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(cipher) 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.

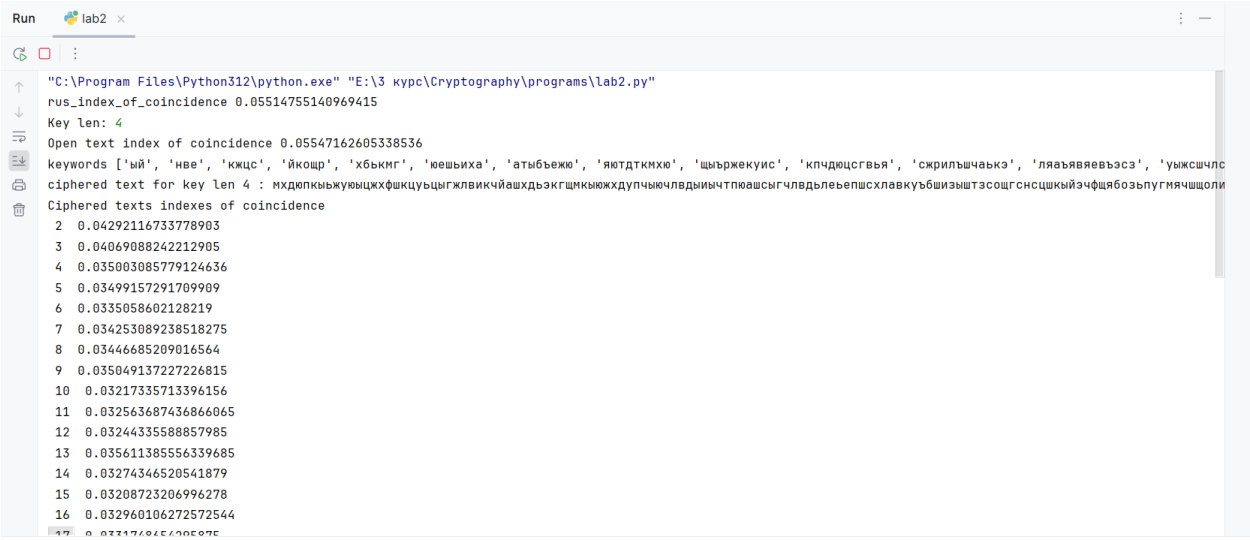
```
import matplotlib.pyplot as plt

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])
```

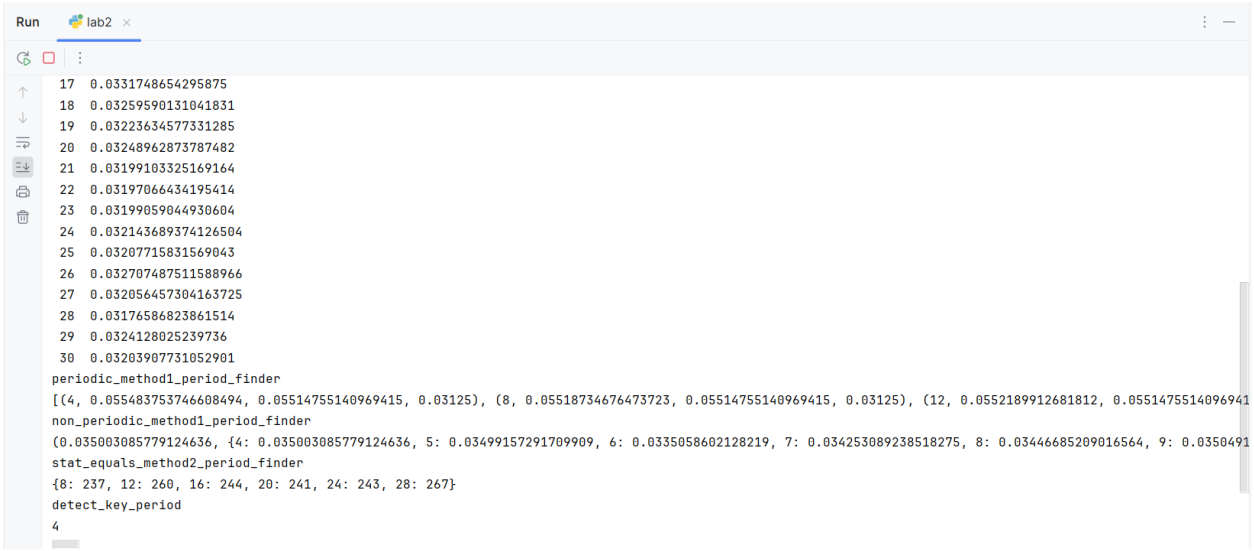
6. Запуск програми

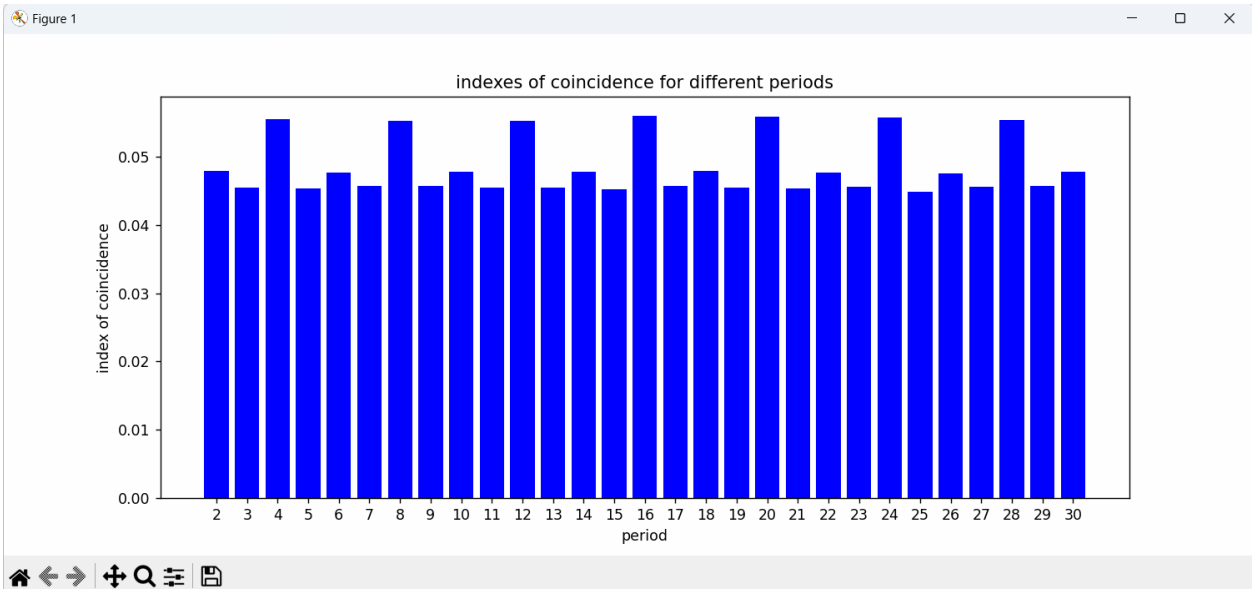
Запустимо програму.

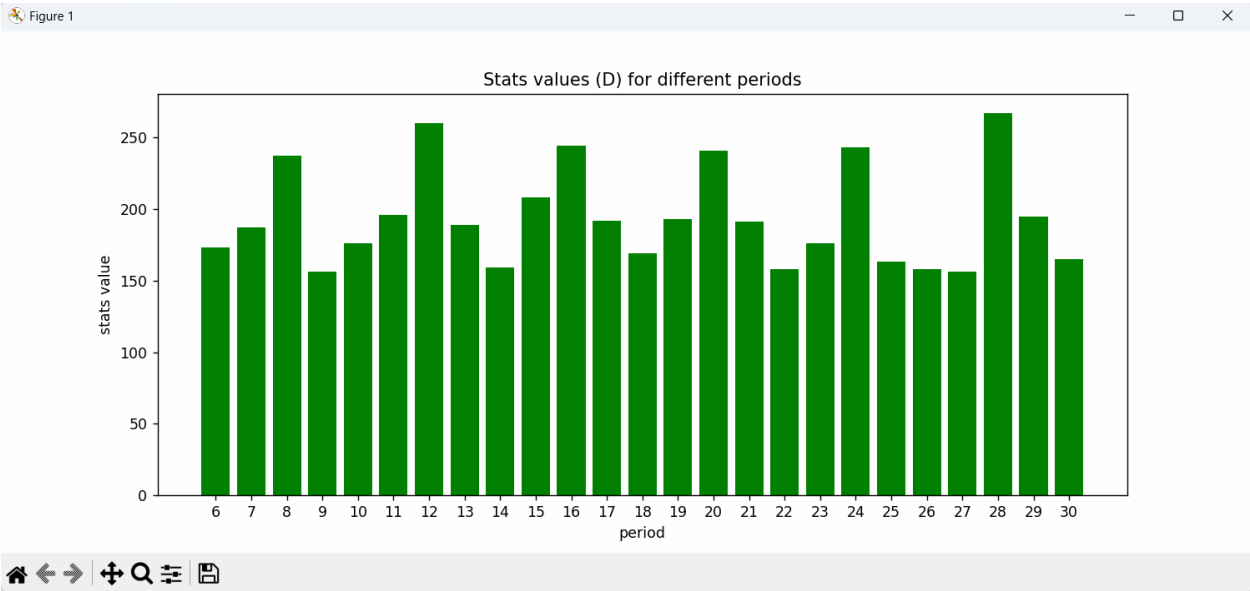


Бачимо спочатку пораховані значення індексу відповідності для мови та невеликого відкритого тексту. Вони відрізняються несуттєво.

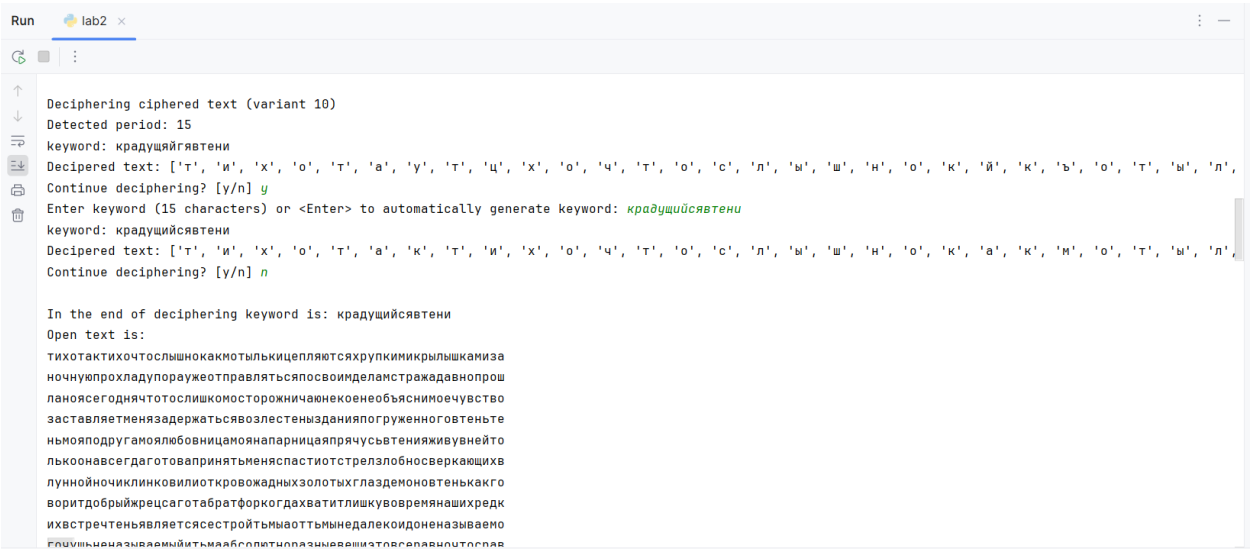
Також бачимо список випадкових ключів для шифрування тексту з періодом 2, 3 і т.д. та зашифрований текст з вказаною довжиною ключа(4). Також виводяться значення індексів відповідності для цих зашифрованих текстів.







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.