



**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  
Власенко Г. В. та  
Перебинос Р. О.  
Бригада №2  
Перевірів/-ла:

---

---

# Криптоаналіз шифру Віженера

## Мета роботи

Засвоєння методів частотного криптоаналізу. Здобуття навичок роботи та аналізу поточкових 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 було обрано текст із минулої лабораторної роботи (кафка) та ініціалізовано 31 випадковий ключ різної довжини (як варіант, що закоментований, були осмислені слова у якості ключів):

```
#keys = ["а", "вы", "дуб", "ночь", "огонь", "вопрос", "медведь", "мельница",  
"организмы", "предложит", "преподавать", "исследовать", "неопределенность",  
"профессионально", "непредсказуемость", "систематизированный"]  
keys = ["".join(random.choices(ALPHABET, k=i)) for i in range(1, 31)]
```

Далі зчитуємо текст для шифрування, після чого підготовлюємо його:

```
text = ""  
with open("texts/pt.txt", "r") as f:  
    text = f.read()  
pt = prepare_text(text)
```

Підготовка тексту: перетворюємо усі літери верхнього регістру у нижній та пропускаємо усі символи окрім літер алфавіту.

```
def prepare_text(text: str) -> str:  
    out = ""  
    for i in text:  
        if i.lower() not in ALPHABET:  
            continue  
  
        out += i.lower()
```

```
return out
```

Виводимо індекс відповідності для рівноімовірного алфавіту, індекс відповідності для відкритого тексту, зашифровуємо цей текст із кожним з періодів та, відповідно, виводимо індекс відповідності для шифротексту з кожним із періодів:

```
print("IC0:", 1/len(ALPHABET))

print("IC(pt):", ic(pt))
print("IC for plaintext encrypted by keys of different length")
ptctr = []
for i in keys:
    ks = VigenereKeyStream(i)
    ct = encrypt_stream(pt, ks)
    ptctr.append(f"IC(ct r={len(i)}): {ic(ct)}")
```

Визначення індексу відповідності відбувається за формулою:

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

де  $Y$  – текст, для якого визначається індекс відповідності,  $N_t(Y)$  – кількість появ букви  $t$  у шифротексті  $Y$ . Реалізується ця формула наступною функцією:

```
def ic(text: str) -> float:
    alph = {i: 0 for i in ALPHABET}
    for i in text:
        if i not in ALPHABET:
            raise Exception("Invalid letter")

        alph[i] += 1

    summ = 0
    for v in alph.values():
        summ += v * (v - 1)

    return summ / (sum(alph.values()) * (sum(alph.values())-1))
```

Для ініціалізації об'єкту потоку ключа Віженера було реалізовано такий клас:

```
class VigenereKeyStream:
    key: str
    key_len: int
    offset: int = 0

    def __init__(self, key: str):
        for i in key:
            if i not in ALPHABET:
                raise Exception("Invalid key")
```

```

        self.key = key
        self.key_len = len(key)

# reset offset for reusing
# existed key
    def reset(self):
        self.offset = 0

# produce next symbol from key
    def next(self) -> str:
        s = self.key[self.offset]
        self.offset = (self.offset + 1) % self.key_len
        return s

```

I, відповідно, функція шифрування:

```

def encrypt_stream(pt: str, stream_key) -> str:
    ct = ""
    for m in pt:
        ct += encrypt(m, stream_key.next())

    return ct
def encrypt(m, k) -> str:
    i = (ALPHABET.find(m) + ALPHABET.find(k)) % len(ALPHABET)
    return ALPHABET[i]

```

Отримуємо такі результати:

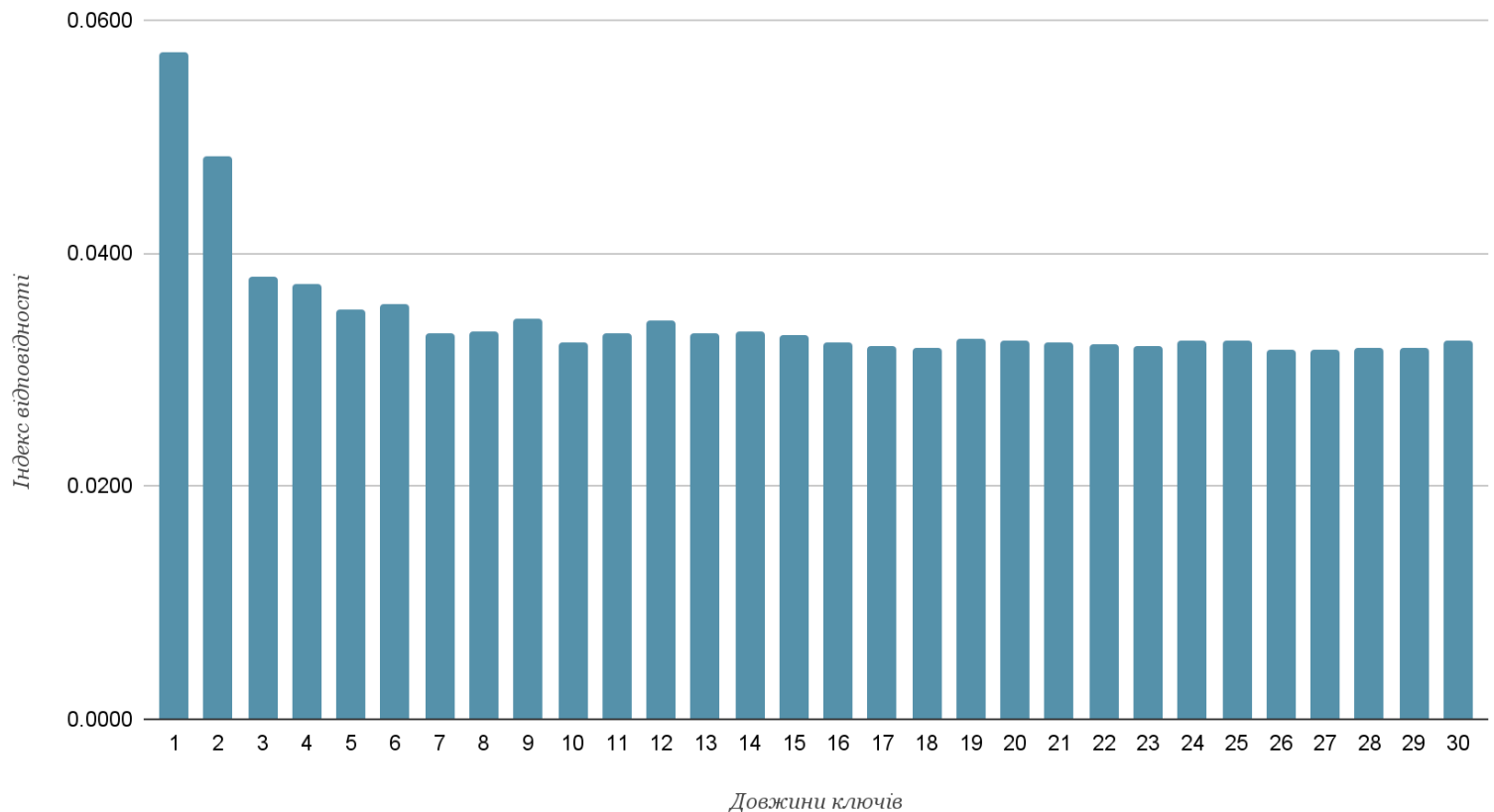
```

(kali@kali)~/crypro-24-25/lab2/perebynos_fb-22_vlasenko_fb-22_cp2]
$ python3 main.py
IC0: 0.03125
IC(pt): 0.05724840132875287
IC for plaintext encrypted by keys of different length

```

IC(ct r=1): 0.05724840132875287	IC(ct r=2): 0.045818850613641535	IC(ct r=3): 0.04041395521290167
IC(ct r=4): 0.03704416603972597	IC(ct r=5): 0.038523870195508826	IC(ct r=6): 0.03636204373987342
IC(ct r=7): 0.037403890321327064	IC(ct r=8): 0.03676358616328764	IC(ct r=9): 0.032358745971428245
IC(ct r=10): 0.03325996455740393	IC(ct r=11): 0.03450761535009283	IC(ct r=12): 0.03285266674740066
IC(ct r=13): 0.033010310801461135	IC(ct r=14): 0.03317464716550889	IC(ct r=15): 0.032277587754585876
IC(ct r=16): 0.03261469508777166	IC(ct r=17): 0.033747260328906646	IC(ct r=18): 0.03278223381775276
IC(ct r=19): 0.0325735836538932	IC(ct r=20): 0.032466082210434594	IC(ct r=21): 0.03290492412795851
IC(ct r=22): 0.03318357126527251	IC(ct r=23): 0.03221731795809353	IC(ct r=24): 0.032313626799911684
IC(ct r=25): 0.03205048823055336	IC(ct r=26): 0.03260748651387107	IC(ct r=27): 0.03244173443944771
IC(ct r=28): 0.03194474055472914	IC(ct r=29): 0.03252634053439553	IC(ct r=30): 0.03199414739365012

## Індекс відповідності ШТ за різними довжинами ключів



Переходимо до розшифрування. Маємо варіант №2. Аналогічним чином зчитуємо та підготовлюємо шифротекст. Так само підраховуємо індекс відповідності шифротексту:

```
text = ""
with open("texts/2_ct.txt", "r") as f:
    text = f.read()
ct = prepare_text(text)

print("IC(ct):", ic(ct))
```

Для розшифрування обчислюємо статистику співпадінь символів та індекси відповідності за усіма можливими періодами:

```
ctic = []
ctcs = []
for i in range(1, 31):
    ctic.append(f"CS(ct r={i}): {colision_stat(ct, i)}")
    ctcs.append(f"IC(ct r={i}): {ic(ct[:i])}")

print("IC for cipher text with different length of key")
print(tabulate([ctic[i:i+3] for i in range(0, len(ctic), 3)], headers=["",
"", ""]))
```

```
print("\nSum of Kroneker symbol for differen length of key")
print(tabulate([ctcs[i:i+3] for i in range(0, len(ctcs), 3)], headers=["",
"", ""]))
```

Статистика співпадінь символів розраховувалась за формулою  $(\delta(a, b))$  – символ Кронекера):

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

із наступною реалізацією:

```
def collision_stat(text: str, r: int) -> int:
    stat = 0
    for i in range(len(text)-r):
        if text[i] == text[i+r]:
            stat += 1
    return stat
```

Результат виконання:

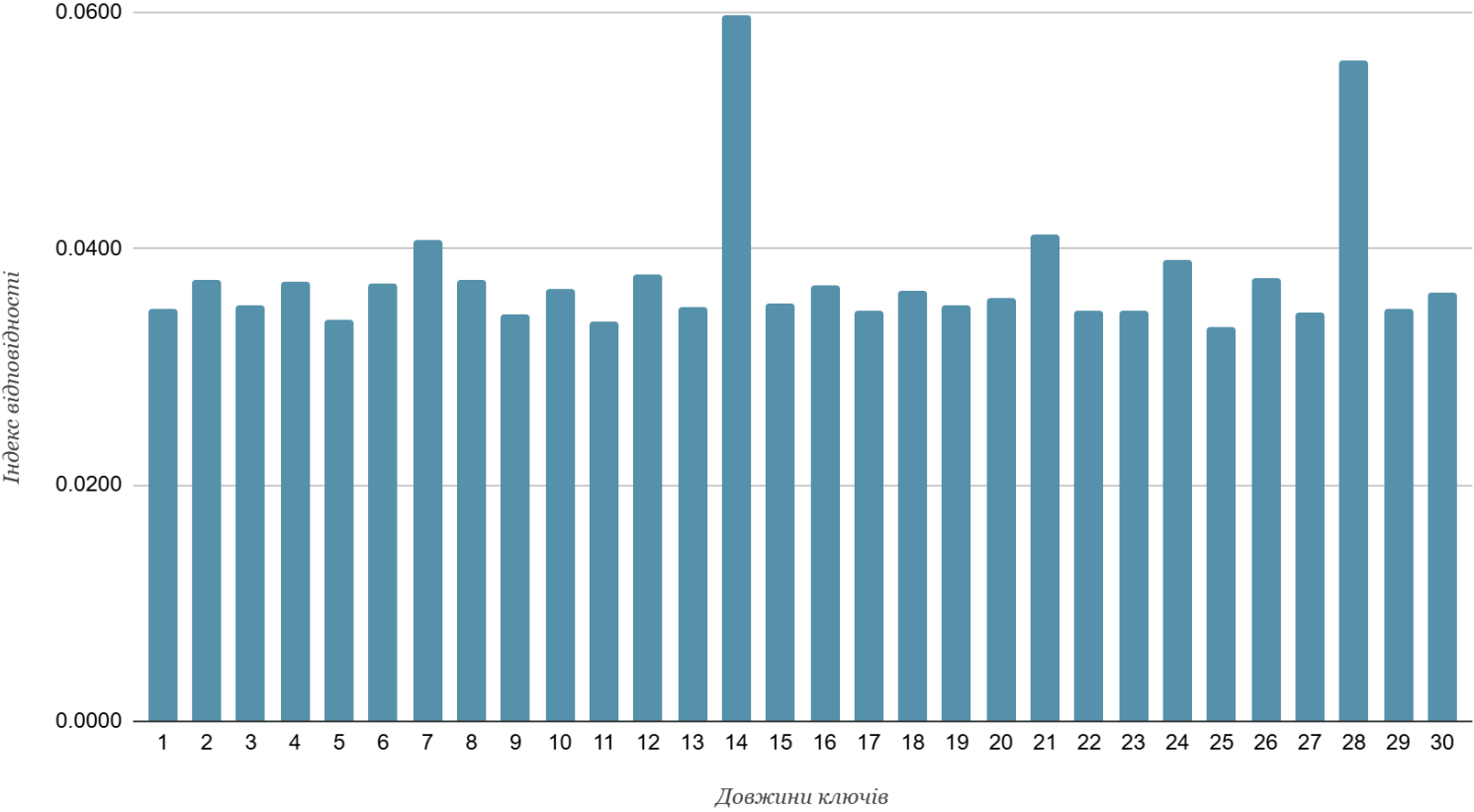
```
IC(ct): 0.03485780146768279
IC for cipher text with different length of key
```

CS(ct r=1): 239	CS(ct r=2): 258	CS(ct r=3): 236
CS(ct r=4): 262	CS(ct r=5): 240	CS(ct r=6): 232
CS(ct r=7): 244	CS(ct r=8): 220	CS(ct r=9): 238
CS(ct r=10): 253	CS(ct r=11): 247	CS(ct r=12): 247
CS(ct r=13): 212	CS(ct r=14): 396	CS(ct r=15): 252
CS(ct r=16): 253	CS(ct r=17): 262	CS(ct r=18): 252
CS(ct r=19): 247	CS(ct r=20): 233	CS(ct r=21): 276
CS(ct r=22): 219	CS(ct r=23): 228	CS(ct r=24): 243
CS(ct r=25): 238	CS(ct r=26): 234	CS(ct r=27): 237
CS(ct r=28): 396	CS(ct r=29): 236	CS(ct r=30): 235

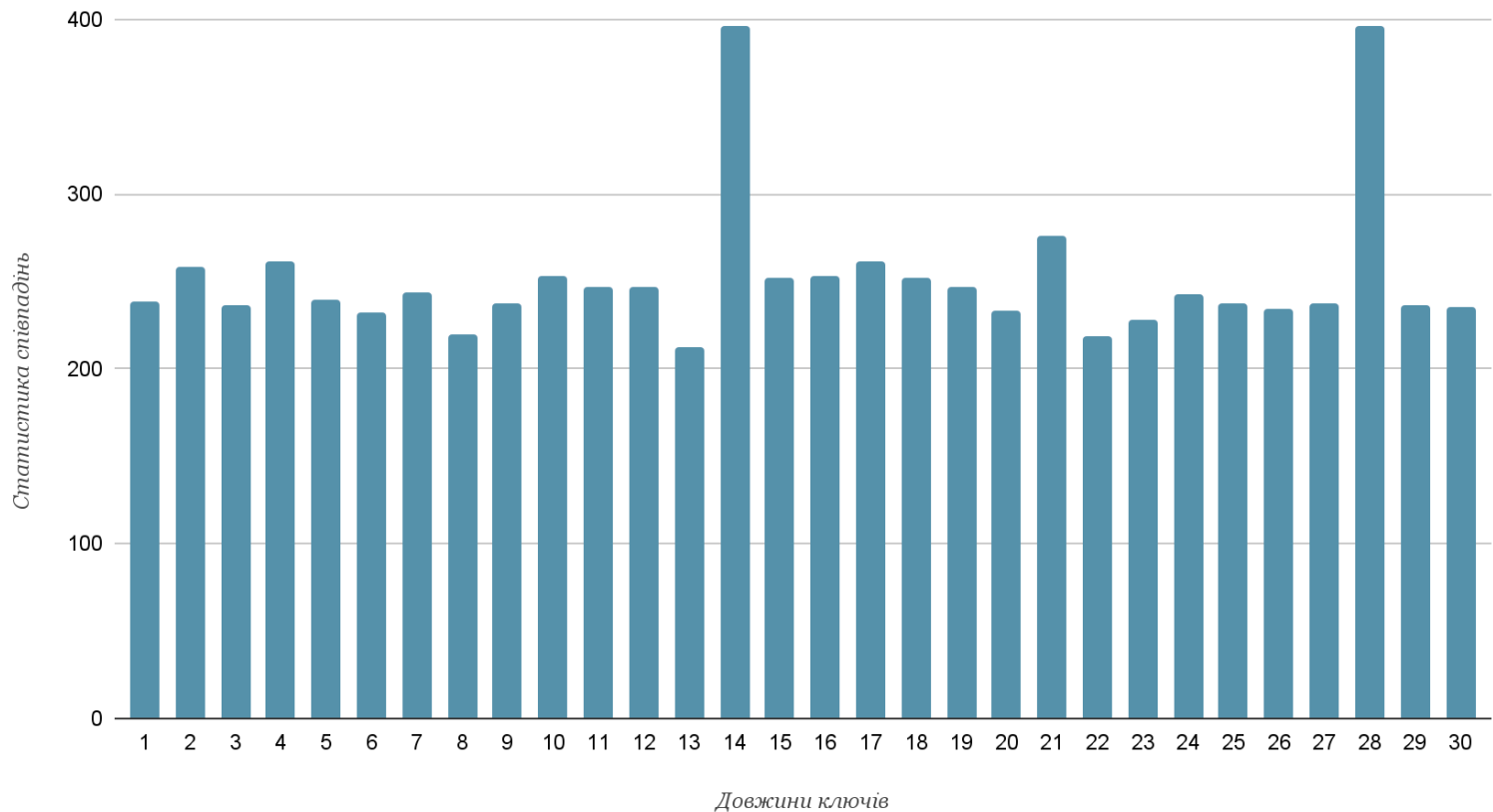
Sum of Kroneker symbol for differen length of key

IC(ct r=1): 0.03485780146768279	IC(ct r=2): 0.03739641785493314	IC(ct r=3): 0.035182840340433466
IC(ct r=4): 0.0372103200818509	IC(ct r=5): 0.03401142623004251	IC(ct r=6): 0.03702393064095192
IC(ct r=7): 0.04073917831334542	IC(ct r=8): 0.03737323120242442	IC(ct r=9): 0.03445531420991544
IC(ct r=10): 0.03653672682814096	IC(ct r=11): 0.03383443413428421	IC(ct r=12): 0.037825655335247246
IC(ct r=13): 0.035026391040902966	IC(ct r=14): 0.059755082976953276	IC(ct r=15): 0.03533474135908009
IC(ct r=16): 0.03695068464353626	IC(ct r=17): 0.03479204262266907	IC(ct r=18): 0.03648166883461001
IC(ct r=19): 0.03521970257721553	IC(ct r=20): 0.03583925194681437	IC(ct r=21): 0.04111912525112802
IC(ct r=22): 0.03472334610059161	IC(ct r=23): 0.034758778415252066	IC(ct r=24): 0.03904425158041359
IC(ct r=25): 0.033317081098651065	IC(ct r=26): 0.037454885035713384	IC(ct r=27): 0.03462122856522683
IC(ct r=28): 0.055891901377555495	IC(ct r=29): 0.03491436100131752	IC(ct r=30): 0.03623285379725661

Індекс відповідності ШТ за різними довжинами ключів



## Статистика співпадінь ШТ за різними довжинами ключів



Бачимо, що значення на  $r = 14$  виділяється найбільше з усіх у обох таблицях результатів, тож це, вірогідно, і є довжина шуканого ключа. Тепер, маючи довжину, можемо перейти до визначення власне ключа. Для цього визначимо частоти символів у шифротексті для кожного символу ключа від 0 до 14 (знайдена довжина) та шляхом порівняння найчастіших символів у тексті із найчастішим символом мови ("o") спробуємо відновити ключ. Отже маємо:

```
print("Letters frequency for every key symbol")
key_len = 14
for i in range(key_len):
    freq = frequencies(ct[i::key_len])
    print(f"Letter {i}:", list(freq.items())[:5])
```

За основу функції визначення частот символів взято таку ж функцію із попередньої лабораторної роботи:

```
def frequencies(text: str) -> dict[str, float]:
    freq = {i: 0 for i in ALPHABET}
    for i in text:
        if i not in ALPHABET:
            raise Exception("Invalid letter")

        freq[i] += 1
```



```
count = sum(freq.values())
for k in freq:
    freq[k] /= count

return dict(sorted(freq.items(), key=lambda item: item[1], reverse=True))
```

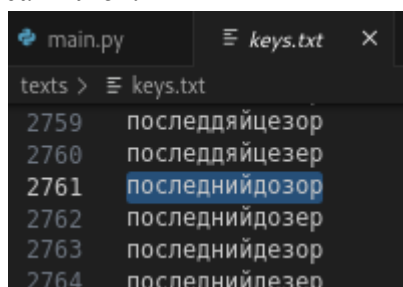
Результати виконання:

```
Letters frequency for every key symbol
Letter 0: [('ф', 0.11641221374045801), ('э', 0.11259541984732824), ('н', 0.08015267175572519), ('ч', 0.07061068702290077), ('ь', 0.06679389312977099)]
Letter 1: [('ь', 0.10877862595419847), ('у', 0.09923664122137404), ('о', 0.08206106870229007), ('а', 0.07251908396946564), ('я', 0.06297709923664122)]
Letter 2: [('я', 0.11450381679389313), ('ц', 0.08969465648854962), ('г', 0.07061068702290077), ('ю', 0.07061068702290077), ('щ', 0.06870229007633588)]
Letter 3: [('р', 0.10305343511450382), ('н', 0.09923664122137404), ('щ', 0.0916030534351145), ('ь', 0.07061068702290077), ('э', 0.061068702290076333)]
Letter 4: [('у', 0.12022900763358779), ('к', 0.09923664122137404), ('е', 0.07251908396946564), ('ц', 0.07061068702290077), ('н', 0.05916030534351145)]
Letter 5: [('й', 0.12022900763358779), ('т', 0.11068702290076336), ('д', 0.07824427480916031), ('с', 0.07824427480916031), ('м', 0.06297709923664122)]
Letter 6: [('т', 0.10133843212237094), ('ы', 0.09560229445506692), ('х', 0.08030592734225621), ('я', 0.07839388145315487), ('э', 0.07074569789674952)]
Letter 7: [('ц', 0.09560229445506692), ('и', 0.08986615678776291), ('н', 0.0841300191204589), ('х', 0.06500956022944551), ('ь', 0.06500956022944551)]
Letter 8: [('о', 0.1089866156787763), ('ч', 0.0994263862332696), ('й', 0.0745697896749522), ('ц', 0.06883365200764818), ('с', 0.06309751434034416)]
Letter 9: [('т', 0.1089866156787763), ('д', 0.08986615678776291), ('й', 0.07648183556405354), ('с', 0.07074569789674952), ('ц', 0.06309751434034416)]
Letter 10: [('ь', 0.10133843212237094), ('у', 0.08986615678776291), ('о', 0.07648183556405354), ('ы', 0.07648183556405354), ('а', 0.06309751434034416)]
Letter 11: [('х', 0.0994263862332696), ('м', 0.0841300191204589), ('э', 0.08221797323135756), ('н', 0.07648183556405354), ('щ', 0.07265774378585087)]
Letter 12: [('ь', 0.1089866156787763), ('у', 0.0994263862332696), ('ц', 0.0745697896749522), ('о', 0.06883365200764818), ('а', 0.05162523900573614)]
Letter 13: [('ю', 0.11089866156787763), ('р', 0.08221797323135756), ('х', 0.0745697896749522), ('б', 0.07074569789674952), ('в', 0.06118546845124283)]
```

Маючи частоти та об'єднавши найчастіші символи у єдиний фрагмент для кожного з періодів, підбираємо ключ (вивід об'ємний, тому одразу зберігаємо до файлу):

```
with open("texts/keys.txt", "w") as f:
    keyfrag = ["фэ", "ьу", "яц", "рлщ", "ук", "йт", "ты", "цин", "ч", "тд",
               "ьу", "х", "ьу", "ю"]
    def reqout(part, symbols):
        if len(part) >= 14:
            f.write(part + "\n")
            return
        mb = symbols[:1][0]
        for i in mb:
            i = decrypt(i, MOST_FREQUENT_SYMBOL)
            reqout(part+i, symbols[1:])
    reqout("", keyfrag)
```

Значення у файлі усі наближені до ключа, що можна здогадатися і вручну. Ось власне і сам ключ:



```
main.py  keys.txt
texts > keys.txt
2759  последдяйцезор
2760  последдяйцезер
2761  последнийдозор
2762  последнийдозер
2763  последнийдезор
2764  последнийдезер
```

Тепер можемо розшифрувати шифротекст, використовуючи знайдений ключ, та зберегти отриманий відкритий текст до окремого файлу:

```
ks = VigenereKeyStream("последнийдозор")
with open("texts/2_pt.txt", "w") as f:
    f.write(decrypt_stream(ct, ks))
```

Власне функція розшифрування:

```
def decrypt_stream(ct: str, stream_key) -> str:
    pt = ""
    for c in ct:
        pt += decrypt(c, stream_key.next())

    return pt

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

## Висновки

У ході виконання комп'ютерного практикуму ми набули навичок роботи із шифрами поліалфавітної перестановки на прикладі шифру Віженера, навчилися ним шифрувати відкритий текст та аналізувати шифротекст (наприклад, визначення періоду/довжини ключа). Ми також засвоїли методи частотного криптоаналізу для знаходження ключа на основі його відомої довжини, засновуючись на частотах символів у шифротексті.