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 2. Шифрование шифрами многозначной 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.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":
            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 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 Плейфера"
                elif mode == "decrypt":
                    open_text_input = "Введите ключевое слово для шифра Плейфера"
        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_())
```

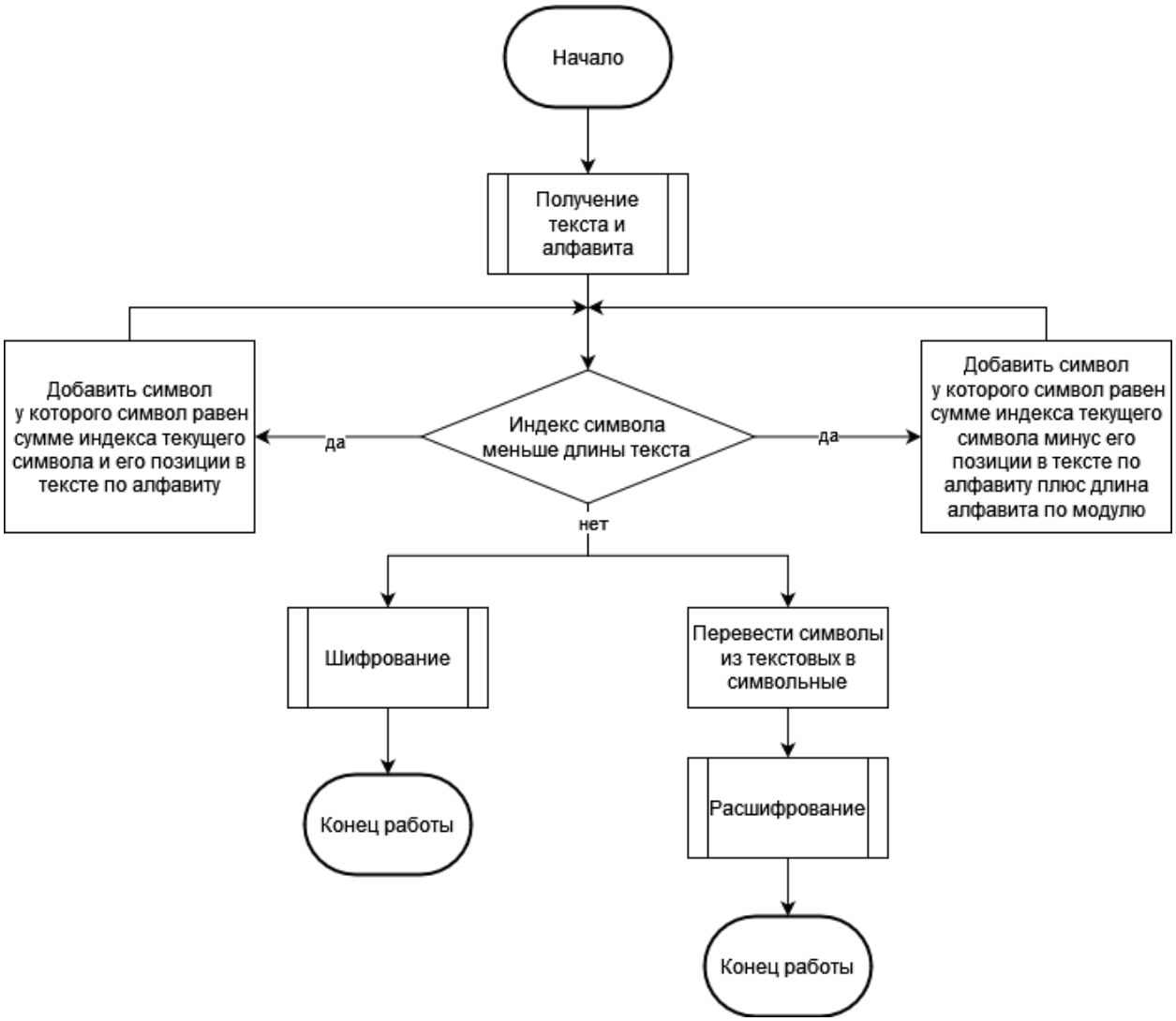
4. Шифр простой замены Тритемия

Шифр Тритемия — система шифрования, разработанная Иоганном Тритемием. Представляет собой усовершенствованный шифр Цезаря, то есть шифр подстановки.

$$Y_j = X_{i+j-1} \bmod n$$

X – исходный (открытый) текст
Y– зашифрованный текст
i – порядковый номер буквы в алфавите таблицы, i=1…n
j – порядковый номер буквы в тексте, j=1…k
k – количество букв в тексте
n – количество букв в выбранном алфавите (мощность алфавита).

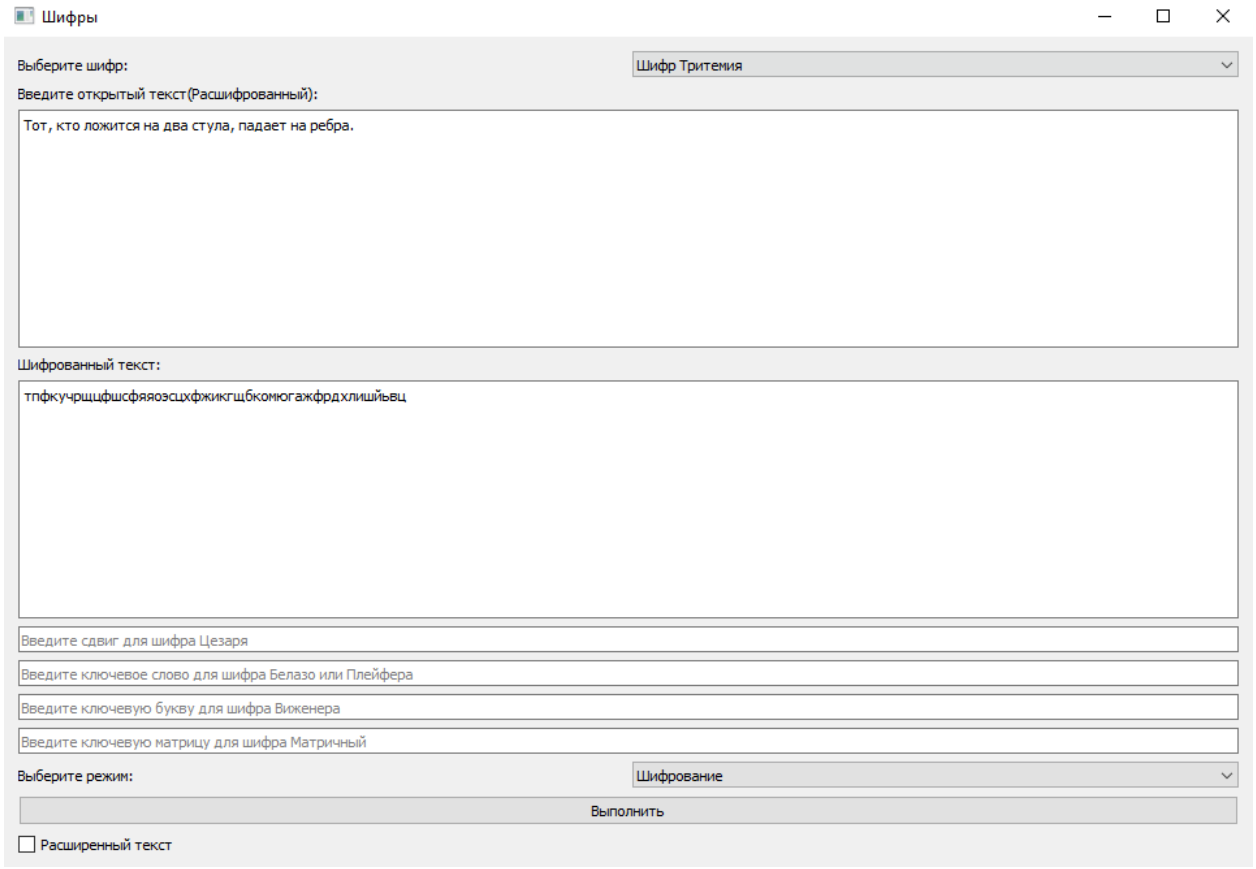
Блок-схема программы

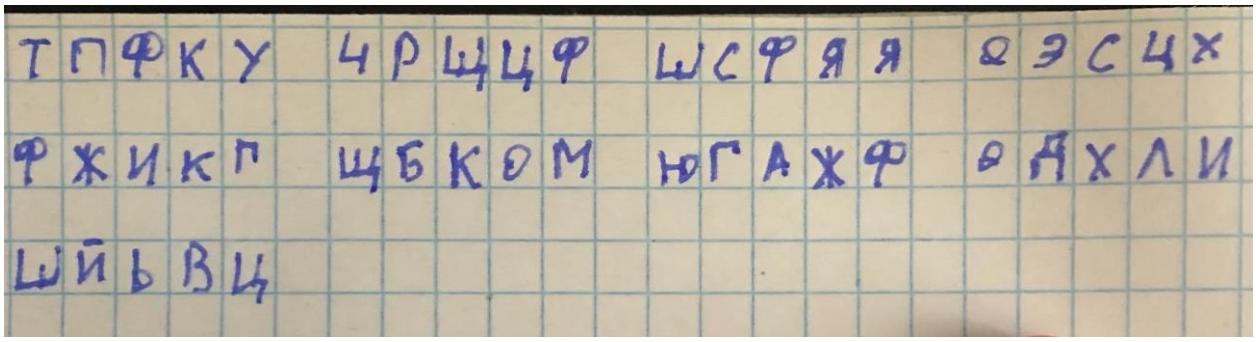


Код программы с комментариями

```
def tritemiy_encrypt(open_text, alphabet):
    encrypted_text = ""  # Шифртекст
    for i in range(len(open_text)):  # Проход по всем символам открытого текста
        element = open_text[i]  # Символ
        encrypted_text += alphabet[(alphabet.index(element) + i) % len(alphabet)]  # Добавление в итоговый шифртекст
зашифрованного символа
    return encrypted_text  # Возврат шифртекста

def tritemiy_decrypt(encrypted_text, alphabet):
    decrypted_text = ""  # Расшифрованный текст
    for i in range(len(encrypted_text)):  # Проход по всем символам шифртекста
        element = encrypted_text[i]  # Символ
        decrypted_text += alphabet[(alphabet.index(element) - i % len(alphabet) + len(alphabet)) % len(alphabet)]  #
Добавление в итоговый текст расшифрованного символа
    # Перевод символов из их текстовых значений в символьные
    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
```

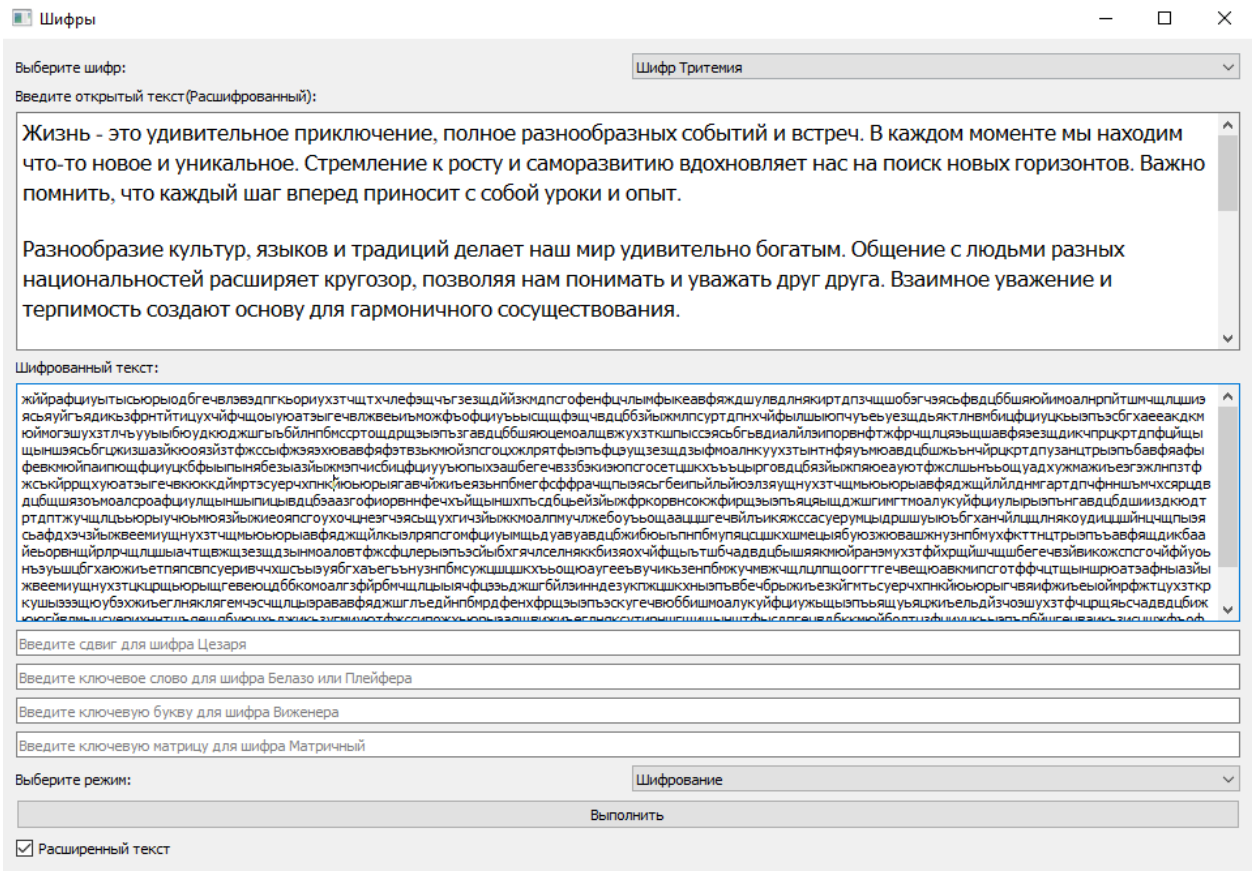
Тестирование



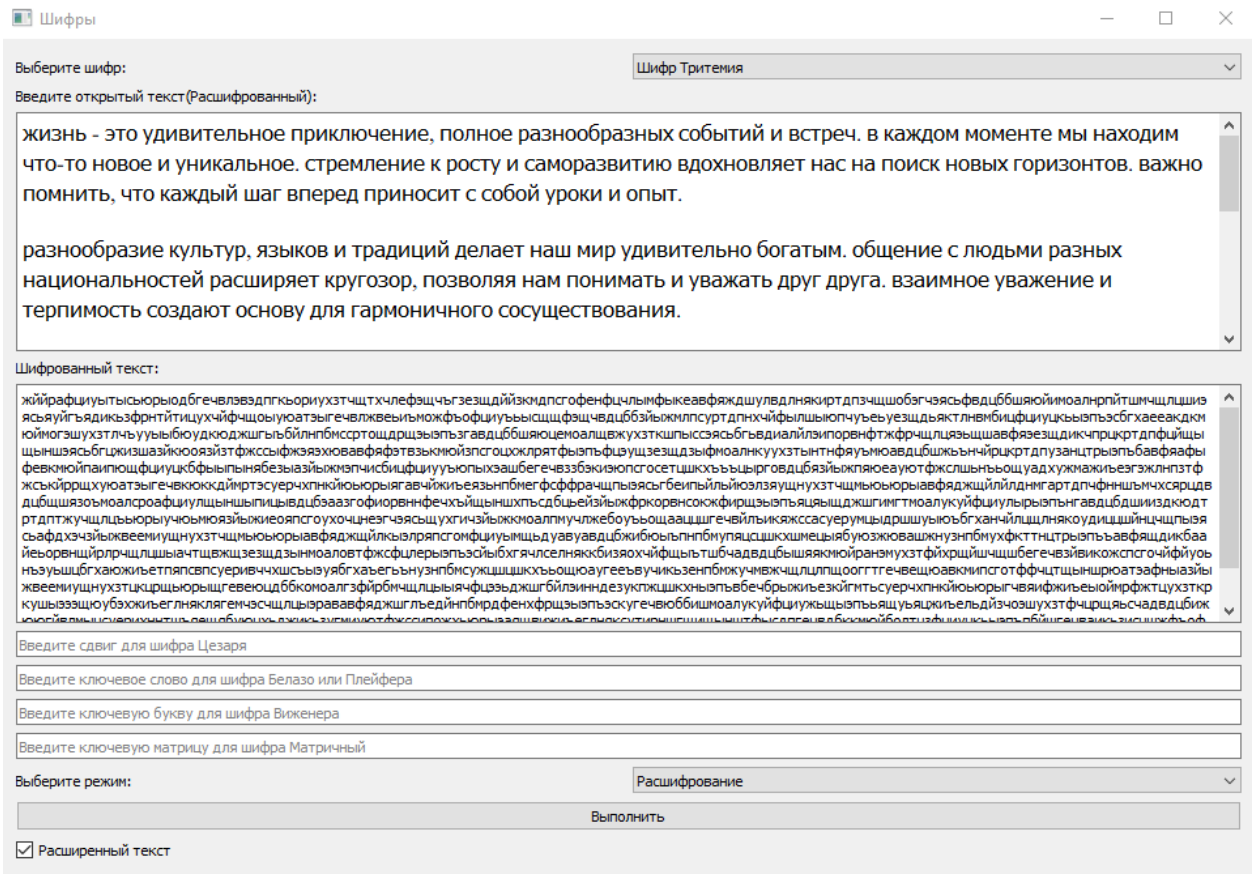


Работа с текстом не менее 1000 знаков

Зашифрование



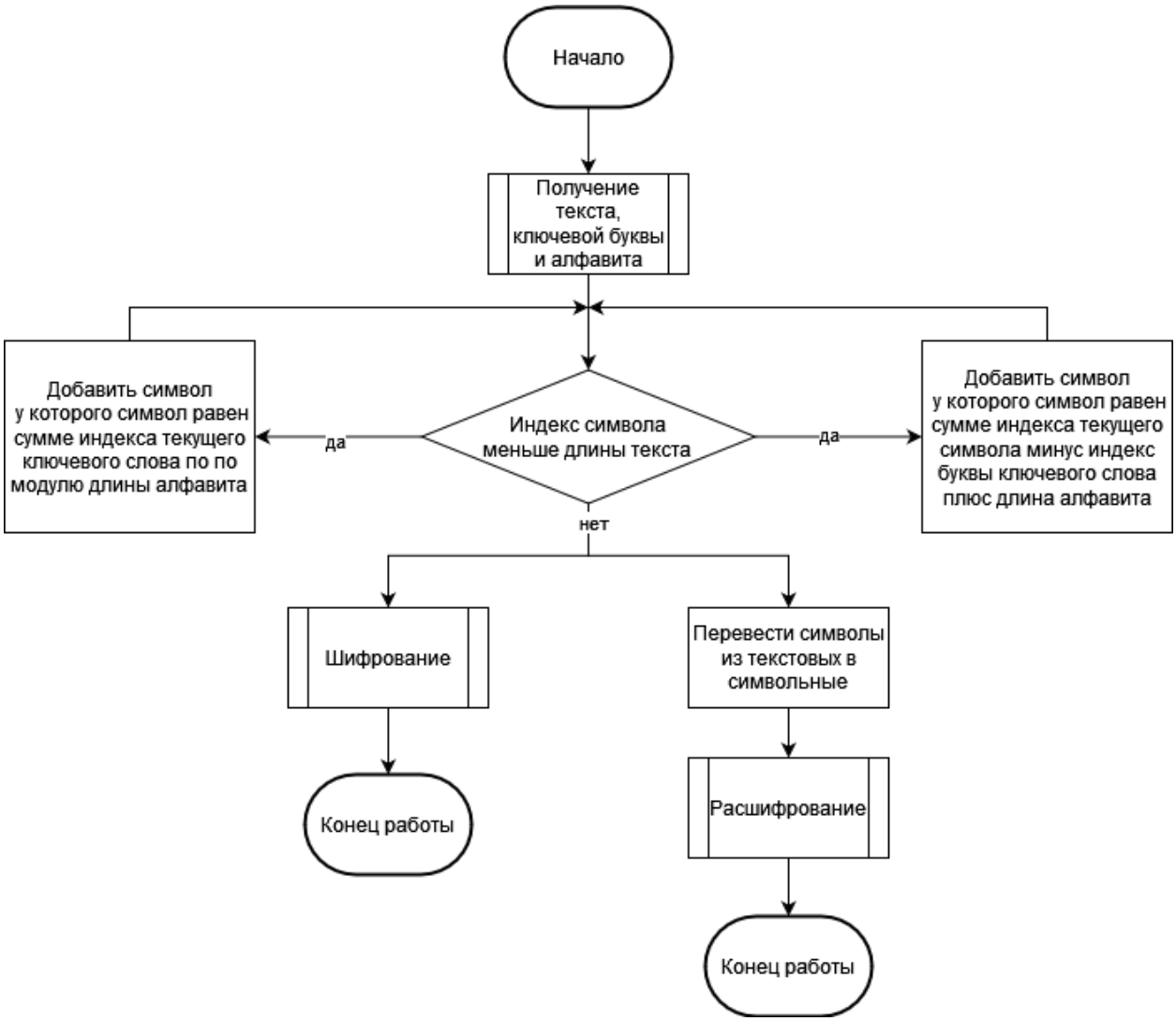
Расшифрование

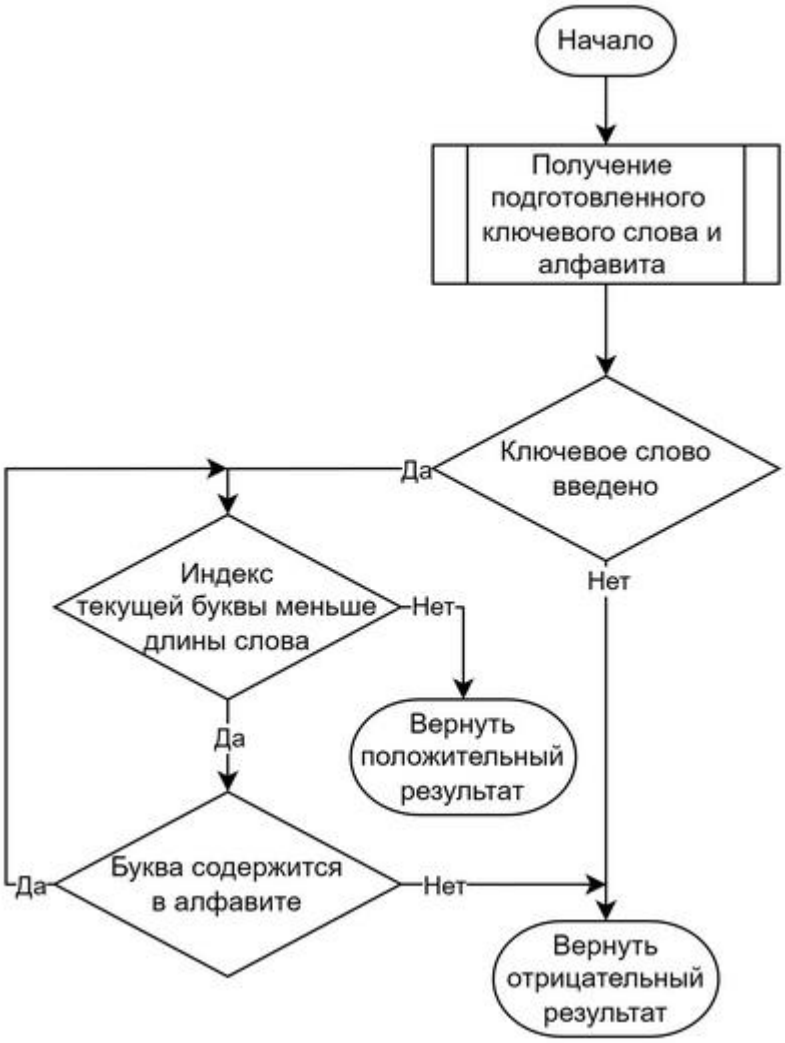


5. Шифр Белазо

Джованни Батиста Белазо в 1553 году (брошюра «Шифр синьора Белазо») предложил использовать для многоалфавитного шифра буквенный, легко запоминаемый ключ, который он назвал паролем. Шифрование осуществляется с помощью пароля-ключа, состоящего из М символов. Из полной таблицы Тритемия выделяется матрица ТШ размерностью [(М+1) x R]. Она включает первую строку и строки, первые элементы которых совпадают с символами ключа

Блок-схема программы

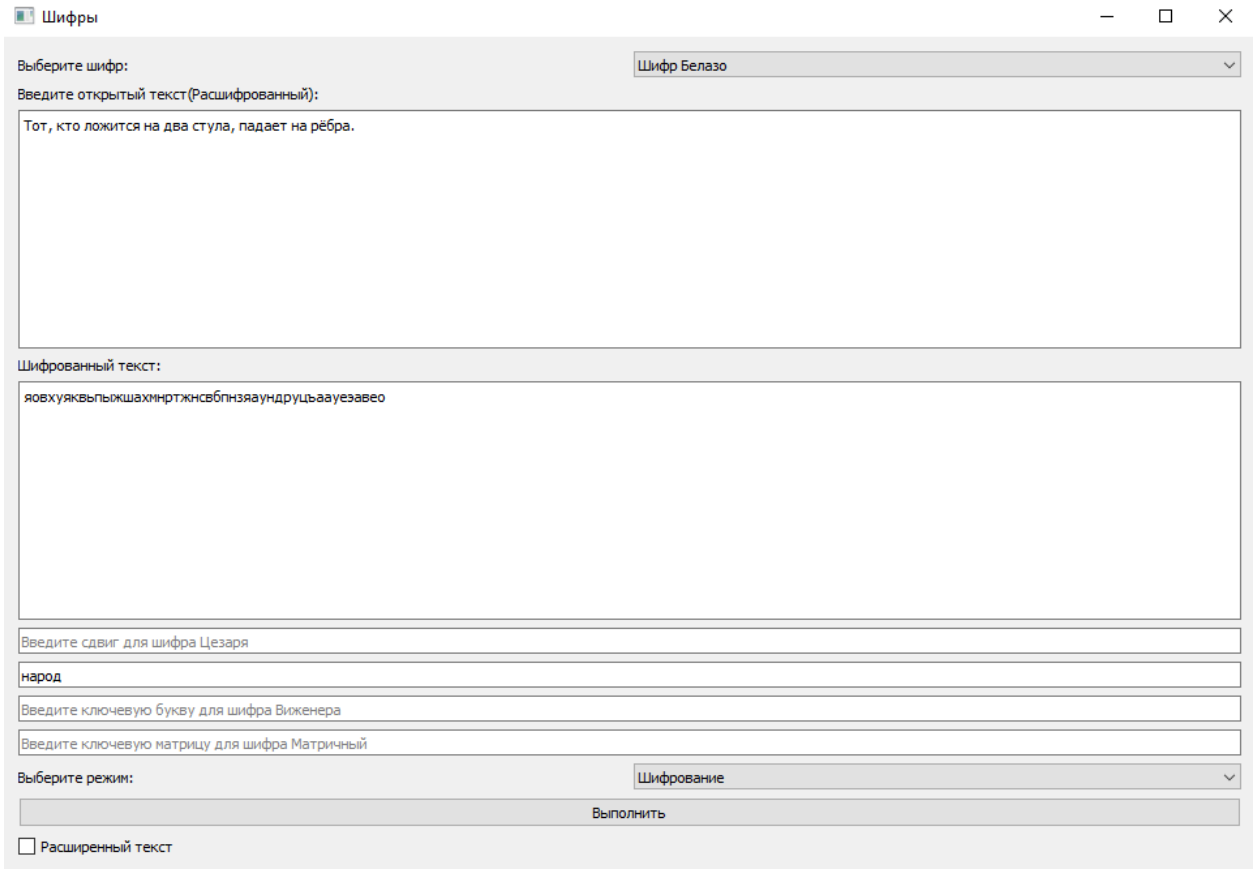


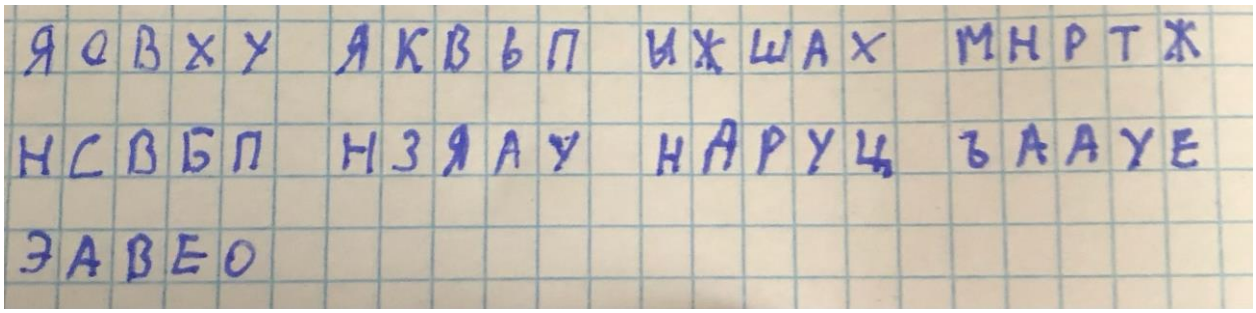

Начало
Получение подготовленного ключевого слова и алфавита
Ключевое слово введено
Да
Нет
Индекс текущей буквы меньше длины слова
Нет
Да
Вернуть положительный результат
Буква содержится в алфавите
Да
Нет
Вернуть отрицательный результат

Код программы с комментариями

```
def belazo_check_parameters(keyword, alphabet):
    if not keyword:
        return False  # Возврат лжи, если ключевое слово не введено
    for keyletter in keyword:
        if keyletter not in alphabet:
            return False  # Возврат лжи, если в ключевом слове присутствуют недопустимые символы
    return True  # Возврат истины, если ключевое слово соответствует требованиям

def belazo_encrypt(open_text, keyword, alphabet):
    encrypted_text = ""  # Шифртекст
    for i in range(len(open_text)):  # Проход по всем символам открытого текста
        element = open_text[i]  # Символ
        encrypted_text += alphabet[(alphabet.index(element) + alphabet.index(keyword[i % len(keyword)])) % len(alphabet)]  #
Добавление в итоговый шифртекст зашифрованного символа
    return encrypted_text  # Возврат шифртекста

def belazo_decrypt(encrypted_text, keyword, alphabet):
    decrypted_text = ""  # Расшифрованный текст
    for i in range(len(encrypted_text)):  # Проход по всем символам шифртекста
        element = encrypted_text[i]  # Символ
        decrypted_text += alphabet[(alphabet.index(element) - alphabet.index(keyword[i % len(keyword)]) + len(alphabet)) %
len(alphabet)]  # Добавление в итоговый текст расшифрованного символа
    # Перевод символов из их текстовых значений в символьные
    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
```

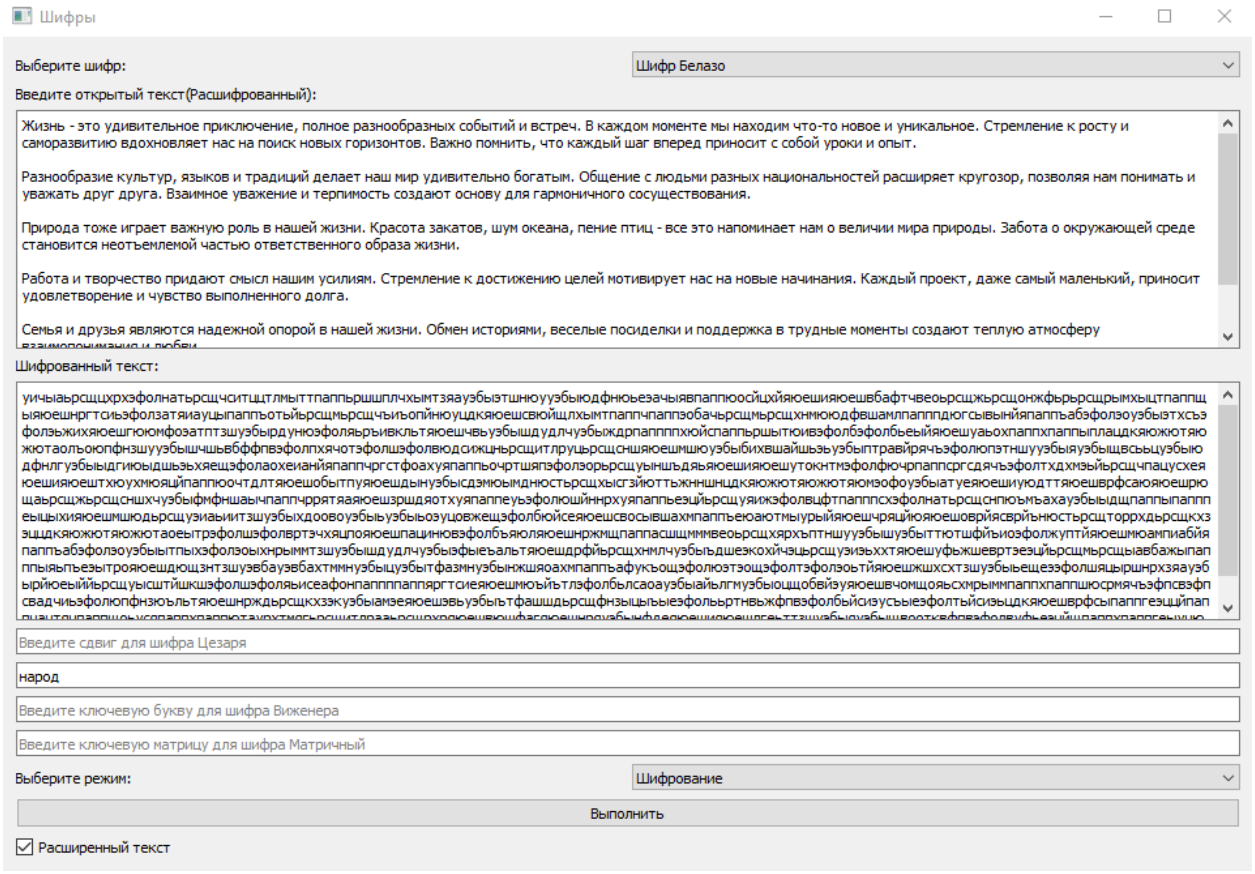
Тестирование



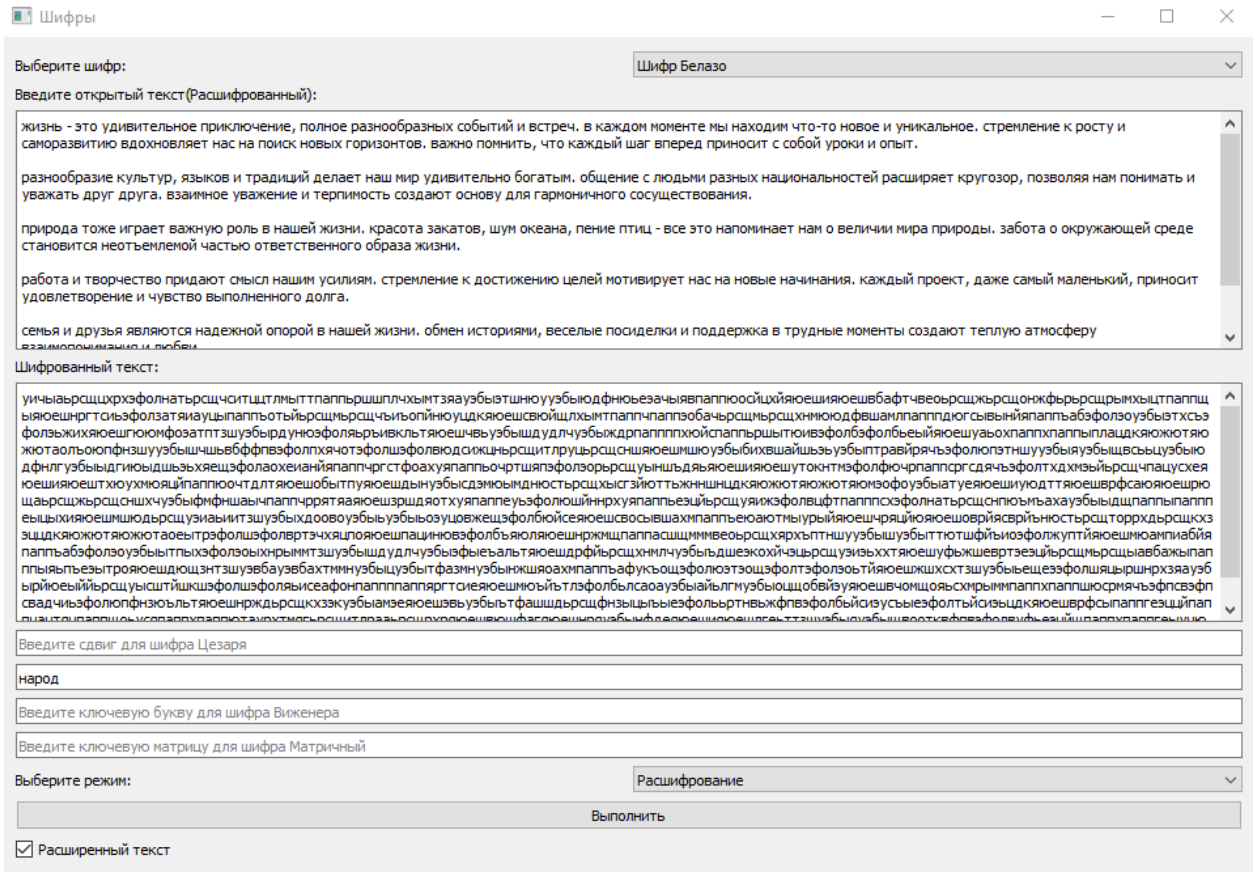


Работа с текстом не менее 1000 знаков

Зашифрование



Расшифрование



3. Шифр Виженера

В книге "Трактат о шифрах" Блез де Виженер описал два шифра, работающих по принципу шифра Белазо (см. выше), но в качестве пароля используется сам шифруемый текст с добавленной перед ним секретной буквой:

1.Шифр с самоключом

$$\begin{array}{rl} & \Gamma = t_0 t_1 t_2 \ldots t_{i-1} \ldots \\ + & T_O = t_1 t_2 t_3 \ldots t_i \ldots \\ \hline & T_ш = s_1 s_2 s_3 \ldots s_i \ldots \end{array}$$

T_O – открытый текст

Г - гамма, накладываемая на текст (сложение по модулю мощности алфавита)

$T_ш$ – шифртекст

t_0 - секретная буква-ключ

t_i, s_i – буквы используемого алфавита в тексте и шифртексте

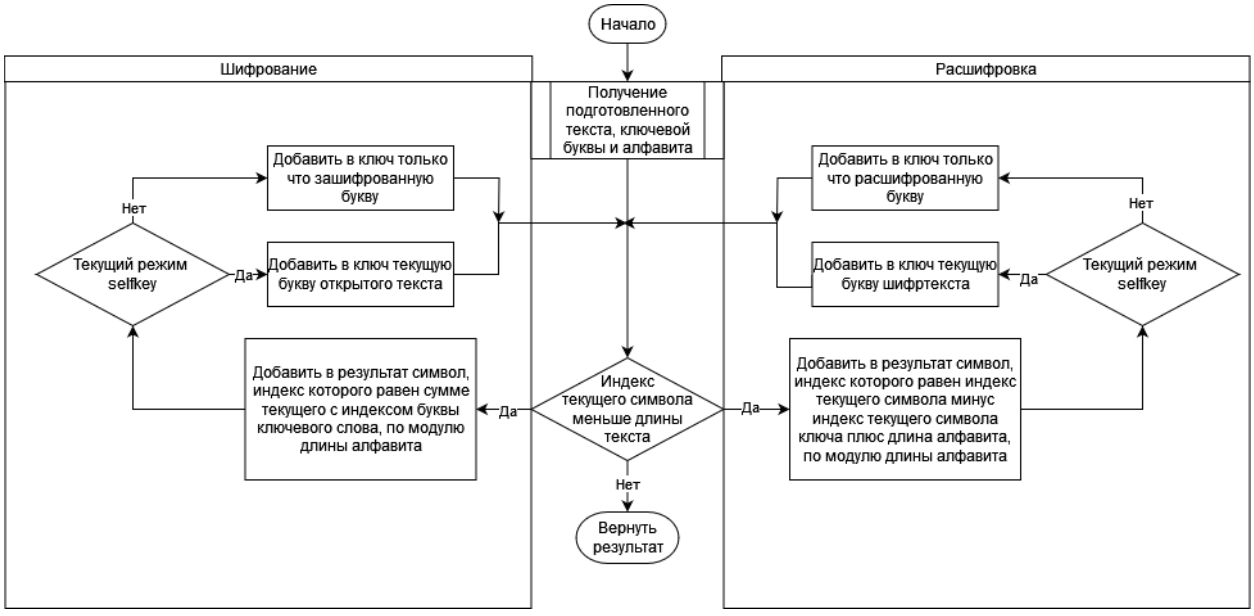
i – порядковый номер буквы в тексте или шифртексте.

2.Шифр ключом-шифртекстом

$$\begin{array}{rl} & \Gamma = s_0 s_1 s_2 \ldots s_{i-1} \ldots \\ + & T_O = t_1 t_2 t_3 \ldots t_i \ldots \\ \hline & T_ш = s_1 s_2 s_3 \ldots s_i \ldots \end{array}$$

s_0 - секретная буква-ключ

Блок-схема программы



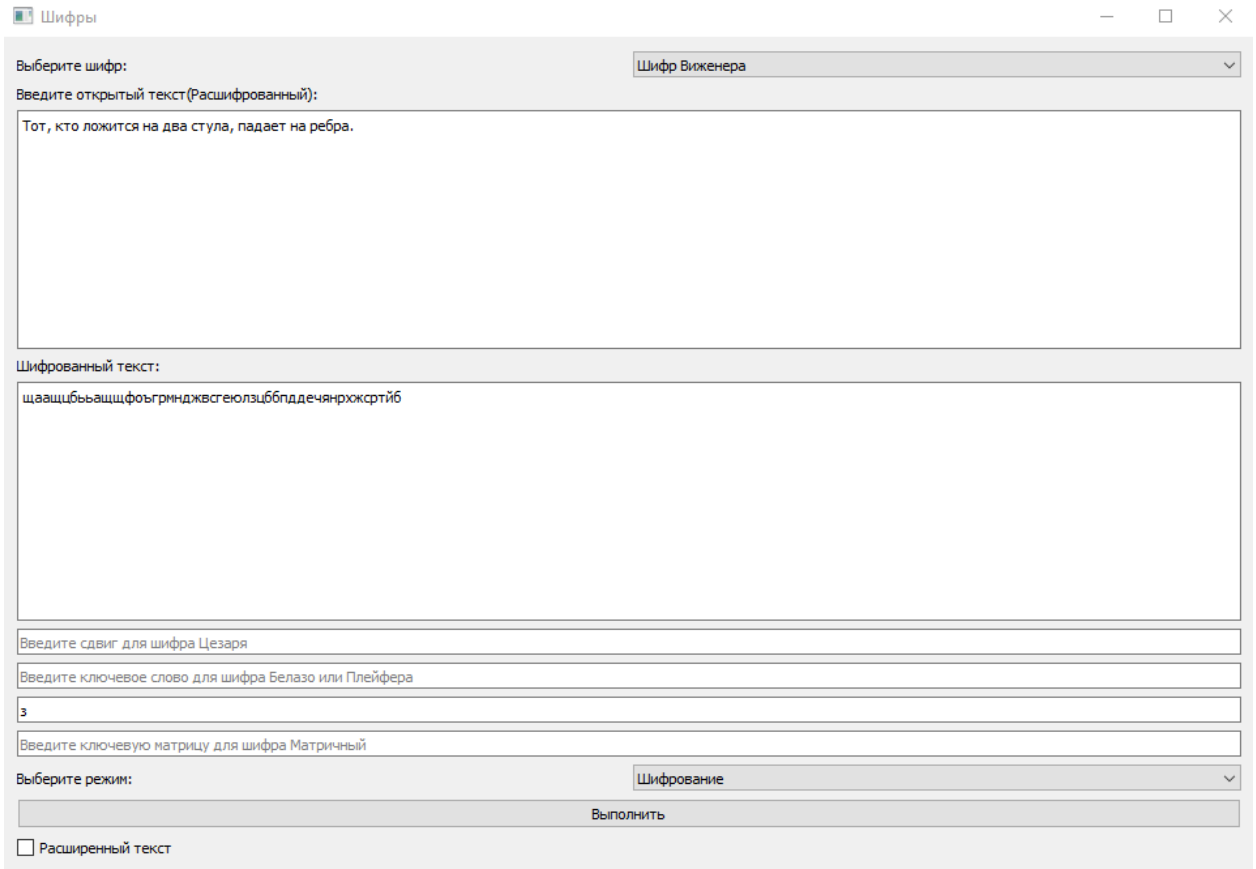

Начало
Получение подготовленного ключевого слова и алфавита
Ключевая буква введена
Да
Нет
Введена одна буква и она есть в алфавите
Да
Вернуть положительный результат
Нет
Вернуть отрицательный результат

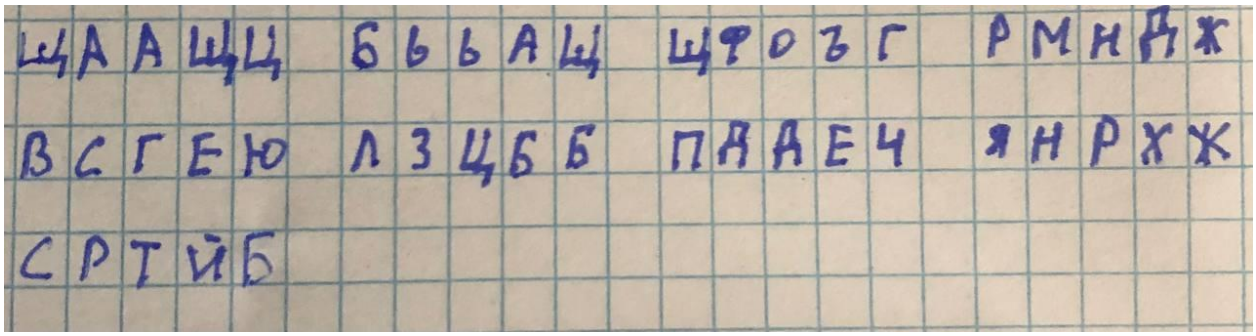
Код программы с комментариями

```
def vigener_check_parameters(key_letter, alphabet):
    if not key_letter:
        return False  # Возврат лжи, если ключевая буква не введена
    if len(key_letter) == 1 and key_letter in alphabet:
        return True  # Возврат истины, если ключевая буква соответствует требованиям
    return False  # Возврат лжи, если в ключевой букве более одного символа или некорректное значение буквы

def vigener_encrypt(open_text, key_letter, mode, alphabet):
    encrypted_text = ""  # Шифртекст
    keyword = key_letter
    for i in range(len(open_text)):  # Проход по всем символам открытого текста
        element = open_text[i]  # Символ
        encrypted_text += alphabet[(alphabet.index(element) + alphabet.index(keyword[i % len(keyword)])) % len(alphabet)]  #
Добавление в итоговый шифртекст зашифрованного символа
        if mode == "selfkey":
            keyword += open_text[i]
        elif mode == "cipherkey":
            keyword += encrypted_text[-1]
    return encrypted_text  # Возврат шифртекста

def vigener_decrypt(encrypted_text, key_letter, mode, alphabet):
    decrypted_text = ""  # Расшифрованный текст
    keyword = key_letter
    for i in range(len(encrypted_text)):  # Проход по всем символам шифртекста
        element = encrypted_text[i]  # Символ
        decrypted_text += alphabet[(alphabet.index(element) - alphabet.index(keyword[i % len(keyword)]) + len(alphabet)) %
len(alphabet)]  # Добавление в итоговый текст расшифрованного символа
        if mode == "selfkey":
            keyword += decrypted_text[-1]  # добавление к ключу расшифрованной буквы
        elif mode == "cipherkey":
            keyword += encrypted_text[i]
```

```
    # Перевод символов из их текстовых значений в символьные
    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
```

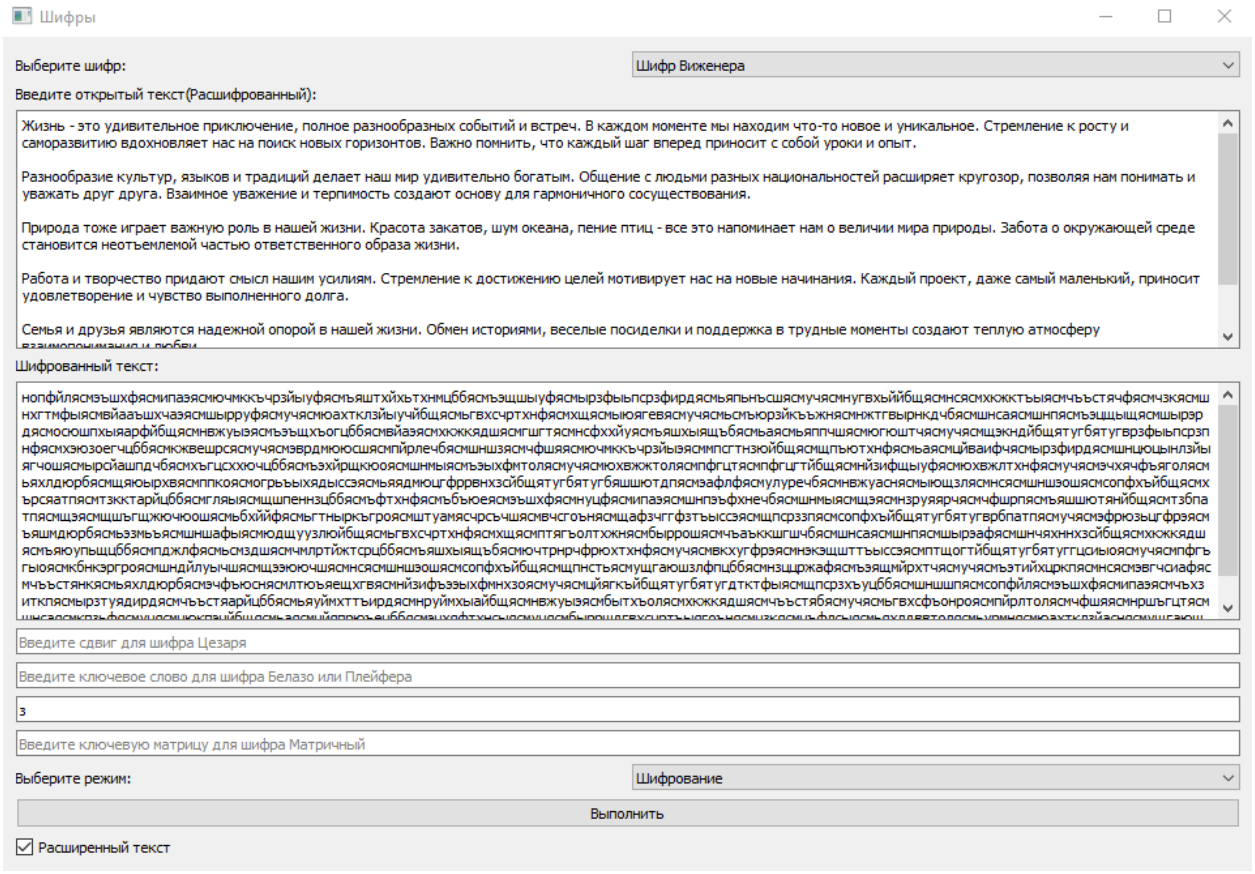
Тестирование





Работа с текстом не менее 1000 знаков

Зашифрование



Расшифрование