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

Фізико-технічний інститут

КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3

Криптоаналіз афінної біграмної підстановки

Варіант №6

Виконали:
ФБ-21 Захожий М.
ФБ-21 Хав’юк А.

Київ 2024

Мета роботи:

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

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.
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.

Виконання:

1.

```python
# Розширений алгоритм Евкліда
def extended_gcd(a, m):
    if a == 0:
        return m, 0, 1
    gcd, x1, y1 = extended_gcd(m % a, a)
    x = y1 - (m // a) * x1
    y = x1
    return gcd, x, y
```

Функція *extended_gcd* реалізує розширений алгоритм Евкліда. Цей алгоритм обчислює найбільший спільний дільник (НСД, англ. GCD) двох чисел a та m, а також коефіцієнти x і y, які задовольняють рівняння:

$$a \cdot x + m \cdot y = \gcd(a, m)$$

```
# Обчислення оберненого елемента за модулем
def modular_inverse(a, m):
    gcd, x, _ = extended_gcd(a, m)
    if gcd != 1:
        raise ValueError(f"Немає оберненого елемента {a} за модулем
{m}")
    return x % m
```

Реалізація обчислення оберненого елементу за модулем із використанням розширеного алгоритму Евкліда, описаного вище. Якщо gcd(a,m)≠1, то оберненого елемента не існує.

```
# Розв'язування лінійних порівнянь
def solve_linear_congruence(a, b, m):
    gcd, x, _ = extended_gcd(a, m)
    if b % gcd != 0:
        return []
    a1 = a // gcd
    b1 = b // gcd
    m1 = m // gcd
    x0 = (x * b1) % m1
    solutions = [(x0 + k * m1) % m for k in range(gcd)]
    return solutions
```

Функція *solve_linear_congruence* реалізує розв’язування лінійних порівнянь:

$$a \cdot x \equiv b \pmod{m}$$

У змінній solutions зберігаються всі розв’язки:

$$x_k = (x_0 + k \cdot m_1) \pmod{m}, \quad k = 0, 1, \ldots, \gcd - 1$$

2.

```
# Топ-5 біграм шифрованого тексту
def find_top_5_bigrams(cipher_text, step=2):
    bigram_counter = Counter()
    for i in range(0, len(cipher_text) - 1, step):
        bigram = cipher_text[i:i+2]
        bigram_counter[bigram] += 1
    return bigram_counter.most_common(5)
```

Функція *find_top_5_bigrams* визначає п'ять найпоширеніших біграм у шифрованому тексті.

За замовчуванням крок дорівнює 2, що означає, що біграми формуються без перекриття. Counter — це зручний клас зі стандартної бібліотеки Python (collections), який рахує кількість появ кожної біграми. Біграми додаються до *bigram_counter*, і для кожної біграми збільшується лічильник її кількості. Метод *most_common(5)* повертає список з п'яти найбільш частих біграм разом із їх кількістю у вигляді списку кортежів.

3.

```
    top_5_bigrams = find_top_5_bigrams(cipher_text)
    print(f"\n => Топ-5 біграм шифрованого тексту:\n => {top_5_bigrams}\n")

    language_bigrams = ['ст', 'но', 'то', 'на', 'ен']
    cipher_bigrams = [bigram for bigram, _ in top_5_bigrams]

    # Генерація всіх комбінацій двох біграм з кожного списку
    lang_bigram_pairs = list(permutations(language_bigrams, 2))
    cipher_bigram_pairs = list(permutations(cipher_bigrams, 2))

    # Перебір всіх пар біграм і знаходження ключів
    for lang_pair in lang_bigram_pairs:
        for cipher_pair in cipher_bigram_pairs:
            try:
                keys = solve_for_keys(lang_pair, cipher_pair, CHAR_TO_NUM, M)
                for a, b in keys:
                    decrypted_text = decrypt_affine_bigram(cipher_text, a, b, CHAR_TO_NUM,
NUM_TO_CHAR, M)
                    if is_russian_text_advanced(decrypted_text, RUSSIAN_CHAR_SET,
FREQUENT_LETTERS, RARE_LETTERS, COMMON_TRIGRAMS):
                        print(f"\n ==> Біграми ВТ: {lang_pair}\n ==> Біграми ШТ:
{cipher_pair}\n")
                        print(f" ==> Ключ знайдений: (a={a}, b={b})")
                        print(" ==> Шифрований текст (перші 500 символів):\n")
                        print("*"*100)
                        print(decrypted_text[:500])
                        print("*"*100)
                        return
            except ValueError:
                continue
```

Генеруються всі можливі комбінації двох біграм із частих біграм мови (*language_bigrams*) і шифртексту (*cipher_bigrams*). Використовується функція *permutations* для формування всіх перестановок пар біграм.

Потім перебираються всі можливі пари біграм: *lang_pair* — дві біграми з частих біграм мови, *cipher_pair* — дві біграми з частих біграм шифртексту.

Для кожної пари біграм викликається функція solve_for_keys. Вона розв'язує систему рівнянь (згідно із системою, що зазначена в завданні):

$$X_1 \cdot a + b \equiv Y_1 \pmod{m^2}$$

$$X_2 \cdot a + b \equiv Y_2 \pmod{m^2}$$

де:

- X_1, X_2 — числові представлення мовних біграм.
- Y_1, Y_2 — числові представлення біграм шифртексту.

Потім повертаються всі можливі ключі *a*,*b*.
Для кожного знайденого ключа (*a*,*b*) виконується дешифрування тексту за допомогою функції *decrypt_affine_bigram*. Потім перевіряється, чи є текст змістовним російською мовою за допомогою *is_russian_text_advanced*.

4.

```
# Розпізнавання змістовного тексту
def is_russian_text_advanced(text, char_set, frequent_letters, rare_letters,
common_trigrams):
    valid_text = ''.join(char for char in text if char in char_set)
    total_chars = len(valid_text)
    if total_chars == 0:
        return False
    letter_freq = Counter(valid_text)
    total_trigrams = sum(valid_text.count(trigram) for trigram in common_trigrams)
    frequent_score = sum((letter_freq[char] / total_chars) for char in frequent_letters if
char in letter_freq)
    rare_score = sum((letter_freq[char] / total_chars) for char in rare_letters if char in
letter_freq)
    trigram_score = total_trigrams / (total_chars - 2) if total_chars > 2 else 0
    return frequent_score > 0.2 and rare_score < 0.05 and trigram_score > 0.01
```

Функція *is_russian_text_advanced* визначає, чи є текст змістовним російською мовою, використовуючи статистичні критерії. Вона аналізує текст за кількома аспектами:

- Виділення валідних символів (текст очищується від символів, які не входять до російського алфавіту, що задається через char_set).
- Частота поширених літер (аналізується частота основних російських літер, наприклад, «о», «а», «е»). Для змістовного тексту вона повинна перевищувати певний поріг (у даному випадку >20% від усіх символів).
- Частота рідкісних літер (аналізується частота малопоширених літер, наприклад, «ф», «щ», «ь»). Для змістовного тексту ця частка повинна бути низькою (менш ніж 5%).
- Частота триграм. Перевіряється наявність поширених триграм (наприклад, «про», «что», «как»). У змістовному тексті частка триграм має перевищувати певний поріг (>1%).

Якщо всі три критерії виконуються, функція повертає True, що означає, що текст, ймовірно, змістовний російською мовою. Інакше повертається False.

Результат:

Висновки

У ході роботи було освоєно основи частотного криптоаналізу та методи його застосування для дешифрування тексту, зашифрованого за допомогою афінної підстановки біграм. Було вивчено алгоритми розв'язування лінійних порівнянь, використання розширеного алгоритму Евкліда для знаходження обернених елементів за модулем, а також методи аналізу тексту за частотними характеристиками.

Результати роботи демонструють практичну значущість методів частотного криптоаналізу та дозволяють поглибити розуміння основ криптографії та методів її аналізу.