Міністерство освіти і науки України

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Фізико-технічний інститут

Криптографія

Лабораторна робота №3

Виконав студент групи ФБ-13

Лагно Костянтин

**Київ 2023**

**Криптоаналіз афінної біграмної підстановки**

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

**Порядок виконання 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.

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

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).

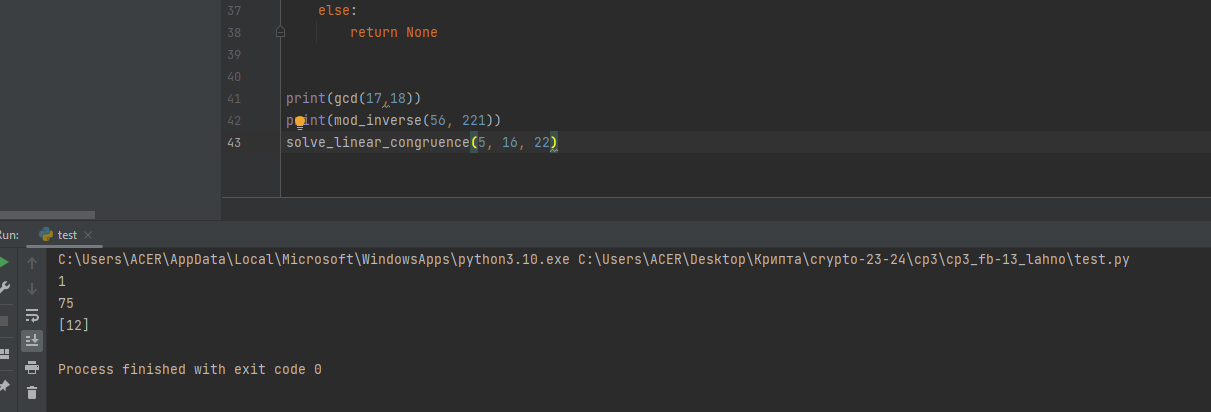
4. Для кожного кандидата на ключ дешифрувати шифртекст. Якщо шифртекст не є змістовним текстом російською мовою, відкинути цього кандидата.

5. Повторювати дії 3-4 доти, доки дешифрований текст не буде змістовним.

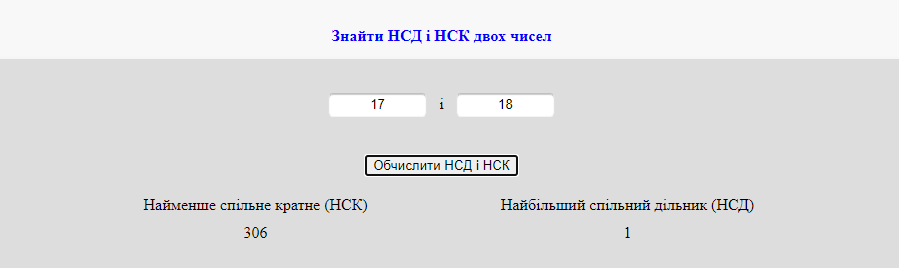
**Хід роботи:**

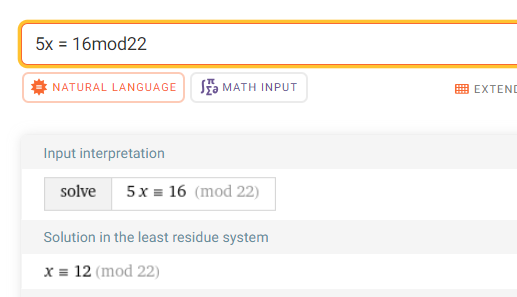
1. Функції для математичних операцій: пошук НСД, розширений алгоритм Евкліда, пошук оберненого та розв’язування лінійних порівнянь:

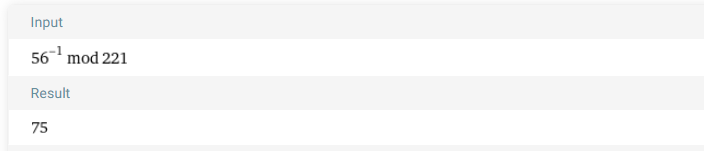
def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return a  
  
  
def extended\_gcd(a, b):  
 if a == 0:  
 return b, 0, 1  
 else:  
 g, x, y = extended\_gcd(b % a, a)  
 return g, y - (b // a) \* x, x  
  
  
def mod\_inverse(a, m):  
 g, x, \_ = extended\_gcd(a, m)  
 return x % m if g == 1 else None  
  
  
def solve\_linear\_congruence(a, b, m):  
 roots = []  
 a, b = a % m, b % m  
 g = gcd(a, m)  
 if g == 1:  
 a\_inv = mod\_inverse(a, m)  
 roots.append((a\_inv \* b) % m)  
 return roots  
 elif g > 1 and b % g == 0:  
 a1 = a // g  
 b1 = b // g  
 m1 = m // g  
 roots = solve\_linear\_congruence(a1, b1, m1)  
 roots.extend((roots[0] + m1 \* i) % m for i in range(g))  
 return roots  
 else:  
 return None

Результати виконання:  


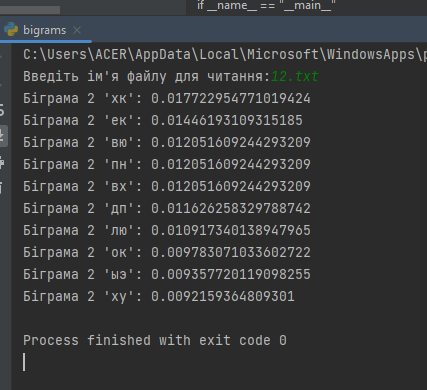
Перевірка:







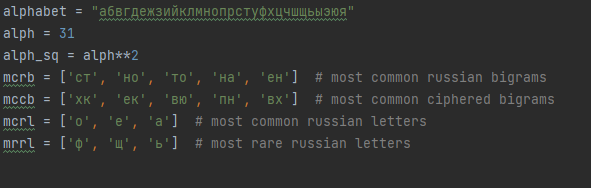
1. Для виконання цієї частини завдання, я використав частини коду з першої лабораторної, а саме форматування тексту (хоч він вже і був без пробілів і лишніх символів, все одно залишились перенесення рядків), і пошук і підрахунок біграм без перетину зашифрованого тексту. Ці функції я трохи модифікував, підігнавши під основне завдання, тому функція підрахунку біграм видає перші 10 біграм з найбільшою частотою. Програму прикріплю окремо файлом з назвою bigrams.py.

В результаті виконання я отримав список найчастіших біграм мого тексту:  


Їх я потім і використовуватиму в процесі розшифрування. Сам же оброблений текст записав в окремий файл var12.txt.

3-5. Тут починається найцікавіше. Розібравшись з завданням, я написав більш-менш робочий і зв’язний код для пошуку ключа і розшифрування:

def find\_possible\_keys(mcrb, mccb, alphabet):  
 possible\_keys = []  
 for i in range(len(mcrb)):  
 for j in range(len(mcrb)):  
 if i != j:  
 for k in range(len(mccb)):  
 for v in range(len(mccb)):  
 if k != v:  
 x1 = bigram\_to\_numeric(mcrb[i], alphabet)  
 y1 = bigram\_to\_numeric(mccb[k], alphabet)  
 x2 = bigram\_to\_numeric(mcrb[j], alphabet)  
 y2 = bigram\_to\_numeric(mccb[v], alphabet)  
 x = x1 - x2  
 y = y1 - y2  
 if gcd(x, alph\_sq) == 1 and mod\_inverse(x, alph\_sq):  
 c = solve\_linear\_congruence(x, y, alph\_sq)  
 if c:  
 for a in c:  
 if a > 0 and gcd(a, alph\_sq) == 1 and mod\_inverse(a, alph\_sq):  
 b = (y1 - a \* x1) % alph\_sq  
 possible\_keys.append((a, b))  
 return possible\_keys  
  
  
def bigram\_to\_numeric(bigram, alphabet):  
 index1 = alphabet.index(bigram[0])  
 index2 = alphabet.index(bigram[1])  
 numeric = index1 \* len(alphabet) + index2  
 return numeric  
  
  
def numeric\_to\_bigram(int, alphabet):  
 bigram = ""  
 letter1 = int // 31  
 letter2 = int % 31  
 bigram += alphabet[letter1]  
 bigram += alphabet[letter2]  
 return bigram  
  
  
def decrypt\_text\_from\_file(text, keys, alphabet):  
 decrypted\_texts = []  
 for key in keys:  
 decrypted = ""  
 for letter in range(0, len(text) - 1, 2):  
 bigram = text[letter: letter + 2]  
 y = bigram\_to\_numeric(bigram, alphabet)  
 c, b = key  
 x = mod\_inverse(c, alph\_sq) \* (y - b) % alph\_sq  
 decrypted += numeric\_to\_bigram(x, alphabet)  
 decrypted\_texts.append([key, decrypted])  
  
 return decrypted\_texts

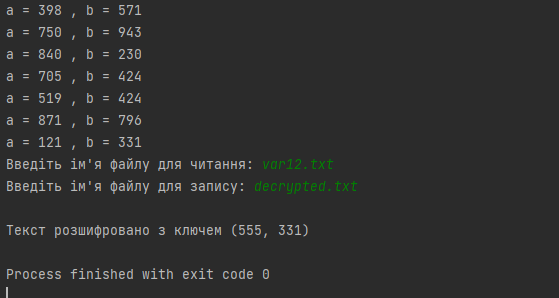
Для розшифрування використовував наступні вхідні дані:  


Біграми відкритого тексту взяв з методички, так само як і найчастіші і найрідкісніші букви. Перейшовши до самого розшифрування, почались проблеми. Все ніяк не виходило розшифрувати текст, а в результаті, після довгого дебагу і перебору усіх ключів (їх на диво було багато), виявилось що я просто в функції numeric\_to\_bigram переплутав місцями букви. Тому в мене виходило розшифрувати текст, але він був нечитабельним, бо букви в біграмах були поміняні місцями. Відповідно, потрібно було дописати ще функцію для перевірки тексту на його умовну змістовність. З допомогою, мені таки вдалось написати таку функцію, в якій рахуються найчастіші і найрідкісніші букви кожного тексту і порівнюються з заданими такими буквами. При певній кількості співпадінь, виводиться правильно розшифрований текст. В моєму випадку при > 4 співпадіннях зразу вивело правильний текст.

Сам код функції:

def find\_readible\_text(text):  
 counter = {char: text.count(char) for char in set(text) if char.isalpha() or char.isspace()}  
 sorted\_letters = dict(sorted(counter.items(), key=lambda item: item[1], reverse=True))  
 mcl = list(sorted\_letters.keys())[:3]  
 mrl = list(sorted\_letters.keys())[-3:]  
 return sum(i in mcrl for i in mcl) + sum(j in mrrl for j in mrl)  
  
  
def final\_text\_checking(texts):  
 for key, text in texts:  
 if find\_readible\_text(text) > 4:  
 return [key, text]

Результат виконання повної програми:



Зашифрований текст (уривок):

Оклйазогтдхвоэшктжсэллыэежяхбчехеквюхуашайейллокйбяктйвндкпйзмихшкшэхкиккптжпнуафйнрцэзкурлюхжыэвхбкдкшхяиожщбтхлцймтдщгхуюклцбьшнцмещврчцчьтакугэвжйулнекчвоцмахухкпюкукгхупнфйллрзгдзкдохксавхцляплэенцяхкпюлюбйчдгцякбуейяждохксавхцляяхкпюлюыптжхквгейчдгцхнцмыжшжхжэьехтшфхййвнхкйеокгхйэоцмахутжджюхбжяежвбурзещежяхбчехжфлююющьчэюхуевхвюйшыьоржлвнкбгхвюдшчжуастхжыэжвашцэежяюэранмафйфквхябчжщжшхкктхзюяжябтмюхййвнгцдпгфздьлпнзюекябзюгцщдэтлюзжжэпнппт

Розшифрованиий текст (уривок):

Когдапожарныеисоседиушлилеоауфманосталсясдедушкойсполдингомдугласомитомомвсеонизадумчивосмотрелинадогорающиеостаткигаражалеоткнулногойвмокруюзолуимедленновысказалточтолежалонадушепервоечтоузнаешьвжизниэточтотыдуракпоследнеечтоузнаешьэточтотывсетотжедуракмногоепередумалязаодинтолькочасисказалсебедаведьтыслепойлеоауфманхотитеувидатьнастоящую

Виглядає наче уривок з книги Рея Бредбері - Вино з кульбабок.

Всі вихідні файли і результати окремо надішлю.

Висновок: в ході виконання даної лабораторної роботи, я опанував навички використання частотного аналізу, модулярної арифметики, розширеного алгоритму Евкліда і алгоритму розв’язку лінійних порівнянь для аналізу, пошуку ключа та розшифрування тексту, зашифрованого афінним шифром.