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 1.4. Шифры перестановки

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
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_check_parameters,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_check_parameters, vertical_transposition_encrypt, vertical_transposition_decrypt
from cardanosGrid import cardanosGridCheckParameters, cardanosGridEncrypt, cardanosGridDecrypt
from feistelsNetwork import feistelsNetworkCheckParameters, feistelsNetwork

available_ciphers = [
    "Шифр АТБАШ", "Шифр Цезаря", "Шифр Полибия",
    "Шифр Тритемия", "Шифр Белазо", "Шифр Виженера", "МАГМА(s_block)",
    "Шифр Матричный", "Шифр Плейфера", "Шифр вертикальной перестановки",
    "Шифр решетка Кардано", "Шифр сеть Фейстель",
]

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, Плейфера, Вертикальной
перестановки')

        # 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 размерности решётки для шифра Кардано
        self.cardanosGrid_edit = QLineEdit()
        self.cardanosGrid_edit.setPlaceholderText('Введите размерность решётки для шифра Кардано (8 8)')

        # Ввод решётки для шифра Кардано
        self.cardanosGridGRIDGRID_edit = QLineEdit()
        self.cardanosGridGRIDGRID_edit.setPlaceholderText('Решётка (0 1, 1 6, 2 1, 2 2, 2 4, 3 6, 3 7, 4 3, 4 5, 5 0, 6
3, 6 5, 6 7, 7 2, 7 4, 7 7)')

        # Ввод решётки для шифра сеть Фейстеля
        self.feistelsNet_edit = QLineEdit()
        self.feistelsNet_edit.setPlaceholderText('Ключ для шифра сети Фейстеля')

        # 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.cardanosGrid_edit)
        layout.addWidget(self.cardanosGridGRIDGRID_edit)
        layout.addWidget(self.feistelsNet_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()
        cardanosGridSizeInput = self.cardanosGrid_edit.text()
        cardanosGridGRIDInput = self.cardanosGridGRIDGRID_edit.text()
        feistelsNetworkInput = self.feistelsNet_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":
                if s_block_check_parameters(open_text_input, alphabet_sblock):
                    cipher_text_input = s_block_encrypt(open_text_input, alphabet_sblock)
                else:
                    cipher_text_input = "Проверьте правильность ввода ключей"
            elif mode == "decrypt":
                if s_block_check_parameters(cipher_text_input, alphabet_sblock):
                    open_text_input = s_block_decrypt(cipher_text_input, alphabet_sblock)
                else:
                    open_text_input = "Проверьте правильность ввода ключей"
        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 == "Шифр вертикальной перестановки":
            keyword = self.keyword_edit.text()
            if keyword:
                if vertical_transposition_check_parameters(keyword, alphabet):
                    if mode == "encrypt":
                        cipher_text_input = vertical_transposition_encrypt(self.text_preparation(open_text_input),
keyword, alphabet)
                    elif mode == "decrypt":
                         open_text_input = vertical_transposition_decrypt(cipher_text_input, keyword, alphabet)
                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 вертикальной перестановки"
                elif mode == "decrypt":
                    open_text_input = "Введите ключевое слово для шифра вертикальной перестановки"
        elif cipher_choose_input == "Шифр решетка Кардано":
            GridSizeInput = list(map(int, cardanosGridSizeInput.split(" ")))
            GridCardano = [list(map(int, x.split(" "))) for x in cardanosGridGRIDInput.split(", ")]
            if cardanosGridCheckParameters(GridSizeInput, GridCardano):
                if mode == "encrypt":
                    cipher_text_input = cardanosGridEncrypt(self.text_preparation(open_text_input), GridSizeInput,
GridCardano, alphabet)
                elif mode == "decrypt":
```

```
                    open_text_input = cardanosGridDecrypt(cipher_text_input, GridSizeInput, GridCardano, alphabet)
            else:
                if mode == "encrypt":
                    cipher_text_input = "Проверьте правильность ввода решетки"
                elif mode == "decrypt":
                    open_text_input = "Проверьте правильность ввода решетки"
        elif cipher_choose_input == "Шифр сеть Фейстель":
            if mode == "encrypt":
                if feistelsNetworkCheckParameters(open_text_input, feistelsNetworkInput, alphabet_sblock):
                    cipher_text_input = feistelsNetwork(open_text_input, feistelsNetworkInput, mode, alphabet_sblock)
                else:
                    cipher_text_input = "Проверьте правильность ввода ключей"
            elif mode == "decrypt":
                if feistelsNetworkCheckParameters(cipher_text_input, feistelsNetworkInput, alphabet_sblock):
                    open_text_input = feistelsNetwork(cipher_text_input, feistelsNetworkInput, mode, alphabet_sblock)
                else:
                    open_text_input = "Проверьте правильность ввода ключей"
        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_())
```

10. Вертикальная перестановка

Шифр вертикальной перестановки – это метод шифрования, который основан на перестановке символов в исходном тексте в соответствии с определенным ключом. В этом шифре символы исходного текста записываются по строкам, а затем считываются по столбцам в порядке, заданном ключом

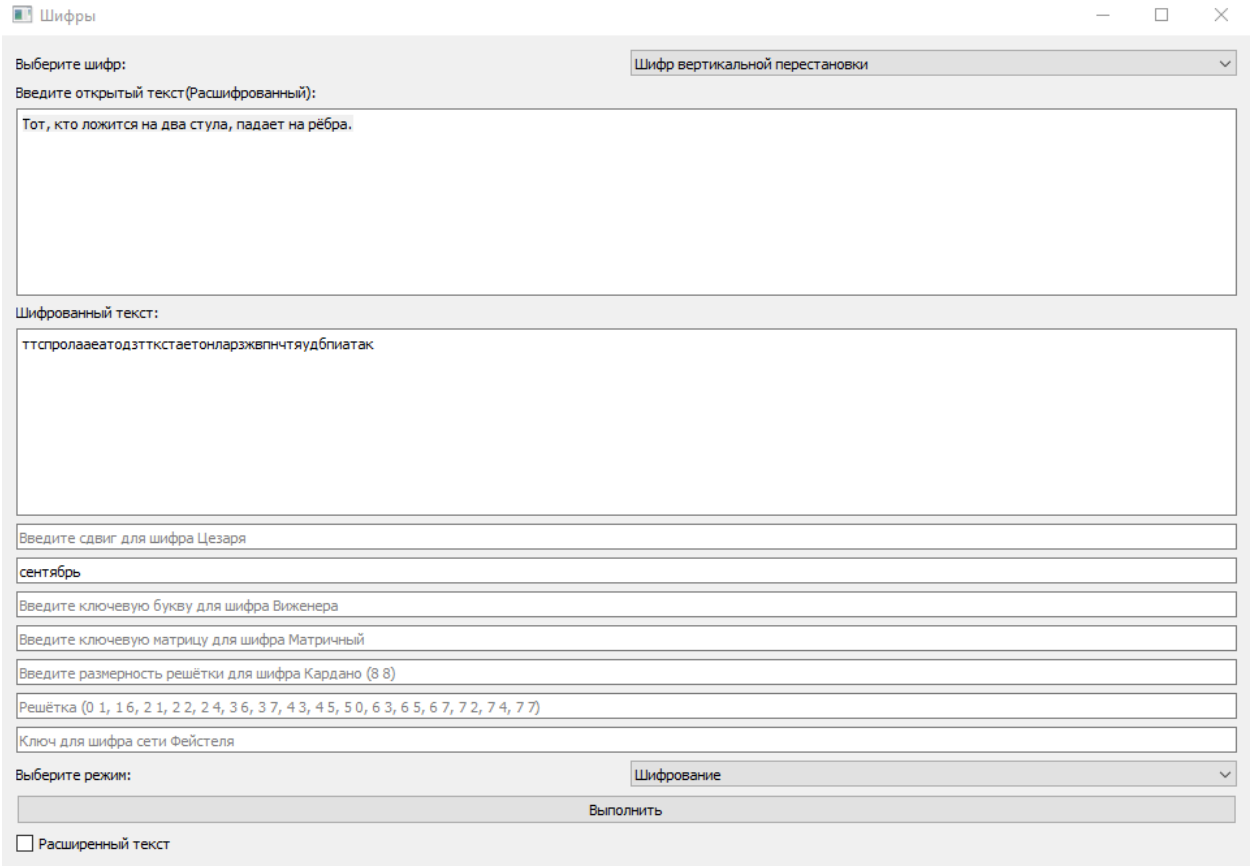
Блок-схема программы

Код программы с комментариями

```
def vertical_transposition_check_parameters(keyword, alphabet):
    for letter in keyword:
        if letter not in alphabet:
            return False  # Буквы ключевого слова отсутствуют в алфавите
    return True

def vertical_transposition_encrypt(open_text, keyword, alphabet):
    for letter in open_text:
        if letter not in alphabet:
            return "Введенный текст содержит запрещенные символы"  # Текстовые буквы не входят в алфавит
    # print("encrypted_text", keyword)
    # print("sorted_keyword", alphabet)
    encrypted_text = ""  # Шифрованный текст
    sorted_keyword = sorted(keyword)
    key = []
    for letter in keyword:  # От ключевого слова к цифровой клавише
        key.append(sorted_keyword.index(letter) + 1)
        sorted_keyword[sorted_keyword.index(letter)] = ""
    cutting_open_text = open_text
    open_text_array = []
    for i in range(len(open_text) // len(keyword)):  # Заполнение матрицы текстом
        open_text_array.append(cutting_open_text[:len(keyword)])
        cutting_open_text = cutting_open_text[len(keyword):]
    open_text_array.append(cutting_open_text)
    print("open_text_array", open_text_array)
    encrypted_text_array = [""] * (len(keyword) + 1)
    for i in range(len(keyword)):  # Заполнение массива путем чтения столбцов
        for row in open_text_array:
            if i < len(row) and row[i]:
                encrypted_text_array[key[i]] += row[i]
    encrypted_text = "".join(encrypted_text_array)
```

```
    print("encrypted_text", encrypted_text)
    return encrypted_text  # Возвращение шифрованного текста

def vertical_transposition_decrypt(encrypted_text, keyword, alphabet):
    for letter in encrypted_text:
        if letter not in alphabet:
            return "Введенный текст содержит запрещенные символы"  # Текстовые буквы не входят в алфавит
    decrypted_text = ""  # Расшифрованный текст
    sorted_keyword = sorted(keyword)
    key = []
    for letter in keyword:  # От ключевого слова к цифровой клавише
        key.append(sorted_keyword.index(letter) + 1)
        sorted_keyword[sorted_keyword.index(letter)] = ""
    cutting_encrypted_text = encrypted_text
    encrypted_text_array = [""] * (len(keyword) + 1)
    for i in range(1, len(keyword) + 1):  # Заполнение матрицы текстом
        if key.index(i) < len(encrypted_text) % len(keyword):
            encrypted_text_array[key.index(i)] = cutting_encrypted_text[:len(encrypted_text) // len(keyword) + 1]
            cutting_encrypted_text = cutting_encrypted_text[len(encrypted_text) // len(keyword) + 1:]
        else:
            encrypted_text_array[key.index(i)] = cutting_encrypted_text[:len(encrypted_text) // len(keyword)]
            cutting_encrypted_text = cutting_encrypted_text[len(encrypted_text) // len(keyword):]
    for i in range(len(encrypted_text) // len(keyword) + 1):
        for row in encrypted_text_array:
            if i < len(row) and row[i]:
                decrypted_text += row[i]
    # Перевод символов из текстовых значений в символы
    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  # Возвращение расшифрованного текста
```

Тестирование



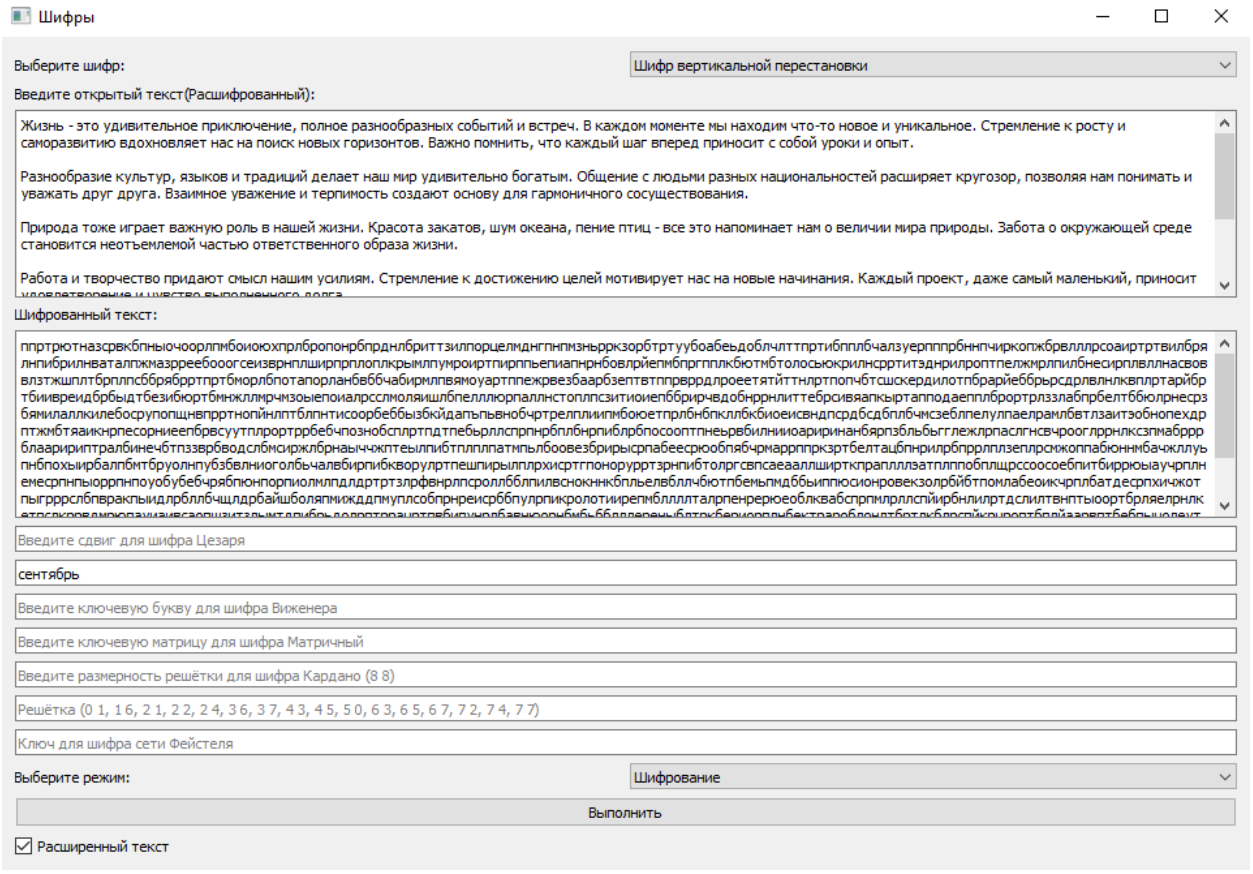
Вертикальная перестановка

Построение частотный гистограммы `Вертикальная перестановка`

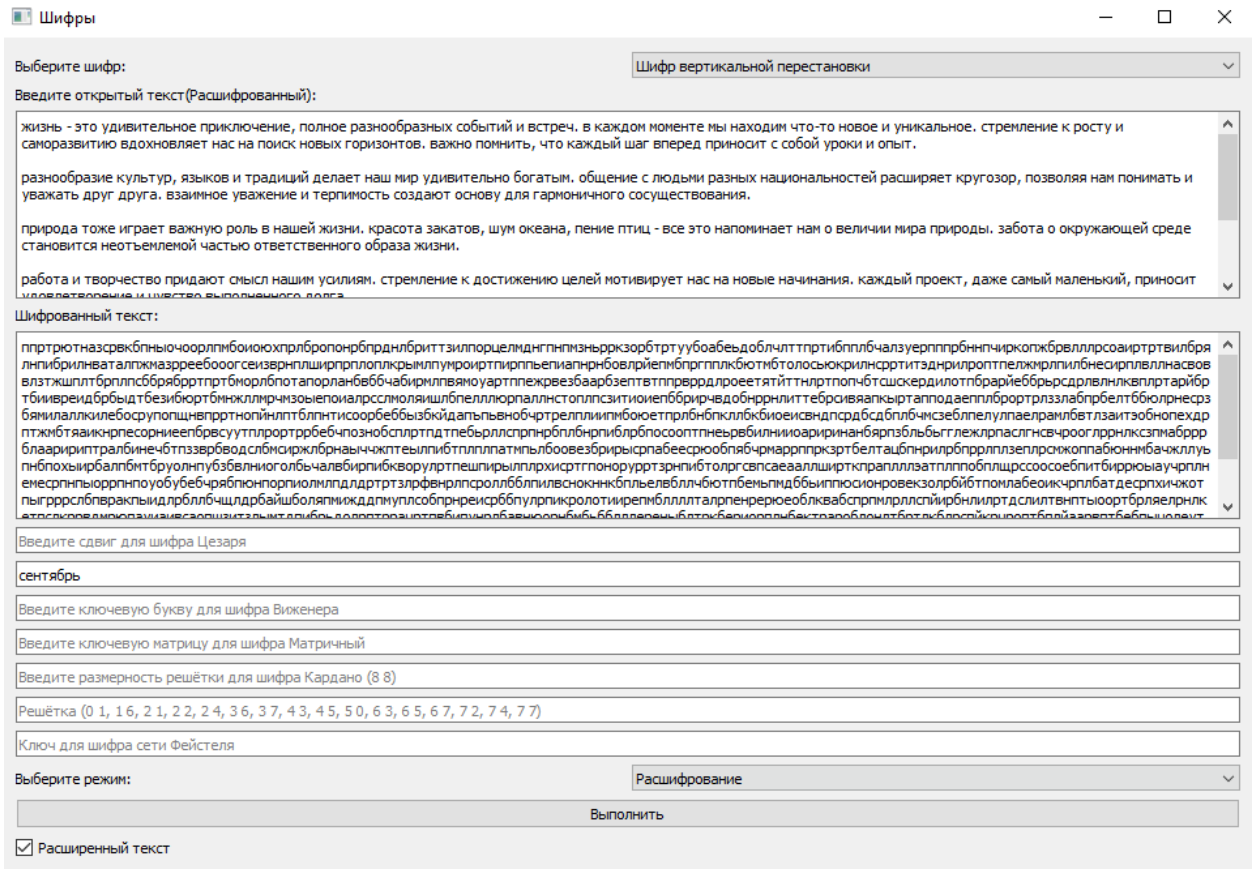
```
create_histogram("ТотЗПТктоложитсянадвастулаЗПТпадаетнаребраТЧК", 1, "Открытый текст")

create_histogram("ттспролаеатодзттккстаетонларжвпнчтяудбпиатак", 2, "Вертикальная перестановка")
```

Работа с текстом не менее 1000 знаков

Зашифрование



Расшифрование



11. Решетка Кардано

Решетка Кардано - частный случай шифра маршрутной перестановки. используется трафарет из прямоугольного листа клетчатой бумаги размером 2m x 2k клеток. В трафарете вырезано m x k клеток так, что при наложении его на чистый лист бумаги того же размера четырьмя возможными способами его вырезы полностью покрывают всю площадь листа. Буквы сообщения последовательно вписываются в вырезы трафарета (по строкам, в каждой строке слева направо) при каждом из четырех его возможных положений в заранее установленном порядке. Пустые клетки, если шифруемый текст не кратен размеру решетки, заполняются «пустышками» (случайными буквами). Шифртекст выписывается построчно, независимо от размеров решетки.

Блок-схема программы

Код программы с комментариями

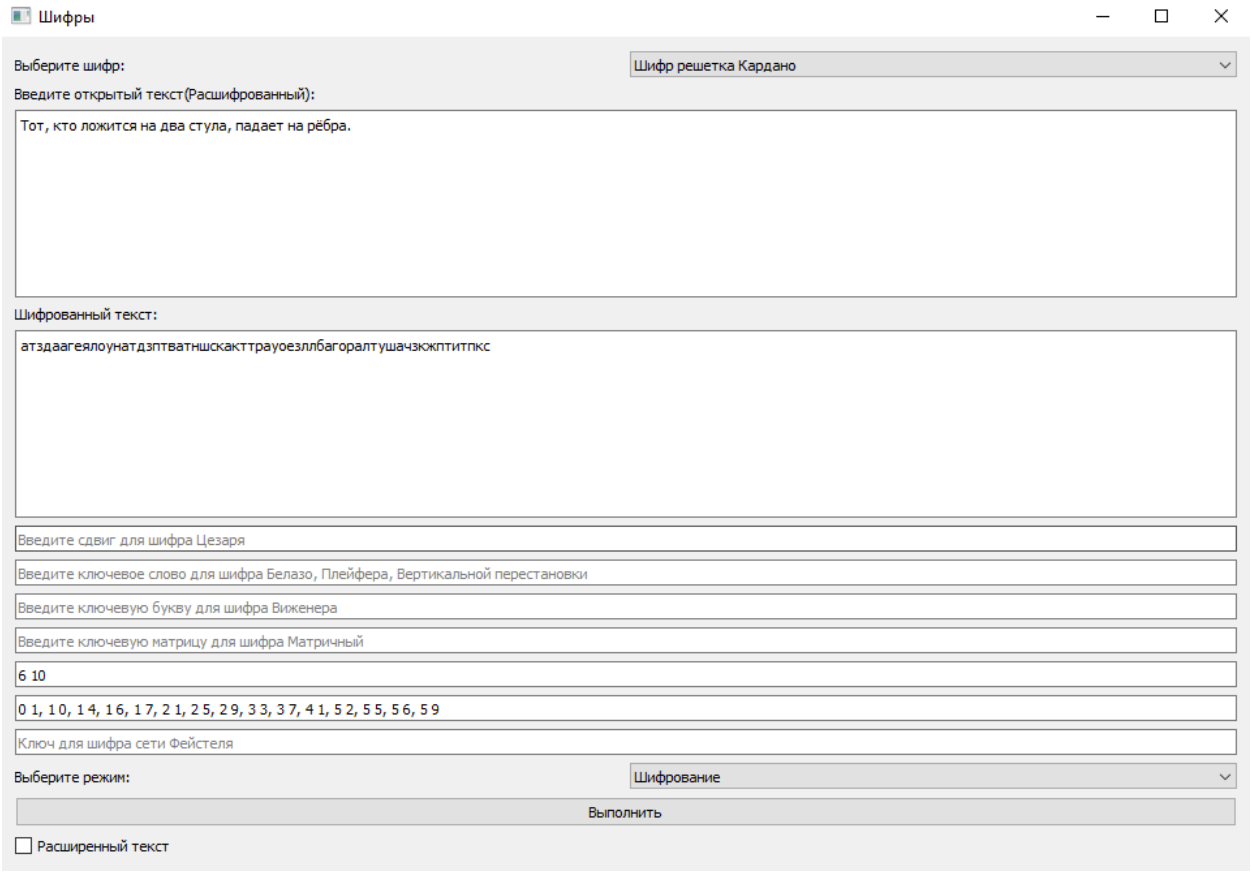
```
import random

def verticalize(GridSizeInput, GridCardano):
    newGrid = []
    for elem in GridCardano:
        newGrid.append([GridSizeInput[0] - elem[0] - 1, elem[1]])
    newGrid.sort()
    return newGrid

def horizontalize(GridSizeInput, GridCardano):
    newGrid = []
    for elem in GridCardano:
        newGrid.append([elem[0], GridSizeInput[1] - elem[1] - 1])
    newGrid.sort()
    return newGrid

def cardanosGridCheckParameters(GridSizeInput, GridCardano):
    def isArrayInArray(arr, item):
        return item in arr

    if len(GridSizeInput) != 2 or not all(GridSizeInput):
        return False

    if GridSizeInput[0] % 2 != 0 or GridSizeInput[1] % 2 != 0:
        return False

    fst = GridCardano
    snd = horizontalize(GridSizeInput, fst)
    trd = verticalize(GridSizeInput, snd)
    fth = horizontalize(GridSizeInput, trd)
```

```
    for elem in fst:
        if isArrayInArray(snd, elem) or isArrayInArray(trd, elem) or isArrayInArray(fth, elem):
            return False

    for elem in snd:
        if isArrayInArray(fst, elem) or isArrayInArray(trd, elem) or isArrayInArray(fth, elem):
            return False

    for elem in trd:
        if isArrayInArray(fst, elem) or isArrayInArray(snd, elem) or isArrayInArray(fth, elem):
            return False

    for elem in fth:
        if isArrayInArray(fst, elem) or isArrayInArray(snd, elem) or isArrayInArray(trd, elem):
            return False

    return True

def cardanosGridEncrypt(openText, GridSizeInput, GridCardano, alphabet):
    for letter in openText:
        if letter not in alphabet:
            return "Введёный текст содержит запрещённые символы"

    encryptedText = ""

    if len(openText) < GridSizeInput[0] * GridSizeInput[1]:
        openText = (openText + "заглушка" * ((GridSizeInput[0] * GridSizeInput[1] - len(openText)) // 8 +
1))[:GridSizeInput[0] * GridSizeInput[1]]
    else:
        openText = openText[:GridSizeInput[0] * GridSizeInput[1]]

    encryptedTextGrid = [["" for _ in range(GridSizeInput[1])] for _ in range(GridSizeInput[0])]
```

```
    def fillTextGrid():
        nonlocal cuttedText
        for coord in GridCardano:
            encryptedTextGrid[coord[0]][coord[1]] = cuttedText[0]
            cuttedText = cuttedText[1:]

    cuttedText = openText
    fillTextGrid()
    GridCardano = horizontalize(GridSizeInput, GridCardano)
    fillTextGrid()
    GridCardano = verticalize(GridSizeInput, GridCardano)
    fillTextGrid()
    GridCardano = horizontalize(GridSizeInput, GridCardano)
    fillTextGrid()

    encryptedText = "".join("".join(row) for row in encryptedTextGrid)
    return encryptedText

def cardanosGridDecrypt(encryptedText, GridSizeInput, GridCardano, alphabet):
    for letter in encryptedText:
        if letter not in alphabet:
            return "Введёный текст содержит запрещённые символы"

    decryptedText = ""
    decryptedTextGrid = [[encryptedText[i * GridSizeInput[1] + j] for j in range(GridSizeInput[1])] for i in
range(GridSizeInput[0])]

    def readTextGrid():
        nonlocal decryptedText
        for coord in GridCardano:
            decryptedText += decryptedTextGrid[coord[0]][coord[1]]
```

```
    readTextGrid()
    GridCardano = horizontalize(GridSizeInput, GridCardano)
    readTextGrid()
    GridCardano = verticalize(GridSizeInput, GridCardano)
    readTextGrid()
    GridCardano = horizontalize(GridSizeInput, GridCardano)
    readTextGrid()

    if "заглушка" in decryptedText:
        decryptedText = decryptedText[:decryptedText.index("заглушка")]

    decryptedText = decryptedText.replace("тчк", ".").replace("зпт", ",").replace("тире", "-").replace("прбл", "
").replace("двтч", ":").replace("тчксзпт", ";").replace("отскб", "(").replace("зкскб", ")").replace("впрзн",
"?").replace("восклзн", "!").replace("првст", "\n")
    return decryptedText
```

Тестирование



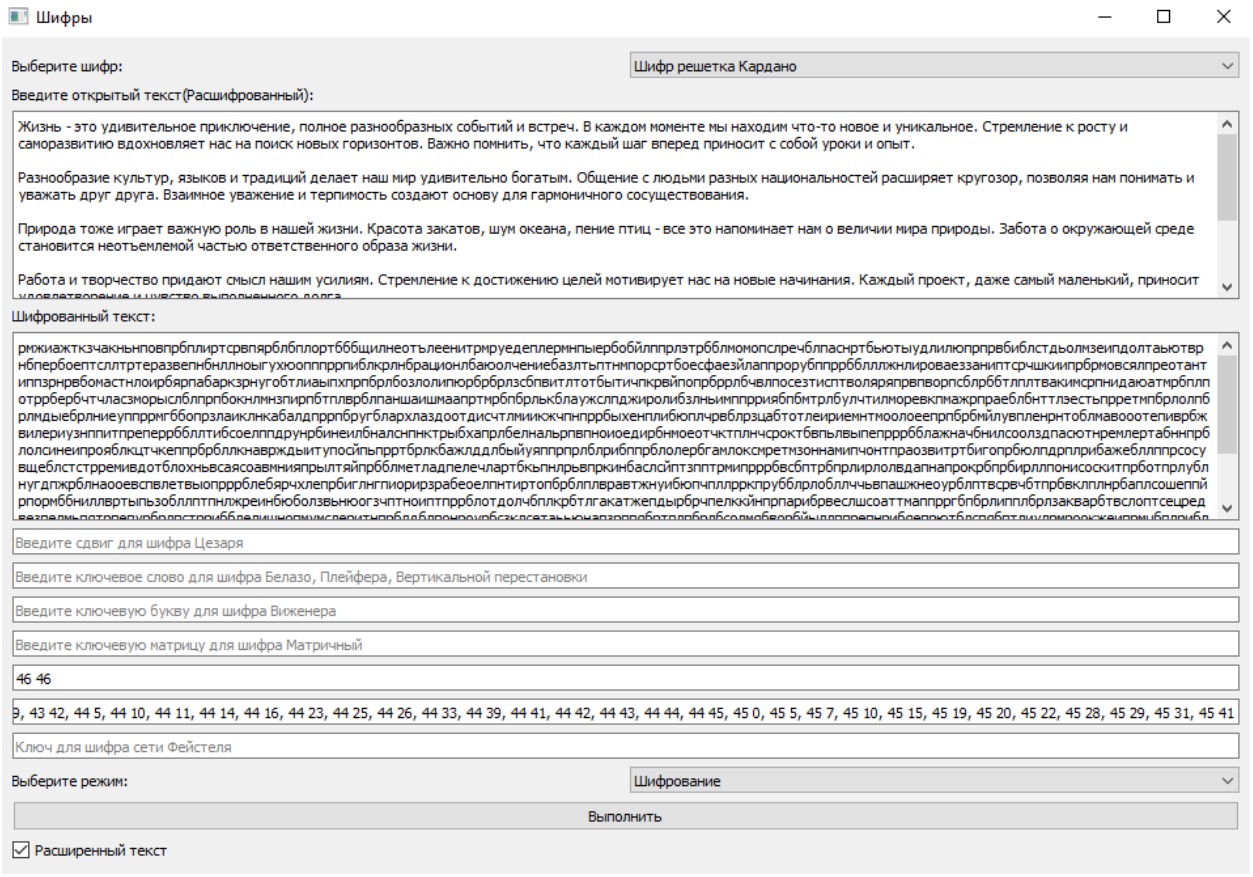
Решетка Кардано

Построение частотный гистограммы `Решетка Кардано`

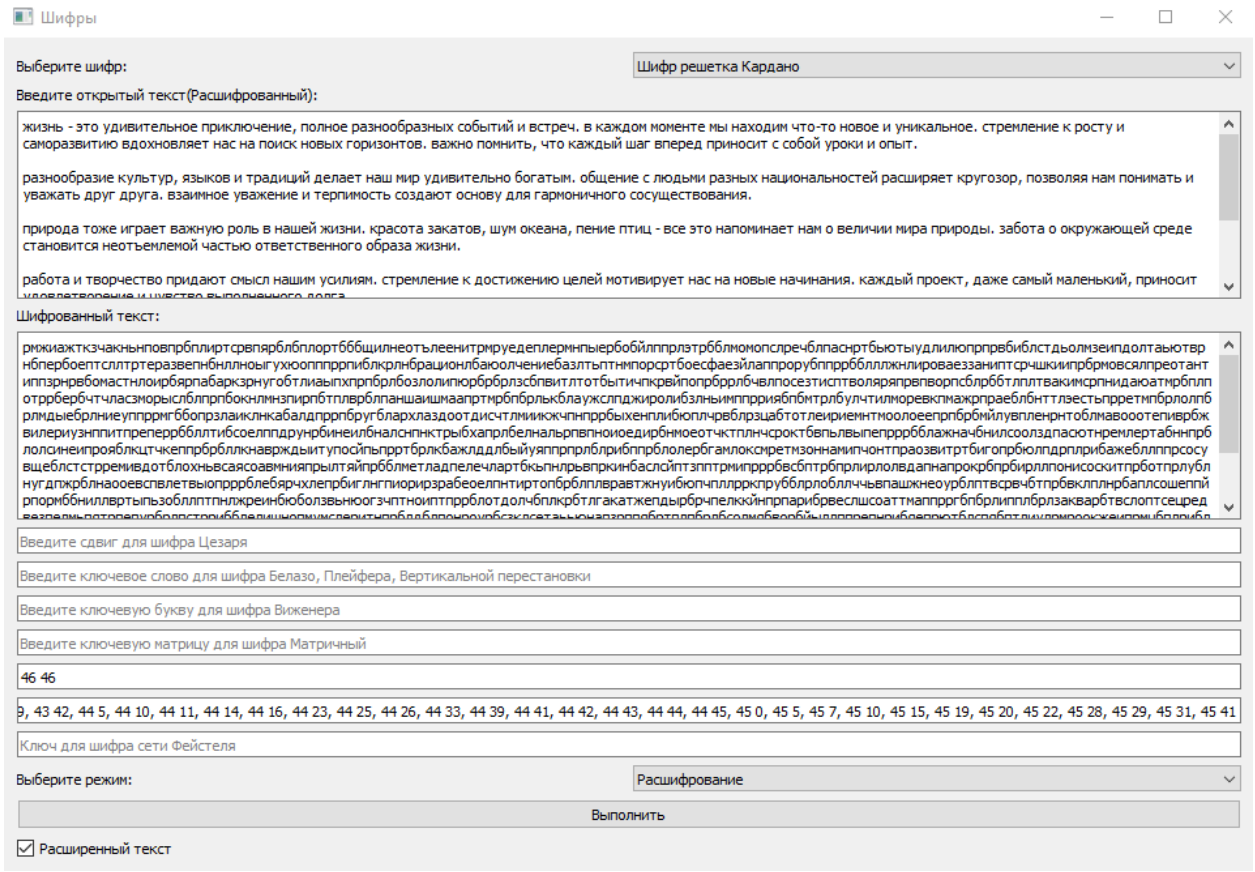
```
create_histogram("ТотЗПТктоложитсянадвастулаЗПТпадаетнаребраТЧК", 1, "Открытый текст")

create_histogram("атздаагеялоунатдзгтватншскакттрауоеэллбагоралтушачзкжптитпкс", 2, "Решётка кардано")
```

Работа с текстом не менее 1000 знаков

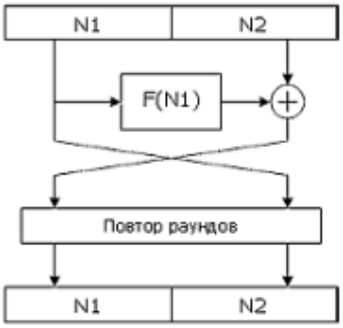
Зашифрование



Расшифрование



12. Перестановка в комбинационных шифрах (DES, МАГМА)



В их основе лежит ***сеть Фейстеля*** - разновидность блочного шифра, предложенная в **1971 году Хорстом Фейстелем**. Для зашифрования открытый текст сначала разбивается на левую и правую половины L и R. На i-ом цикле используется подключ k_i:

$R_{i+1} = L_i$

$L_{i+1} = R_i \oplus f(L_i, K_i)$

где ($\oplus$ = xor)

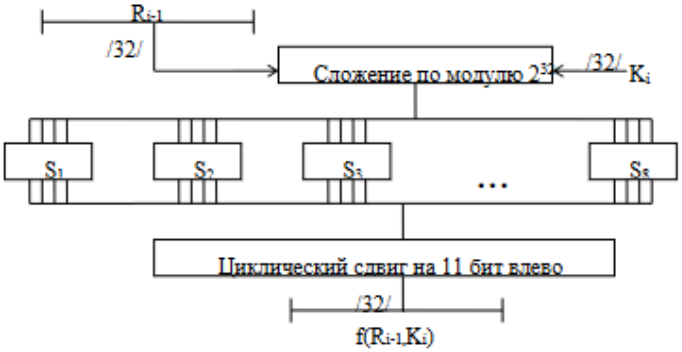
Для генерации подключей исходный 256-битный ключ разбивается на восемь 32-битных блоков: $K_1 \ldots K_8$.

Расшифрование выполняется так же, как и зашифрование, но инвертируется порядок подключей K_i.

Функция $f(L_i,K_i)$ вычисляется следующим образом:

R_{i-1} и K_i складываются по модулю 2^{32}.

Результат разбивается на восемь 4-битовых подпоследовательностей, каждая из которых поступает на вход своего S-блока.



Блок-схема функции шифрования f алгоритма ГОСТ 28147-89 и ГОСТ Р 34.12-2015 (Магма).

Код программы с комментариями

```
from S_block import s_block_encrypt

def addBinary(a, b):
    result = ""
    i = len(a) - 1
    j = len(b) - 1
    carry = 0
    while i >= 0 or j >= 0:
        sum = carry
        if i >= 0:
            sum += int(a[i]) - int('0')
            i -= 1
        if j >= 0:
            sum += int(b[j]) - int('0')
            j -= 1
        result = str(sum % 2) + result
        carry = int(sum / 2)
    if carry > 0:
        result = '1' + result
    return result

def feistelsNetworkCheckParameters(text, key, alphabet):
    if len(text) != 16:
        return False
    if len(key) != 64:
        return False
    for letter in text:
        if letter not in alphabet:
            return False
    for letter in key:
        if letter not in alphabet:
```

```
            return False
    return True

def feistelsNetwork(openText, key, mode, alphabet_sblock):
    def hex2bin(hex):
        ans = ""
        for letter in hex:
            ans += bin(int(letter, 16))[2:].zfill(4)
        return ans

    def bin2hex(bin):
        return hex(int(bin, 2))[2:]

    def f(rt, ki):
        sm = format((int(rt, 16) + int(ki, 16)) % (2 ** 32), '08x')
        sBlockOutput = s_block_encrypt(sm, alphabet_sblock)
        shift11 = hex2bin(sBlockOutput)[11:] + hex2bin(sBlockOutput)[:11]
        return shift11

    encryptedText = ""
    keys = [key[i:i+8] for i in range(0, len(key), 8)]
    revKeys = keys[::-1]
    keys = keys + keys + keys
    keys = keys + revKeys
    if mode == "decrypt":
        keys = keys[::-1]

    lt = openText[:8]
    rt = openText[8:]

    for key in keys:
        ff = f(rt, key)
```

```
        ltBin = ""
        for i in lt:
            ltBin += hex2bin(i)
        ffXorlt = ""
        for i in range(32):
            ffXorlt += str(int(ff[i]) ^ int(ltBin[i]))

        lt = rt
        rt = format(int(ffXorlt, 2), '08x')

    return rt + lt
```

Тестирование

В настоящем контрольном примере ключ имеет значение:

K = ffeeddccbbaa99887766554433221100f0f1f2f3f4f5f6f7f8f9fafbfcfdfeff.

А.2.4 Алгоритм зашифрования

В настоящем контрольном примере зашифрование производится при значениях итерационных ключей из А.2.3. Пусть открытый текст, подлежащий зашифрованию, равен

$$a = \text{fedcba9876543210},$$

Результатом зашифрования является шифртекст

$$b = G^*[K_{32}]G[K_{31}]\ldots G[K_1](a_1, a_0) = \text{4ee901e5c2d8ca3d}.$$