МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені Ігоря Сікорського»

«ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ»

КРИПТОГРАФІЯ

Комп'ютерний практикум №2

Виконали

студенти 3-го курсу групи ФБ-22

Лаптєв Д. М. та Проскурня А. С.

Бригада №5

Перевірив/-ла: _____ _____

Київ - 2024

Криптоаналіз шифру Віженера

Мета роботи: Засвоєння методів частотного криптоаналізу. Здобуття навичок роботи та аналізу потокових шифрів гамування адитивного типу на прикладі шифру Віженера.

Порядок виконання роботи:

0. Уважно прочитати методичні вказівки до виконання комп'ютерного практикуму.

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 номеру варіанта).

Хід роботи

Для виконання п.1 обрали текст із файлу text.txt - текст Достоєвського "Злочин і кара".

Підготовлюємо текст:

```
def prep_text(path):
    file = open(path, "r", encoding="utf-8")
    text = file.read()
    text = text.lower()
    text = ''.join(filter(lambda char: char in set(ALPHABET), text))
    text = text.replace('ё', 'е')

    # with open('filtered_text.txt', "w", encoding="utf-8") as file:
    #         file.write(text)

    return text
```

- Очищуємо текст від символів, які не входять до алфавіту.
- Переводимо всі літери в нижній регістр.

- Видаляємо символи, які не є буквами кирилиці, і замінюємо ё на е.

Генеруємо випадкові ключі:

```
def random_keys():
    return [''.join(random.choices(ALPHABET, k=i)) for i in range(1, 31)]
```

Зашифровуємо текст:

```
def vigenere_encrypt(text, key):
    encrypted_text = ''
    key_length = len(key)
    indexed_key = [ALPHABET.index(i) for i in key]
    indexed_text = [ALPHABET.index(i) for i in text]

    for i in range(len(indexed_text)):
        index = (indexed_text[i] + indexed_key[i % key_length]) % len(ALPHABET)
        encrypted_text += ALPHABET[index]

    return encrypted_text
```

Індекс відповідності тексту:

Виведення відбувається наступним кодом

```
print('I0', 1/M)
print('I(plain text)', AffinityIndex(plain_text))
for key in random_keys():
    cipher_text = vigenere_encrypt(plain_text, key)
    print(f'r{len(key)}: ', AffinityIndex(cipher_text))
```

Обчислюємо індекс за формулою

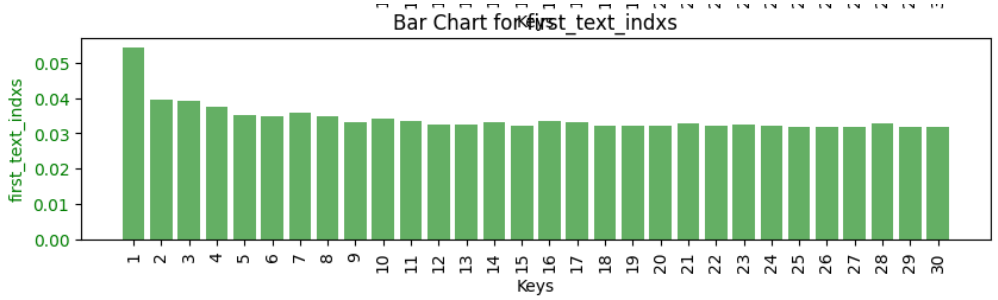
$$I(Y) = \frac{1}{n(n-1)} \sum_{t \in Z_m} N_t(Y)(N_t(Y) - 1),$$

```python
def AffinityIndex(text):
    symbols = Counter(text)

    _sum = 0
    for count in symbols.values():
        _sum += count * (count - 1)

    return _sum / (sum(symbols.values()) * (sum(symbols.values()) - 1))
```

Маємо наступний вивід:

```
I0 0.03125

I(plain text) 0.05430962032383196

r1:  0.05430962032383196

r2:  0.03980150982357874

r3:  0.03937027167061246

r4:  0.03779308597829523

r5:  0.03448472789306048

r6:  0.03434054205957306

r7:  0.034064232998275615

r8:  0.034365421183782656

r9:  0.0330046084808034

r10:  0.03378679306769591

r11:  0.03421237687425092

r12:  0.032991603484057476

r13:  0.033339157310136945

r14:  0.03315180996571016

r15:  0.03245783318283348

r16:  0.03257789380496614

r17:  0.03228556409550342

r18:  0.032372829508450704

r19:  0.033493897923592066
```

```
r20:  0.033640533973857696

r21:  0.032042050243391344

r22:  0.03241825275795458

r23:  0.03217831999190297

r24:  0.03270737833778425

r25:  0.031994742211750374

r26:  0.03244614753358352

r27:  0.03195949678578678

r28:  0.03158838318299369

r29:  0.03181210682266632

r30:  0.03205599763120581
```



2. Розшифрування. Варіант №5

Обробляємо шифрований текст, як і перед цим у пункті 1.

Так само як і вище підраховуємо індекси відповідності і кількості збігів. Це робиться для визначення довжини ключа.

```python
indexes = {}
collision_counts={}
for i in range(1, 31):
    index = AffinityIndex(cipher_text[::i])
    indexes[i] = index
    print(f'r{i} ',index)

    col_count = count_repeated_letters(cipher_text, i)
    collision_counts[i] = col_count
    print('collision count', col_count, '\n')
```

```python
def count_repeated_letters(text, r):
    count = 0
    for i in range(len(text) - r):
        if text[i] == text[i + r]:
            count += 1
    return count
```

За формулою:

$$D_r = \sum_{i=1}^{n-r} \delta(y_i, y_{i+r})$$

На основі підрахунків виводимо топ-5 довжин ключа:

```python
print('Top 5 indexes: ', dict(sorted(indexes.items(), key=lambda item: item[1], reverse=True)[:5]))
print('Top 5 collision counts:', dict(sorted(collision_counts.items(), key=lambda item: item[1], reverse=True)[:5]))
```

```
////////////////////////////////_analyzing_////////////////////////////////

r1  0.03532444245066751

collision count 182

r2  0.03364463294467222

collision count 211

r3  0.03573092683194233

collision count 192

r4  0.034531214146324624

collision count 180

r5  0.035344276138367596

collision count 192

r6  0.03398235751176928

collision count 206

r7  0.03584982612133835

collision count 173
```

r8 0.045344257897752906

collision count 189

r9 0.03691813804173355

collision count 180

r10 0.033062464714128936

collision count 211

r11 0.03479528797820498

collision count 165

r12 0.03406907154230495

collision count 196

r13 0.03857308584686775

collision count 214

r14 0.03549875311720698

collision count 198

r15 0.03585026737967915

collision count 198

r16 0.05325193325193325

collision count 302

r17 0.03427629772040648

collision count 182

r18 0.0327314700305054

collision count 216

r19 0.032959230416857534

collision count 208

r20 0.0340620233858668

collision count 207

r21 0.036978031192352843

collision count 172

r22 0.031924019607843135

collision count 169

r23 0.037537638006022084

collision count 184

r24 0.04350537397747698

collision count 173

r25 0.034761904761904765

collision count 187

r26 0.03531438415159346

collision count 195

r27 0.035256410256410256

collision count 192

r28 0.035671641791044775

collision count 175

r29 0.03680358955184018

collision count 214

r30 0.033962908180680394

collision count 215

r30 0.033962908180680394

collision count 215

Top 5 indexes: {16: 0.05325193325193325, 8: 0.045344257897752906, 24: 0.04350537397747698, 13: 0.03857308584686775, 23: 0.037537638006022084}

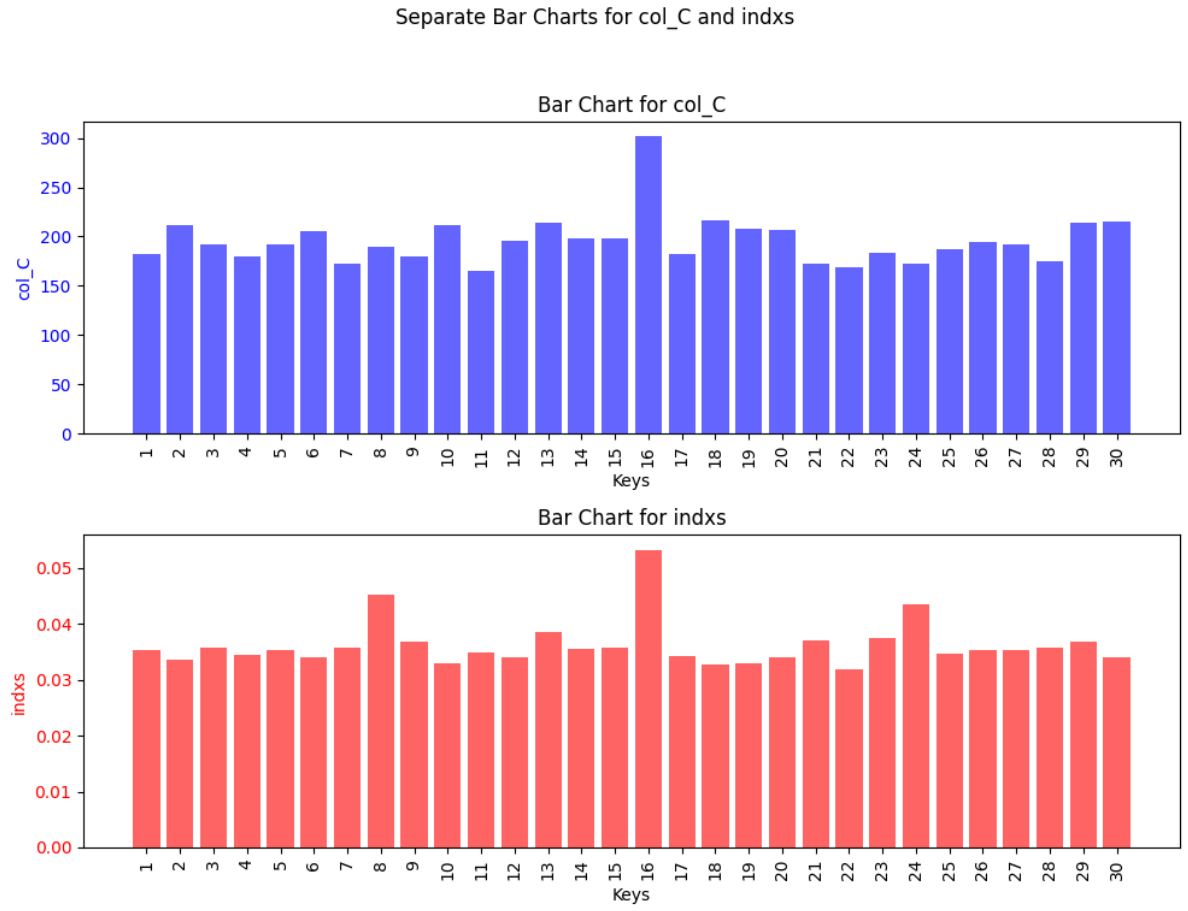
Top 5 indexes: {16: 0.05325193325193325, 8: 0.045344257897752906, 24: 0.04350537397747698, 13: 0.03857308584686775, 23: 0.037537638006022084}

Top 5 collision counts: {16: 302, 18: 216, 30: 215, 13: 214, 29: 214}

o frequency: 0.03562522265764161

Top 5 collision counts: {16: 302, 18: 216, 30: 215, 13: 214, 29: 214}

o frequency: 0.03562522265764161



По діаграмах і виводу: довжина ключа - 16 (найвищий показник серед інших значень).

Частотний аналіз для символів ключа:

```
key_len = 16
for i in range(key_len):
    freq_s = freq(cipher_text[i::key_len])
    print(f"Letter {i}:", dict(sorted(freq_s.items(), key=lambda item: item[1], reverse=True)[:5]))
```

Визначаємо частотою символів ключа від 0 до 16.

```
Letter 0: {'т': 0.09971509971509972, 'й': 0.08262108262108261, 'с':
0.07122507122507123, 'п': 0.06837606837606838, 'х': 0.06552706552706553}

Letter 1: {'у': 0.11396011396011396, 'т': 0.08262108262108261, 'н':
0.07692307692307693, 'е': 0.07122507122507123, 'к': 0.06837606837606838}

Letter 2: {'ш': 0.09116809116809117, 'р': 0.09116809116809117, 'щ':
0.07977207977207977, 'у': 0.07977207977207977, 'л': 0.07122507122507123}

Letter 0: {'т': 0.09971509971509972, 'й': 0.08262108262108261, 'с':
0.07122507122507123, 'п': 0.06837606837606838, 'х': 0.06552706552706553}
```

Letter 1: {'у': 0.11396011396011396, 'т': 0.08262108262108261, 'н': 0.07692307692307693, 'е': 0.07122507122507123, 'к': 0.06837606837606838}

Letter 2: {'ш': 0.09116809116809117, 'р': 0.09116809116809117, 'щ': 0.07977207977207977, 'у': 0.07977207977207977, 'л': 0.07122507122507123}

Letter 3: {'у': 0.09116809116809117, 'ц': 0.08262108262108261, 'ь': 0.08262108262108261, 'о': 0.07977207977207977, 'ы': 0.06837606837606838}

Letter 3: {'у': 0.09116809116809117, 'ц': 0.08262108262108261, 'ь': 0.08262108262108261, 'о': 0.07977207977207977, 'ы': 0.06837606837606838}

Letter 4: {'щ': 0.1168091168091168, 'р': 0.09116809116809117, 'л': 0.08547008547008547, 'у': 0.06837606837606838, 'ш': 0.06267806267806268}

Letter 4: {'щ': 0.1168091168091168, 'р': 0.09116809116809117, 'л': 0.08547008547008547, 'у': 0.06837606837606838, 'ш': 0.06267806267806268}

Letter 5: {'ц': 0.1168091168091168, 'н': 0.09971509971509972, 'р': 0.09686609686609686, 'х': 0.08831908831908832, 'и': 0.06837606837606838}

Letter 5: {'ц': 0.1168091168091168, 'н': 0.09971509971509972, 'р': 0.09686609686609686, 'х': 0.08831908831908832, 'и': 0.06837606837606838}

Letter 6: {'я': 0.09686609686609686, 'ц': 0.09686609686609686, 'ю': 0.09116809116809117, 'с': 0.08831908831908832, 'в': 0.06837606837606838}

Letter 6: {'я': 0.09686609686609686, 'ц': 0.09686609686609686, 'ю': 0.09116809116809117, 'с': 0.08831908831908832, 'в': 0.06837606837606838}

Letter 7: {'ь': 0.14814814814814814, 'у': 0.11396011396011396, 'ц': 0.06837606837606838, 'о': 0.06267806267806268, 'щ': 0.05413105413105413}

Letter 7: {'ь': 0.14814814814814814, 'у': 0.11396011396011396, 'ц': 0.06837606837606838, 'о': 0.06267806267806268, 'щ': 0.05413105413105413}

Letter 8: {'п': 0.08831908831908832, 'й': 0.08262108262108261, 'ж': 0.08262108262108261, 'у': 0.06837606837606838, 'б': 0.06552706552706553}

Letter 8: {'п': 0.08831908831908832, 'й': 0.08262108262108261, 'ж': 0.08262108262108261, 'у': 0.06837606837606838, 'б': 0.06552706552706553}

Letter 9: {'ь': 0.10826210826210826, 'у': 0.10541310541310542, 'о': 0.08831908831908832, 'ы': 0.07692307692307693, 'ц': 0.07122507122507123}

Letter 9: {'ь': 0.10826210826210826, 'у': 0.10541310541310542, 'о': 0.08831908831908832, 'ы': 0.07692307692307693, 'ц': 0.07122507122507123}

Letter 10: {'ю': 0.11965811965811966, 'б': 0.09116809116809117, 'х': 0.08831908831908832, 'р': 0.07407407407407407, 'ш': 0.06552706552706553}

Letter 11: {'ь': 0.1111111111111111, 'ы': 0.07692307692307693, 'у': 0.07692307692307693, 'о': 0.06837606837606838, 'ц': 0.06837606837606838}

```
Letter 10: {'ю': 0.11965811965811966, 'б': 0.09116809116809117, 'х':
0.08831908831908832, 'р': 0.07407407407407407, 'ш': 0.06552706552706553}

Letter 11: {'ь': 0.1111111111111111, 'ы': 0.07692307692307693, 'у':
0.07692307692307693, 'о': 0.06837606837606838, 'ц': 0.06837606837606838}

Letter 12: {'ч': 0.1168091168091168, 'а': 0.09971509971509972, 'ъ':
0.08262108262108261, 'э': 0.07692307692307693, 'я': 0.07122507122507123}

Letter 13: {'т': 0.09686609686609686, 'ы': 0.09116809116809117, 'н':
0.07977207977207977, 'ъ': 0.06837606837606838, 'п': 0.05982905982905983}

Letter 12: {'ч': 0.1168091168091168, 'а': 0.09971509971509972, 'ъ':
0.08262108262108261, 'э': 0.07692307692307693, 'я': 0.07122507122507123}

Letter 13: {'т': 0.09686609686609686, 'ы': 0.09116809116809117, 'н':
0.07977207977207977, 'ъ': 0.06837606837606838, 'п': 0.05982905982905983}

Letter 14: {'у': 0.1, 'к': 0.09428571428571429, 'р': 0.06571428571428571,
'т': 0.06571428571428571, 'е': 0.06285714285714286}

Letter 14: {'у': 0.1, 'к': 0.09428571428571429, 'р': 0.06571428571428571,
'т': 0.06571428571428571, 'е': 0.06285714285714286}

Letter 15: {'ч': 0.14285714285714285, 'с': 0.09142857142857143, 'о':
0.08571428571428572, 'ц': 0.07428571428571429, 'й': 0.07142857142857142}

Letter 15: {'ч': 0.14285714285714285, 'с': 0.09142857142857143, 'о':
0.08571428571428572, 'ц': 0.07428571428571429, 'й': 0.07142857142857142}
```

Формуємо можливі ключі:

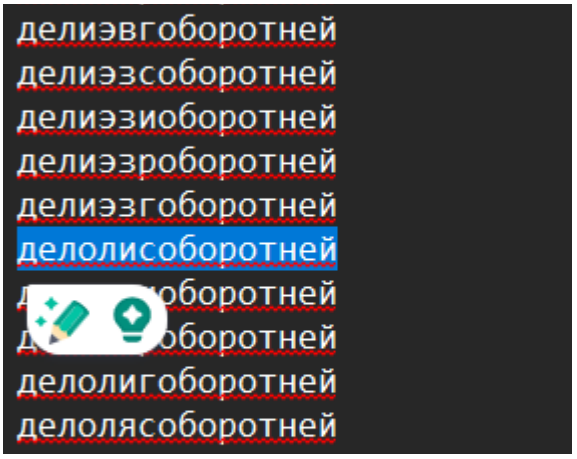
```python
with open("keys.txt", "w", encoding='utf-8') as f:
    def reqout(part, symbols):
        if len(part) >= 16:
            f.write(part + '\n')
            return
        mb = symbols[:1][0]
        for i in mb:
            i = decrypt(i, 'о')
            reqout(part+i, symbols[1:])
    reqout("", keyfrag)
```

```python
def decrypt(c, k) -> str:
    i = (ALPHABET.find(c) - ALPHABET.find(k)) % len(ALPHABET)
    return ALPHABET[i]
```

Аналізуємо утворені ключі:

```
# keyfrag = ['ти', 'ул', 'шрш', 'уцьс', 'шрл', 'цнрх', 'яцюс', 'ьу', 'пйж', 'ьус', 'обх', 'ь', 'чаъ', 'тын', 'у', 'чсс'] # freq > 0.079
# keyfrag = ['ти', 'ул', 'шрш', 'уцьс', 'шрл', 'цнрх', 'я', 'ь', 'п', 'ь', 'о', 'ь', 'чаъ', 'тын', 'у', 'чсс'] # with собор
# keyfrag = ['т', 'у', 'шрш', 'уцьс', 'шрл', 'цнрх', 'я', 'ь', 'п', 'ь', 'о', 'ь', 'чаъ', 'тын', 'у', 'чсс'] # without ддк & lower
# keyfrag = ['т', 'у', 'шрш', 'уцьс', 'шрл', 'цнрх', 'яцюс', 'ь', 'п', 'ь', 'о', 'ь', 'ь', 'ъ', 'у', 'ч'] # with оборотней
# keyfrag = ['т', 'у', 'ш', 'уцьс', 'шрл', 'цнрх', 'яцюс', 'ь', 'п', 'ь', 'о', 'ь', 'ь', 'ъ', 'у', 'ч'] # with дел
keyfrag = ['т', 'у', 'ш', 'цьс', 'шрл', 'цнрх', 'яцюс', 'ь', 'п', 'ь', 'о', 'ь', 'ь', 'ъ', 'у', 'ч'] # without деле
```

При аналізі ми поступово відкидали різні буквосполучення, що не могли б стояти поруч (на скріні вище показано). Проглядаючи файл ми все зменщували і зменшували кількість імовірних ключів отаким відкиданням, потім стали бачити законосірності - трапляння слів “дел”, “оборотней”. У кінці ми скоротили кількість ключв до близько 150, і цю кількість проглянули і знайшли підходязий ключ, яким мав би зміст.



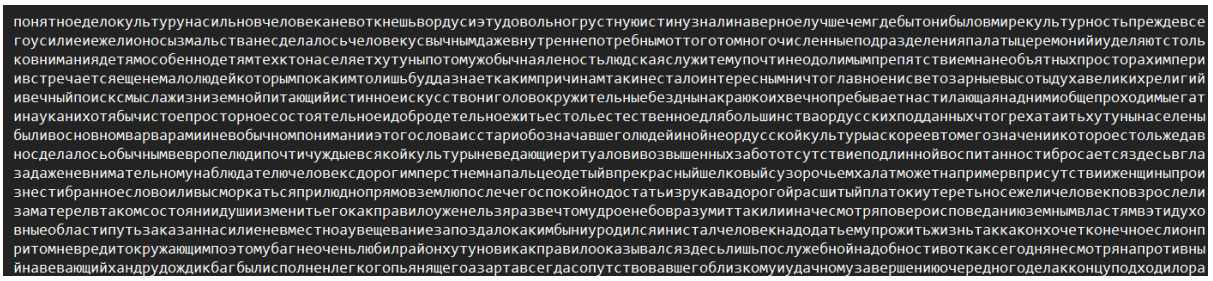
Імовірно, ми знайшли ключ - “делолисоборотней”.

Тепер розшифровуємо текст:

```
with open("de_text.txt", "w", encoding='utf-8') as f:
    key = 'делолисоборотней'
    f.write(vigenere_decrypt(cipher_text, key))
```

```
def vigenere_decrypt(text, key):
    decrypted_text = ''
    key_length = len(key)
    key_as_int = [ALPHABET.index(i) for i in key]
    text_as_int = [ALPHABET.index(i) for i in text]

    for i in range(len(text_as_int)):
        value = (text_as_int[i] - key_as_int[i % key_length]) % len(ALPHABET)
        decrypted_text += ALPHABET[value]

    return decrypted_text
```

Отриманий текст:



Вийшов осмислений текст - ключ підібраний правильно.

Висновки

У процесі виконання комп'ютерного практикуму ми опанували навички роботи з шифрами поліалфавітної перестановки на прикладі шифру Віженера. Ми навчились шифрувати відкритий текст, проводити аналіз шифрованого тексту (наприклад, визначати довжину ключа за допомогою індексу відповідності та кількості збігів), а також виконувати дешифрування. Окрім цього, ми засвоїли методи частотного криптоаналізу для знаходження ключа на основі статистичних властивостей тексту.