КРИПТОГРАФІЯ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №3

Криптоаналіз афінної біграмної підстановки

Виконав: Стадник Юрій ФБ-12

ВР12

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

Порядок виконання 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.

Розширений алгоритм Евкліда:

```python
def euclidean_algorithm(a, b):
    if not a:
        return b, 0, 1

    gcd, u1, v1 = euclidean_algorithm(b % a, a)
    u = v1 - (b // a) * u1
    v = u1

    return gcd, u, v
```

Скрипт для розв'язування лінійних порівнянь, що виводить усі можливі розв'язки

```python
def modular_linear_equation_solver(a, b, n):
    a = a % n
    b = b % n

    d, u, v = euclidean_algorithm(a, n)
    if b % d != 0:
        return []

    x0 = (u * (b // d)) % n
    if x0 < 0:
        x0 += n

    return [(x0 + i * (n // d)) % n for i in range(d)]
```

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

5 найчастіших біграм відкритого тексту були взяті з методичних вказівок до лабораторної роботи

```
FREQUENT_BIGRAMS_OPEN_TEXT = ['ст', 'но', 'то', 'на', 'ен']
```

А 5 найчастіших біграм шифртексту обчислили за допомогою цієї функції

```
def calculate_frequencies(text):
    frequencies = {}
    bigrams_count = 0
    for i in range(0, len(text) - 1, 2):
        bigram = text[i:i + 2]
        bigrams_count += 1
        if bigram in frequencies:
            frequencies[bigram] += 1
        else:
            frequencies[bigram] = 1
    normalized_frequencies = {bigram: frequency / bigrams_count for bigram, frequency in frequencies.items()}
    return normalized_frequencies
```

```
frequencies = calculate_frequencies(input_text)
frequencies_list = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)[:5]
top_bigrams_cyphered = [bigram[0] for bigram in frequencies_list]
```

Можемо вивести найчастіші біграми відкритого (зверху) та шифрованого (знизу) текстів

```
['ст', 'но', 'то', 'на', 'ен']
['хк', 'ек', 'вю', 'пн', 'вх']
```

За допомогою функції **generate_key** (що використовує описану вище функцію для розв'язання лінійних порівнянь) та **generate_keys** (що викликає функцію generate_key для кожної унікальної пари відкритої/шифрованої біграм) можемо згенерувати усі можливі ключі для цього шифртексту

```
def generate_key(bigram1, bigram2):
    x1, y1 = get_bigram_id(bigram1[0]), get_bigram_id(bigram1[1])
    x2, y2 = get_bigram_id(bigram2[0]), get_bigram_id(bigram2[1])
    roots = modular_linear_equation_solver(x1 - x2, y1 - y2, len(ALPHABET) ** 2)
    if not roots:
        return
    keys = []
    for root in roots:
        a = root
        b = (y1 - a * x1) % (len(ALPHABET) ** 2)
        keys.append((a, b))
    return keys


def generate_keys(open_bigrams, cyphered_bigrams):
    pairs = [(open_bigram, cyphered_bigram) for open_bigram in open_bigrams for cyphered_bigram in cyphered_bigrams]
    keys = []
    for i in pairs:
        for j in pairs:
            if i == j or i[1] == j[1]:
                continue
            roots = generate_key(i, j)
            if roots:
                keys.extend(roots)
    return set(keys)
```

Отримали 229 можливих ключів

```
Found 229 possible keys
```

3. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата. Повторювати дії 3-4 доти, доки дешифрований текст не буде змістовним.

Оскільки даний текст є досить великим, то можна обійтися використанням лише методу неможливих біграм, проте для менших текстів його слід комбінувати з іншими методами

Збережемо деякі неможливі біграми російської мови у список

```
IMPOSSIBLE_BIGRAMS = ['аы', 'оы', 'иы', 'ыы', 'уы', 'еы', 'аь', 'оь', 'иь', 'ыь', 'уь', 'еь', 'юы', 'яы', 'эы', 'юь',
                      'яь', 'эь', 'ць', 'хь', 'кь']
```

Функція для дешифровки

```python
def decipher_text(text, a=None, b=None):
    if a is None or b is None:
        return text
    deciphered_text = ''
    for i in range(0, len(text) - 1, 2):
        y = get_bigram_id(text[i:i + 2])
        x = (euclidean_algorithm(a, len(ALPHABET) ** 2)[1] * (y - b)) % (len(ALPHABET) ** 2)
        deciphered_text += ALPHABET[x // len(ALPHABET)] + ALPHABET[x % len(ALPHABET)]
    return deciphered_text
```

Проходячи по усім можливим ключам з попереднього етапу, розшифровуємо текст за допомогою вищеописаної функції та, якщо отриманий текст не містить жодної з неможливих біграм, виводимо його перші 100 символів для ручної перевірки

```python
for a_value, b_value in possible_keys:
    deciphered_text = decipher_text(input_text, a=a_value, b=b_value)
    if all(impossible_bigram not in deciphered_text for impossible_bigram in IMPOSSIBLE_BIGRAMS):
        print(
            f'Found possible solution: a = {a_value}, b = {b_value}, deciphered text = {deciphered_text[:100]}...')
```

У цьому випадку отримали **a = 555, b = 331** та розшифрований текст

«когдапожарныеисоседиушлилеоауфманосталсясдедушкойсполдингомдугласомитомомвсеонизадумчивосмотрелинадо»

```
Found possible solution: a = 555, b = 331, deciphered text = когдапожарныеисоседиушлилеоауфманосталсясдедушкойсполдингомдугласомитомомвсеонизадумчивосмотрелинадо...
```

Цей текст являється уривком з книги Рея Бредбері «Страус»

Висновки: Під час виконання даного комп'ютерного практикуму ми розглянули моноалфавітні підстановки та провели їх порівняння з поліалфавітними, що використовувалися у другому практикумі. Ми також вивчили концепцію модульної арифметики та успішно використали її практично при пошуку ключів для шифрованого тексту.