КОМП’ЮТЕРНИЙ ПРАКТИКУМ №3

Гоголєва Поліна ФБ-12

Варіант 5

Криптоаналіз афінної біграмної підстановки

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

Хід 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.

```python
def gcd(b, n):
    x0, x1 = 1, 0
    while n:
        q, b, n = b // n, n, b % n
        x0, x1 = x1, x0 - q * x1

    return b, x0

def modinv(a, m):
    # Знаходження оберненого елементу за модулем m
    d, x = gcd(a, m)
    if d == 1:
        return d, x % m
    return d, None

def solve_linear_congruence(a, b, m):
    d, a_inv = modinv(a, m)

    if a_inv:
        return [(a_inv * b) % m]

    if not b % d:
        a, b, m = a / d, b / d, m / d
        x0 = (b * modinv(a, m)[1]) % m
        return [int(x0 + (i - 1) * m) for i in range(1, d + 1)]
```

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

```
def count_bigrams(file_path):
    bigram_with_overlap_counts = {}
    bigram_without_overlap_counts = {}

    with open(file_path, 'r', encoding='windows-1251') as file:
        text = file.read()

    cleaned_text = text.lower()

    # Підрахунок частоти біграм
    for i in range(len(cleaned_text) - 1):
        # Підрахунок частоти біграм з перетином
        bigram_with_overlap = cleaned_text[i:i + 2]
        if bigram_with_overlap in bigram_with_overlap_counts:
            bigram_with_overlap_counts[bigram_with_overlap] += 1
        else:
            bigram_with_overlap_counts[bigram_with_overlap] = 1

        if i % 2 == 0:
            # Підрахунок частоти біграм без перетину
            bigram_without_overlap = cleaned_text[i:i + 2]
            if bigram_without_overlap in bigram_without_overlap_counts:
                bigram_without_overlap_counts[bigram_without_overlap] += 1
            else:
                bigram_without_overlap_counts[bigram_without_overlap] = 1

    # Сортування біграм за частотою
    sorted_bigram_with_overlap = sorted(bigram_with_overlap_counts.items(),
key=lambda x: x[1], reverse=True)
    sorted_bigram_without_overlap =
sorted(bigram_without_overlap_counts.items(), key=lambda x: x[1],
reverse=True)

    # Виведення п'ятьох найчастіших біграм
    print("П'ять найчастіших біграм з перетином:")
    for i in range(5):
        print(f"{sorted_bigram_with_overlap[i][0]}:
{sorted_bigram_with_overlap[i][1]} разів")

    print("\nП'ять найчастіших біграм без перетину:")
    for i in range(5):
        print(f"{sorted_bigram_without_overlap[i][0]}:
{sorted_bigram_without_overlap[i][1]} разів")

# Шлях до файлу
```

```
file_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-
24\tasks\cp3\variants\05.txt"
count_bigrams(file_path)
```

```
П'ять найчастіших біграм з перетином:
вн: 56 разів
тн: 51 разів
дк: 50 разів
ун: 50 разів
хщ: 48 разів

П'ять найчастіших біграм без перетину:
вн: 33 разів
тн: 30 разів
дк: 29 разів
ун: 26 разів
зт: 23 разів
```

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).

Ну тоді нам треба дізнатися п’ять найчастіших біграм російського тексту, використаєм той же код із першої лаби і перевіримо це на тому ж великому тексті російською. Отримали значення:

```
П'ять найчастіших біграм з перетином в російській мові:
го: 47191 разів
во: 39491 разів
то: 39441 разів
на: 38361 разів
по: 38049 разів

П'ять найчастіших біграм без перетину в російській мові:
го: 23609 разів
во: 19809 разів
то: 19522 разів
на: 19183 разів
по: 19021 разів
```

(потім я прибрала частину для розрахунку біграм з перетином, бо поняла що це not the case)

Хоча теоретично найчастішими біграмами російської мови є СТ, НО, ТО, НА, ЕН

П’ятьма найчастішими біграмами російської мови (в порядку спадання частот) є біграми «ст», «но», «то», «на», «ен». Перевірте ці відомості за допомогою програми

Можливо, наш текст був недостатньо довгим, щоб отримати більш точні значення, але в цілому вони +- співпадають, тому використовуватимемо «офіційні» теоретичні дані, бо їм я наче трохи більше довіряю :)

Тобто за даними частотності біграм виходить, що біграми вн тн дк ун зт у нашому шифртексті відповідають біграмам ст но то на ен у відповідному порядку по спаданню частотності. Це дає нам можливість розрахувати ймовірні пари ключів для афінного шифру. Для цього використаємо формули вказані у методичці:

$$\begin{cases} Y^{*} \equiv aX^{*} + b \pmod{m^2} \\ Y^{**} \equiv aX^{**} + b \pmod{m^2} \end{cases}, \qquad Y^{*} - Y^{**} \equiv a(X^{*} - X^{**}) \pmod{m^2}. \qquad b = (Y^{*} - aX^{*}) \bmod m^2.$$

Код для розрахунку пари ключів виглядатиме так

```
alphabet = 'абвгдежзийклмнопрстуфхцчшщьыэюя'
theoretical_bigrams = ('ст', 'но', 'то', 'на', 'ен')

def count_a(x1, x2, y1, y2):
    x1 = alphabet.index(x1[0]) * len(alphabet) + alphabet.index(x1[1])
    x2 = alphabet.index(x2[0]) * len(alphabet) + alphabet.index(x2[1])

    y1 = alphabet.index(y1[0]) * len(alphabet) + alphabet.index(y1[1])
    y2 = alphabet.index(y2[0]) * len(alphabet) + alphabet.index(y2[1])

    results = solve_linear_congruence(y1 - y2, x1 - x2, len(alphabet) ** 2)

    if results is not None:
        return [x for x in [modinv(i, len(alphabet) ** 2)[1] for i in
results] if x is not None]


def count_b(x1, y1, a):
    x1 = alphabet.index(x1[0]) * len(alphabet) + alphabet.index(x1[1])
    y1 = alphabet.index(y1[0]) * len(alphabet) + alphabet.index(y1[1])

    return (y1 - a * x1) % len(alphabet) ** 2
```

Але це загальні формули, для знаходження всіх можливих кандидатів пар ключів треба перевірити всі можливі перестановки (співставлення) біграм відкритого тексту і шифртексту, бо далеко не обов’язково найчастіша біграма російської мови буде переходити у найчастішу біграму шифртексту.

Для розрахунку всіх можливих пар ключів напишемо таку функцію, використовуючи ф-ю permutations з itertools, щоб не прописувати алгоритм знаходження всіх перестановок вручну.

```
def all_possible_keys(string, key_size=5):
    f_list = count_bigrams(string)[:key_size]
    possible_keys = []

    for i in permutations(theoretical_bigrams, 2):
        for j in range(len(f_list) - 1):
            key_1 = count_a(i[0], i[1], f_list[j][0], f_list[j + 1][0])

            if key_1 is None:
                continue

            for solution in key_1:
                key = solution, count_b(i[0], f_list[j][0], solution)
                possible_keys.append(key)

    return possible_keys

file_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-
24\tasks\cp3\variants\05.txt"
keys = all_possible_keys(file_path)
for key in keys:
    print('Possible Key:', key)
```

Отримали довжелезний список всіх можливих ключів

```
Possible Key: (251, 653)
Possible Key: (933, 715)
Possible Key: (654, 777)
Possible Key: (375, 839)
Possible Key: (96, 901)
Possible Key: (122, 940)
Possible Key: (219, 757)
Possible Key: (318, 335)
Possible Key: (189, 95)
Possible Key: (741, 83)
Possible Key: (918, 212)
Possible Key: (315, 478)
Possible Key: (274, 227)
Possible Key: (569, 13)
Possible Key: (312, 726)
Possible Key: (308, 940)
Possible Key: (829, 943)
Possible Key: (839, 726)
Possible Key: (742, 940)
Possible Key: (643, 943)
Possible Key: (832, 664)
Possible Key: (501, 41)
Possible Key: (609, 230)
Possible Key: (111, 182)
Possible Key: (664, 58)
Possible Key: (951, 360)
Possible Key: (307, 889)
Possible Key: (685, 374)
Possible Key: (942, 911)
Possible Key: (772, 610)
Possible Key: (220, 653)
Possible Key: (43, 105)
Possible Key: (129, 41)
Possible Key: (460, 695)
Possible Key: (352, 87)
```

Але ж вручну ми не будемо їх всіх перевіряти, правда?)

Маючи ключі дуже легко розшифрувати афінний шифр за допомогою однієї функції.

```
def decrypt(string, key):
    new_str = ''
```

```
    for i in range(0, len(string), 2):
        y = alphabet.index(string[i]) * len(alphabet) +
alphabet.index(string[i + 1])
        x = (modinv(key[0], len(alphabet) ** 2)[1] * (y - key[1])) %
len(alphabet) ** 2
        new_str += alphabet[x // len(alphabet)] + alphabet[x % len(alphabet)]
```

Але нам треба, щоб текст був змістовний. Тут ми плавно переходимо до наступного завдання.

4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.
Я вирішила використовувати ентропійний фактор порівняння (бо він мені здався найлегшим not gonna lie)
Як сказав великий інтернет:

Энтропия одной буквы русского языка равна примерно $E_1 \approx 4{,}35$ бит.

Що навіть сходиться з нашими практично знайденими значеннями при виконанні першої лаби

```
H1 у тексті з пробілами: 4.345653112220377
```

Тож використовуватимемо це значення як істинне.

Думала, які саме параметри порівняння задати для ентропії, вирішила спробувати, щоб ентропія розшифрованого тексту була у межах $4.2<H<4.4$, подивимся, що вийде. Для цього створимо кілька функцій. Перша буде стирена з першої лаби і буде розраховувати ентропію однієї літери, а друга буде приймати на ввід текст, використовувати попередню функцію і порівнювати значення ентропії з дозволеними межами.

```
def find_entropy(decrypted_text):
    entropy_letter = 0.0
    total_letters = len(decrypted_text)

    # Підрахунок ентропії для літер
    sorted_letter_counts = {}
    for letter in decrypted_text:
        if letter in sorted_letter_counts:
            sorted_letter_counts[letter] += 1
        else:
            sorted_letter_counts[letter] = 1

    for count in sorted_letter_counts.values():
        probability = count / total_letters
```

```
        entropy_letter -= probability * math.log2(probability)

    return entropy_letter

def check_the_text(decrypted_text):
    if find_entropy(decrypted_text)>4.2 and find_entropy(decrypted_text)<4.4:
        return print(decrypted_text)
```

Well, щоб перевірити, що це спрацює напишемо ще одну функцію, котра одразу і дешифруватиме і перевірятиме і виводитиме текст.

```
def decrypt_and_check(file_path):
    try:
        with open(file_path, 'r', encoding='windows-1251') as file:
            text = file.read()
    except FileNotFoundError:
        print(f"File '{file_path}' not found.")
        return

    cleaned_text = text.lower()

    keys = all_possible_keys(cleaned_text)

    for key in keys:
        decrypted_text = decrypt(cleaned_text, key)

        if check_the_text(decrypted_text):
            print(f"Decrypted Text with Key {key}:")
            print(decrypted_text)
```

і тут у мене все поламалось)))))))

```
      f_list = count_bigrams(string)[:key_size]
               ^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp3\gogoleva_fb-12_cp3\lab3.py", line 39, in count_bigrams
    with open(file_path, 'r', encoding='windows-1251') as file:
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'кеюибщаефдфмдкдкролрццисвнуншвйняэшскевдтнюдаобсюсыэиихзтмдьлыохуу
шпддкясввцгтнавпьгнуьввйнлхиьерддыцрихэкьзцэижцьехщмсэкжлрибуждэм\пхимьпьявсттнзцюсфспьуэйпдкнхркхульацкчашяьнсибжяксэкцц
```

Тому new commit coming soon when i`ll find an energy to deal with it.

апдейт! Я знайшла помилку!

Проблема полягала у тому, що функція all_possible_keys очікує шлях до файлу як аргумент, а замість цього я передавала рядок очищеного тексту.

Потім у мене виникла ще одна помилка

```
  File "c:\Users\Polya\Desktop\KPI\crypto\crypto-23-24\cp3\gogoleva_fb-12_cp3\test.py", line 93, in decrypt
    y = alphabet.index(string[i]) * len(alphabet) + alphabet.index(string[i + 1])
        ^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: substring not found
```

Але вона швидко вирішилась додаванням error handling у функцію (пропуск елементів що не є частиною алфавіту). Також трошки змінила функцію all_possible_keys, щоб прибратии можливі дублікати пар ключей котрі ми перебираємо.

Ну і з усіма пофікшеними маленькими моментами фінальний код став виглядати так:

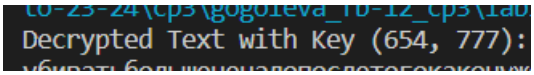
```
from itertools import permutations
import math


alphabet = 'абвгдежзийклмнопрстуфхцчшщьыэюя'
theoretical_bigrams = ('ст', 'но', 'то', 'на', 'ен')


def gcd(b, n):
    x0, x1 = 1, 0
    while n:
        q, b, n = b // n, n, b % n
        x0, x1 = x1, x0 - q * x1

    return b, x0

def modinv(a, m):

    d, x = gcd(a, m)
    if d == 1:
        return d, x % m
    return d, None

def solve_linear_congruence(a, b, m):
    d, a_inv = modinv(a, m)

    if a_inv:
        return [(a_inv * b) % m]

    if not b % d:
        a, b, m = a / d, b / d, m / d
        x0 = (b * modinv(a, m)[1]) % m
        return [int(x0 + (i - 1) * m) for i in range(1, d + 1)]


def count_bigrams_from_text(text):
    bigram_without_overlap_counts = {}


    for i in range(len(text) - 1):
        bigram_without_overlap = text[i:i + 2]
        if bigram_without_overlap in bigram_without_overlap_counts:
            bigram_without_overlap_counts[bigram_without_overlap] += 1
```

```
        else:
            bigram_without_overlap_counts[bigram_without_overlap] = 1


    sorted_bigram_without_overlap =
sorted(bigram_without_overlap_counts.items(), key=lambda x: x[1],
reverse=True)

    return sorted_bigram_without_overlap

def count_a(x1, x2, y1, y2):
    x1 = alphabet.index(x1[0]) * len(alphabet) + alphabet.index(x1[1])
    x2 = alphabet.index(x2[0]) * len(alphabet) + alphabet.index(x2[1])

    y1 = alphabet.index(y1[0]) * len(alphabet) + alphabet.index(y1[1])
    y2 = alphabet.index(y2[0]) * len(alphabet) + alphabet.index(y2[1])

    results = solve_linear_congruence(y1 - y2, x1 - x2, len(alphabet) ** 2)

    if results is not None:
        return [x for x in [modinv(i, len(alphabet) ** 2)[1] for i in
results] if x is not None]


def count_b(x1, y1, a):
    x1 = alphabet.index(x1[0]) * len(alphabet) + alphabet.index(x1[1])
    y1 = alphabet.index(y1[0]) * len(alphabet) + alphabet.index(y1[1])

    return (y1 - a * x1) % len(alphabet) ** 2


def all_possible_keys(cleaned_text, key_size=5):
    f_list = count_bigrams_from_text(cleaned_text)[:key_size]
    possible_keys = set()

    for i in permutations(theoretical_bigrams, 2):
        for j in range(len(f_list) - 1):
            key_1 = count_a(i[0], i[1], f_list[j][0], f_list[j + 1][0])

            if key_1 is None:
                continue

            for solution in key_1:
                key = solution, count_b(i[0], f_list[j][0], solution)
                possible_keys.add(key)

    return list(possible_keys)

def decrypt(string, key):
    new_str = ''
```

```
    for i in range(0, len(string), 2):
        try:
            y = alphabet.index(string[i]) * len(alphabet) +
alphabet.index(string[i + 1])
            x = (modinv(key[0], len(alphabet) ** 2)[1] * (y - key[1])) %
len(alphabet) ** 2
            new_str += alphabet[x // len(alphabet)] + alphabet[x %
len(alphabet)]
        except ValueError:
            print(f"Non-alphabet character found at position {i}:
'{string[i]}' or '{string[i + 1]}'")
            return None

    return new_str

def find_entropy(decrypted_text):
    entropy_letter = 0.0
    total_letters = len(decrypted_text)

    # Підрахунок ентропії для літер
    sorted_letter_counts = {}
    for letter in decrypted_text:
        if letter in sorted_letter_counts:
            sorted_letter_counts[letter] += 1
        else:
            sorted_letter_counts[letter] = 1

    for count in sorted_letter_counts.values():
        probability = count / total_letters
        entropy_letter -= probability * math.log2(probability)

    return entropy_letter

def check_the_text(decrypted_text):
    if find_entropy(decrypted_text)>4.2 and find_entropy(decrypted_text)<4.4:
        return decrypted_text

def decrypt_and_check(file_path):
    with open(file_path, 'r', encoding='windows-1251') as file:
        text = file.read()

    cleaned_text = ''.join([char for char in text.lower() if char in
alphabet])

    keys = all_possible_keys(cleaned_text)
    for key in keys:
        decrypted_text = decrypt(cleaned_text, key)
        if check_the_text(decrypted_text):
            print(f"Decrypted Text with Key {key}:")
            print(decrypted_text)
```

```
file_path = r"C:\Users\Polya\Desktop\KPI\crypto\crypto-23-
24\tasks\cp3\variants\05.txt"
decrypt_and_check(file_path)
```

І мені пощастило і виявилось, що у мене існує тільки одна пара ключів, при розшифровці якою текст підходить під range значень ентропії мови. І це пара ключів:



І розшифрований текст і справді виявився змістовним.

убиватьбольшененадопослетогокаконужеубилноследуетемубытьблагодарныминачепришлосьбыубиватьсамомуэтонеоднолишьдоброесо
страданиеэтоотождествлениенаоснованииодинаковыхимпульсовкубийствусобственноговорялишьвминимальнойстепенисмещенныйнар
циссизмэтическаяценностьэтойдоброты этимнеоспариваетсяможетбытьэтовообщемеханизмнашегодоброгоучастияпоотношениюкдруго
мучеловекуособенноясноспроступающийвчрезвычайномслучаеобремененногосознаниясвоейвиныписателянетсомнениячтоэтасимпатияп
опричинеотождествлениярешительноопределилавыборматериаладостоевскогоносначалаонизэгоистическихпобужденийвыводилобыкно
венногопреступникаполитическогоирелигиозногопреждечемкконцусвоейжизнивернутьсякпервопреступникукотцеубийцеисделатьвегол
ицесвоепоэтическоепризнаниеопубликованиеегопосмертногонаследияидневниковегоженыяркоосветилоодинэпизодегожизнитовремяко
гдадостоевскийвгерманиибылобуреваемигорнойстрастьюдостоевскийзарулеткойявныйприпадокпатологическойстрастикоторыйнеподд
аетсяиной оценкенискакойстороны небылонедостаткавоправданияхэтогостранногоинедостойногоповедениячувствовиныкакэтонередкоб
ываетуневротиковнашлоконкретнуюзаменувобремененностидолгамиидостоевскийможготговариватьсятемчтоонпривыигрышеполучилбы
возможностьвернутьсяврассию избежавзаключенияв тюрьмукредитораминоэтобылтолькопредлогдостоевскийбылдостаточнопроницател
енчтобыэтопонятьидостаточночестенчтобывэтомпризнатьсяонзналчтоглавнымбылаиграсамапосебевсеподробностиегообусловленногоп
ервичнымипозывамибезрассудногоповеденияслужаттомудоказательствомищекоечемуиномуоннеуспокаивалсяпоканетерялвсегоиграб
ыладлянеготакжесредствомсамонаказаниянесчетноеколичествораздавалонмолодойженесловоиличестноесловобольшенеиграть илинеигр
атьвэтотденьионнарушалэтословокаконарассказываетпочтивсегдаеслионсвоимипроигрышамидоводилсебяиеедокрайнебедственногопо
ложенияэтослужилодлянегоещеоднимпатологическимудовлетворениемонмогпереднеюпоноситьиунижатьсебяпроситьеепрезиратьегора
скаиватьсявтомчтоонавышлазамужзанегостарогогрешникаипослевсейэтойразгрузкисовестинаследующийденьиграначиналасьсноваимо
лодаяженапривыклакэтомуциклутаккказаметилачтотоотчеговдействительноститолькоиможнобылоожидатьспасенияписательствониког
данепродвигалосьвпередлучшечемпослепотеривсегоизакладыванияпоследнегоимуществавсвязивсегоэтогоонаконечнонепонималакогдае
гочувствовиныбылоудовлетворенонаказаниямиккоторымонсамсебяприговорилтогдаисчезалазатрудненностьвработетогдаонпозволялсе
беседелатьнесколькошаговнапутикуспехурассматриваярассказболееемолодогописателянетрудноугадатькакиедавнопозабытыедетскиепер
еживаниянаходятвыявленияивигорнойстрастиустефанацвейгапосвятившегомеждупрочимдостоевскомуодинизсвоихочерковтримастеравс
борникесмятениечувствестьновелладвадцатьчетыречасавжизниженщиныэтотмаленькийшедеврпоказываеткакбудтолишьтокакимбезотв
етственнымсуществомявляетсяженщинаинакакиеудивительныедлянеесамойзакононарушенияеетолкаетнеожиданноежизненноевпечатл
ениеноновеллаэтаеслиподвергнутьеепсихоаналитическомутолкованиюговоритоднакобезтакойоправдывающейтенденциигораздобольш
епоказываетсовсеминоеобщечеловеческоеилискорееобщемужскоеитакоетолкованиестольявноподсказаночтонетвозможностиегонедопу
ститьдлясущностихудожественноготворчестваxарактерночтописательскоторымменясвязываютдружескиеотношенияэтоответнамоирасспр
осыутверждалчтоупомянутоетолкованиеемучуждоивовсеневходиловегонамерениянесмотряначточтоврассказвплетенынекоторыедеталик
акбырассчитанныенаточтобыуказыватьнатайныйследвэтойновеллевеликосветскаяпожилаядамаповеряетписателюотомчтоейпришлосьп
ережитьболеедвадцатилеттомуназадранооводовевшаяматьдвухсыновейкоторыевнейболеененуждалисьотказавшаясяоткакихбытонибыло
надежднасороквторомгодужизнионапопадаетвовремяодногоизсвоихбесцельныхпутешествийвигорныйзалмонакскогоказиногдесредивсе
хдиковинеевниманиеприковываютдверукикоторыеспотрясающейнепосредственностьюисилойотражаютвсепереживаемыенесчастныминг
рокомчувстварукиэтирукикрасивогоюноши писателькакбыбезовсякогоумысладелаетегоровесникомстаршегосынанаблюдающейзаигрой
женщиныпотерявшеговсеивглубочайшемотчаянииипокидающегозалчтобывпаркепоконч итьсосвоеюбезнадежнойжизньюнеизяснимаясим
патиязаставляетженщинуследоватьзаюношейвпредпринятьвседляегоспасенияонпринимаетеезаоднуизмногочисленныхв томгороденавяз
чивыхженщинихочетотнееотделатьсяноонанепокидаетегоивынужденавконцеконцоввсилусложившихсяобстоятельствостатьсявегономе
реотеляиразделитьегопостельпослеэтойимпровизированнойлюбовнойночионавелитказалосьбыуспокоившемусяюношедатьейторжестве
нноеобещаниечтоонникогдабольшенебудетиграть снабжаетегоденьгаминаобратныйпутьисосвоейстороныдаетобещаниевстретитьсяним
передуходомпоезданавокзаленозатемвнейпробуждаетсябольшаянежностькюношеонаготовапожертвоватьвсемчтобытолькосохранитьего
длясебяионарешаетотправитьсяснимвместевпутешествиевместотогочтобыснимпроститьсявсяческиепомехизадерживаютееионаопаздыв
аетнапоездвтоскепоисчезнувшемуюношеонасноваприходитвигорныйдомисвозмущениемобнаруживаеттамтежерукинакануневозбудивш
иевнейтакуюгорячуюсимпатиюнарушительдолгавернулсякигреонанапоминаетемуобегообещанииноодержимыйстрастьюонбранитсорва
вшуюегоигрувелитейубиратьсявонишвыряетденьгикоторымионахотелаеговыкупитьопозореннаяонапокидаетгородавпоследствииузнает
чтоейнеудалосьспастиегоотсамоубийстваэтаблестящеибезпробеловвмотивировкенаписаннаяновеллаимеетконечноправонасуществован
иекактаковаяинеможетнепроизвестиначитателябольшоговпечатленияоднакопсихоанализучитчтоонавозниклана основеумопострояемого
вожделенияпериодаполовогосозреванияокаковомвожделениинекоторыевспоминаютсовершенносознательносогласноумопострояемому
вожделениюматьдолжнасамаввестиюношувполовуюжизньдляспасенияегоотзаслуживающегоопасениявредаонанизмастольчастыесубли
мирующиехудожественныепроизведенияв ытекаютизтогожепервоисточникапороконанизмазамещаетсяпорокомигорнойстрастиударение
поставленноенастрастнуюдеятельностьрукпредательскисвидетельствуетобэтомотводеэнергиидействительноигорнаяодержимостьявляет
сяэквивалентомстаройпотребностивонанизмениоднимсловомкромесловаигранельзяназвать

Висновки

В цілому лабораторна знову ґрунтувалась на частотному аналізі і перебиранні перестановок, а значення ключів розраховувались просто за формулами, що в цілому було дуже схоже на попередню роботу, тому не скажу, що було важко чи багато нового. А от що реально зацікавило це автоматизація визначення змістовності тексту, бо це якраз було моїм основним питанням при захисті попередньої роботи – невже треба буде постійно робити це вручну? А якщо там буде тисяча варіантів – читати всі??? Ну от і відповідь знайшлась і, до речі, це виявилось набагато простіше, ніж я думала.