Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Московский политехнический университет»

(Московский политех)

Отчёт по курсу «Программирование криптографических алгоритмов»

Лабораторная работа 1.3. Шифры блочной замены



Выполнил:

Студент группы 221-352

Иванов В. В.

Проверил преподаватель: Бутакова Н. Г.

Москва 2024г.

**Аннотация**

* **Среда программирования**
  + Visual Studio Code
* **Язык программирования**
  + Python
* **Процедуры для запуска программы** 
  + Visual Studio Code (main.py)
* **Пословица-тест**
  + Тот, кто ложится на два стула, падает на ребра.
* **Текст для проверки работы (не меньше 1000 знаков (1430))**

Жизнь - это удивительное приключение, полное разнообразных событий и встреч. В каждом моменте мы находим что-то новое и уникальное. Стремление к росту и саморазвитию вдохновляет нас на поиск новых горизонтов. Важно помнить, что каждый шаг вперед приносит с собой уроки и опыт.

Разнообразие культур, языков и традиций делает наш мир удивительно богатым. Общение с людьми разных национальностей расширяет кругозор, позволяя нам понимать и уважать друг друга. Взаимное уважение и терпимость создают основу для гармоничного сосуществования.

Природа тоже играет важную роль в нашей жизни. Красота закатов, шум океана, пение птиц - все это напоминает нам о величии мира природы. Забота о окружающей среде становится неотъемлемой частью ответственного образа жизни.

Работа и творчество придают смысл нашим усилиям. Стремление к достижению целей мотивирует нас на новые начинания. Каждый проект, даже самый маленький, приносит удовлетворение и чувство выполненного долга.

Семья и друзья являются надежной опорой в нашей жизни. Обмен историями, веселые посиделки и поддержка в трудные моменты создают теплую атмосферу взаимопонимания и любви.

Таким образом, наша жизнь - это мозаика различных моментов, соединенных воедино. Важно ценить каждый момент и стремиться делать мир вокруг нас ярче и лучше. С любовью, терпением и целеустремленностью мы можем создавать свою уникальную историю, наполненную смыслом и радостью.

* **Код программы-интерфейса**
* import sys
* import random
* from PyQt5.QtWidgets import QApplication, QWidget, QVBoxLayout, QHBoxLayout, QLabel, QLineEdit, QPushButton, QComboBox, QTextEdit, QCheckBox
* from PyQt5.QtCore import Qt
* from atbash import atbash\_encrypt, atbash\_decrypt
* from cesar import cesar\_encrypt, cesar\_decrypt, cesar\_check\_parameters
* from polibia import polibia\_encrypt, polibia\_decrypt
* from tritemiy import tritemiy\_encrypt, tritemiy\_decrypt
* from belazo import belazo\_encrypt, belazo\_decrypt, belazo\_check\_parameters
* from vigener import vigener\_encrypt, vigener\_decrypt, vigener\_check\_parameters
* from S\_block import s\_block\_encrypt, s\_block\_decrypt
* from matrix import matrix\_encrypt, matrix\_decrypt, matrix\_check\_parameters, multiply\_matrix, determinant, adjugate\_matrix, inverse\_matrix
* from playfair import playfair\_encrypt, playfair\_decrypt, playfair\_check\_parameters
* *# from veritcalTransposition import vertical\_transposition\_encrypt, vertical\_transposition\_decrypt, vertical\_transposition\_check\_parameters*
* available\_ciphers = [
* "Шифр АТБАШ", "Шифр Цезаря", "Шифр Полибия",
* "Шифр Тритемия", "Шифр Белазо", "Шифр Виженера", "МАГМА(s\_block)",
* "Шифр Матричный", "Шифр Плейфера", *# "Вертикальная Транспозиция",*
* ]
* alphabet = [
* "а", "б", "в", "г", "д", "е", "ж", "з", "и", "й", "к", "л", "м",
* "н", "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ",
* "ъ", "ы", "ь", "э", "ю", "я"
* ]
* alphabet\_polibia = [
* ["а", "б", "в", "г", "д", "е"],
* ["ж", "з", "и", "й", "к", "л"],
* ["м", "н", "о", "п", "р", "с"],
* ["т", "у", "ф", "х", "ц", "ч"],
* ["ш", "щ", "ъ", "ы", "ь", "э"],
* ["ю", "я"]
* ]
* alphabet\_playfair = [
* "а", "б", "в", "г", "д", "е", "ж", "з", "и", "к", "л", "м", "н",
* "о", "п", "р", "с", "т", "у", "ф", "х", "ц", "ч", "ш", "щ", "ь",
* "ы", "э", "ю", "я"
* ]
* alphabet\_sblock = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"]
* mem = {
* "bigTextFlag": False,
* "vigenerSwitch": False,
* "mode": "encrypt",
* }
* class CipherApp(QWidget):
* def \_\_init\_\_(self):
* super().\_\_init\_\_()
* self.initUI()
* def initUI(self):
* self.setWindowTitle('Шифры')
* self.resize(960, 640)
* layout = QVBoxLayout()
* *# Выбор шифра*
* cipher\_layout = QHBoxLayout()
* cipher\_label = QLabel('Выберите шифр:')
* self.cipher\_combo = QComboBox()
* self.cipher\_combo.addItems(available\_ciphers)
* cipher\_layout.addWidget(cipher\_label)
* cipher\_layout.addWidget(self.cipher\_combo)
* *# Ввод открытого текста*
* open\_text\_label = QLabel('Введите открытый текст(Расшифрованный):')
* self.open\_text\_edit = QTextEdit()
* *# Ввод зашифрованного текста*
* cipher\_text\_label = QLabel('Шифрованный текст:')
* self.cipher\_text\_edit = QTextEdit()
* *# Ввод сдвига для шифра Цезаря*
* self.cesar\_shift\_edit = QLineEdit()
* self.cesar\_shift\_edit.setPlaceholderText('Введите сдвиг для шифра Цезаря')
* self.cesar\_shift\_edit.textChanged.connect(self.check\_cesar\_shift)
* *# Ввод ключевого слова для шифра Белазо или Плейфера*
* self.keyword\_edit = QLineEdit()
* self.keyword\_edit.setPlaceholderText('Введите ключевое слово для шифра Белазо или Плейфера')
* *# Ввод ключевой буквы для шифра Виженера*
* self.vigener\_key\_edit = QLineEdit()
* self.vigener\_key\_edit.setPlaceholderText('Введите ключевую букву для шифра Виженера')
* self.vigener\_key\_edit.textChanged.connect(self.check\_vigener\_key)
* *# Ввод ключевой матрицы для шифра Матричный*
* self.matrix\_edit = QLineEdit()
* self.matrix\_edit.setPlaceholderText('Введите ключевую матрицу для шифра Матричный')
* *# Ввод ключа для шифра вертикальной транспозиции*
* *# self.vertical\_transposition\_keyword\_edit = QLineEdit()*
* *# self.vertical\_transposition\_keyword\_edit.setPlaceholderText('Введите ключ для шифра вертикальной транспозиции')*
* *# Режим работы шифра (шифрование или дешифрование)*
* mode\_layout = QHBoxLayout()
* mode\_label = QLabel('Выберите режим:')
* self.mode\_combo = QComboBox()
* self.mode\_combo.addItems(['Шифрование', 'Расшифрование'])
* mode\_layout.addWidget(mode\_label)
* mode\_layout.addWidget(self.mode\_combo)
* *# Кнопка для запуска шифрования/дешифрования*
* self.encrypt\_button = QPushButton('Выполнить')
* layout.addLayout(cipher\_layout)
* layout.addWidget(open\_text\_label)
* layout.addWidget(self.open\_text\_edit)
* layout.addWidget(cipher\_text\_label)
* layout.addWidget(self.cipher\_text\_edit)
* layout.addWidget(self.cesar\_shift\_edit)
* layout.addWidget(self.keyword\_edit)
* layout.addWidget(self.vigener\_key\_edit)
* layout.addWidget(self.matrix\_edit)
* *# layout.addWidget(self.vertical\_transposition\_keyword\_edit)*
* layout.addLayout(mode\_layout)
* layout.addWidget(self.encrypt\_button)
* self.setLayout(layout)
* *# Переключатель для выбора режима текста*
* self.text\_mode\_checkbox = QCheckBox('Расширенный текст')
* layout.addWidget(self.text\_mode\_checkbox)
* *# Подключение слотов к сигналам*
* self.encrypt\_button.clicked.connect(self.cipher\_parser)
* self.text\_mode\_checkbox.stateChanged.connect(self.handle\_text\_mode\_change)
* def handle\_text\_mode\_change(self, state):
* if state == Qt.Checked:
* mem["bigTextFlag"] = True
* else:
* mem["bigTextFlag"] = False
* def check\_cesar\_shift(self):
* shift\_text = self.cesar\_shift\_edit.text()
* try:
* shift = int(shift\_text)
* if shift < 0 or shift >= len(alphabet):
* self.cesar\_shift\_edit.setStyleSheet("QLineEdit { color: red; }")
* else:
* self.cesar\_shift\_edit.setStyleSheet("")
* except ValueError:
* self.cesar\_shift\_edit.setStyleSheet("QLineEdit { color: red; }")
* def check\_vigener\_key(self):
* key\_text = self.vigener\_key\_edit.text()
* if len(key\_text) != 1 or key\_text.lower() not in alphabet:
* self.vigener\_key\_edit.setStyleSheet("QLineEdit { color: red; }")
* else:
* self.vigener\_key\_edit.setStyleSheet("")
* def text\_preparation(self, text):
* bigTextFlag = mem["bigTextFlag"]
* if bigTextFlag:
* *# Обработка расширенного текста*
* return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ", "прбл").replace(":", "двтч").replace(";", "тчсзп").replace("(", "отскб").replace(")", "зкскб").replace("?", "впрзн").replace("!", "восклзн").replace("\n", "првст").lower()
* else:
* *# Обработка обычного текста*
* return text.replace("ё", "е").replace(".", "тчк").replace(",", "зпт").replace("-", "тире").replace(" ", "").replace(":", "").replace(";", "").replace("(", "").replace(")", "").replace("?", "").replace("!", "").replace("\n", "").lower()
* def cipher\_parser(self):
* cipher\_choose\_input = self.cipher\_combo.currentText()
* open\_text\_input = self.open\_text\_edit.toPlainText()
* cipher\_text\_input = self.cipher\_text\_edit.toPlainText()
* cesar\_shift = self.cesar\_shift\_edit.text()
* keyword = self.keyword\_edit.text()
* vigener\_keyletter = self.vigener\_key\_edit.text()
* matrix\_input = self.matrix\_edit.text()
* *# Определение режима работы (шифрование или дешифрование)*
* mode = 'encrypt' if self.mode\_combo.currentText() == 'Шифрование' else 'decrypt'
* *# Определение флага для обработки больших текстов*
* bigTextFlag = len(open\_text\_input) > 1000  *#ваш порог длины текста*
* if cipher\_choose\_input == "Шифр АТБАШ":
* if mode == "encrypt":
* cipher\_text\_input = atbash\_encrypt(self.text\_preparation(open\_text\_input), alphabet)
* elif mode == "decrypt":
* open\_text\_input = atbash\_decrypt(cipher\_text\_input, alphabet)
* elif cipher\_choose\_input == "Шифр Цезаря":
* if cesar\_shift:  *# Проверка на пустую строку*
* cesar\_shift = int(cesar\_shift)
* if cesar\_check\_parameters(cesar\_shift, alphabet):
* if mode == "encrypt":
* cipher\_text\_input = cesar\_encrypt(self.text\_preparation(open\_text\_input), cesar\_shift, alphabet)
* elif mode == "decrypt":
* open\_text\_input = cesar\_decrypt(cipher\_text\_input, cesar\_shift, alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода сдвига"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода сдвига"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите сдвиг для шифра Цезаря"
* elif mode == "decrypt":
* open\_text\_input = "Введите сдвиг для шифра Цезаря"
* elif cipher\_choose\_input == "Шифр Полибия":
* if mode == "encrypt":
* cipher\_text\_input = polibia\_encrypt(self.text\_preparation(open\_text\_input), alphabet\_polibia)
* elif mode == "decrypt":
* open\_text\_input = polibia\_decrypt(cipher\_text\_input, alphabet\_polibia)
* elif cipher\_choose\_input == "Шифр Тритемия":
* if mode == "encrypt":
* cipher\_text\_input = tritemiy\_encrypt(self.text\_preparation(open\_text\_input), alphabet)
* elif mode == "decrypt":
* open\_text\_input = tritemiy\_decrypt(cipher\_text\_input, alphabet)
* elif cipher\_choose\_input == "Шифр Белазо":
* if keyword:
* if belazo\_check\_parameters(keyword.lower(), alphabet):
* if mode == "encrypt":
* cipher\_text\_input = belazo\_encrypt(self.text\_preparation(open\_text\_input), keyword.lower(), alphabet)
* elif mode == "decrypt":
* open\_text\_input = belazo\_decrypt(cipher\_text\_input, keyword.lower(), alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода ключевого слова"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода ключевого слова"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевое слово для шифра Белазо"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевое слово для шифра Белазо"
* elif cipher\_choose\_input == "Шифр Виженера":
* if vigener\_keyletter:
* if vigener\_check\_parameters(vigener\_keyletter, alphabet):
* mode = "encrypt" if self.mode\_combo.currentText() == 'Шифрование' else 'decrypt'
* if mode == "encrypt":
* cipher\_text\_input = vigener\_encrypt(self.text\_preparation(open\_text\_input), vigener\_keyletter, "selfkey", alphabet)
* elif mode == "decrypt":
* open\_text\_input = vigener\_decrypt(cipher\_text\_input, vigener\_keyletter, "selfkey", alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода ключевой буквы"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода ключевой буквы"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевую букву для шифра Виженера"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевую букву для шифра Виженера"
* elif cipher\_choose\_input == "МАГМА(s\_block)":
* if mode == "encrypt":
* cipher\_text\_input = s\_block\_encrypt(self.text\_preparation(open\_text\_input), alphabet\_sblock)
* elif mode == "decrypt":
* open\_text\_input = s\_block\_decrypt(cipher\_text\_input, alphabet\_sblock)
* elif cipher\_choose\_input == "Шифр Матричный":
* input\_matrix = list(map(int, matrix\_input.split()))
* matrix\_input = [input\_matrix[:3], input\_matrix[3:6], input\_matrix[6:]]
* if matrix\_input:
* if matrix\_check\_parameters(matrix\_input):
* if mode == "encrypt":
* cipher\_text\_input = matrix\_encrypt(self.text\_preparation(open\_text\_input), matrix\_input, alphabet)
* elif mode == "decrypt":
* open\_text\_input = matrix\_decrypt(cipher\_text\_input, matrix\_input, alphabet)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ввода матрицы"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ввода матрицы"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевую матрицу для шифра Матричный"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевую матрицу для шифра Матричный"
* elif cipher\_choose\_input == "Шифр Плейфера":
* keyword = self.keyword\_edit.text()
* if keyword:
* if playfair\_check\_parameters(keyword, alphabet\_playfair):
* if mode == "encrypt":
* cipher\_text\_input = playfair\_encrypt(self.text\_preparation(open\_text\_input), keyword, alphabet\_playfair)
* elif mode == "decrypt":
* open\_text\_input = playfair\_decrypt(cipher\_text\_input, keyword, alphabet\_playfair)
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Проверьте правильность ключевого слова"
* elif mode == "decrypt":
* open\_text\_input = "Проверьте правильность ключевого слова"
* else:
* if mode == "encrypt":
* cipher\_text\_input = "Введите ключевое слово для шифра Плейфэра"
* elif mode == "decrypt":
* open\_text\_input = "Введите ключевое слово для шифра Плейфэра"
* *# elif cipher\_choose\_input == "Вертикальная Транспозиция":*
* *#     keyword = self.vertical\_transposition\_keyword\_edit.text()*
* *#     if keyword:*
* *#         matrix\_input = [int(num) for num in keyword.split() if num.isdigit()]*
* *#         if vertical\_transposition\_check\_parameters(matrix\_input, cipher\_text\_input):*
* *#             if mode == "encrypt":*
* *#                 cipher\_text\_input = vertical\_transposition\_encrypt(cipher\_text\_input, matrix\_input)*
* *#             elif mode == "decrypt":*
* *#                 open\_text\_input = vertical\_transposition\_decrypt(cipher\_text\_input, matrix\_input)*
* *#         else:*
* *#             if mode == "encrypt":*
* *#                 cipher\_text\_input = "Проверьте правильность ввода ключа"*
* *#             elif mode == "decrypt":*
* *#                 open\_text\_input = "Проверьте правильность ввода ключа"*
* *#     else:*
* *#         if mode == "encrypt":*
* *#             cipher\_text\_input = "Введите ключ и текст для шифра вертикальной транспозиции"*
* *#         elif mode == "decrypt":*
* *#             open\_text\_input = "Введите ключ и текст для шифра вертикальной транспозиции"*
* else:
* pass
* *# Обновление текста в виджетах*
* self.open\_text\_edit.setPlainText(open\_text\_input)
* self.cipher\_text\_edit.setPlainText(cipher\_text\_input)
* if \_\_name\_\_ == '\_\_main\_\_':
* app = QApplication(sys.argv)
* ex = CipherApp()
* ex.show()
* sys.exit(app.exec\_())

**8. Матричный шифр**

Принцип работы шифра можно разделить на несколько шагов:

1. Генерация ключа — создание квадратной матрицы, которая называется

ключевой матрицей.

2. Преобразование текста в числовую форму — каждый символ текста

преобразуется в числовое значение.

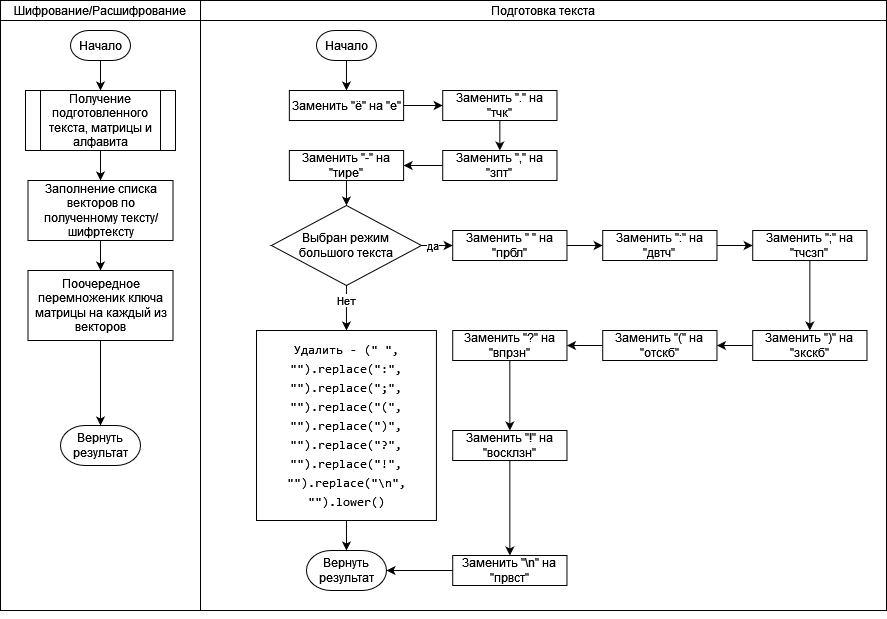
3. Умножение ключевой матрицы на векторы текста — ключевая матрица

умножается на каждый вектор текста.

4. Дешифрование текста — для дешифрования текста выполняются

обратные операции.

**Блок-схема программы**



**Код программы с комментариями**

def multiply\_matrix(A, B):

    rowsA, colsA = len(A), len(A[0])

    rowsB, colsB = len(B), len(B[0])

    C = []

    if colsA != rowsB:

        return False

    for i in range(rowsA):

        C.append([])

    for k in range(colsB):

        for i in range(rowsA):

            t = 0

            for j in range(rowsB):

                t += A[i][j] \* B[j][k]

            C[i].append(t)

    return C

def determinant(A):

    N = len(A)

    B = [row[:] for row in A]

    denom = 1

    exchanges = 0

    for i in range(N - 1):

        maxN = i

        maxValue = abs(B[i][i])

        for j in range(i + 1, N):

            value = abs(B[j][i])

            if value > maxValue:

                maxN = j

                maxValue = value

        if maxN > i:

            B[i], B[maxN] = B[maxN], B[i]

            exchanges += 1

        else:

            if maxValue == 0:

                return maxValue

        value1 = B[i][i]

        for j in range(i + 1, N):

            value2 = B[j][i]

            B[j][i] = 0

            for k in range(i + 1, N):

                B[j][k] = (B[j][k] \* value1 - B[i][k] \* value2) / denom

        denom = value1

    if exchanges % 2:

        return -B[N - 1][N - 1]

    else:

        return B[N - 1][N - 1]

def adjugate\_matrix(A):

    N = len(A)

    adjA = []

    for i in range(N):

        adjA.append([])

        for j in range(N):

            B = []

            sign = 1 if (i + j) % 2 == 0 else -1

            for m in range(j):

                B.append([A[m][n] for n in range(i)] + [A[m][n] for n in range(i + 1, N)])

            for m in range(j + 1, N):

                B.append([A[m][n] for n in range(i)] + [A[m][n] for n in range(i + 1, N)])

            adjA[i].append(sign \* determinant(B))

    return adjA

def inverse\_matrix(A):

    det = determinant(A)

    if det == 0:

        return False

    N = len(A)

    B = adjugate\_matrix(A)

    for i in range(N):

        for j in range(N):

            B[i][j] /= det

    return B

def matrix\_check\_parameters(matrix):

    print(matrix)

    if determinant(matrix) == 0:

        return False

    for row in matrix:

        for num in row:

            if not isinstance(num, (int, float)):

                return False

    if len(matrix) == 3 and len(matrix[2]) == 3:

        return True  *# True, если матрица 3х3*

    return False  *# False, если матрица не соответствует требованиям*

def matrix\_encrypt(open\_text, key\_matrix, alphabet):

    encrypted\_text = ""

    open\_text\_array = []

    if len(open\_text) % len(key\_matrix) != 0:

        open\_text += "ф" \* (3 - len(open\_text) % len(key\_matrix))

    for i in range(0, len(open\_text), len(key\_matrix)):

        vector = []

        for letter in open\_text[i:i + len(key\_matrix)]:

            vector.append([alphabet.index(letter)])

        open\_text\_array.append(vector)

    for vector in open\_text\_array:

        result\_matrix = multiply\_matrix(key\_matrix, vector)

        if not result\_matrix:

            break

        for el in result\_matrix:

            encrypted\_text += "".join(str(e).rjust(3, "0") for e in el)

    return encrypted\_text

def matrix\_decrypt(encrypted\_text, key\_matrix, alphabet):

    decrypted\_text = ""

    inverse\_matrix\_ = inverse\_matrix(key\_matrix)

    encrypted\_text\_array = []

    for i in range(0, len(encrypted\_text), 9):

        vector = []

        current\_str = encrypted\_text[i:i + 9]

        for j in range(len(key\_matrix)):

            vector.append([int(current\_str[j \* 3:j \* 3 + 3])])

        encrypted\_text\_array.append(vector)

    for vector in encrypted\_text\_array:

        if not inverse\_matrix\_:

            break

        result\_matrix = multiply\_matrix(inverse\_matrix\_, vector)

        if not result\_matrix:

            break

        for el in result\_matrix:

            index = int(round(el[0][0].real)) if isinstance(el[0][0], complex) else int(round(el[0][0]))

            if index >= 0:

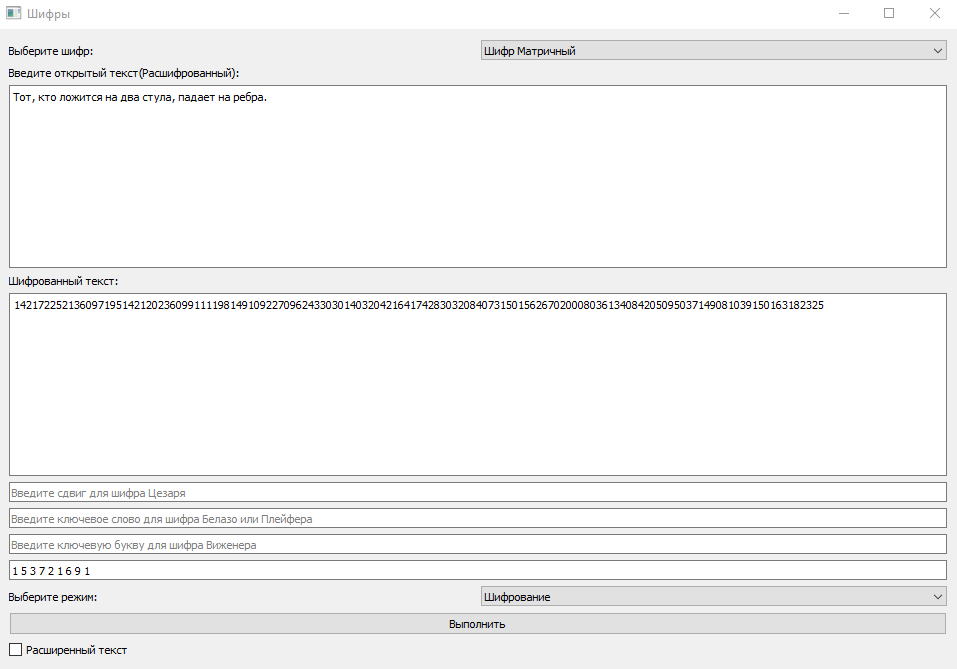
                decrypted\_text += alphabet[index % len(alphabet)]

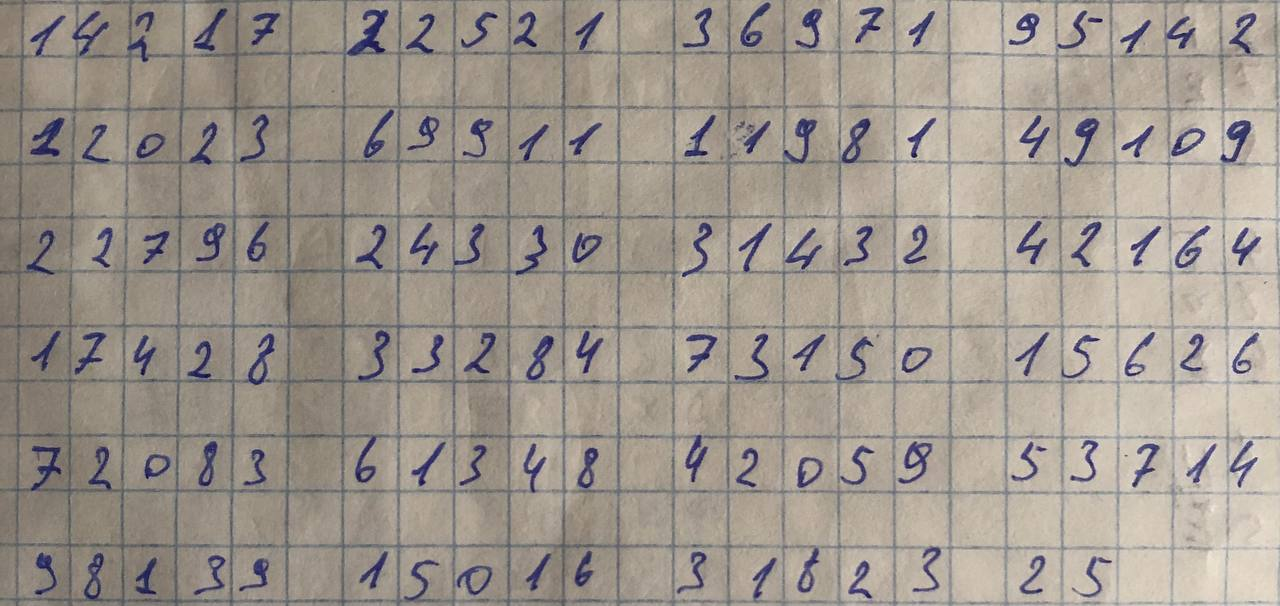
            else:

                decrypted\_text += alphabet[index % len(alphabet)]

    return decrypted\_text

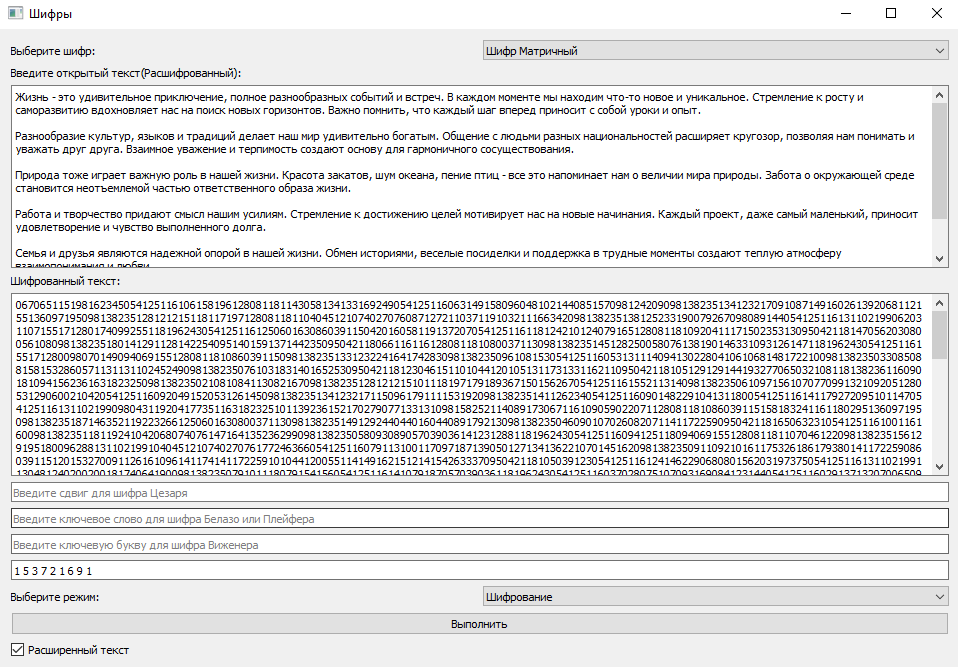
**Тестирование**



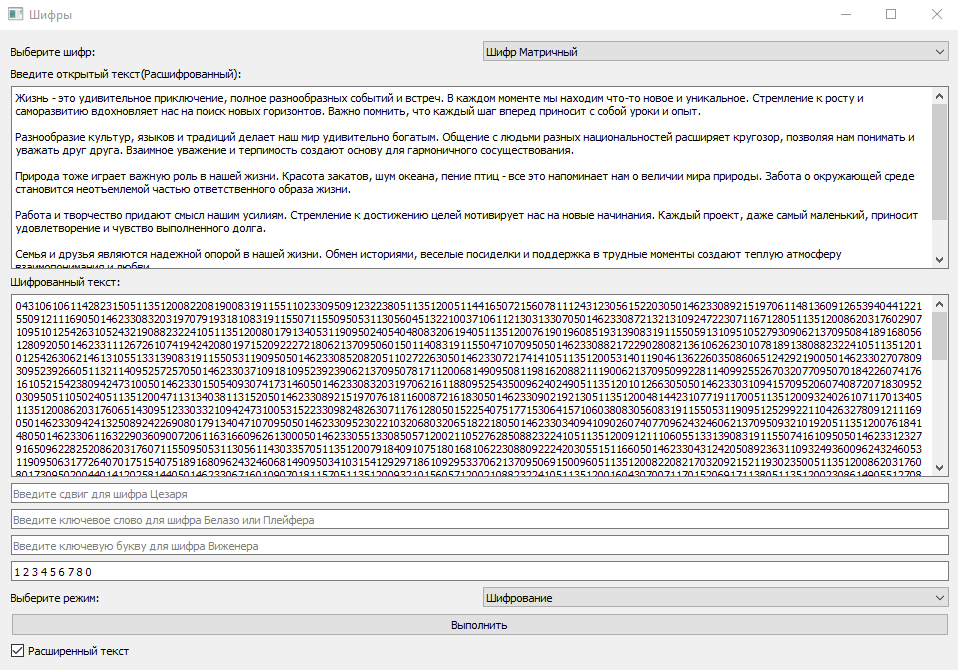


**Работа с текстом не менее 1000 знаков**

**Зашифрование**



**Расшифрование**



**2. Шифр Плэйфера**

Шифр Плейфера — это метод шифрования, названный в честь Чарльза

Уитстона Плейфера. Он был представлен в XIX веке и отличался простотой

и эффективностью, что делало его популярным среди военных и

разведывательных служб в разные исторические периоды.

**Структура шифра:**

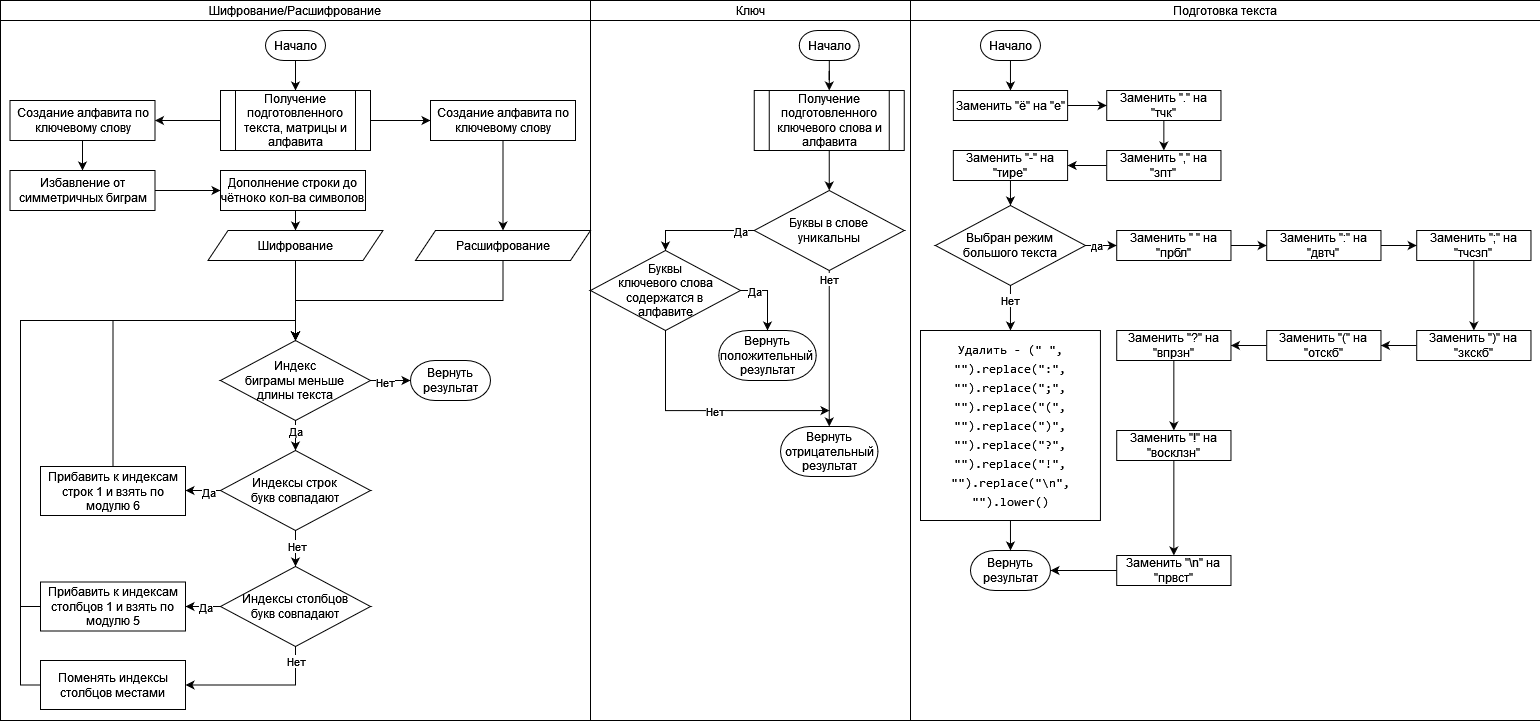
● 6x5 матрица, в которую помещаются буквы алфавита;

● ключевое слово или фраза записывается в верхней части матрицы,

при этом повторяющиеся буквы в ключевом слове игнорируются;

● оставшиеся буквы алфавита заполняют оставшиеся места в матрице.

**Блок-схема программы**



**Код программы с комментариями**

def get\_alphabet\_index(element, alphabet):

    for row in range(len(alphabet)):

        for col in range(len(alphabet[row])):

            if element == alphabet[row][col]:

                return [row, col]

    return []

def playfair\_check\_parameters(keyword, alphabet):

    if len(set(keyword)) != len(keyword):

        return False  *# Ложь, если буквы в слове не уникальны*

    for letter in keyword:

        if letter not in alphabet:

            return False  *# Ложь, если буквы в ключевом слове не содержатся в алфавите*

    return True  *# Истина, если всё ок*

def playfair\_encrypt(open\_text, keyword, alphabet):

    encrypted\_text = ""  *# Шифртекст*

*# Формирование алфавита с учетом ключевого слова*

*# print("keyword", keyword)*

*# print("alphabet", alphabet)*

    unprepared\_alphabet = list(keyword) + [char for char in alphabet if char not in keyword]

    unprepared\_alphabet = list(dict.fromkeys(unprepared\_alphabet))

*# print(unprepared\_alphabet, "unprepared\_alphabet")*

    while len(unprepared\_alphabet) < 30:

        unprepared\_alphabet.append('\_')

    new\_alphabet = [unprepared\_alphabet[i:i+6] for i in range(0, len(unprepared\_alphabet), 6)]

*# print(new\_alphabet, "newAlphabet")*

    for i in range(0, len(open\_text) - 1, 2):  *# Избавление от биграм из одинаковых букв*

        if open\_text[i] == open\_text[i + 1] and not (open\_text[i] == 'ф' and open\_text[i + 1] == 'ф'):  *# Если биграма не "фф"*

            open\_text = open\_text[:i + 1] + "ф" + open\_text[i + 1:]

        elif open\_text[i] == open\_text[i + 1] and open\_text[i] == 'ф' and open\_text[i + 1] == 'ф':  *# Если биграма "фф"*

            open\_text = open\_text[:i + 1] + "х" + open\_text[i + 1:]

    open\_text += "ф" \* (len(open\_text) % 2)  *# Дополнение строки до чётного кол-ва букв*

    for i in range(0, len(open\_text), 2):  *# Зашифрование*

        first\_index = get\_alphabet\_index(open\_text[i], new\_alphabet)

        second\_index = get\_alphabet\_index(open\_text[i + 1], new\_alphabet)

        if first\_index[0] == second\_index[0]:

            first\_index[1] = (first\_index[1] + 1) % 6

            second\_index[1] = (second\_index[1] + 1) % 6

            encrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

        elif first\_index[1] == second\_index[1]:

            first\_index[0] = (first\_index[0] + 1) % 5

            second\_index[0] = (second\_index[0] + 1) % 5

            encrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

        else:

            first\_index[1], second\_index[1] = second\_index[1], first\_index[1]

            encrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

    return encrypted\_text  *# Возврат шифртекста*

def playfair\_decrypt(encrypted\_text, keyword, alphabet):

    decrypted\_text = ""  *# Расшифрованный текст*

*# Формирование алфавита с учетом ключевого слова*

    unprepared\_alphabet = list(keyword) + [char for char in alphabet if char not in keyword]

    unprepared\_alphabet = list(dict.fromkeys(unprepared\_alphabet))

    while len(unprepared\_alphabet) < 30:

        unprepared\_alphabet.append('\_')

    new\_alphabet = [unprepared\_alphabet[i:i+6] for i in range(0, len(unprepared\_alphabet), 6)]

    for i in range(0, len(encrypted\_text), 2):  *# Расшифрование*

        first\_index = get\_alphabet\_index(encrypted\_text[i], new\_alphabet)

        second\_index = get\_alphabet\_index(encrypted\_text[i + 1], new\_alphabet)

        if first\_index[0] == second\_index[0]:

            first\_index[1] = (first\_index[1] - 1 + 6) % 6

            second\_index[1] = (second\_index[1] - 1 + 6) % 6

            decrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

        elif first\_index[1] == second\_index[1]:

            first\_index[0] = (first\_index[0] - 1 + 5) % 5

            second\_index[0] = (second\_index[0] - 1 + 5) % 5

            decrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

        else:

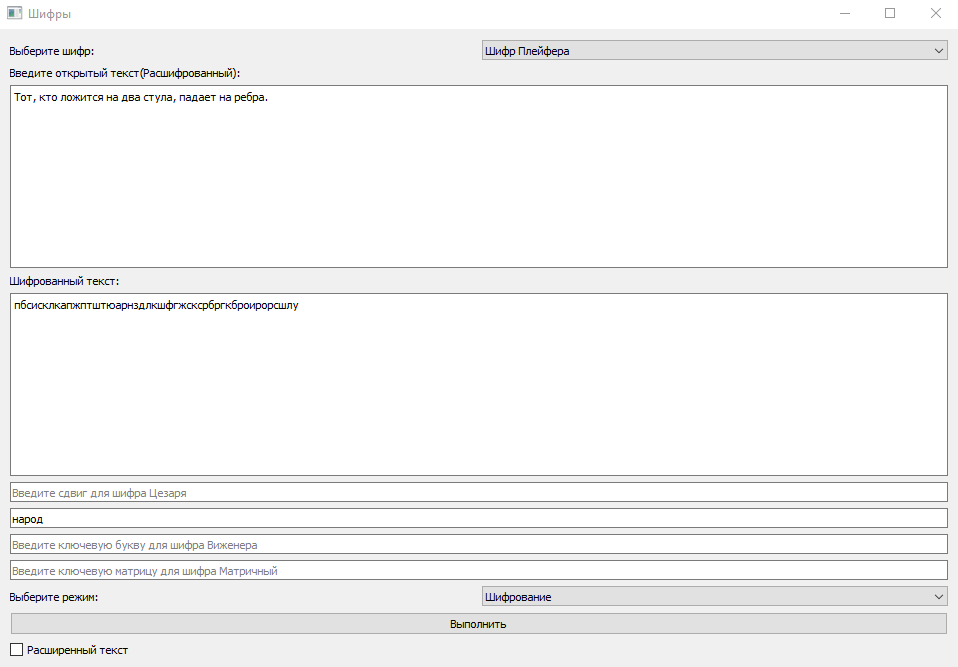
            first\_index[1], second\_index[1] = second\_index[1], first\_index[1]

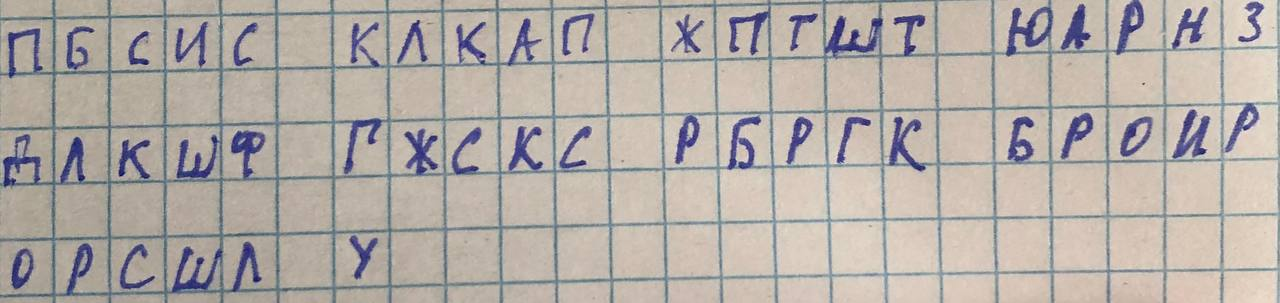
            decrypted\_text += new\_alphabet[first\_index[0]][first\_index[1]] + new\_alphabet[second\_index[0]][second\_index[1]]

    decrypted\_text = decrypted\_text.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace('прбл', ' ').replace('двтч', ':').replace('тчсзп', ';').replace('отскб', '(').replace('зкскб', ')').replace('впрзн', '?').replace('восклзн', '!').replace('првст', '\n')

    return decrypted\_text  *# Возврат расшифрованного текста*

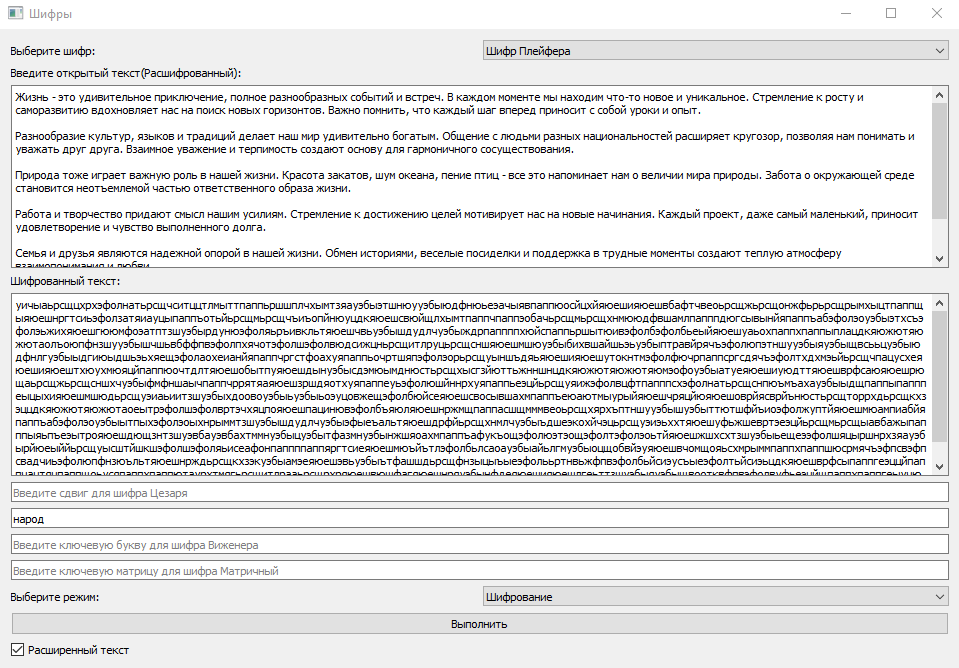
**Тестирование**





**Работа с текстом не менее 1000 знаков**

**Зашифрование**



**Расшифрование**

