НИУ ИТМО

Факультет программной инженерии и

компьютерной техники

**Лабораторная работа №1**

**Основы шифрования данных**

по дисциплине

‘Информационная безопасность’

Из раздела

**Криптографические системы с секретным ключом**

Вариант 2

Описание задания

Реализовать шифрование и дешифрацию файла по методу Виженера. Ключевая фраза вводится. Реализовать в программе частотный криптоанализ зашифрованного текста.

# Программные и аппаратные средства

Процессор: AMD Ryzen 5 4600H with Radeon Graphics

Видеокарта: NVIDIA GeForce GTX 1650

Объем оперативной памяти: 16 GB

Язык программирования: Python

# Листинг разработанной программы с комментариями

**main.py:**

from model.affine import аffine\_сipher  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 with open("encrypted\_text.txt", 'r', encoding='utf-8') as file:  
 encr\_text = file.read()  
 affine = аffine\_сipher('keywords')  
 text = encr\_text  
 a = 15  
 b = 4  
 encrypt\_text = affine.encrypt(text, a, b)  
 print(f"Encrypt word = {encrypt\_text}")  
 decrypt\_text = affine.decrypt(encrypt\_text, a, b)  
 print(f"Decrypt word = {decrypt\_text}")  
 frequency\_analysis = affine.analyze\_frequency(encrypt\_text)  
 print("Частотный анализ:")  
 for char, freq in affine.analyze\_frequency(encrypt\_text).items():  
 print(f"{char}: {freq // 2}")  
 print("\n")  
 affine.check\_keywords(encrypt\_text)  
 print("\nТоп 10 наиболее часто встречающихся слов:")  
 top\_words = affine.get\_top\_n\_words(encrypt\_text)  
 for word, freq in top\_words:  
 print(f"{word}: {freq}")

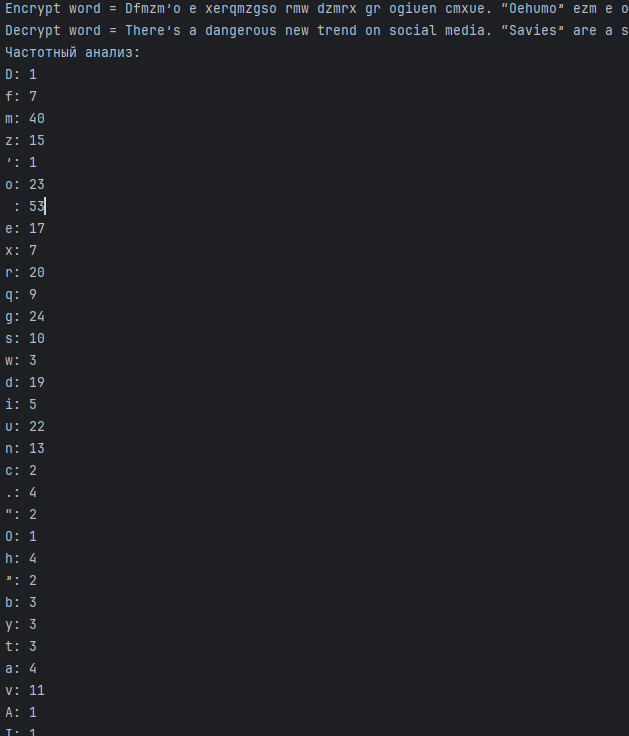
**helpers.py:**

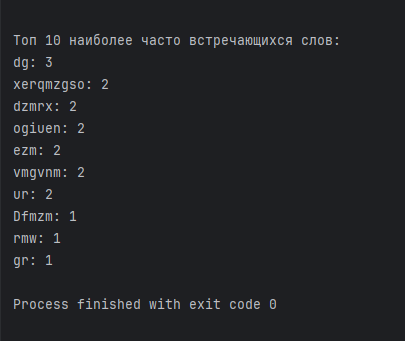
def is\_relatively\_prime(a, b):  
 *"""  
 Проверяет, являются ли числа a и b взаимно простыми.  
  
 Args:  
 a (int): Первое число для проверки.  
 b (int): Второе число для проверки.  
  
 Returns:  
 bool: True, если числа взаимно простые; False в противном случае.  
 """* a = abs(a)  
 b = abs(b)  
 if a == 0 or b == 0:  
 return False  
 if a == 1 and b == 1:  
 return True  
 while b != 0:  
 a, b = b, a % b  
 return a == 1  
  
  
def load\_keyword\_list(file\_path):  
 *"""  
 Метод загрузки слов в массив  
 :param file\_path: Путь до файла со словами  
 :return: Массив слов  
 """* with open(file\_path, 'r', encoding='utf-8') as f:  
 return [line.strip() for line in f]

**affine.py:**

import re  
  
from lab1.helpers import is\_relatively\_prime  
from collections import Counter  
from lab1.helpers import load\_keyword\_list  
  
  
class аffine\_сipher:  
 *"""  
 Класс который реализует афинный шифр  
 """* def \_\_init\_\_(self, keyword\_list\_path):  
 *"""  
 Init метод  
 :param keyword\_list\_path: Путь до массива с ключевыми словами  
 """* self.keyword\_list = load\_keyword\_list(keyword\_list\_path)  
 self.frequency\_analysis = Counter()  
 self.ru = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'  
 self.en = 'abcdefghijklmnopqrstuvwxyz'  
 self.answer = ''  
  
 def encrypt(self, text, a, b):  
 *"""  
 Шифрование текста афинным методом  
 :param text: Текст для кодировки  
 :param a: Параметр а из формулы кодирования  
 :param b: Параметр b из формулы кодирования  
 :return: Зашифрованный текст  
 """* self.answer = ''  
 index = 0  
 for letter in text:  
 letter\_status = False  
 if letter.lower() in self.ru:  
 if not is\_relatively\_prime(len(self.ru), a):  
 return "Русский алфавит и число a взаимно простые"  
 if letter not in self.ru:  
 letter\_status = True  
 new\_letter = self.ru[(a \* self.ru.index(letter.lower()) + b) % len(self.ru)]  
 if letter\_status:  
 self.answer += new\_letter.upper()  
 else:  
 self.answer += new\_letter  
 elif letter.lower() in self.en:  
 if not is\_relatively\_prime(len(self.en), a):  
 return "Английский алфавит и число a взаимно простые"  
 if letter not in self.en:  
 letter\_status = True  
 new\_letter = self.en[(a \* self.en.index(letter.lower()) + b) % len(self.en)]  
 if letter\_status:  
 self.answer += new\_letter.upper()  
 else:  
 self.answer += new\_letter  
 else:  
 self.answer += letter  
 index += 1  
 return self.answer  
  
 def decrypt(self, text, a, b):  
 *"""  
 Расшифровка текста афинным методом  
 :param text: Зашифрованный текст  
 :param a: Изначальный параметр а для формулы  
 :param b: Изначальный параметр b для формулы  
 :return: Расшифрованный текст  
 """* self.answer = ''  
 index = 0  
 for letter in text:  
 letter\_status = False  
 if letter.lower() in self.ru:  
 a\_ru = pow(a, -1, len(self.ru))  
 if not is\_relatively\_prime(len(self.ru), a):  
 return "Русский алфавит и число a взаимно простые"  
 if letter not in self.ru:  
 letter\_status = True  
 new\_letter = self.ru[(a\_ru \* self.ru.index(letter.lower()) + b) % len(self.ru)]  
 if letter\_status:  
 self.answer += new\_letter.upper()  
 else:  
 self.answer += new\_letter  
 elif letter.lower() in self.en:  
 a\_en = pow(a, -1, len(self.en))  
 if not is\_relatively\_prime(len(self.en), a):  
 return "Английский алфавит и число a взаимно простые"  
 if letter not in self.en:  
 letter\_status = True  
 new\_letter = self.en[(a\_en \* (self.en.index(letter.lower()) - b)) % len(self.en)]  
 if letter\_status:  
 self.answer += new\_letter.upper()  
 else:  
 self.answer += new\_letter  
 else:  
 self.answer += letter  
 index += 1  
 return self.answer  
  
 def analyze\_frequency(self, encrypted\_text):  
 *"""  
 Частотый анализ символов в тексте  
 :param encrypted\_text: Зашифрованный текст  
 :return: Словарь с символом и их количеством в тексте  
 """* for char in encrypted\_text:  
 self.frequency\_analysis[char] += 1  
 return dict(self.frequency\_analysis)  
  
 def check\_keywords(self, encrypted\_text):  
 *"""  
 Метод для проверки наличия ключевых слов в тексте  
 :param encrypted\_text: Зашифрованный текст  
 """* for keyword in self.keyword\_list:  
 if keyword in encrypted\_text:  
 pattern = rf'\b{keyword}\b'  
 print(f"Ключевое слово {keyword} встречается {len(re.findall(pattern, encrypted\_text))} раз")  
 else:  
 print(f"Ключевое слово '{keyword}' не найдено в зашифрованном тексте.")  
  
 def get\_top\_n\_words(self, encrypted\_text, n=10):  
 *"""  
 Метод для вывода 10 самых частых слов в тексте  
 :param encrypted\_text: Зашифрованный текст  
 :param n: Изначально задано 10 - количество слов  
 :return: Массив со словами  
 """* word\_freq = Counter(re.findall(r'\b[a-zA-Zа-яА-Я]{2,}\b', encrypted\_text))  
 return sorted(word\_freq.items(), key=lambda x: x[1], reverse=True)[:n]

# Результаты работы программы





# Выводы по работе

В результате выполнения лабораторной работы были получены навыки реализации Аффиного алгоритма на языке программирования Python.