Міністерство освіти і науки України Національний технічний університет України “Київський політехнічний інститут ім. Ігоря Сікорського” Фізико-технічний інститут

КРИПТОГРАФІЯ
КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3
Криптоаналіз афінної біграмної підстановки
Варіант 7

Виконали студенти:
ФБ-23 Лишиленко Ангеліна
ФБ-23 Тіщенко Олександр

Київ-2024

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

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

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. Спочатку зробили функцію GCD яка обчислює НСД за допомогою розширеного алгоритму Евкліда.

```python
def GCD(a, b):
    if a == 0:
        return b, 0, 1
    gcd, x1, y1 = GCD(b % a, a)
    x = y1 - (b // a) * x1
    y = x1
    return gcd, x, y
```

Потім написали функцію ModularInverse що обраховує обернений елемент. Обраховується НСД за допомогою функції GCD: якщо НСД(a,m)≠1, обернений елемент не існує (викидається помилка), в іншому випадку знаходимо обернений елемент за формулою a * a^-1 = 1(mod m)

```python
def ModularInverse(a, m):                                   #Обчислення a^-1

    gcd, x, _ = GCD(a, m)
    if gcd != 1:
        raise ValueError(f"Обернений елемент для {a} за модулем {m} не існує")
    return x % m
```

Також написали функцію для обчислення лінійних порівнянь LinearCongruence. В цій функції також обчислюємо НСД за допомогою функції GCD. А потім враховуємо три можливі випадки розв'язання:

1) $\gcd(a,n)=1$. В цьому випадку порівняння має один розв'язок: $x \equiv a^{-1}b \pmod{n}$.
2) $\gcd(a,n)=d>1$. Маємо дві можливості:
   2.1) Якщо b не ділиться на d, то порівняння не має розв'язків.
   2.2) Якщо b ділиться на d, то порівняння має рівно d розв'язків x_0, x_0+n_1, $x_0+2n_1 \ldots$, $x_0+(d-1)n_1$, де $a=a_1d$, $b=b_1d$, $n=n_1d$ і x_0 є єдиним розв'язком порівняння $a_1x \equiv b_1 \pmod{n_1}$: $x_0 = b_1 \cdot a_1^{-1} \bmod n_1$.

```python
def LinearCongruence(a, b, m):                              #Лінійне порівняння ax ≡ b (mod m)
    gcd, x, _ = GCD(a, m)
    if b % gcd != 0:
        return []                                           # Розв'язків немає

    x0 = (x * (b // gcd)) % m                               #знаходимо один частковий розв'язок
    if x0 < 0:
        x0 += m

    solutions = []                                          #знаходимо всі розв'язки
    step = m // gcd
    for i in range(gcd):
        solutions.append((x0 + i * step) % m)

    return solutions
```

Що ми отримали:

```
Оберіть опцію:
1. Обчислити обернений елемент за модулем
2. Розв'язати лінійне порівняння
3. Завершити програму
-->: 1
Введіть число a: 3
Введіть модуль m: 5
Обернений елемент числа 3 за модулем 5: a^-1= 2

Оберіть опцію:
1. Обчислити обернений елемент за модулем
2. Розв'язати лінійне порівняння
3. Завершити програму
-->: 1
Введіть число a: 2
Введіть модуль m: 4
Обернений елемент для 2 за модулем 4 не існує
```

```
Оберіть опцію:
1. Обчислити обернений елемент за модулем
2. Розв'язати лінійне порівняння
3. Завершити програму
-->: 2
Введіть коефіцієнт a: 4
Введіть число b: 12
Введіть модуль m: 8
Розв'язки лінійного порівняння 4x ≡ 12 (mod 8): [3, 5, 7, 1]
```

```
Оберіть опцію:
1. Обчислити обернений елемент за модулем
2. Розв'язати лінійне порівняння
3. Завершити програму
-->: 2
Введіть коефіцієнт a: 4
Введіть число b: 5
Введіть модуль m: 8
Розв'язків немає
```

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

Ми генеруємо всі біграми в тексті, рахуємо кількість появ кожної біграми, обираємо 5 найпоширеніших біграм та виводимо на екран

```
# Обчислення біграм
bigrams_list = generate_bigrams(raw_text)
bigram_counts = Counter(bigrams_list)

# Пошук найчастіших біграм
popular_bigrams_with_counts = top_bigrams(bigram_counts, 5)
print("Найчастіші біграми шифртексту з кількістю появ:")
for bigram, count in popular_bigrams_with_counts:
    print(f"{bigram}: {count}")

# Отримання лише біграм без частоти
popular_bigrams = [bigram for bigram, _ in popular_bigrams_with_counts]
```

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

Створюємо всі можливі відповідності між п’ятьма найчастішими біграмами шифртексту та частими біграмами мови(common_bigrams = ['ст', 'но', 'то', 'на', 'ен'])

```
# Генерація можливих ключів
unique_pairs = set()
possible_permutations = permutations_with_mapping(popular_bigrams, common_bigrams)
for mapping in possible_permutations:
    unique_pairs.update(set(mapping.items()))
```

Ця частина коду розв’язує систему рівнянь для кожної пари біграм:

```python
# Розв'язання системи рівнянь для ключа
def solve_linear_congruence(pair):
    solutions = []
    y1, y2 = bigram_to_value(pair[0][0]), bigram_to_value(pair[1][0])
    x1, x2 = bigram_to_value(pair[0][1]), bigram_to_value(pair[1][1])
    dy, dx = (y1 - y2) % modulus, (x1 - x2) % modulus

    if dx == 0 or dy == 0:
        return []

    divisor = gcd_extended(dx, modulus)
    if divisor > 1:
        if dy % divisor != 0:
            return []
        dy //= divisor
        dx //= divisor
        modulus_reduced = modulus // divisor
    else:
        modulus_reduced = modulus

    base_a = compute_a(dy, dx, modulus_reduced)
    for k in range(divisor):
        a = (base_a + k * modulus_reduced) % modulus
        b = (y1 - x1 * a) % modulus
        solutions.append((a, b))

    return solutions
```

```python
key_candidates = []
print("\nКандидати на ключі (a, b):")
for pair_combination in combinations(unique_pairs, 2):
    solutions = solve_linear_congruence(pair_combination)
    for solution in solutions:
        key_candidates.append(solution)
        print(f"Пари {pair_combination}: ключ {solution}")
```

Генерує всі пари з унікальних відповідностей біграм та для кожної пари біграм розв'язує систему рівнянь виду:

$$y_1 = a x_1 + b \ (mod\, m)$$

$$y_2 = a x_2 + b \ (mod\, m)$$

Де x_1, x_2 — біграми шифртексту, а y_1, y_2 — біграми мови. Повертає можливі значення a і b

4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.

```python
# Розшифрування біграми
def decrypt_bigram(bigram, key_a, key_b):
    num = bigram_to_value(bigram)
    decrypted = (num - key_b) * pow(key_a, -1, modulus) % modulus
    return value_to_bigram(decrypted)

# Чи є текст валідним
def is_valid_text(text_sample):
    consonants = 'бвгджзклмнпрстфхцчшщ'
    vowels = 'аеиоуыэюя'
    for i in range(len(text_sample) - 4):
        if (all(ch in consonants for ch in text_sample[i:i+5]) or
            all(ch in vowels for ch in text_sample[i:i+5])):
            return False
    return True
```

```python
valid_keys = []
with open("decrypted_texts.txt", "w", encoding="utf8") as output_file:
    for key in set(key_candidates):
        test_decryption = ""
        full_decryption = ""
        for bigram in bigrams_list:  # Розшифровуємо весь текст
            try:
                decrypted_bigram = decrypt_bigram(bigram, key[0], key[1])
                test_decryption += decrypted_bigram
                full_decryption += decrypted_bigram
            except:
                break
        else:
            if is_valid_text(test_decryption[:200]):  # Перевірка тільки на першій частині тексту
                valid_keys.append(key)
                output_file.write(f"Ключ: {key}, Розшифрований текст: {full_decryption}\n\n")
                print(f"\nКлюч: {key}, Розшифрований текст записаний у файл.")
```

Дешифруємо біграму за формулою:

$$x = (y - b) * a^{-1} (mod\, m)$$

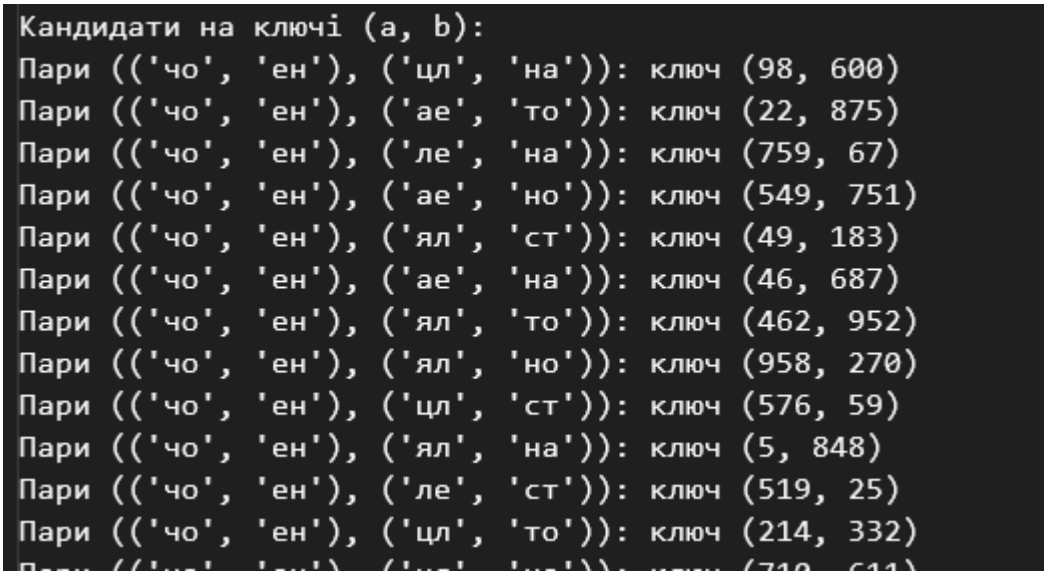
де a^{-1} — обернене до a за модулем.

Та перевіряємо, чи є розшифрований текст змістовним.

Наприклад, текст відкидається, якщо 5 поспіль букв — голосні або приголосні. А результат розшифрованого тексту записується у файл

Й в результаті ми отримаємо:

```
Найчастіші біграми шифртексту з кількістю появ:
цл: 51
ял: 49
ае: 43
ле: 42
чо: 39
```



Ключ: (200, 900), Розшифрований текст:
атызнаешьсколькоразмывэтомгодуиграливбейсболавпрошломавпозапрошл
омнистогониссегоспросилтомгубыегодвигалисьбыстробыстроявсезаписалт
ысячпятьсотшестьдесятвосемьразасколькоразячистилзубызадесятьлетжизн
ишестьтысячразарукимылпятнадцатьтысячразспалчетыресл…