

Лабораторна робота №2

З предмета "Криптографія"

Хід роботи:

Завдання:

1. Самостійно підібрати текст для шифрування (2-3 кб) та ключі довжини $r = 2, 3, 4, 5$, а також довжини 10-20 знаків. Зашифрувати обраний відкритий текст шифром Віженера з цими ключами.
2. Підрахувати індекси відповідності для відкритого тексту та всіх одержаних шифртекстів і порівняти їх значення.
3. Використовуючи наведені теоретичні відомості, розшифрувати наданий шифртекст (згідно свого варіанта №2).

Імпорт бібліотек та класів з lab_2_code.py та створення екземплярів

```
In [1]: from lab_2_code import *

processor = TextProcessor()
cipher = VigenereCipher()
analyzer = CryptoAnalyzer()
visualizer = Visualizer()
print("Все імпортовано та ініціалізовано!")
```

Все імпортовано та ініціалізовано!

=====

Пункт 1

=====

Зчитування та підготовка тексту.

щодо генерації тексту - ми вирішили використати наступний підхід - ми взяли текст з лабораторної роботи №1. Далі, ми ділимо текст на рандомні за довжиною відрізки тексту, потім також рандомним чином об'єднуємо їх в один текст. Тому виходить доволі випадковий текст, але в той же час який зберігає структуру та частоти появи літер,

подібні до оригіналу. Тобто таким чином ми створюємо більш природний (лексичний?) текст (а не просто ф-ія random).

```
In [2]: donor_text = processor.read_text(r'C:\Users\rdk\d_disk\5sem\cryptography\lab
plaintext = processor.generate_random_text(donor_text, 1250)
print(f"[+] Перші 50 символів згенерованого тексту: {plaintext[:50]}...")
print(f"[+] Довжина згенерованого тексту: {len(plaintext)} символів")
```

[+] Перші 50 символів згенерованого тексту: ениепотомучтоониспуганнообвехвмаг
азинахвквартирахв...

[+] Довжина згенерованого тексту: 1250 символів

Генеруємо ключі шифрування (ф-ія generate_keys) та виконуємо шифрування віженером (ф-ія encrypt).

ми створюємо ключі довжиною, 2-5 та 10-20 (за допомогою ф-ії generate_keys) та виводимо їх.

```
In [3]: key_lengths = [2, 3, 4, 5, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
keys = processor.generate_keys(key_lengths)
print("\n[+] Згенеровані ключі:")
for i, key in enumerate(keys):
    print(f" [-] Ключ {i+1} (довжина {len(key)}): {key}")
```

[+] Згенеровані ключі:

[-] Ключ 1 (довжина 2): чв
[-] Ключ 2 (довжина 3): есы
[-] Ключ 3 (довжина 4): агам
[-] Ключ 4 (довжина 5): ъпэрм
[-] Ключ 5 (довжина 10): уямзййючэш
[-] Ключ 6 (довжина 11): рвуфпыжезфы
[-] Ключ 7 (довжина 12): йаефчвякжхщ
[-] Ключ 8 (довжина 13): щющйцмпбдхбыч
[-] Ключ 9 (довжина 14): шхйпъодзщдэурь
[-] Ключ 10 (довжина 15): ажкзфкбюафпслк
[-] Ключ 11 (довжина 16): арнпепбеоузхсиюч
[-] Ключ 12 (довжина 17): жьмхзфутшчслчктпй
[-] Ключ 13 (довжина 18): ьфхшшяшъйлхибаэнше
[-] Ключ 14 (довжина 19): лваммхендъуаачджьюъ
[-] Ключ 15 (довжина 20): ъгцътквссчйрнлпбжсп

=====

Пункт 2

=====

Обчислюємо індекси відповідності (calculate_ioc).

індекс відповідності (ioc) в коді рахується для того, щоб визначити, наскільки символи в тексті повторюються між собою. Уявімо, що ми рахуємо кожну літеру в тексті і дивимосся,

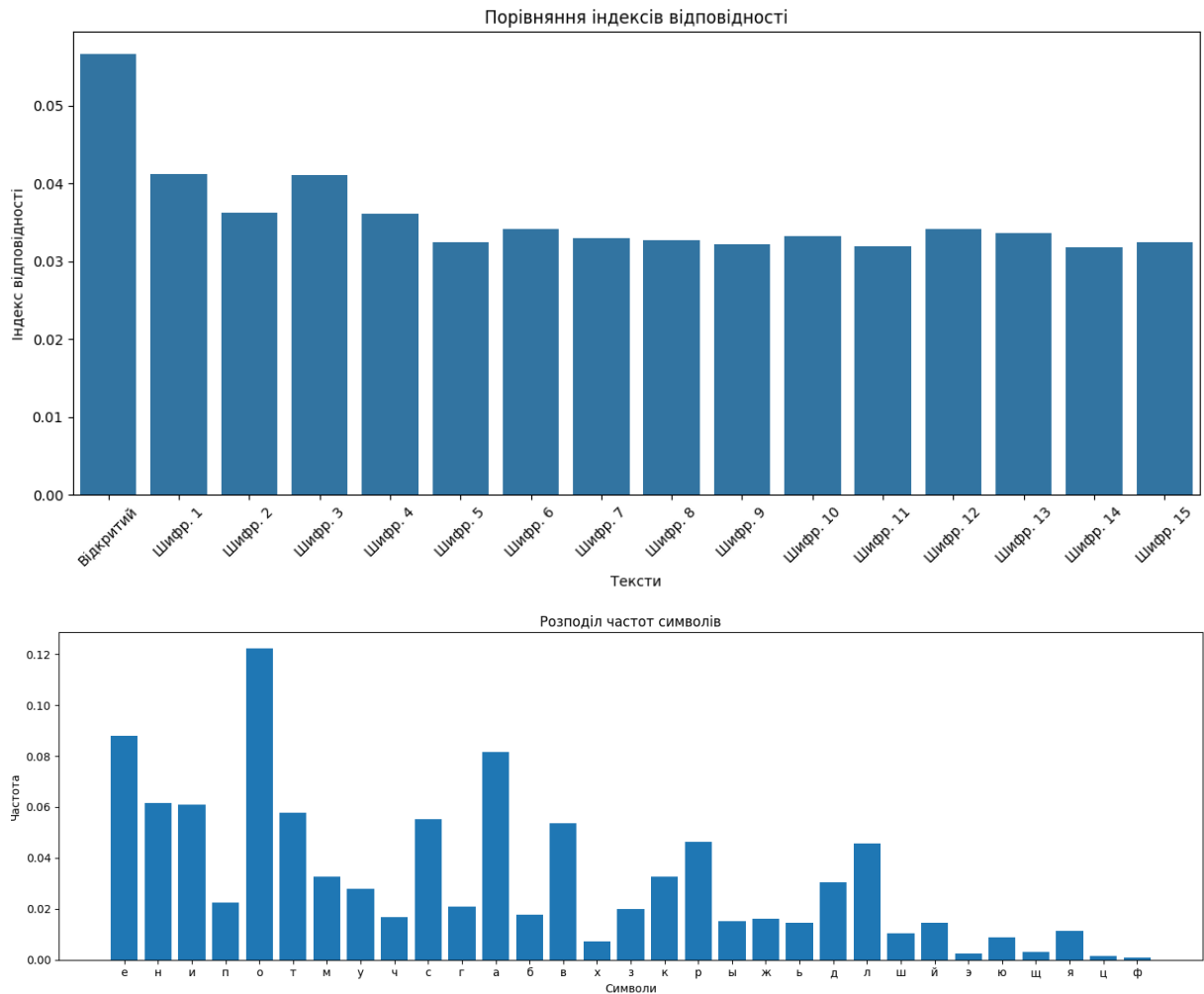
скільки разів вона зустрічається з іншою такою ж літерою. Формула бере кількість кожної літери, множить її на (к-сть - 1) і ділить на загальну кількість можливих пар символів у тексті. Якщо простіше - це показник того, яка ймовірність зустріти однакові літери в тексті. Для звичайного тексту цей показник буде вищим (бо деякі літери, як "о" чи "а", часто повторюються), а для зашифрованого - нижчим (бо там символи розподілені більш випадково).

```
In [4]: ciphertexts = [cipher.encrypt(plaintext, key) for key in keys]
ioc_plaintext = cipher.calculate_ioc(plaintext)
ioc_ciphertexts = [cipher.calculate_ioc(ciphertext) for ciphertext in ciphertexts]

print("\n[=] Індокси відповідності [=]")
print(f" [+] Відкритий текст: {ioc_plaintext:.10f}")
for i, ioc in enumerate(ioc_ciphertexts):
    print(f" [-] Шифротекст {i+1}: {ioc:.10f}")

# показуємо результати візуально
visualizer.plot_ioc_comparison(ioc_plaintext, ioc_ciphertexts)
visualizer.plot_frequency_distribution(plaintext)
```

```
[=] Індокси відповідності [=]
[+] Відкритий текст: 0.0566610088
[-] Шифротекст 1: 0.0412208167
[-] Шифротекст 2: 0.0362530024
[-] Шифротекст 3: 0.0410939952
[-] Шифротекст 4: 0.0361197758
[-] Шифротекст 5: 0.0324855084
[-] Шифротекст 6: 0.0341598078
[-] Шифротекст 7: 0.0329761409
[-] Шифротекст 8: 0.0327481185
[-] Шифротекст 9: 0.0322203363
[-] Шифротекст 10: 0.0332515612
[-] Шифротекст 11: 0.0319308247
[-] Шифротекст 12: 0.0342225781
[-] Шифротекст 13: 0.0336320256
[-] Шифротекст 14: 0.0317476381
[-] Шифротекст 15: 0.0324573259
```



Як видно з виводу та графіка №1 - індекси відповідності найкращі* для відкритого тексту - 0,057 (це відповідає рос мові), в той час як всі інші зашифровані текст мають менші значення. З другого графіка видно частоти букв для нашого відкритого тексту - вони також відповідають рос мові.

Пункт 3

Аналіз варіанту (в нашому випадку - №2) та пробуємо розшифрувати.

В цій частині ми аналізуємо зашифрований текст (за варіантом, в нашому випадку №2). Для цього ми спочатку намагаємося дізнатися довжину ключа (ф-ія `find_key_length`), після цього, за знайденою довжиною, знаходимо ключ (ф-ія `find_key`). В кінці ми дешифруємо наш текст цим ключем (ф-ія `decrypt`).

Як працює find_key_length - ми беремо наш зашифрований текст і робимо таку штуку: пробуємо порізати його на відрізки різної довжини (від 1 до 30 символів). Для кожної довжини дивимось на символи через однакові проміжки. Наприклад, якщо думаємо що довжина ключа 3, то беремо: перший символ, четвертий, сьомий... потім другий, п'ятий, восьмий... і так далі. Для кожного такого набору рахуємо індекс відповідності (ІОС). Фішка в тому, що коли ми вгадаємо правильну довжину ключа, то в цих наборах будуть символи, зашифровані одним і тим же символом ключа - і ІОС буде найбільш схожим на ІОС звичайного тексту!

Як працює знаходження самого ключа (find_key) - коли ми знайшли довжину, починається ще цікавіше. Для кожної позиції в ключі (наприклад, якщо довжина 3, то окремо для першої, другої і третьої позицій):

Беремо всі символи, які зашифровані цією позицією ключа

Рахуємо частоту кожної літери в цих символах

Пробуємо всі можливі зсуви алфавіту і дивимось, при якому зсуві частоти літер найбільше схожі на частоти в звичайному

російському тексті (які ми знаємо з теорії*)

Той зсув, який дає найкращий збіг - це і є символ ключа на цій позиції.

*до речі, значення природної мови ми взяли з минулої лабораторної роботи. Тільки трохи там підправили алфавіт, щоб він відповідав цій лабораторній та отримали результат.

```
In [5]: print("\n[===] Аналіз варіанту №2 [===]")
encrypted_variant = processor.read_text(r'C:\Users\rdk\d_disk\5sem\cryptogra
recommended_length, top_key_lengths = analyzer.find_key_length(encrypted_var

found_key = analyzer.find_key(encrypted_variant, recommended_length)
decrypted_text = cipher.decrypt(encrypted_variant, found_key)

print(f"\n[+] Знайдений ключ: {found_key}")
print(f"[+] Розшифрований текст (перші 50 символів): {decrypted_text[:50]}..
```

```
[===] Аналіз варіанту №2 [===]
```

```
[!] Топ-5 можливих довжин ключа [!]
```

```
[+] Довжина 28: ІОС = 0.055361
```

```
[+] Довжина 14: ІОС = 0.055283
```

```
[+] Довжина 7: ІОС = 0.044625
```

```
[+] Довжина 21: ІОС = 0.044470
```

```
[+] Довжина 8: ІОС = 0.036423
```

```
[✓] Рекомендована довжина ключа: 14
```

```
[+] Знайдений ключ: последнийдозор
```

```
[+] Розшифрований текст (перші 50 символів): какясмогэтосделатьспросилгесерип
очемуэтогонесмогсд...
```

Як ми бачимо з результатів - оптимальна довжина ключа - 14 (за ІОС зверху 28, але в коді є перевірка - він створює "плато" з довжин, які мають ІОС близький до макс (більше 90% від макс значення), і потім обирає найменшу довжину з цього плато. ми можемо отримати кратні довжини з дуже схожими ІОС, але нам потрібна найменша з них.) По факту, це як знайти найменший спільний дільник для числа.

Додатковий аналіз через статистику Dr.

```
In [6]: print("\n[===] Додатковий аналіз через статистику Dr [===]")
recommended_length_dr, coincidence_stats, peaks = analyzer.calculate_coincidences
visualizer.visualize_dr_stats(coincidence_stats)
```

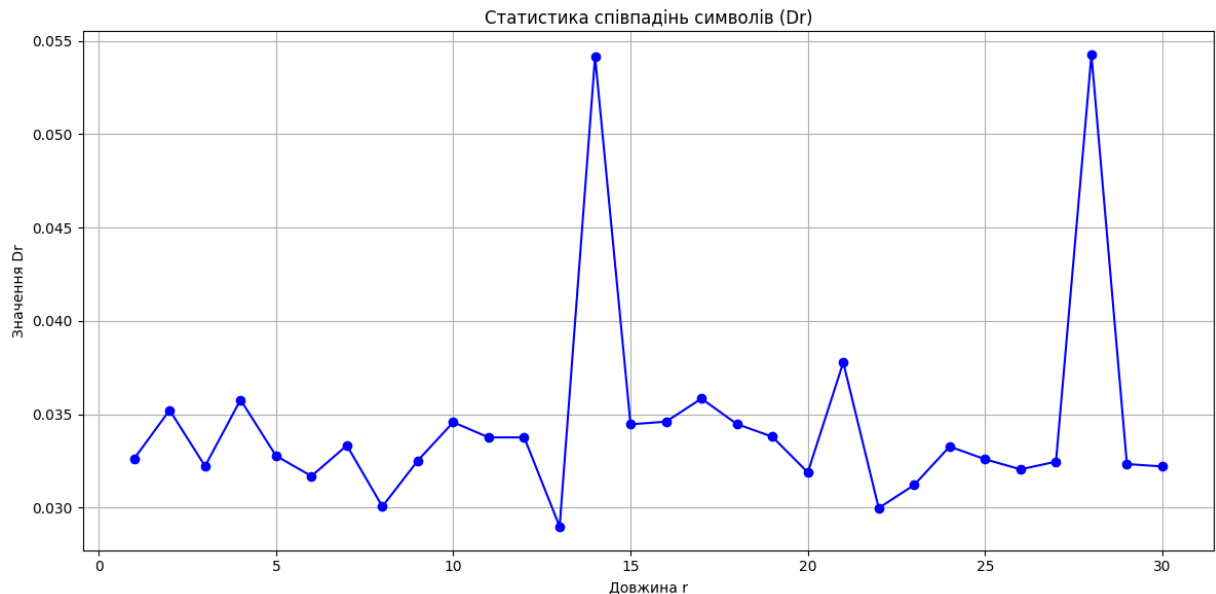
[===] Додатковий аналіз через статистику Dr [===]

[!] Аналіз статистики співпадінь Dr [!]

Топ-5 піків статистики Dr:

```
[+] Довжина 28: Dr = 0.054247
[+] Довжина 14: Dr = 0.054143
[+] Довжина 21: Dr = 0.037772
[+] Довжина 17: Dr = 0.035836
[+] Довжина 4: Dr = 0.035773
```

[✓] Рекомендована довжина ключа (за Dr): 14



```
In [7]: print("\nПорівняння результатів двох методів:")
print(f"1. Метод індексу відповідності: довжина ключа = {recommended_length}")
print(f"2. Метод статистики співпадінь: довжина ключа = {recommended_length}")
```

Порівняння результатів двох методів:

1. Метод індексу відповідності: довжина ключа = 14
2. Метод статистики співпадінь: довжина ключа = 14

Dr - це метод для знаходження довжини ключа в шифрі Віженера шляхом підрахунку однакових символів на певній відстані один від одного

Його основна ідея - якщо в тексті взяти символи на відстані, що дорівнює довжині ключа, то вони будуть зашифровані одним і тим самим символом ключа.

Наприклад, якщо ключ має довжину 5, то кожний 5-й символ шифрується однаково. Тому на відстані 5 (і кратній 5) будуть частіше зустрічатися однакові символи

Після підрахунку співпадінь ми нормалізуємо значення (ділимо на кількість порівнянь), шукаємо піки - місця де значення більше ніж у сусідів та вибираємо найбільші піки як ймовірні довжини ключа.

В результаті, ми маємо знайдену правильну довжину ключа - 14

Додатковий аналіз з іншими довжинами ключа (першим методом) (щоб точно переконатися, що ми обрали оптимальний/правильний ключ).

```
In [8]: additional_results = []
additional_lengths = [length for length, _ in top_key_lengths[:5] if length

for length in additional_lengths:
    print(f"\n{'=' * 30} Аналіз з довжиною {length} {'=' * 30}")
    key = analyzer.find_key(encrypted_variant, length)
    text = cipher.decrypt(encrypted_variant, key)
    additional_results.append((length, key, text))
    print(f"[+] Ключ: {key}")
    print(f"[+] Текст: {text[:50]}...")

===== Аналіз з довжиною 28 =====
=====
[+] Ключ: последнийдозорпоследнийдозор
[+] Текст: какясмогэтосделатьсяспросилгесерипочемуэтогонесмогсд...

===== Аналіз з довжиною 7 =====
=====
[+] Ключ: послздн
[+] Текст: какяпмьшессоилатьппрзмыогпферипмчееорхонснесммгсэ...

===== Аналіз з довжиною 21 =====
=====
[+] Ключ: мослздрпосоеднийслздн
[+] Текст: накаяпмльшеоуоитетьппркмыогпсеримочемурхонсресммгоз...

===== Аналіз з довжиною 8 =====
=====
[+] Ключ: одозозои
[+] Текст: лкнгийнгштосдомжхгимппмдлжеъжчлтечдноътнгъмфреари...
```

З виводу видно, що ключ з довжиною 28 підходить, але сенс - це по факту подвоєний ключ довжини 14) Інші ключі вже не правильно розшифровують текст.

Загальний аналіз та результати.

```
In [9]: print("\n" + "="*50)
print("Аналіз результатів:")
print("="*50)
print(f"1. Довжина відкритого тексту: {len(plaintext)} символів")
print(f"2. Кількість згенерованих ключів: {len(keys)}")
print(f"3. Довжини ключів:", ", ".join(map(str, key_lengths)))
print(f"4. Індекс відповідності відкритого тексту: {ioc_plaintext:.6f}")
print("5. Індокси відповідності шифротекстів:")
for i, ioc in enumerate(ioc_ciphertexts):
    print(f" Шифротекст {i+1}: {ioc:.6f}")
print("6. Аналіз можливих довжин ключа:")
print("-"*30)
print("Метод 1 (IOC) - топ-5:")
for length, ioc in top_key_lengths[:5]:
    print(f" Довжина {length}: IOC = {ioc:.6f}")

print("\nМетод 2 (Dr) - топ-5:")
for length, dr in peaks[:5]: # peaks отримуємо з методу calculate_coincidence
    print(f" Довжина {length}: Dr = {dr:.6f}")
print(f"7. Знайдений ключ для варіанту: {found_key}")
print(f"8. Довжина розшифрованого тексту: {len(decrypted_text)} символів")
print("="*50)
```


=====

Аналіз результатів:

=====

1. Довжина відкритого тексту: 1250 символів
2. Кількість згенерованих ключів: 15
3. Довжини ключів: 2, 3, 4, 5, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
4. Індекс відповідності відкритого тексту: 0.056661
5. Індеси відповідності шифротекстів:
 - Шифротекст 1: 0.041221
 - Шифротекст 2: 0.036253
 - Шифротекст 3: 0.041094
 - Шифротекст 4: 0.036120
 - Шифротекст 5: 0.032486
 - Шифротекст 6: 0.034160
 - Шифротекст 7: 0.032976
 - Шифротекст 8: 0.032748
 - Шифротекст 9: 0.032220
 - Шифротекст 10: 0.033252
 - Шифротекст 11: 0.031931
 - Шифротекст 12: 0.034223
 - Шифротекст 13: 0.033632
 - Шифротекст 14: 0.031748
 - Шифротекст 15: 0.032457
6. Аналіз можливих довжин ключа:

Метод 1 (IOC) – топ-5:

- Довжина 28: IOC = 0.055361
- Довжина 14: IOC = 0.055283
- Довжина 7: IOC = 0.044625
- Довжина 21: IOC = 0.044470
- Довжина 8: IOC = 0.036423

Метод 2 (Dr) – топ-5:

- Довжина 28: Dr = 0.054247
- Довжина 14: Dr = 0.054143
- Довжина 21: Dr = 0.037772
- Довжина 17: Dr = 0.035836
- Довжина 4: Dr = 0.035773

7. Знайдений ключ для варіанту: последнийдозор
 8. Довжина розшифрованого тексту: 7328 символів
- =====

Висновки

У ході лабораторної роботи досліджено принципи роботи та методи криптоаналізу шифру Віженера.

Знайдений індекс відповідності (IOC) для відкритого тексту російською мовою становить приблизно 0.056 - відповідає теоретичним очікуванням рос мови.

Зі збільшенням довжини ключа значення IOC зменшується: для ключів довжиною 2–5 символів - близько 0.040–0.045, а для ключів довжиною 10–20 символів - 0.032–0.034.

Для зашифрованого тексту варіанту №2 найвірогіднішою виявилася довжина 14 та був знайдений ключ - "последнийдозор" (+ це підтвердилося успішним розшифруванням).

Отримані результати підтверджують теоретичні відомості про вразливість шифру Віженера до частотного криптоаналізу та демонструють практичну можливість його зламу (особливо для коротких ключів (2–5 символів)) при наявності достатньої кількості шифротексту.