ФБ-21 Редько-Шпак Родислав, Серяков Владислав

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 [9]:
```
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 [10]:
```
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: ясвесьмасомнительнымипосетитсястр
ахиярostьрассказч...
[+] 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 [11]:
```
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): зе
 [-] Ключ 2 (довжина 3): квк
 [-] Ключ 3 (довжина 4): йстм
 [-] Ключ 4 (довжина 5): юъхпп
 [-] Ключ 5 (довжина 10): лрьтдбгжсг
 [-] Ключ 6 (довжина 11): фжаримыйчиз
 [-] Ключ 7 (довжина 12): лрмцичхоочщы
 [-] Ключ 8 (довжина 13): лтшлбскакйюшл
 [-] Ключ 9 (довжина 14): шбадубшонщыюйщ
 [-] Ключ 10 (довжина 15): ормпонщфдиьшънб
 [-] Ключ 11 (довжина 16): оюднорйьоцьсъбуи
 [-] Ключ 12 (довжина 17): бьебаъвколююещочщ
 [-] Ключ 13 (довжина 18): щомфцавюсфойжюкнвй
 [-] Ключ 14 (довжина 19): нжфьлгцгувнамзсеофч
 [-] Ключ 15 (довжина 20): йццоуьачумкнсшпрзхтм
```

==========

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 дивимося,

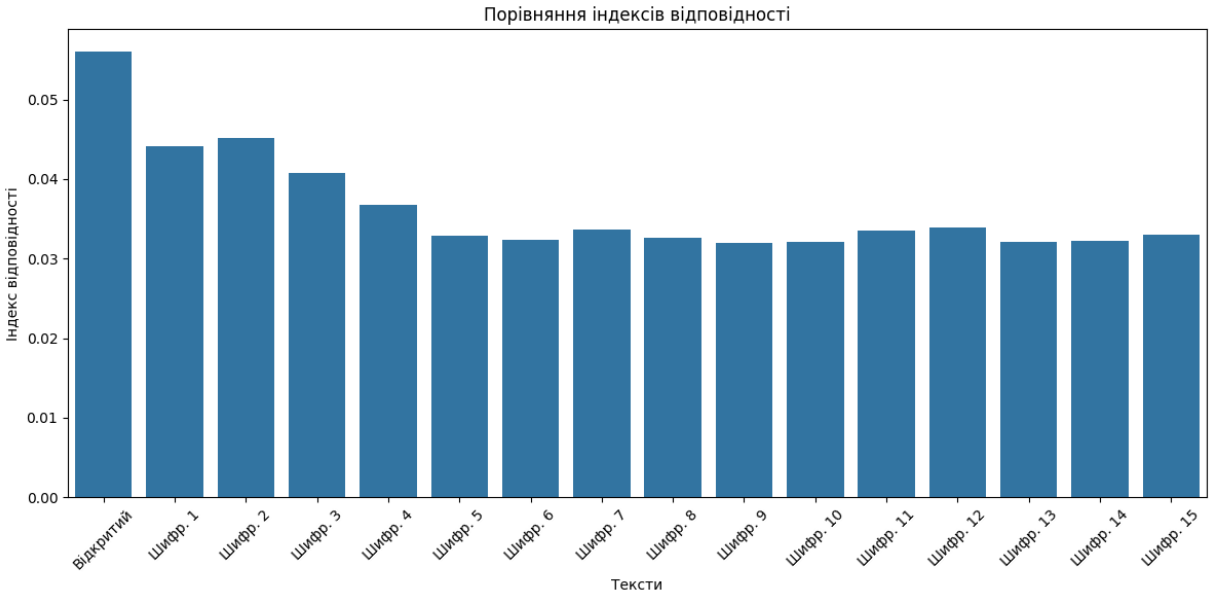
скільки разів вона зустрічається з іншою такою ж літерою. Формула бере кількість кожної літери, множить її на (к-сть - 1) і ділить на загальну кількість можливих пар символів у тексті. Якщо простіше - це показник того, яка ймовірність зустріти однакові літери в тексті. Для звичайного тексту цей показник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

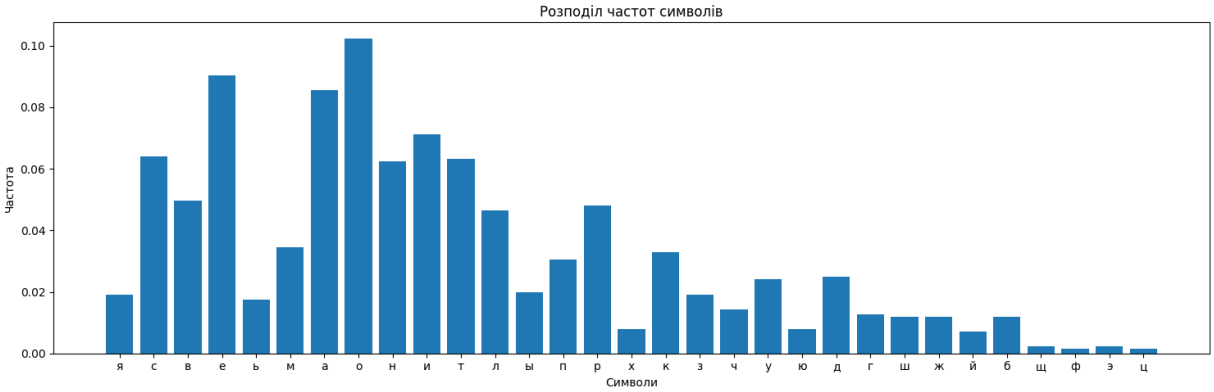
In [12]:
```
ciphertexts = [cipher.encrypt(plaintext, key) for key in keys]
ioc_plaintext = cipher.calculate_ioc(plaintext)
ioc_ciphertexts = [cipher.calculate_ioc(ciphertext) for ciphertext in cipher

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

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

```
[=] Індекси відповідності [=]
 [+] Відкритий текст: 0.0560217774
 [-] Шифротекст 1: 0.0441197758
 [-] Шифротекст 2: 0.0451279424
 [-] Шифротекст 3: 0.0407993595
 [-] Шифротекст 4: 0.0367987190
 [-] Шифротекст 5: 0.0329184948
 [-] Шифротекст 6: 0.0323407526
 [-] Шифротекст 7: 0.0337101681
 [-] Шифротекст 8: 0.0326405124
 [-] Шифротекст 9: 0.0319590072
 [-] Шифротекст 10: 0.0320717374
 [-] Шифротекст 11: 0.0335269816
 [-] Шифротекст 12: 0.0338690152
 [-] Шифротекст 13: 0.0320435548
 [-] Шифротекст 14: 0.0321844676
 [-] Шифротекст 15: 0.0330171337
```





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 (IOC). Фішка в тому, що коли ми вгадаємо правильну довжину ключа, то в цих наборах будуть символи, зашифровані одним і тим же символом ключа - і IOC буде найбільш схожим на IOC 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 [13]:
```
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: IOC = 0.055361
 [+] Довжина 14: IOC = 0.055283
 [+] Довжина 7: IOC = 0.044625
 [+] Довжина 21: IOC = 0.044470
 [+] Довжина 8: IOC = 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 IOC зверху 28, але в коді є перевірка - він створює "плато" з довжин, які мають IOC близький до макс (більше 90% від макс значення), і потім обирає найменшу довжину з цього плато. ми можемо отримати кратні довжини з дуже схожими IOC, але нам потрібна найменша з них.) По факту, це як найти найменший спільний дільник для числа.

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

In [14]:
```
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: какясмогэтосделатьспросилгесерипочемуэтогонесмогсд...

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

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

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

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 [15]:
```
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("3. Довжини ключів:", ", ".join(map(str, key_lengths)))
```

```
print(f"4. Індекс відповідності відкритого тексту: {ioc_plaintext:.6f}")
print("5. Індекси відповідності шифротекстів:")
for i, ioc in enumerate(ioc_ciphertexts):
    print(f" Шифротекст {i+1}: {ioc:.6f}")
print("6. Можливі довжини ключа для варіанту (топ-5):")
for length, ioc in top_key_lengths[:5]:
    print(f" Довжина {length}: IOC = {ioc:.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: 0.056022
5. Індекси відповідності шифротекстів:
 Шифротекст 1: 0.044120
 Шифротекст 2: 0.045128
 Шифротекст 3: 0.040799
 Шифротекст 4: 0.036799
 Шифротекст 5: 0.032918
 Шифротекст 6: 0.032341
 Шифротекст 7: 0.033710
 Шифротекст 8: 0.032641
 Шифротекст 9: 0.031959
 Шифротекст 10: 0.032072
 Шифротекст 11: 0.033527
 Шифротекст 12: 0.033869
 Шифротекст 13: 0.032044
 Шифротекст 14: 0.032184
 Шифротекст 15: 0.033017
6. Можливі довжини ключа для варіанту (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
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.