Міністерство освіти і науки України
Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"
Фізико-технічний інститут

КРИПТОГРАФІЯ
КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3
Криптоаналіз афінної біграмної підстановки

Виконали студенти групи ФБ-32:
Красноок Юлія та Водяник Дмитро

Київ - 2025

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

Порядок виконання 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.
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.

Хід роботи:

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.

Код програми:

```
import sys
from typing import List, Tuple, Optional

ALPHABET = "абвгдежзийклмнопрстуфхцчшщыьэюя"
M = len(ALPHABET)
M_SQUARED = M * M

print(f"--- Налаштування ---")
print(f"Алфавіт (m = {M} символів): {ALPHABET}")
print(f"Модуль (m^2): {M_SQUARED} (тобто {M}*{M})")

def extended_gcd(a: int, b: int) -> Tuple[int, int, int]:
```

```
    if a == 0:
        return (b, 0, 1)
    gcd, u1, v1 = extended_gcd(b % a, a)
    u = v1 - (b // a) * u1
    v = u1
    return (gcd, u, v)

def mod_inverse(a: int, n: int) -> Optional[int]:
    gcd, u, v = extended_gcd(a, n)
    if gcd != 1:
        return None
    else:
        return u % n

def solve_linear_congruence(a: int, b: int, n: int) ->
List[int]:
    a = a % n
    b = b % n
    d, u, v = extended_gcd(a, n)
    solutions = []
    if b % d != 0:
        return []
    a1 = a // d
    b1 = b // d
    n1 = n // d
    a1_inv = mod_inverse(a1, n1)
    if a1_inv is None:
        print("Помилка: неможливо знайти обернений елемент для
a1.", file=sys.stderr)
        return []
    x0 = (b1 * a1_inv) % n1
    for i in range(d):
        solutions.append(x0 + i * n1)
    return solutions

print("\n--- Тестування математичного ядра ---")

a_test_1 = 5
n_test = M_SQUARED
inv_1 = mod_inverse(a_test_1, n_test)
print(f"1. Обернений до {a_test_1} mod {n_test} = {inv_1}")
if inv_1 is not None:
    print(f" Перевірка: ({a_test_1} * {inv_1}) % {n_test} =
{(a_test_1 * inv_1) % n_test} (має бути 1)")
```

```
a_test_2 = 31
inv_2 = mod_inverse(a_test_2, n_test)
print(f"\n2. Обернений до {a_test_2} mod {n_test} = {inv_2} (має бути None)")

a_test_3 = 5
b_test_3 = 7
sols_3 = solve_linear_congruence(a_test_3, b_test_3, n_test)
print(f"\n3. Розв'язки для {a_test_3}x ≡ {b_test_3} (mod {n_test}): {sols_3}")
if sols_3:
    print(f" Перевірка: ({a_test_3} * {sols_3[0]}) % {n_test} = {(a_test_3 * sols_3[0]) % n_test} (має бути {b_test_3})")

a_test_4 = 31
b_test_4 = 7
sols_4 = solve_linear_congruence(a_test_4, b_test_4, n_test)
print(f"\n4. Розв'язки для {a_test_4}x ≡ {b_test_4} (mod {n_test}): {sols_4} (має бути [])")

a_test_5 = 31
b_test_5 = 62
sols_5 = solve_linear_congruence(a_test_5, b_test_5, n_test)
print(f"\n5. Розв'язки для {a_test_5}x ≡ {b_test_5} (mod {n_test}):")
print(f" Кількість розв'язків: {len(sols_5)} (має бути 31)")
print(f" Перші 5 розв'язків: {sols_5[:5]} (має бути [2, 33, 64, 95, 126])")
if sols_5:
    print(f" Перевірка (перший): ({a_test_5} * {sols_5[0]}) % {n_test} = {(a_test_5 * sols_5[0]) % n_test} (має бути {b_test_5})")
    print(f" Перевірка (останній): ({a_test_5} * {sols_5[-1]}) % {n_test} = {(a_test_5 * sols_5[-1]) % n_test} (має бути {b_test_5})")
```

Результат:

```
--- Налаштування ---
Алфавіт (m = 31 символів): абвгдежзийклмнопрстуфхцчшщыьэюя
Модуль (m^2): 961 (тобто 31*31)

--- Тестування математичного ядра ---
1. Обернений до 5 mod 961 = 769
   Перевірка: (5 * 769) % 961 = 1 (має бути 1)

2. Обернений до 31 mod 961 = None (має бути None)

3. Розв'язки для 5x ≡ 7 (mod 961): [578]
   Перевірка: (5 * 578) % 961 = 7 (має бути 7)

4. Розв'язки для 31x ≡ 7 (mod 961): [] (має бути [])

5. Розв'язки для 31x ≡ 62 (mod 961):
   Кількість розв'язків: 31 (має бути 31)
   Перші 5 розв'язків: [2, 33, 64, 95, 126] (має бути [2, 33, 64, 95, 126])
   Перевірка (перший): (31 * 2) % 961 = 62 (має бути 62)
   Перевірка (останній): (31 * 932) % 961 = 62 (має бути 62)

Process finished with exit code 0
```

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

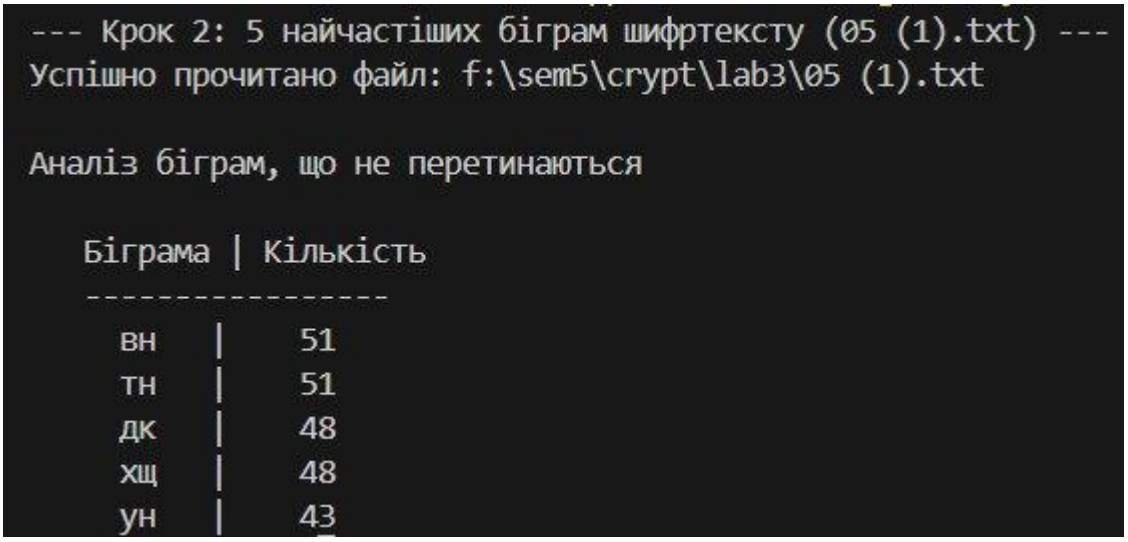
Код:

```
from collections import Counter
import re
import os
from pathlib import Path

def get_non_overlapping_bigram_counts(text: str) -> Counter:
    bigrams = [text[i:i+2] for i in range(0, len(text) - 1, 2)]
    counts = Counter(bigrams)
    return counts

def main():
    filename_to_find = "05 (1).txt"
    script_dir = Path(__file__).parent
    filepath = script_dir / filename_to_find
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            ciphertext = file.read()
        cleaned_text = re.sub(r'\s+', '', ciphertext)
```

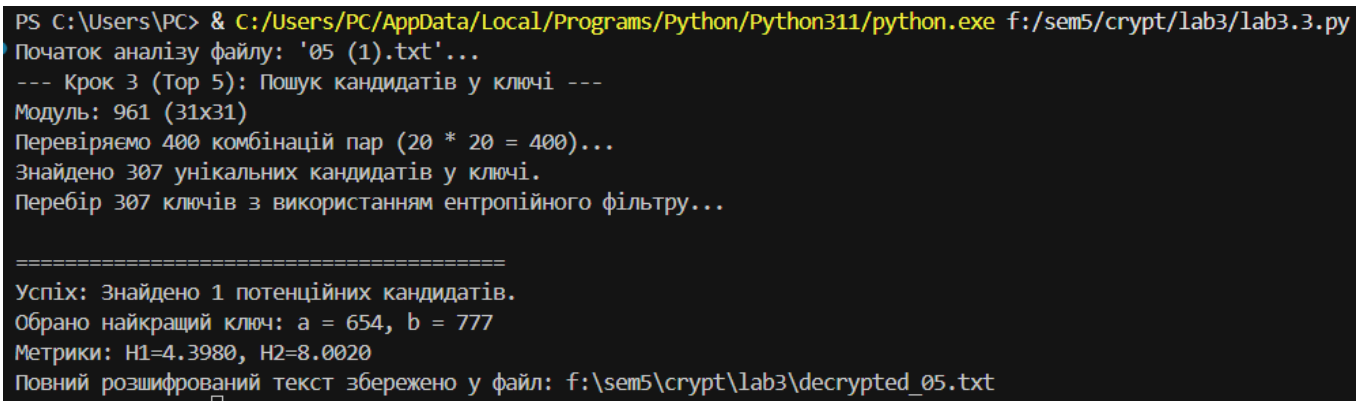
```
        bigram_counts = get_non_overlapping_bigram_counts(cleaned_text)
        top_5 = bigram_counts.most_common(5)
                print(f"--- Крок 2: 5 найчастіших біграм шифртексту
({filename_to_find}) ---")
        print(f"Успішно прочитано файл: {filepath}\n")
        print("Аналіз біграм, що не перетинаються\n")
        print(" Біграма | Кількість")
        print(" -----------------")
        for bigram, count in top_5:
            print(f" {bigram} | {str(count).ljust(2)}")
    except FileNotFoundError:
        print(f"ПОМИЛКА: Файл '{filename_to_find}' не знайдено.")
        print(f"Я шукав його тут: {filepath}")
            print(f"Будь ласка, переконайтеся, що файл з таким іменем
знаходиться в тій же папці, що і скрипт.")
    except Exception as e:
        print(f"Виникла інша помилка: {e}")

if __name__ == "__main__":
    main()
```

Варіант 5

Після виконання коду ми визначили, що 5 найчастіших біграм у нашому шифротексті це:



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.



Відповідно ми знайшли два найкращі ключі a-654 та b-777, які розшифрували текст

Висновок: У ході виконання комп'ютерного практикуму було набуто практичних навичок частотного аналізу шифртекстів та розкриття афінної біграмної підстановки. Реалізовано основні математичні підпрограми для роботи з модулярною арифметикою обчислення оберненого елемента за модулем за допомогою розширеного алгоритму Евкліда та розв'язання лінійних конгруенцій. Проведено аналіз частот біграм шифртексту, визначено найчастіші біграми, підібрано можливі ключі шифрування та виконано дешифрування тексту. У результаті було отримано змістовний відкритий текст, що підтвердило правильність знайденого ключа та засвоєння методів криптоаналізу моноалфавітних підстановок.