

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 №1, 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 m^2 .

```
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` – розв'язання порівняння вигляду $ax \equiv b \pmod n$, `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 a можна використати таке порівняння:

$$(Y^* - Y^{**}) \equiv a(X^* - X^{**}) \pmod{m^2}$$

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

$$(Y^* - Y^{**})a^{-1} \equiv (X^* - X^{**}) \pmod{m^2}$$

У цьому порівнянні невідомим є a^{-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 a^{-1} .

Отже, для отримання значень a 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 a у розумінні модульної арифметики, тобто розв'язку порівняння $a \equiv 1 \pmod m$. 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 = "абвгдежзийклмнопрстуфхцчшщъыьэюя"
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()
    criterial1 = 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(criterial1, criteria2, criteria3, criteria4)
    return criterial1 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)
            deciphered = affine_bigram.decipher(source.filtered_as_string)
            if deciphered is None:
                continue
```

```

        open_text_candidate = ''.join(deciphered)
        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_delimiters", "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)

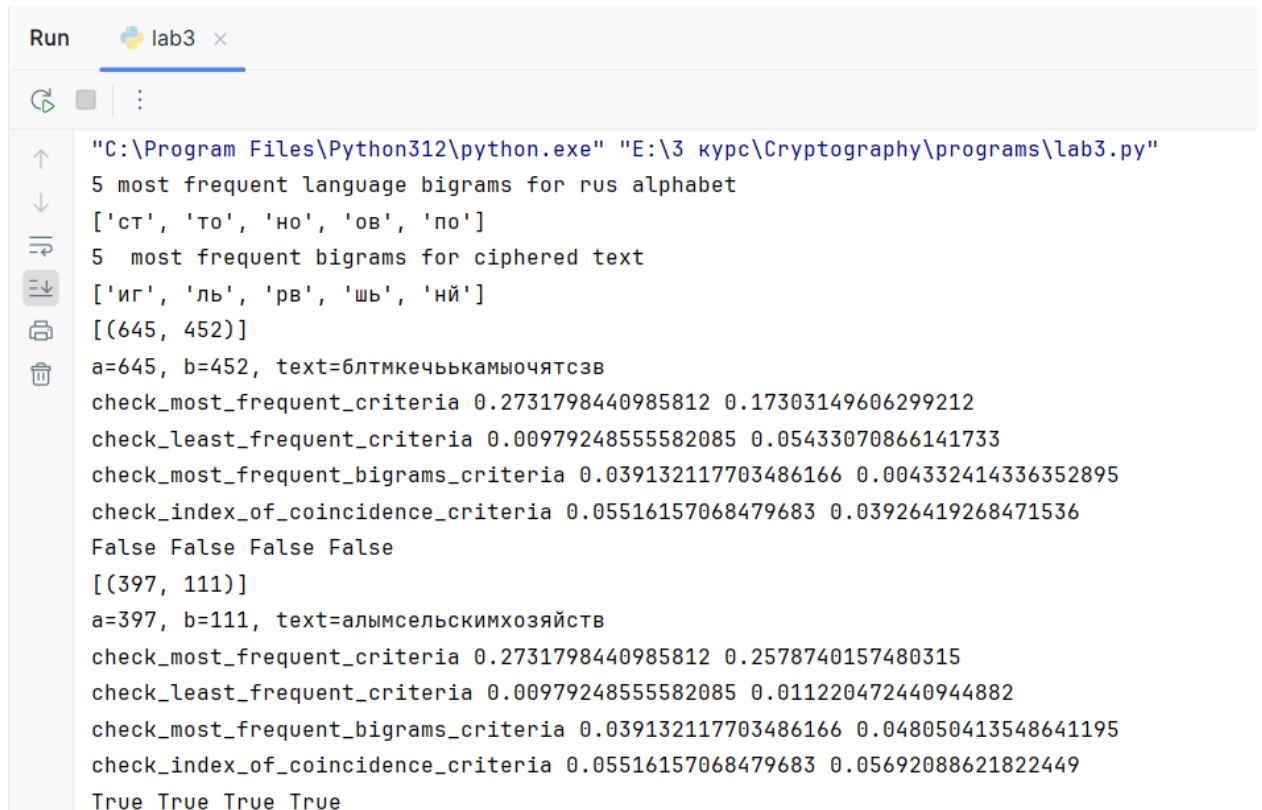
open_text = decipher(ciphered_source, bigram_alphabet,
                     bigrams_most_frequent, rus_bigrams_most_frequent,
                     rus_unigrams.get_ngrams_frequencies(to_sort=True),
                     rus_bigrams.get_ngrams_frequencies(to_sort=True),
                     rus_index_of_coincidence)

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.



```
Run lab3 x
"C:\Program Files\Python312\python.exe" "E:\3 курс\Cryptography\programs\lab3.py"
5 most frequent language bigrams for rus alphabet
['ст', 'то', 'но', 'ов', 'по']
5 most frequent bigrams for ciphered text
['иг', 'ль', 'рв', 'шь', 'нй']
[(645, 452)]
a=645, b=452, text=блтикечькамьочятсзв
check_most_frequent_criteria 0.2731798440985812 0.17303149606299212
check_least_frequent_criteria 0.00979248555582085 0.05433070866141733
check_most_frequent_bigrams_criteria 0.039132117703486166 0.004332414336352895
check_index_of_coincidence_criteria 0.05516157068479683 0.03926419268471536
False False False False
[(397, 111)]
a=397, b=111, text=алымсельскимхозяйств
check_most_frequent_criteria 0.2731798440985812 0.2578740157480315
check_least_frequent_criteria 0.00979248555582085 0.011220472440944882
check_most_frequent_bigrams_criteria 0.039132117703486166 0.048050413548641195
check_index_of_coincidence_criteria 0.05516157068479683 0.05692088621822449
True True True True
```

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 a, 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.

Run lab3 x





In the end of deciphering open text is:

алымсельскимхозяйствомвструктуреееэкономикипреобладалосельск
оохозяйствовегодывнебылоболееполовинычисленностизанятыхионо
давалоприблизительнонациональногодоходасгодасталаосуществлят
сяпрограммаиндустриализациивенгриизапоследующиелетпромышлен
ноепроизводствостранывозрасловразпосравнениюсдовоеннымуровне
мнаиболеебыстрыми темпамиразвивалисьтакиеважнейшиесточкизрени
ятехническогоразвитиявсегонародногохозяйстваотрасликакэлектр
оэнергетикамашиностроениеихимияпосуществузановобылсозданрядот
раслейсовременногомашиностроениянапримерпроизводствоавтобусо
виузловдляавтомашинтехникисвязимедицинскогооборудованияприбо
ростроенияидрповыпускунекоторыхизделийпромышленностивенгрияз
анимаетзаметноеместовмировомпроизводствеиэкспортевчастностиэ
танебольшаястранавсегонаселениямираобеспечиваетоколомирового
экспортаавтобусовэлектралампмедикаментовразвитиеведущихотрас
лейнародногохозяйстваобусловилосущественныйподъемэкономикист
ранывцеломобъемнациональногодоходавгодувозросприблизительнов
разпосравнениюсдовоеннымуровнемврезультатезначительногоинду
стриальногоразвитияпроизошлиизменениявэкономическойструктуре
венгриидоляпромышленностивнациональномдоходестраныувеличилас
ьсвгодудоприблизительноизмениласьструктурасамойпрмышленност
ивкоторойдолямашиностроенияиметаллообработкиподняласьдохимияд
овсеэтипоказателихарактеризуютвенгриюкаксреднеразвитоеиндуст
риальноаграрноегосударствосразвитымсельскимхозяйствомглавное
местосредиотраслейвенгерскойиндустрииизанимаетмашиностроениеи
акотороеприходитсяоколополовойпродукцииисвышезанятыхвпромышл
ностичастуотраслейприходитсяпримернополовоэкспортстраныного



а которого приходится около половины продукции и свыше занятых в промышленности на эту отрасль приходится примерно всего экспорта страны. Номенклатура производимых изделий весьма широка и составляет по оценкам до 10% номенклатуры мировой машиностроительной продукции. Венгрия специализируется главным образом на производстве автобусов, дизельных моторов, станков, порталных и плавающих кранов. Производятся также телефонные центры и различные средства связи, в том числе периферийные устройства, в том числе электрооборудование и измерительные приборы на экспорт. Отправляется более 60% продукции венгерского машиностроения. Наиболее крупным центром развития машиностроения является Будапешт, здесь выпускается почти 80% продукции отрасли. Крупными центрами машиностроения являются также Едьерд, Брецен, Секешфехервар, на черную металлургию приходится около 10% индустриального производства. Она представлена крупными комбинатами. В Дунауйвароше, где и находится производство, базируется в большой степени на привозном сырье, импортируется вся железная руда, основная часть коксующегося угля, более половины кокса, легированные металлы. Главный поставщик страны бывшего советского союза цветной меллургии, высокие уровни развития выделяется алюминевая промышленность, использующая крупные запасы бокситов. Крупнейшие заводы в Варпаноте и Кетабане. Алюминиевый прокат выпускается Секешфехерварским комбинатом. Производство и потребление алюминевых изделий на душу населения Венгрии занимает одно из первых мест в мире. Значительная часть этих изделий и проката поступает на экспорт. Основные предприятия химической промышленности нафтехимические комплексы в Сазхаломбаште и Ленивароше существуют. Производство пластмасс, синтетических материалов, минеральных удобрений. Основное направление специализации Венгрии в химической промышленности — выпуск лекарств, средств защиты растений и полупроводников.


```
Run lab3 x
↑
↓
↺
↻
🖨
🗑
койпромышленнистивыпусклекарствсредствзащитырастенийиполуфаб
рикатовдляихпроизводстваосновныефармацевтическиепредприятиях
иноинигедеонрихтерещедовоенныекчислуважныхотраслеймеждународ
нойспециализациивенгрииотносятсялегкаяипищеваяпромышленность
нанихприходитсяоколополовойпродукциииболеечемзанятыхпозтимпо
казателямстранаопережаетостальныегосударствавосточноевропыв
которыевывозитсяоколополовиныэкспортируемойэтимитотраслямпро
дукциитемпыростааграрногопроизводстваввенгрииивпериодхгодовбы
лиоднимиизсамыхвысокихвмирезаэтимстояликрупныекапиталовложен
иявсельскоехозяйствосозданиедлянегосовременнойматериальнотех
ническойбазыширокоевнедрениевхозяйственнуюпрактикунаучнотехн
ическихдостиженийбольшоеразвитиеполучиливсевозможныевесьмаде
йственныеформыматериальногостимулированиячтонарядусвысокимуро
внеморганизованностиспособствовалобыстромуруступроизводствав
итоgetherпериодхгодовобъемсельскогохозяйственнойпродукцииудвоилс
яапродукциипищевойпромышленностиувеличилсябольшечемвразкчисл
усущественныхособенностейиодновременноважныхфакторовдинамиче
скогоразвитиявенгрииотноситсяорганическоевключениевсистемуаг
рарногопроизводстваличныхподсобныххозяйствсредисущественныхо
собенностейвенгерскойэкономикиследуетотметитьвысокуюстепенье
еучастиявмеждународномразделениитрудаобъемвнешнеторговлисос
тавляетпочтипосравнениюстоимостиваловогонациональногопродукт
анаэкспортпоступаетвсреднеоколовыпускаемойпродукциивтомчисл
еболеечемвдесятикрупнейшихотрасляхэтадолясоставляетсвышескак
имотрасляотносятсявчастностиалюминиеваяпромышленностьфармац
евтикаприборостроениепроизводствоавтобусовбувицелыйрядподот
раслейсельскогохозяйстваипищевойпромышленностипоказателивовл
раслейсельскогохозяйстваипищевойпромышленностипоказателивовл
еченностивсистемумировыхэкономическихсвязейиэкспортнойориент
ацииувенгрииивышечемуостальныхвосточноевропейскихстранибольши
нствазападноевропейскихгосударствовзначительнойстепениобъяс
няетсяспецификойразвитиянародногохозяйстваотносительнойбедно
стьюполезнымиископаемымиузоотствомвнутреннегорынкавпоследнеевре
мяопределяющейтенденциейразвитияэкономикивенгрииивляетсядаль
нейшаяинтенсификациявнешнеторговывнешнеэкономическихсвязей
активноеучастиеевропейскихинтеграционны
Process finished with exit code 0
```

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.