_frequent[i1], language\_most\_frequent[j1]  
  
  
def is\_likely\_open\_text(source: TextSourcePlus, alphabet,  
 language\_frequencies\_sorted,  
 language\_bigrams\_frequencies\_sorted,  
 language\_index\_of\_coincidence):  
 source\_cut = TextSourcePlus.from\_string(source.filtered\_as\_string[1:], alphabet)  
 source\_cut.apply\_filter\_chain()  
 criteria1 = check\_most\_frequent\_criteria(source, alphabet, language\_frequencies\_sorted)  
 criteria2 = check\_least\_frequent\_criteria(source, alphabet, language\_frequencies\_sorted)  
 criteria3 = check\_most\_frequent\_bigrams\_criteria(  
 source\_cut, alphabet, language\_bigrams\_frequencies\_sorted)  
 criteria4 = check\_index\_of\_coincidence\_criteria(source, language\_index\_of\_coincidence)  
 print(criteria1, criteria2, criteria3, criteria4)  
 return criteria1 and criteria2 and criteria3 and criteria4  
  
  
def decipher(source: TextSourcePlus, bigram\_alphabet: BigramAlphabet,  
 bigrams\_most\_frequent, language\_most\_frequent,  
 language\_frequencies\_sorted,  
 language\_bigrams\_frequencies\_sorted,  
 language\_index\_of\_coincidence):  
 *# f = open("out.txt", "w", encoding='utf-8')* alphabet = bigram\_alphabet.alphabet  
 for y1, x1, y2, x2 in gen\_combinations(bigrams\_most\_frequent, language\_most\_frequent):  
 a\_b\_list = get\_affine\_keys\_from\_congruence(y1, y2, x1, x2, bigram\_alphabet.m2)  
 if not a\_b\_list:  
 continue  
  
 print(a\_b\_list) *#, file=f)* for a, b in a\_b\_list:  
 affine\_bigram = BigramAffineCipher(a, b, bigram\_alphabet)  
 decipered = affine\_bigram.decipher(source.filtered\_as\_string)  
 if decipered is None:  
 continue  
  
 open\_text\_candidate = ''.join(decipered)  
 print(f"a={a}, b={b}, text={open\_text\_candidate[:20]}") *#, file=f)* open\_source\_candidate = TextSourcePlus.from\_string(open\_text\_candidate, alphabet)  
 open\_source\_candidate.apply\_filter\_chain()  
 if is\_likely\_open\_text(open\_source\_candidate,  
 alphabet,  
 language\_frequencies\_sorted,  
 language\_bigrams\_frequencies\_sorted,  
 language\_index\_of\_coincidence):  
 *# f.close()* return open\_text\_candidate  
 *# f.close()  
  
# calculate characteristics of language using long russian text*alphabet = Alphabet(RUS\_LOWERCASE)  
bigram\_alphabet = BigramAlphabet(RUS\_LOWERCASE)  
long\_source = TextSourcePlus(PATH\_LAB1 + LONG\_FILE\_NAME, alphabet, "to\_lower", "replace\_ru\_yo\_hard",  
 "delete\_delimeters", "delete\_spaces")  
long\_source.apply\_filter\_chain()  
rus\_index\_of\_coincidence = index\_of\_coincidence(long\_source)  
rus\_unigrams = NGrams(alphabet, 1, long\_source)  
rus\_unigrams.feed()  
rus\_bigrams = NGrams(alphabet, 2, long\_source)  
rus\_bigrams.feed()  
print(f"{MOST\_FREQUENT\_NUMBER} most frequent language bigrams for rus alphabet")  
rus\_bigrams\_most\_frequent = get\_most\_frequent(rus\_bigrams, bigram\_alphabet, MOST\_FREQUENT\_NUMBER)  
  
*# calculate characteristics of ciphered text*ciphered\_source = TextSourcePlus(PATH\_LAB3 + CIPHERED\_FILE\_NAME, alphabet)  
ciphered\_source.apply\_filter\_chain()  
unigrams = NGrams(alphabet, 1, ciphered\_source)  
unigrams.feed()  
bigrams = NGrams(alphabet, 2, ciphered\_source)  
bigrams.feed()  
print(f"{MOST\_FREQUENT\_NUMBER} most frequent bigrams for ciphered text")  
bigrams\_most\_frequent = get\_most\_frequent(bigrams, bigram\_alphabet, MOST\_FREQUENT\_NUMBER)  
  
open\_text = decipher(ciphered\_source, bigram\_alphabet,  
 bigrams\_most\_frequent, rus\_bigrams\_most\_frequent,  
 rus\_unigrams.get\_ngrams\_frequencies(to\_sort=True),  
 rus\_bigrams.get\_ngrams\_frequencies(to\_sort=True),  
 rus\_index\_of\_coincidence)  
  
line\_len = 60  
print("\nIn the end of deciphering open text is:")  
if open\_text:  
 for i in range(0, len(open\_text), line\_len):  
 print(open\_text[i: i + line\_len])

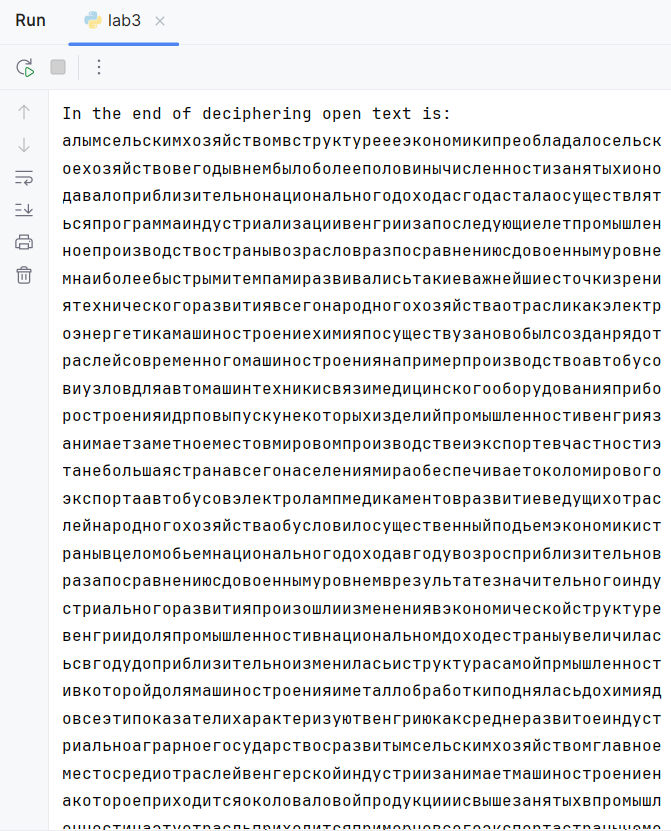
**7. Запуск програми**

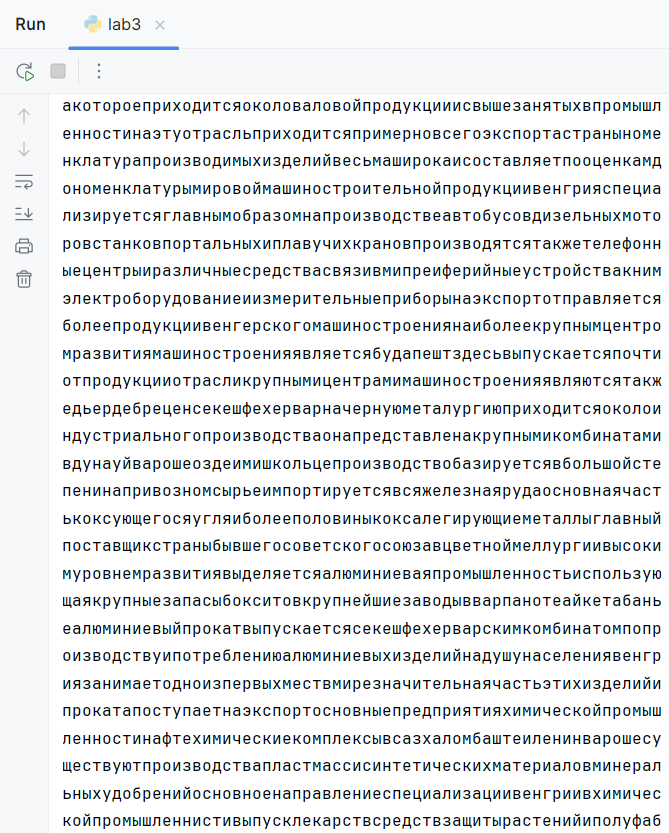
Запустимо програму.

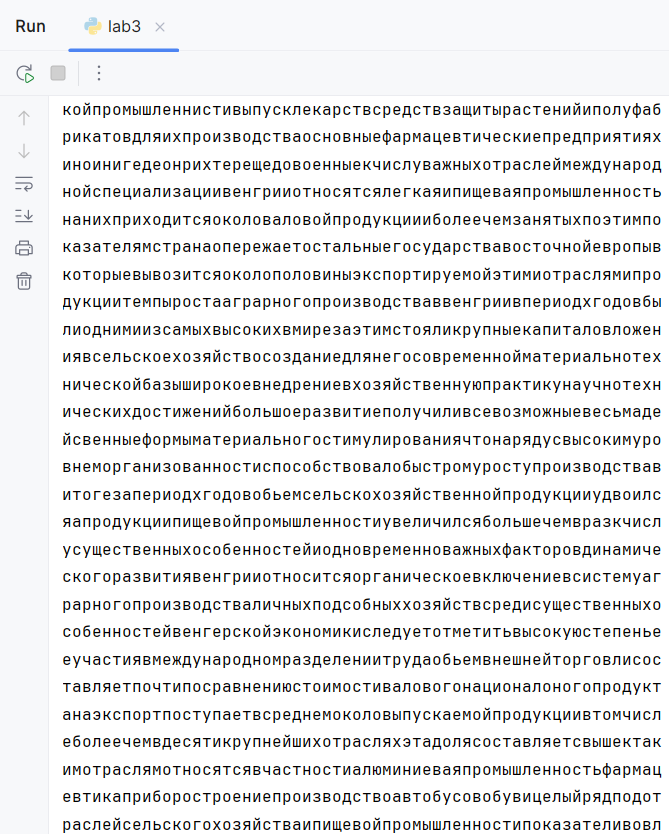


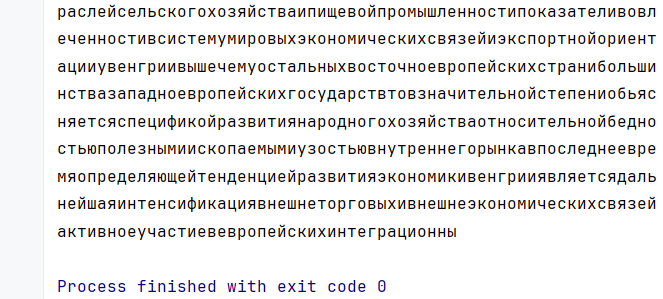
Бачимо 5 найчастіших біграм мови та шифртексту. При цьому 5 найчастіших біграм мови дещо відрізняються від вказаних у методичці за порядком слідування або за набором. У будь-якому випадку, ми будемо використовувати обчислені нами значення.

Далі бачимо кандидатів у ключі(пари а, b) та відповідність дешифрованого кожним ключем тексту зазначеним вище критеріям. Стає зрозуміло, що вже другий кандидат дає при дешифруванні відкритий текст. Він відповідає усім чотирьом критеріям.









Візуально бачимо, що це відкритий текст без пропусків.

Висновок

У цій роботі було виконано криптоаналіз тексту, зашифрованого шифром біграмної афінної підстановки. На підставі частотного аналізу знайдено послідовність ключів-кандидатів шифру. Розроблено розпізнавач відкритого тексту з використанням декількох критеріїв. Також розроблено деякі функції для роботи з модульною арифметикою. З використанням всіх цих інструментів вдалось успішно дешифрувати наданий шифртекст.