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 2

Виконав : студент групи ФБ-24 Луняка Артем

Київ – 2024

**Тема:**

Криптоаналіз шифру Віженера

**Мета:**

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

**Варіант 10**

***Завдання до виконання***

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

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.

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

3. Використовуючи наведені теоретичні відомості, розшифрувати наданий шифртекст (згідно свого номеру варіанта).

**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 лабораторної роботи №1: alphabet, text\_source, ngram.

**2. Модуль text\_source\_plus.**

Модуль text\_source\_plus призначено для додаткових дій над текстовими джерелами. Він містить клас TextSourcePlus, який розширює функціональність класу TextSource.

У класі TextSourcePlus додано методи from\_string для отримання текстового джерела з рядка замість файла та periodic для побудови списку текстових джерел при розбитті тексту на r блоків.

Також у цьому класі додано властивість alphabet, властивість filtered\_as\_string для представлення тексту після фільтрації у вигляді рядка, а також фільтр replace\_ru\_yo\_filter, який міняє літеру «ё» на «е».

from alphabet import Alphabet  
from text\_source import TextSource  
  
  
class TextSourcePlus(TextSource):  
  
 @property  
 def alphabet(self):  
 return self.\_alphabet  
  
 @property  
 def filtered\_as\_string(self):  
 return ''.join(self.\_source\_filtered)  
  
 @classmethod  
 def from\_string(cls, source\_string, alphabet: Alphabet, \*filters):  
 source = cls(None, alphabet, \*filters)  
 source.\_start\_source = source\_string  
 return source  
  
 @classmethod  
 def periodic(cls, source\_string, alphabet: Alphabet, r, \*filters):  
 sources = list()  
 for i in range(r):  
 periodic\_list = [source\_string[j] for j in range(i, len(source\_string), r)]  
 periodic\_string = ''.join(periodic\_list)  
 sources.append(cls.from\_string(periodic\_string, alphabet, \*filters))  
 return sources  
  
 @staticmethod  
 def replace\_ru\_yo\_filter(source: str, to\_replace=(('ё', 'е'), )):  
 to\_replace\_dict = dict(to\_replace)  
 filtered = [c if c not in to\_replace\_dict else to\_replace\_dict[c] for c in source]  
 return ''.join(filtered)

**3. Клас VizhenerCipher**

Клас VizhenerCipher описано у модулі vizhener, який був створений для виконання домашньої роботи з завданнями на шифр Віженера. Цей клас містить методи для шифрування(сipher) та дешифрування(decipher) для заданого алфавіту. Метод build\_key розповсюджує ключове слово на довжину тексту.

class VizhenerCipher:  
  
 def \_\_init\_\_(self, keyword, alphabet: Alphabet):  
 self.\_keyword = keyword  
 self.\_alphabet = alphabet  
 self.\_m = self.\_alphabet.m  
  
 @property  
 def keyword(self):  
 return self.\_keyword  
  
 def build\_key(self, n):  
 key = list(self.\_keyword) \* (n // len(self.\_keyword) + 1)  
 key = key[:n]  
 return key  
  
 def cipher(self, letters):  
 key = self.build\_key(len(letters))  
 key\_numbers = self.\_alphabet.get\_numbers\_list(key)  
 numbers\_list = [(x + y) % self.\_m  
 for x, y in zip(self.\_alphabet.get\_numbers\_list(letters),  
 key\_numbers)]  
 return self.\_alphabet.get\_letters\_list(numbers\_list)  
  
 def decipher(self, ciphered\_letters):  
 key = self.build\_key(len(ciphered\_letters))  
 key\_numbers = self.\_alphabet.get\_numbers\_list(key)  
 numbers\_list = [(x + self.\_m - y) % self.\_m  
 for x, y in zip(self.\_alphabet.get\_numbers\_list(ciphered\_letters),  
 key\_numbers)]  
 return self.\_alphabet.get\_letters\_list(numbers\_list)

**4.** **Модуль vizhener\_decipher**

Модуль vizhener\_decipher містить функції та класи для виконання лабораторної роботи. Зокрема, цей модуль містить функцію index\_of\_coincidence для підрахунку індексу відповідності. Також модуль містить клас VizhenerDecipher, в якому описано методи для знаходження періоду ключа та дешифрування тексту, зашифрованого шифром Віженера.

Для знаходження періоду ключа написано методи, що реалізують підходи, наведені у методичних рекомендаціях:

* розбиття шифрованого тексту на блоки та обчислення індексу відповідності поблочно - periodic\_method1\_period\_finder;
* обчислення індексів відповідності текстів, зашифрованих шифрами Віженера з довжиною ключа 2,3 і т.д. та порівняння їх з індексом відповідності шифрованого тексту – non\_periodic\_method1\_period\_finder;
* обчислення статистики збігів символів, які віддалені на r позицій - stat\_equals\_method2\_period\_finder

На підставі тестування цих методів було розроблено метод detect\_key\_period. Він використовує методи periodic\_method1\_period\_finder та stat\_equals\_method2\_period\_finder для визначення періоду ключа.

Метод decipher здійснює дешифрування тексту за вже готовим ключовим словом або за сконструйованим ключовим словом, яке побудовано на підставі частот символів шифртексту та частот символів мови.

from random import randint  
  
from alphabet import Alphabet  
from text\_source\_plus import TextSourcePlus  
from ngram import NGrams  
from vizhener import VizhenerCipher  
  
  
def index\_of\_coincidence(source: TextSourcePlus):  
 if source.size <= 1:  
 return 0  
  
 ngrams = NGrams(source.alphabet, 1, source)  
 ngrams.feed()  
 s = 0  
 for x in ngrams.get\_ngrams\_quantities().values():  
 s += x \* (x - 1)  
 return s / (source.size \* (source.size - 1))  
  
  
def create\_random\_keyword(alphabet: Alphabet, length):  
 key\_list = [alphabet.get\_letter(randint(0, alphabet.m - 1)) for i in range(length)]  
 return ''.join(key\_list)  
  
  
def cipher\_source(source: TextSourcePlus, rmin, rmax):  
 ciphered\_dict = dict()  
 keywords = list()  
 for r in range(rmin, rmax + 1):  
 keyword = create\_random\_keyword(source.alphabet, r)  
 keywords.append(keyword)  
 v = VizhenerCipher(keyword, source.alphabet)  
 ciphered\_dict[r] = ''.join(v.cipher(source.filtered\_as\_string))  
 return keywords, ciphered\_dict  
  
  
class VizhenerDecipher:  
 METHOD\_1\_R\_MIN = 2  
 METHOD\_2\_R\_MIN = 6  
 R\_MAX = 30  
  
 def \_\_init\_\_(self, source: TextSourcePlus, alphabet\_co\_index=1):  
 self.\_source = source  
 self.\_source.apply\_filter\_chain()  
 self.\_alphabet\_co\_index = alphabet\_co\_index  
 self.\_m = self.\_source.alphabet.m  
  
 def get\_index\_of\_coincidence\_for\_period(self, r):  
 sources = TextSourcePlus.periodic(  
 self.\_source.filtered\_as\_string, self.\_source.alphabet, r)  
 for source in sources:  
 source.apply\_filter\_chain()  
 indexes = [ind for ind in map(index\_of\_coincidence, sources)]  
 return sum(indexes) / len(indexes) *# average index of coincidence* def periodic\_method1\_period\_finder(self, return\_all=False):  
 uniform\_index\_of\_coincidence = 1 / self.\_m  
 sigma = abs(self.\_alphabet\_co\_index - uniform\_index\_of\_coincidence) / 4  
 candidate\_periods = list()  
 all\_periods = list()  
 for r in range(self.METHOD\_1\_R\_MIN, self.R\_MAX + 1):  
 ind = self.get\_index\_of\_coincidence\_for\_period(r)  
 all\_periods.append(  
 (r, ind, self.\_alphabet\_co\_index, uniform\_index\_of\_coincidence))  
 if abs(ind - self.\_alphabet\_co\_index) <= sigma:  
 candidate\_periods.append(  
 (r, ind, self.\_alphabet\_co\_index, uniform\_index\_of\_coincidence))  
 return candidate\_periods if not return\_all else all\_periods  
  
 def non\_periodic\_method1\_period\_finder(self, ciphered\_texts: dict, return\_all=False):  
 *# ciphered\_texts - dictionary of ciphered known text with key period as key  
 # and text as value* ciphered\_indexes = dict()  
 ind = index\_of\_coincidence(self.\_source)  
 for r, ciphered in ciphered\_texts.items():  
 source = TextSourcePlus.from\_string(ciphered, self.\_source.alphabet)  
 source.apply\_filter\_chain()  
 ciphered\_indexes[r] = index\_of\_coincidence(source)  
 sigma = (max(ciphered\_indexes.values()) - min(ciphered\_indexes.values())) / 6  
 candidates = {r: idx for r, idx in ciphered\_indexes.items() if abs(ind - idx) < sigma}  
 return (ind, candidates) if not return\_all else (ind, ciphered\_indexes)  
  
 def find\_equals\_num(self, source: TextSourcePlus):  
 s = source.filtered\_as\_string  
 return sum([1 if s[i] == s[i + 1] else 0 for i in range(len(s) - 1)])  
  
 def stat\_equals\_method2\_period\_finder(self, return\_all=False):  
 stats = dict()  
 for r in range(self.METHOD\_2\_R\_MIN, self.R\_MAX + 1):  
 sources = TextSourcePlus.periodic(  
 self.\_source.filtered\_as\_string, self.\_source.alphabet, r)  
 d\_n = 0  
 for source in sources:  
 source.apply\_filter\_chain()  
 d\_n += self.find\_equals\_num(source)  
 stats[r] = d\_n  
 max\_d\_n = max(stats.values())  
 sigma = (max\_d\_n - min(stats.values())) / 4  
 candidates = {r: d\_n for r, d\_n in stats.items() if abs(max\_d\_n - d\_n) < sigma}  
 return candidates if not return\_all else stats  
  
 def detect\_key\_period(self):  
 period\_candidates = self.periodic\_method1\_period\_finder()  
 stats\_candidates = self.stat\_equals\_method2\_period\_finder()  
 r = 0  
 if not period\_candidates:  
 return r  
  
 potential\_period = min(period\_candidates)[0]  
 min\_stats\_candidate = min(stats\_candidates.keys())  
 if min\_stats\_candidate % potential\_period == 0:  
 r = potential\_period  
 return r  
  
 def \_get\_ith\_most\_frequent(self, ngrams: NGrams, i):  
 frequencies = ngrams.get\_ngrams\_frequencies(to\_sort=True)  
 return list(frequencies.keys())[i]  
  
 def \_construct\_keyword(self, sources, frequencies\_numbers, language\_ngrams):  
 keyword = ""  
 for j, source in enumerate(sources):  
 ngrams = NGrams(source.alphabet, 1, source)  
 ngrams.feed()  
 y = self.\_get\_ith\_most\_frequent(ngrams, 0)  
 x = self.\_get\_ith\_most\_frequent(language\_ngrams, frequencies\_numbers[j])  
 number = (source.alphabet.get\_number(y) - source.alphabet.get\_number(x)  
 + source.alphabet.m) % source.alphabet.m  
 keyword += source.alphabet.get\_letter(number)  
 return keyword  
  
 def decipher(self, period, language\_source: TextSourcePlus,  
 keyword="", frequencies\_numbers=()):  
 if not period:  
 return "", ""  
  
 if not keyword:  
 language\_ngrams = NGrams(self.\_source.alphabet, 1, language\_source)  
 language\_ngrams.feed()  
  
 sources = TextSourcePlus.periodic(  
 self.\_source.filtered\_as\_string, self.\_source.alphabet, period)  
 for source in sources:  
 source.apply\_filter\_chain()  
 keyword = self.\_construct\_keyword(sources, frequencies\_numbers, language\_ngrams)  
 vizhener = VizhenerCipher(keyword, self.\_source.alphabet)  
 return keyword, vizhener.decipher(self.\_source.filtered\_as\_string)

**5. Основна програма**

Основна програма для лабораторної роботи міститься у модулі lab2\_reworked. У цьому модулі, зокрема описано константи для шляхів та назв файлів, що використовуються у програмі , а також рядок символів використовуваного алфавіту. Функція frequencies\_numbers\_gen генерує списки з номерами найбільш частих літер мови, що використовуються для розшифрування ключа(0 – найбільш часта літера, 1 – друга за частотою тощо). Функція decipher виконує розшифрування тексту за участі користувача. Користувачу пропонується продовжити або зупинити розшифрування, а також ввести правильне ключове слово. Функція plot\_bar зображує стовпчасту діаграму за допомогою matplotlib.

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

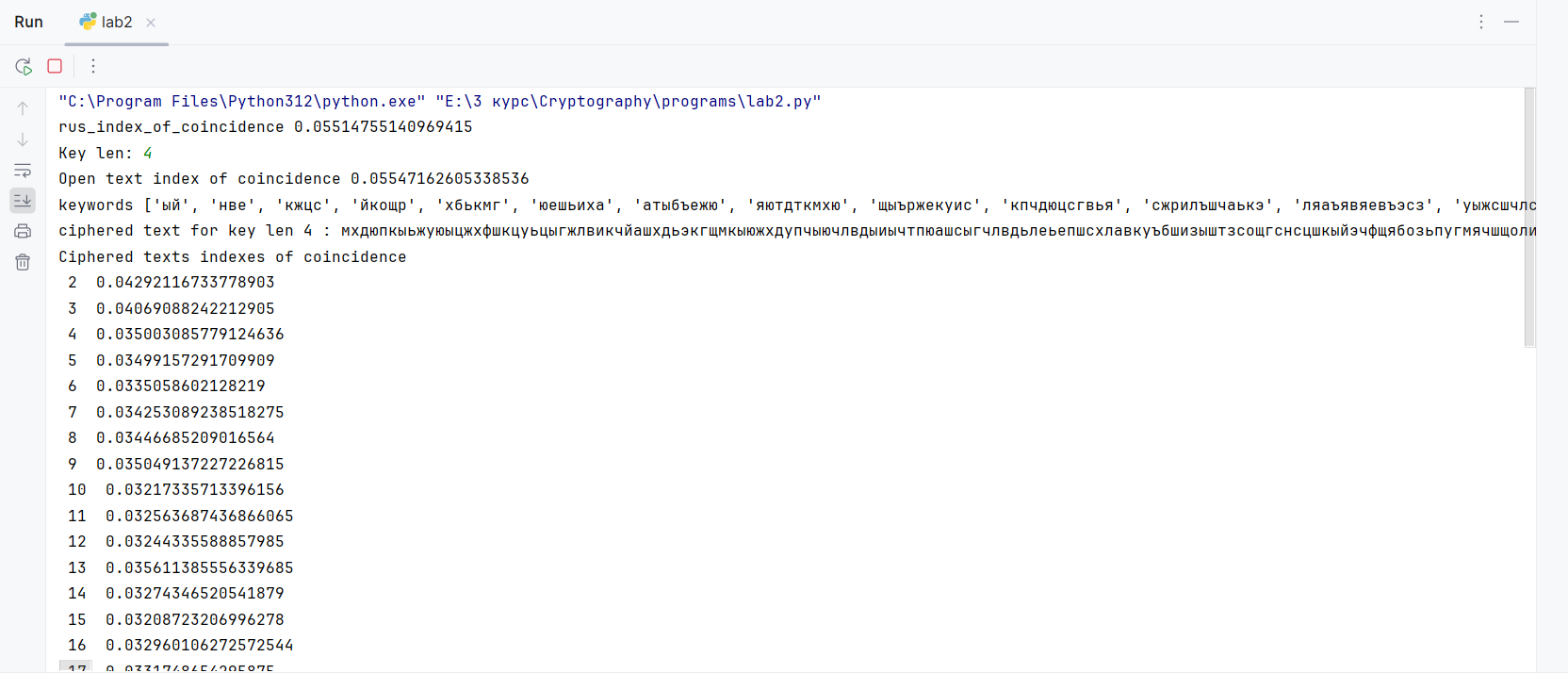
Зашифровуємо невеликий текст шифром Віженера з випадковими ключами довжиною від 2 до 30 символів. Вибираємо довжину ключа та для цієї довжини тестуємо методи визначення періоду ключа з модуля vizhener\_decipher. Пересвідчуємось у том, що ці методи працюють. Показуємо отримані значення індексів відповідності та статистики у вигляді таблиць та стовпчастих діаграм.

Після цього пробуємо дешифрувати текст з 10 варіанту за допомогою функції decipher.

import matplotlib.pyplot as plt  
  
from text\_source\_plus import TextSourcePlus  
from alphabet import Alphabet  
from vizhener\_decipher import VizhenerDecipher, index\_of\_coincidence, cipher\_source  
  
PATH\_LAB1 = "..\\lab1\\"  
LONG\_FILE\_NAME = "rus\_text.txt"  
PATH\_LAB2 = "..\\lab2\\"  
SHORT\_FILE\_NAME = "short\_rus\_text.txt"  
CIPHERED\_FILE\_NAME = "ciphered.txt"  
  
RUS\_LOWERCASE\_WITH\_HARD = "абвгдежзийклмнопрстуфхцчшщъыьэюя"  
  
  
def frequencies\_numbers\_gen(r, alphabet: Alphabet):  
 k = 0  
 while True:  
 frequencies\_numbers = [k] \* r  
 yield frequencies\_numbers  
 k = (k + 1) % alphabet.m  
  
  
def decipher(v\_d: VizhenerDecipher, alphabet: Alphabet, language\_source: TextSourcePlus):  
 period = v\_d.detect\_key\_period()  
 if period:  
 print(f"Detected period: {period}")  
 else:  
 print("Can't detect key period")  
 return  
  
 gen = frequencies\_numbers\_gen(period, alphabet)  
 keyword = ""  
 while True:  
 if not keyword:  
 frequiencies\_numbers = next(gen)  
 keyword, open\_list = v\_d.decipher(period, language\_source, keyword, frequiencies\_numbers)  
 print(f"keyword: {keyword}")  
 print(f"Decipered text: {open\_list}")  
 to\_continue = input("Continue deciphering? [y/n] ")  
 if to\_continue.lower() != 'y':  
 break  
  
 keyword = input(f"Enter keyword ({period} characters) "  
 "or <Enter> to automatically generate keyword: ")  
 return keyword, ''.join(open\_list)  
  
  
def plot\_bar(x, y, xlabel, ylabel, title, color='b'):  
 plt.rcParams["figure.figsize"] = (12, 5)  
 plt.bar(x, y, width=0.8, color=color)  
 plt.xticks(x)  
 plt.xlabel(xlabel)  
 plt.ylabel(ylabel)  
 plt.title(title)  
 plt.show()  
  
  
alphabet = Alphabet(RUS\_LOWERCASE\_WITH\_HARD)  
long\_source = TextSourcePlus(PATH\_LAB1 + LONG\_FILE\_NAME, alphabet, "to\_lower", "replace\_ru\_yo",  
 "delete\_delimeters", "delete\_spaces")  
long\_source.apply\_filter\_chain()  
rus\_index\_of\_coincidence = index\_of\_coincidence(long\_source)  
print(f"rus\_index\_of\_coincidence {rus\_index\_of\_coincidence}")  
  
key\_len = int(input("Key len: "))  
  
short\_source = TextSourcePlus(PATH\_LAB2 + SHORT\_FILE\_NAME, alphabet, "to\_lower", "replace\_ru\_yo",  
 "delete\_delimeters", "delete\_spaces")  
short\_source.apply\_filter\_chain()  
short\_index = index\_of\_coincidence(short\_source)  
print(f"Open text index of coincidence {short\_index}")  
  
keywords, ciphered\_dict = cipher\_source(short\_source, 2, 30)  
print(f"keywords {keywords}")  
print(f"ciphered text for key len {key\_len} : {ciphered\_dict[key\_len]}")  
print("Ciphered texts indexes of coincidence")  
for r, ciphered in ciphered\_dict.items():  
 ciph\_source = TextSourcePlus.from\_string(ciphered, alphabet)  
 ciph\_source.apply\_filter\_chain()  
 ciphered\_ind = index\_of\_coincidence(ciph\_source)  
 print(f" {r} {ciphered\_ind}")  
  
v\_d = VizhenerDecipher(TextSourcePlus.from\_string(ciphered\_dict[key\_len], alphabet),  
 rus\_index\_of\_coincidence)  
print("\nperiodic\_method1\_period\_finder")  
periodic\_ids\_list = v\_d.periodic\_method1\_period\_finder(return\_all=True)  
print("period index of coincidence")  
x = []  
y = []  
for r, id\_of\_coid, \_, \_ in periodic\_ids\_list:  
 print(f"{r:5} {id\_of\_coid:6.4f}")  
 x.append(r)  
 y.append(id\_of\_coid)  
plot\_bar(x, y, "period", "index of coincidence", "indexes of coincidence for different periods")  
  
print("non\_periodic\_method1\_period\_finder")  
print(v\_d.non\_periodic\_method1\_period\_finder(ciphered\_dict))  
  
print("\nstat\_equals\_method2\_period\_finder")  
stats\_dict = v\_d.stat\_equals\_method2\_period\_finder(return\_all=True)  
print("period equals stats")  
x = []  
y = []  
for r, stat\_eq in stats\_dict.items():  
 print(f"{r:5} {stat\_eq:6}")  
 x.append(r)  
 y.append(stat\_eq)  
plot\_bar(x, y, "period", "stats value", "Stats values (D) for different periods", color='g')  
  
print("detect\_key\_period")  
period = v\_d.detect\_key\_period()  
if period:  
 print(period)  
else:  
 print("Can't detect key period")  
  
print("\nDeciphering ciphered text (variant 10)")  
ciphered\_source = TextSourcePlus(PATH\_LAB2 + CIPHERED\_FILE\_NAME, alphabet)  
v\_d = VizhenerDecipher(ciphered\_source, rus\_index\_of\_coincidence)  
keyword, open\_text = decipher(v\_d, alphabet, long\_source)  
  
line\_len = 60  
print(f"\nIn the end of deciphering keyword is: {keyword}")  
print("Open text is:")  
for i in range(0, len(open\_text), line\_len):  
 print(open\_text[i: i + line\_len])

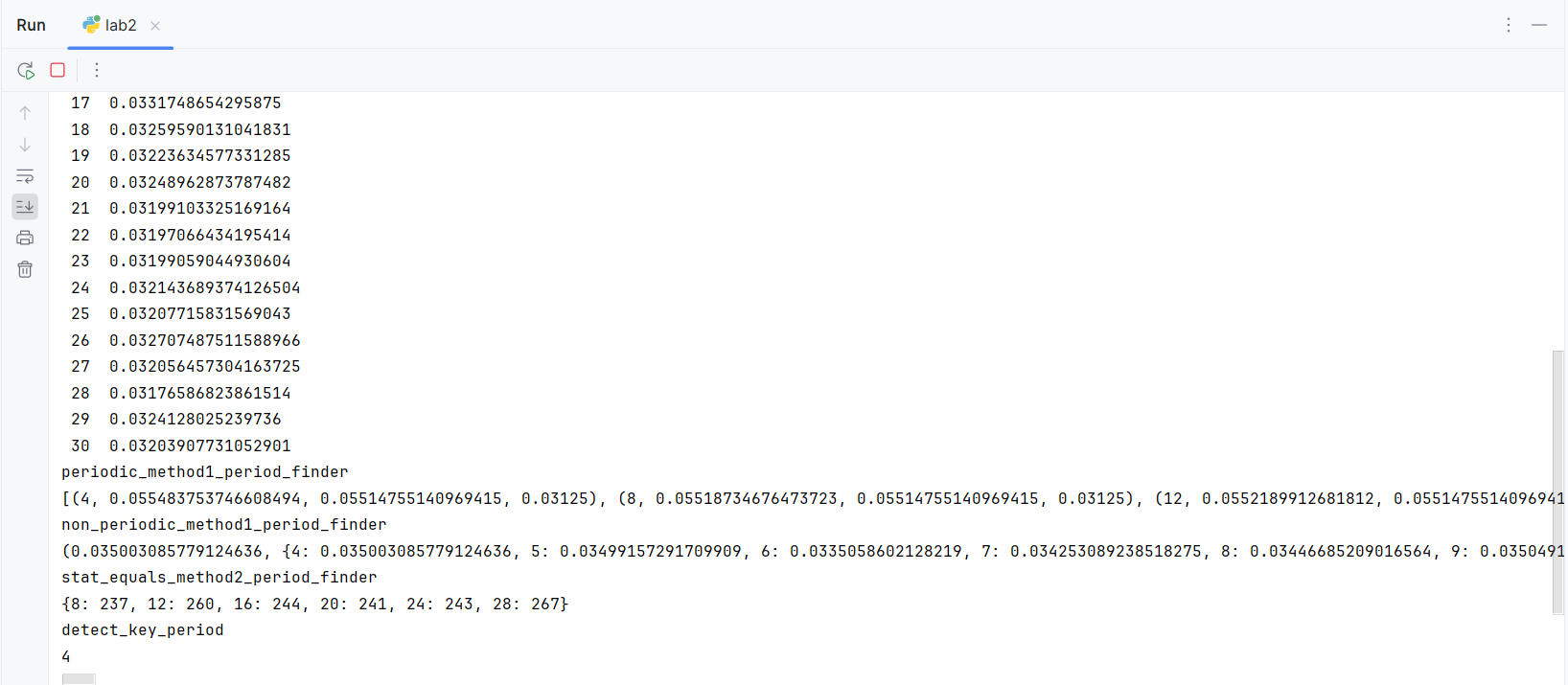
**6. Запуск програми**

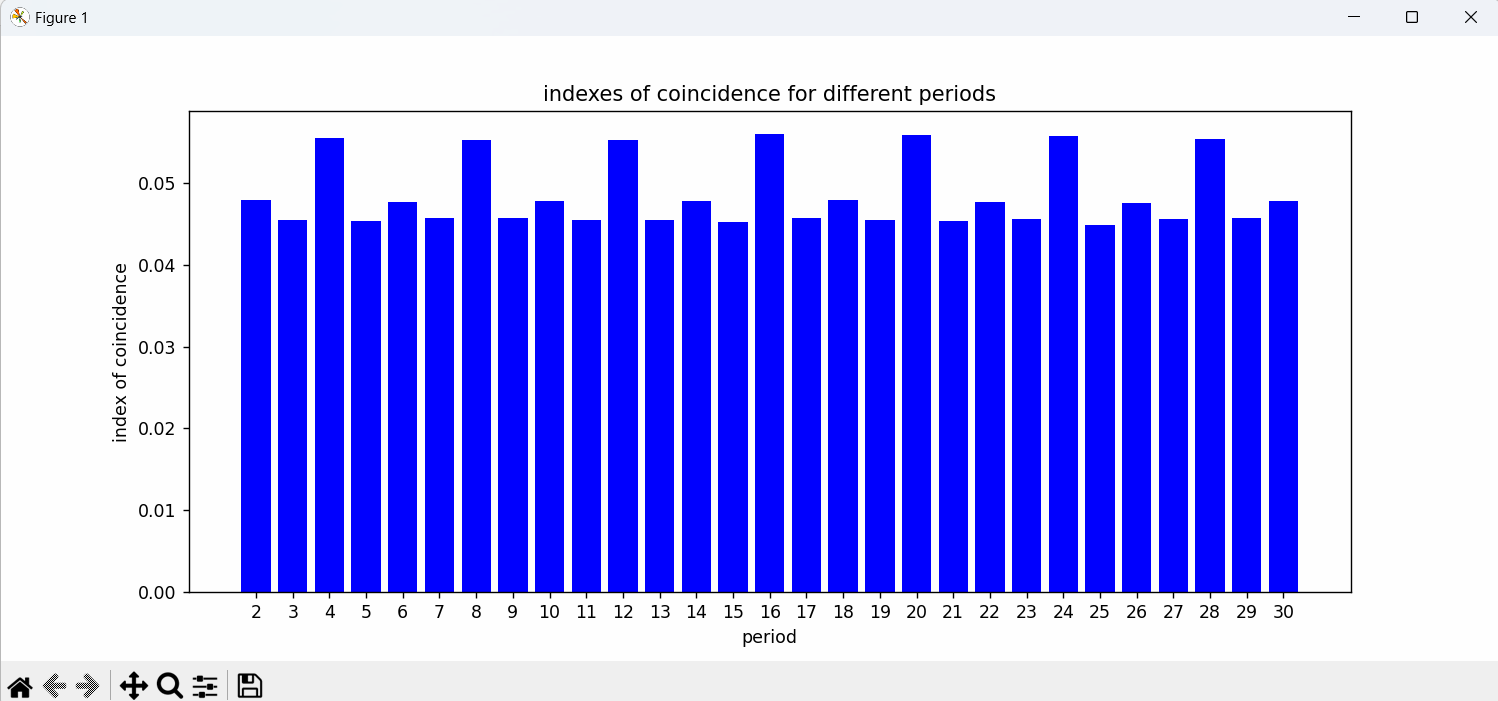
Запустимо програму.

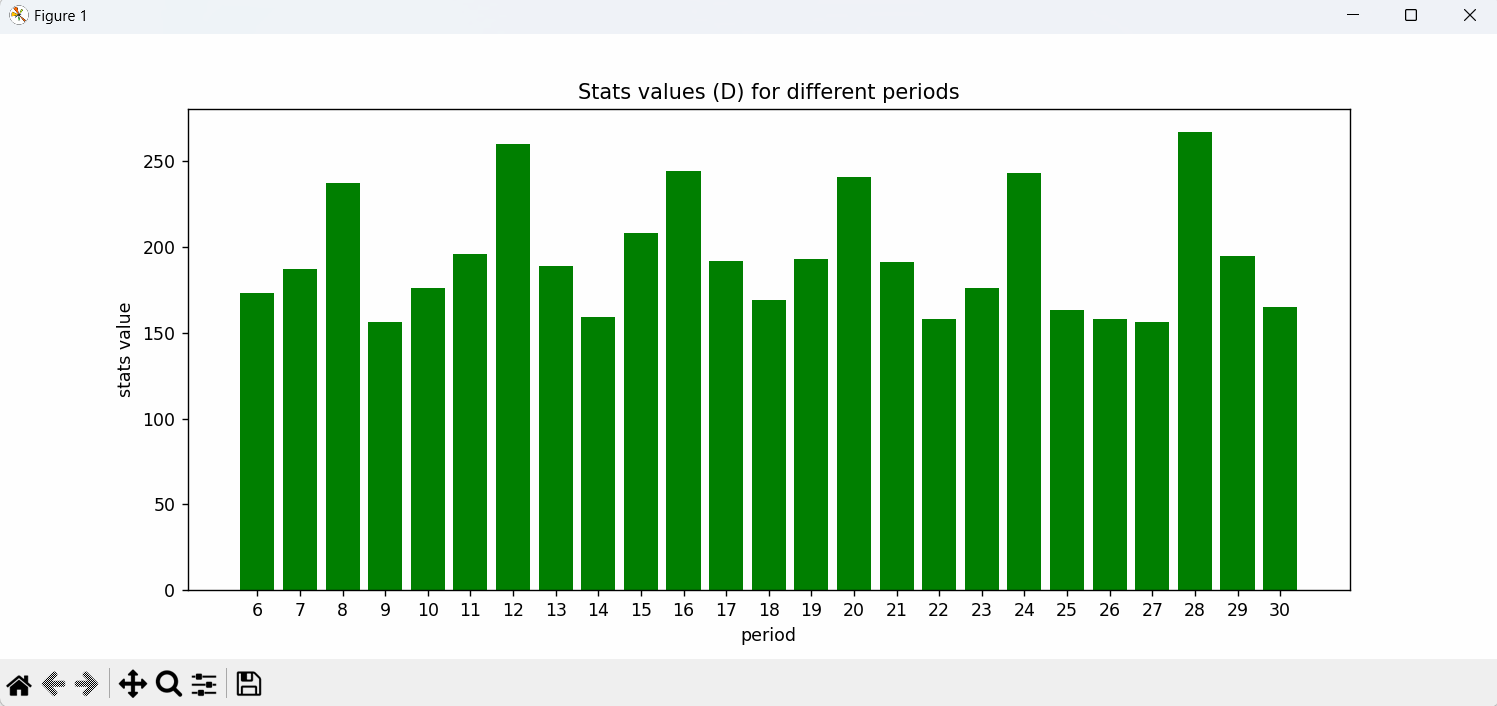


Бачимо спочатку пораховані значення індексу відповідності для мови та невеликого відкритого тексту. Вони відрізняються несуттєво.

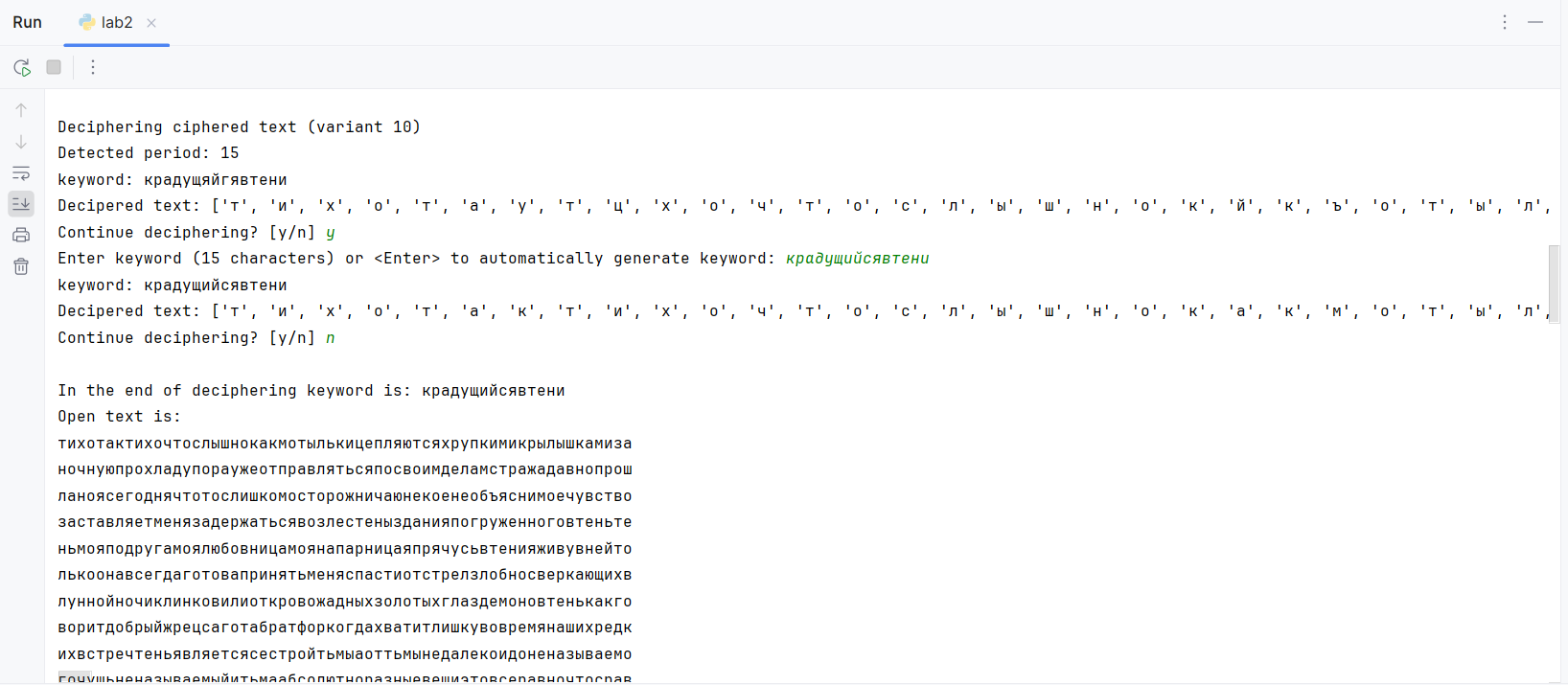
Також бачимо список випадкових ключів для шифрування тексту з періодом 2, 3 і т.д. та зашифрований текст з вказаною довжиною ключа(4). Також виводяться значення індексів відповідності для цих зашифрованих текстів.







Далі бачимо результати викликів методів визначення періодів ключа та визначення періоду ключа для введеного значення(4). Також бачимо діаграми індексів відповідності та статистики для різних значень ключа.



Після початку дешифрування зашифрованого тексту бачимо, що ключ та текст майже розшифровані. Вводимо у діалозі правильне на наш погляд ключове слово та пересвідчуємось, що текст повністю дешифрований.

Висновок

У цій роботі було виконано криптоаналіз тексту, зашифрованого шифром Віженера. Знайдено період ключа та дешифровано зашифрований текст. Вивчено різні підходи до знайдення періоду ключа та проаналізовано їх працездатність. На підставі цього аналізу розроблено розпізнавач періоду ключа шифру Віженера. Також розроблено програми для дешифрування тексту в інтерактивному режимі.