In [ ]:
```python
# Розв'язання методом Фрідмана (тест Фрідмана) для шифру Віженера
# з урахуванням великого і малого регістрів.

import string
import math
```

In [ ]:
```python
# 1. Зчитуємо вихідний шифротекст (у якому можуть бути великі та малі літери),
#    а також будь-які розділові знаки, пробіли тощо.
with open("cipher.txt", "r", encoding="utf-8") as f:
    cipher_text = f.read()

# ------------------------------------------------------------------------
# Щоб потім ПОВЕРНУТИ регістр і символи на свої місця, нам треба:
#  1) Зібрати окремо лише букви (для аналізу).
#  2) Запам'ятати:
#     - чи була літера великою / маленькою,
#     - які саме символи (і де) були неалфавітними.
#  3) Після дешифрування «суцільного» тексту відтворити вихідний формат.
# ------------------------------------------------------------------------

def extract_alpha_and_record_case(text: str):
    """
    Обходимо весь оригінальний текст:
      - Якщо це літера (A-Za-z):
          зберігаємо її (в UPPERCASE) у окремому рядку cleaned
          фіксуємо True, якщо була велика, інакше False
      - Будь-які інші символи (пробіли, пунктуація тощо)
          просто запам'ятовуємо у positions як None
    Повертає:
      (cleaned, positions)
    де:
      cleaned  - суцільний рядок з великими літерами (для аналізу)
      positions - список тих самих розмірів, що text, де:
          • True  => на цьому місці була літера (верхній регістр)
          • False => на цьому місці була літера (нижній регістр)
          • None  => неалфавітний символ
    """
    cleaned = []
    positions = []
    for ch in text:
        if ch.isalpha():
            cleaned.append(ch.upper())
            positions.append(ch.isupper())  # True, якщо велика
        else:
            positions.append(None)  # Не літера
    return "".join(cleaned), positions

def restore_format(uppercase_text: str, original_text: str, positions: list) ->
    """
    Відновлює формат (включно з регістром і неалфавітними символами)
    на основі:
      - uppercase_text: дешифрований рядок (усі літери у верхньому регістрі),
      - original_text: той, що був спочатку,
      - positions: список з інформацією про те,
                   де в original_text були літери (True/False) і де - None.
    Логіка:
      - Ітеруємося за original_text.
      - Якщо positions[i] is True -> беремо поточну літеру з uppercase_text,
```

```
            але робимо її великою.
      - Якщо positions[i] is False -> беремо поточну літеру з uppercase_text,
            але робимо її маленькою.
      - Якщо positions[i] is None -> відтворюємо оригінальний неалфавітний симво
    """
    result = []
    alpha_index = 0  # індекс, яким рухаємося по uppercase_text

    for i, ch in enumerate(original_text):
        pos = positions[i]
        if pos is None:
            # неалфавітний символ => залишаємо як було
            result.append(ch)
        else:
            # це була літера; беремо символ з uppercase_text[alpha_index]
            letter = uppercase_text[alpha_index]
            alpha_index += 1

            if pos is True:
                # була велика літера, відновимо велику
                result.append(letter.upper())
            else:
                # була мала літера, повернемо в нижній
                result.append(letter.lower())

    return "".join(result)

# Виділимо букви у верхньому регістрі + запишемо позиції
cleaned_cipher, positions = extract_alpha_and_record_case(cipher_text)
N = len(cleaned_cipher)

print("Очищений шифротекст (тільки літери A-Z) - фрагмент:")
print(cleaned_cipher[:200], "...")
print(f"Загальна кількість літер (N) = {N}")
```

```
Очищений шифротекст (тільки літери A-Z) - фрагмент:
VYCPKHOJTXZRJVAGXOZFRDMZGRSIBTACTWPLIJRDKSBVAAHPVRLSVCTTEPSRJVYGMWYKIHHPVJYXFHNVC
GPRKTGHASCYORHLVIYCLZGKEXURQRLDMVKIMPULGIMGTBKNMPACTZYAAWYZMEYCUJGDGCLSEPBRKWSAMV
OEGHAFGYGVASYKAHAFGCMLXGZWOGTMHTPXMWIZ ...
Загальна кількість літер (N) = 1621
```

In [ ]:

```
# 2. Функції для обчислення індексу збігу (IC), методу Фрідмана тощо

def compute_index_of_coincidence(text: str) -> float:
    """
    Обчислює індекс збігу (IC) для заданого тексту,
    вважаючи, що text вже містить лише A-Z.
    """
    freq = [0]*26
    for ch in text:
        freq[ord(ch) - ord('A')] += 1

    N = len(text)
    if N <= 1:
        return 0.0

    numerator = 0
    for f in freq:
        numerator += f * (f - 1)
```

```
    denominator = N * (N - 1)
    return numerator / denominator

def friedman_test(text: str) -> float:
    """
    Повертає орієнтовну довжину ключа за формулою Фрідмана.
    """
    ic = compute_index_of_coincidence(text)
    N = len(text)
    if N == 0:
        return 1.0

    # Коефіцієнти для англійської мови
    K_approx = (0.0265 * N) / ((1.0 - ic) + (0.0385 * N))
    return K_approx
```

In [ ]:

```
# 3. Тест Фрідмана: приблизна довжина ключа
ic_value = compute_index_of_coincidence(cleaned_cipher)
print(f"Індекс збігу (IC) для всього шифротексту: {ic_value:.4f}")

K_estimated = friedman_test(cleaned_cipher)
print(f"Орієнтовна довжина ключа за тестом Фрідмана: ~ {K_estimated:.2f}")

approx_len = round(K_estimated)

# Наприклад, пошукаємо у діапазоні ±3
search_range = range(max(1, approx_len - 3), approx_len + 50)
print(f"Перевірятимемо довжини ключа у діапазоні: {list(search_range)}")

def average_ic_for_key_length(text: str, key_len: int) -> float:
    """
    Розбиває text на key_len стовпчиків (кожен стовпець – символи,
    що шифруються одним і тим же зсувом), і обчислює середній IC стовпців.
    """
    columns = [''] * key_len
    for i, ch in enumerate(text):
        col_index = i % key_len
        columns[col_index] += ch

    ic_sum = 0.0
    for col in columns:
        if len(col) > 1:
            ic_sum += compute_index_of_coincidence(col)
    return ic_sum / key_len

ic_scores = {}
for kl in search_range:
    ic_col = average_ic_for_key_length(cleaned_cipher, kl)
    ic_scores[kl] = ic_col

# Сортуємо за зменшенням середнього IC:
sorted_ic = sorted(ic_scores.items(), key=lambda x: x[1], reverse=True)

print("Середній IC по стовпцях для кожної довжини ключа:")
for kl, val in sorted_ic:
    print(f"  Довжина = {kl}, середній IC = {val:.4f}")

best_guess_length = sorted_ic[0][0]
print(f"\nНайвищий середній IC отримано для довжини ключа: {best_guess_length}")
```

```
Індекс збігу (IC) для всього шифротексту: 0.0409
Орієнтовна довжина ключа за тестом Фрідмана: ~ 0.68
Перевірятимемо довжини ключа у діапазоні: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 3
3, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
Середній IC по стовпцях для кожної довжини ключа:
  Довжина = 24, середній IC = 0.0703
  Довжина = 12, середній IC = 0.0697
  Довжина = 48, середній IC = 0.0691
  Довжина = 36, середній IC = 0.0689
  Довжина = 6, середній IC = 0.0583
  Довжина = 42, середній IC = 0.0580
  Довжина = 30, середній IC = 0.0580
  Довжина = 18, середній IC = 0.0574
  Довжина = 3, середній IC = 0.0477
  Довжина = 45, середній IC = 0.0475
  Довжина = 15, середній IC = 0.0474
  Довжина = 21, середній IC = 0.0472
  Довжина = 40, середній IC = 0.0471
  Довжина = 27, середній IC = 0.0471
  Довжина = 8, середній IC = 0.0470
  Довжина = 9, середній IC = 0.0470
  Довжина = 4, середній IC = 0.0469
  Довжина = 33, середній IC = 0.0469
  Довжина = 44, середній IC = 0.0468
  Довжина = 20, середній IC = 0.0467
  Довжина = 28, середній IC = 0.0466
  Довжина = 16, середній IC = 0.0462
  Довжина = 32, середній IC = 0.0459
  Довжина = 39, середній IC = 0.0457
  Довжина = 10, середній IC = 0.0444
  Довжина = 2, середній IC = 0.0444
  Довжина = 38, середній IC = 0.0442
  Довжина = 14, середній IC = 0.0437
  Довжина = 46, середній IC = 0.0436
  Довжина = 22, середній IC = 0.0436
  Довжина = 34, середній IC = 0.0432
  Довжина = 26, середній IC = 0.0431
  Довжина = 50, середній IC = 0.0424
  Довжина = 47, середній IC = 0.0415
  Довжина = 1, середній IC = 0.0409
  Довжина = 5, середній IC = 0.0407
  Довжина = 7, середній IC = 0.0406
  Довжина = 19, середній IC = 0.0406
  Довжина = 11, середній IC = 0.0404
  Довжина = 23, середній IC = 0.0404
  Довжина = 17, середній IC = 0.0403
  Довжина = 13, середній IC = 0.0401
  Довжина = 43, середній IC = 0.0400
  Довжина = 31, середній IC = 0.0399
  Довжина = 29, середній IC = 0.0398
  Довжина = 37, середній IC = 0.0396
  Довжина = 25, середній IC = 0.0395
  Довжина = 49, середній IC = 0.0394
  Довжина = 41, середній IC = 0.0392
  Довжина = 35, середній IC = 0.0387

Найвищий середній IC отримано для довжини ключа: 24
```

In [ ]:

```python
# 4. Відновлення самого ключа через chi-squared частотний аналіз стовпчиків

english_freq = {
    'A': 0.08167, 'B': 0.01492, 'C': 0.02782, 'D': 0.04253, 'E': 0.12702,
    'F': 0.02228, 'G': 0.02015, 'H': 0.06094, 'I': 0.06966, 'J': 0.00153,
    'K': 0.00772, 'L': 0.04025, 'M': 0.02406, 'N': 0.06749, 'O': 0.07507,
    'P': 0.01929, 'Q': 0.00095, 'R': 0.05987, 'S': 0.06327, 'T': 0.09056,
    'U': 0.02758, 'V': 0.00978, 'W': 0.02360, 'X': 0.00150, 'Y': 0.01974,
    'Z': 0.00074
}

def chi_squared_statistic(text_segment: str) -> float:
    """
    Порівнюємо частотний розподіл text_segment (A-Z)
    з типовим для англійської мови (english_freq).
    """
    segment_length = len(text_segment)
    if segment_length == 0:
        return 1e9

    freq_count = [0]*26
    for ch in text_segment:
        freq_count[ord(ch) - ord('A')] += 1

    chi2 = 0.0
    for i in range(26):
        letter = chr(ord('A') + i)
        observed = freq_count[i]
        expected = english_freq[letter] * segment_length
        if expected > 0:
            chi2 += (observed - expected)**2 / expected
    return chi2

def decrypt_caesar(cipher_segment: str, shift: int) -> str:
    """
    Дешифруємо підрядок шифром Цезаря зі зсувом shift (0..25).
    Припускаємо, що cipher_segment містить тільки A-Z.
    """
    res = []
    for ch in cipher_segment:
        old_pos = ord(ch) - ord('A')
        new_pos = (old_pos - shift) % 26
        res.append(chr(new_pos + ord('A')))
    return "".join(res)

def guess_shift_for_column(col_text: str) -> int:
    """
    Знаходимо shift (0..25), який дає мінімальний chi-squared
    (найближчий до англ. частот).
    """
    best_shift = 0
    min_chi2 = 1e9
    for s in range(26):
        candidate = decrypt_caesar(col_text, s)
        chi2_val = chi_squared_statistic(candidate)
        if chi2_val < min_chi2:
            min_chi2 = chi2_val
            best_shift = s
    return best_shift
```

```
def guess_key_for_length(text: str, key_len: int) -> str:
    """
    Для заданої довжини ключа:
      1) Розбиваємо на key_len стовпців;
      2) Для кожного стовпця знаходимо зсув (shift) -> літера ключа;
      3) Складаємо ключ (A..Z).
    """
    columns = [''] * key_len
    for i, ch in enumerate(text):
        col_index = i % key_len
        columns[col_index] += ch

    key_chars = []
    for col in columns:
        best_s = guess_shift_for_column(col)
        key_letter = chr(ord('A') + best_s)
        key_chars.append(key_letter)
    return "".join(key_chars)
```

In [ ]:
```
# 5. Визначаємо ключ за знайденою довжиною
best_key_length = best_guess_length
print(f"\nСпробуємо побудувати ключ для k={best_key_length}:")

guessed_key = guess_key_for_length(cleaned_cipher, best_key_length)
print(f"Кандидат на ключ: {guessed_key}")
```

```
Спробуємо побудувати ключ для k=24:
Кандидат на ключ: CRYPTOGRAPHYCRYPTOGRAPHY
```

In [ ]:
```
# 5.1. Мануальна корекція ключа (якщо потрібно)
best_key_length = 12 # вибираємо кращий варіант
chosen_key = guess_key_for_length(cleaned_cipher, best_key_length)
print(f"Обраний ключ: {chosen_key}")
```

```
Обраний ключ: CRYPTOGRAPHY
```

In [ ]:
```
# 6. Дешифруємо увесь очищений текст, а потім відновлюємо формат
def decrypt_vigenere(cipher: str, key: str) -> str:
    """
    Дешифруємо (лише літери A-Z у cipher).
    Для спрощення – cipher і key мають бути у верхньому регістрі.
    """
    decrypted = []
    key_index = 0
    key_len = len(key)

    for ch in cipher:
        shift = ord(key[key_index % key_len]) - ord('A')
        old_pos = ord(ch) - ord('A')
        new_pos = (old_pos - shift) % 26
        decrypted.append(chr(new_pos + ord('A')))
        key_index += 1

    return "".join(decrypted)

decrypted_upper = decrypt_vigenere(cleaned_cipher, chosen_key)

# Тепер повертаємо регістр (та інші символи) з оригінального cipher_text
restored_decrypted = restore_format(decrypted_upper, cipher_text, positions)
```

```
print("\n=== Фрагмент дешифрованого тексту з відновленим форматом ===")
print(restored_decrypted[:500], "...")
```

```
=== Фрагмент дешифрованого тексту з відновленим форматом ===
The artist is the creator of beautiful things. To reveal art and conceal the arti
st is art's aim. The critic is he who can translate into another manner or a new
material his impression of beautiful things. The highest, as the lowest, form of
criticism is a mode of autobiography. Those who find ugly meanings in beautiful t
hings are corrupt without being charming. This is a fault. Those who find beautif
ul meanings in beautiful things are the cultivated. For these there is hope. They
are the elect ...
```