Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра информационных](https://www.belstu.by/fakultety/fit/vm) систем и технологий

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №6**

по дисциплине Информационная безопасность

Тема: Изучение устройства и функциональных особенностей шифровальной машины «Энигма»

Студент: Станчик М.А. ФИТ курс 3 группа 4

Преподаватель: Савельева М.Г.

Минск, 2025

**Лабораторная работа №6**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации перестановочных шифров.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости перестановочных шифров.

2. Изучить структуру, принципы функционирования, реализацию процедур зашифрования сообщений в машинах семейства «Энигма».

3. Изучить и приобрести практические навыки выполнения криптопреобразований информации на платформе «Энигма», реализованной в виде симуляторов.

4. Получить практические навыки оценки криптостойкости подстановочных и перестановочных шифров на платформе «Энигма».

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# Теоретические сведения

Машина «Энигма» – это электромеханическое устройство.

Как и другие роторные машины, «Энигма» состоит из комбинации механических и электрических подсистем. Механическая часть включает в себя клавиатуру, набор вращающихся дисков – роторов, которые расположены вдоль вала и прилегают к нему, и ступенчатого механизма, двигающего один или несколько роторов при каждом нажатии на клавишу. Электрическая часть, в свою очередь, состоит из электрической схемы, соединяющей между собой клавиатуру, коммутационную панель, лампочки и роторы (для соединения роторов использовались скользящие контакты).

При каждом нажатии на клавишу самый правый ротор сдвигался на одну позицию, а при определенных условиях сдвигались и другие роторы. Движение роторов приводило к различным криптографическим преобразованиям при каждом следующем нажатии на клавишу на клавиатуре, т. е. зашифрование/расшифрование сообщений основано на выполнении ряда замен (подстановок) одного символа другим.

Механические части двигались и, замыкая контакты, образовывали меняющийся электрический контур. При нажатии на клавишу клавиатуры контур замыкается, ток проходит через созданную (для зашифрования/расшифрования одного конкретного символа сообщения) цепь и в результате включает одну из набора лампочек, отображающую искомую букву шифртекста (или расшифрованного сообщения).

# Ход работы

Целью работы была разработка приложения-симулятора шифровальной машины «Энигма», состоящей из клавиатуры, трех роторов и отражателя. Для этого был разработан класс Enigma.

Для реализации шифрования знать и хранить следующие данные:

* подстановочные алфавиты, соответствующие каждому ротору;
* текущую позицию каждого ротора;
* общее количество смещений каждого ротора;
* количество полных оборотов роторов;
* шаги смещения каждого ротора.

Для хранения этих данных созданы следующие переменные, представленные на листинге 2.1.

public static final String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

public static final String alphabetRightRotor = "LEYJVCNIXWPBQMDRTAKZGFUHOS";;

public static final String alphabetMiddleRotor = "BDFHJLCPRTXVZNYEIWGAKMUSQO";

public static final String alphabetLeftRotor = "FSOKANUERHMBTIYCWLQPZXVGJD";

Map<Character, Character> alphabetReflector;

int length = alphabet.length();

private int rotorRightCurrentPosition = 0;

private int rotorMiddleCurrentPosition = 0;

private int rotorLeftCurrentPosition = 0;

private int rotorRightTotalOffsets = 0;

private int rotorMiddleTotalOffsets = 0;

private int rotorLeftTotalOffsets = 0;

private int rotorRightFullRotations = 0;

private int rotorMiddleFullRotations = 0;

private int rotorLeftFullRotations = 0;

private int rotorRightStep = 4;

private int rotorMiddleStep = 0;

private int rotorLeftStep = 0;

Листинг 2.1 – Переменные для хранения данных для реализации шифрования

Для инициализации объекта шифровальной машины Энигма с заданными начальными позициями роторов был разработан конструктор класса Enigma, который принимает как параметры начальную позицию правого ротора, начальную позицию среднего ротора, начальную позицию левого ротора. Это необходимо в первую очередь для выполнения задания, касающегося шифрования сообщения применяя различные варианты начальных установок роторов. Код конструктора представлен в листинге 2.2.

public Enigma(int rightRotorPosition, int middleRotorPosition, int leftRotorPosition) {

if (rightRotorPosition >= 0 && rightRotorPosition < length && middleRotorPosition >= 0 && middleRotorPosition < length && leftRotorPosition >= 0 && leftRotorPosition < length) {

rotorRightCurrentPosition = rightRotorPosition;

rotorMiddleCurrentPosition = middleRotorPosition;

rotorLeftCurrentPosition = leftRotorPosition;

}

else {

throw new IllegalArgumentException("Позиции роторов должны быть от 0 до " + (length - 1));

}

alphabetReflector = fillTheRelector();

}

Листинг 2.2 – Конструктор класса

Для создания и инициализации отображения для отражателя шифровальной машины Энигма был создан метод fillTheRelector, который возвращает объект типа Map<Character, Character>, который представляет собой ассоциативный массив (или словарь), где каждый символ сопоставляется с другим символом. Отражатель отвечает за отражение сигнала, позволяя символу, входящему в роторы, быть преобразованным в другой символ при выходе. Код метода представлен в листинге 2.3.

private static Map<Character, Character> fillTheRelector() {

Map<Character, Character> reflector = new HashMap<>();

reflector.put('A', 'F');

reflector.put('B', 'V');

reflector.put('C', 'P');

reflector.put('D', 'J');

reflector.put('E', 'I');

reflector.put('G', 'O');

reflector.put('H', 'Y');

reflector.put('K', 'R');

reflector.put('L', 'Z');

reflector.put('M', 'X');

reflector.put('N', 'W');

reflector.put('T', 'Q');

reflector.put('S', 'U');

return reflector;

}

Листинг 2.3 – Метод для создания отражателя

Далее было создано перечисление (enum) Operation для определения набора констант, которые представляют два типа операций, выполняемых в шифровальной машине Энигма: шифрование и расшифровка. Код перечисления представлен в листинге 2.4.

enum Operation {

ENCRYPT,

DECRYPT,

}

Листинг 2.4 – Перечисление Operation

Метод isLetterInAlphabet используется для проверки, является ли передаваемый символ допустимым для шифрования или расшифровки. Это важно, чтобы избежать ошибок при обработке символов, которые не входят в алфавит (например, пробелов, цифр или специальных символов). Код метода представлен в листинге 2.5.

private boolean isLetterInAlphabet(char letter) {

return alphabet.indexOf(letter) != -1;

}

Листинг 2.5 – Метод для проверки передаваемого символа

Метод processLetter отвечает за обработку символа в шифровальной машине Энигма, выполняя как шифрование, так и расшифровку. Принимает как параметры символ, который нужно обработать и операцию, которую нужно выполнить (шифрование или расшифровка). Метод processLetter реализует основную логику шифрования и расшифровки в Энигме, обрабатывая символ через три ротора и отражатель, а затем возвращая результат. Это ключевая часть алгоритма, обеспечивающая корректное преобразование символов. Код метода представлен в листинге 2.6.

private char processLetter(char letter, Operation operation) {

char letterAfterRightRotor = (operation == Operation.ENCRYPT)

? encryptWithRotor(letter, alphabet, alphabetRightRotor, rotorRightCurrentPosition)

: decryptWithRotor(letter, alphabet, alphabetRightRotor, rotorRightCurrentPosition);

char letterAfterMiddleRotor = (operation == Operation.ENCRYPT)

? encryptWithRotor(letterAfterRightRotor, alphabet, alphabetMiddleRotor, rotorMiddleCurrentPosition)

: decryptWithRotor(letterAfterRightRotor, alphabet, alphabetMiddleRotor, rotorMiddleCurrentPosition);

char letterAfterLeftRotor = (operation == Operation.ENCRYPT)

? encryptWithRotor(letterAfterMiddleRotor, alphabet, alphabetLeftRotor, rotorLeftCurrentPosition)

: decryptWithRotor(letterAfterMiddleRotor, alphabet, alphabetLeftRotor, rotorLeftCurrentPosition);

char letterAfterReflector = encryptWithReflector(letterAfterLeftRotor);

char letterAfterLeftRotorBackwards = (operation == Operation.ENCRYPT)

? encryptWithRotor(letterAfterReflector, alphabetLeftRotor, alphabet, rotorLeftCurrentPosition)

: decryptWithRotor(letterAfterReflector, alphabetLeftRotor, alphabet, rotorLeftCurrentPosition);

char letterAfterMiddleRotorBackwards = (operation == Operation.ENCRYPT)

? encryptWithRotor(letterAfterLeftRotorBackwards, alphabetMiddleRotor, alphabet, rotorMiddleCurrentPosition)

: decryptWithRotor(letterAfterLeftRotorBackwards, alphabetMiddleRotor, alphabet, rotorMiddleCurrentPosition);

char letterAfterRightRotorBackwards = (operation == Operation.ENCRYPT)

? encryptWithRotor(letterAfterMiddleRotorBackwards, alphabetRightRotor, alphabet, rotorRightCurrentPosition)

: decryptWithRotor(letterAfterMiddleRotorBackwards, alphabetRightRotor, alphabet, rotorRightCurrentPosition);

updateRotorPositions();

return letterAfterRightRotorBackwards;

}

Листинг 2.6 – Код метода для обработки символа в шифровальной машине

Метод encryptWithRotor принимает как параметры символ, который нужно зашифровать, строку, которая представляет алфавит, строку, представляющую зашифрованный алфавит и текущее смещение ротора. Отвечает за шифрование символа через ротор, применяя смещение для получения зашифрованного символа. Код метода представлен в листинге 2.7.

private char encryptWithRotor(char letter, String alphabet, String alphabetEncryption, int offset) {

int index = alphabet.indexOf(letter);

int indexEncrypted = (index + offset) % length;

return alphabetEncryption.charAt(indexEncrypted);

}

Листинг 2.7 – Код метода для шифрования символа через ротор

Метод decryptWithRotor принимает как параметры символ, который нужно расшифровать, строку, представляющую алфавит, строку, представляющую зашифрованный алфавит, текущее смещение ротора. Отвечает за расшифровку символа через ротор, применяя смещение для получения исходного символа. Код метода представлен в листинге 2.8.

private char decryptWithRotor(char letter, String alphabet, String alphabetEncryption, int offset) {

int index = alphabet.indexOf(letter);

int indexEncrypted = (index - offset + length) % length;

return alphabetEncryption.charAt(indexEncrypted);

}

Листинг 2.8 – Код метода для шифрования символа через ротор

Метод encryptWithReflector принимает как параметр символ, который нужно обработать через отражатель. Отвечает за обработку символа через отражатель. Он обеспечивает взаимное преобразование символов, что является ключевым элементом шифрования Энигмы. Код метода представлен в листинге 2.9.

private char decryptWithRotor(char letter, String alphabet, String alphabetEncryption, int offset) {

int index = alphabet.indexOf(letter);

int indexEncrypted = (index - offset + length) % length;

return alphabetEncryption.charAt(indexEncrypted);

}

Листинг 2.9 – Код метода для обработки символа через отражатель

Метод updateRotorPositions управляет изменением позиций роторов в процессе шифрования. Это важно, так как каждая операция шифрования изменяет конфигурацию роторов, что, в свою очередь, влияет на результат шифрования. Корректное обновление позиций роторов обеспечивает безопасность шифрования, так как каждый символ будет обработан с использованием уникальных настроек роторов. Код метода представлен в листинге 2.10.

private void updateRotorPositions() {

rotorRightTotalOffsets += rotorRightStep;

rotorRightCurrentPosition = rotorRightTotalOffsets % length;

if (rotorRightTotalOffsets / length > 0) {

rotorRightFullRotations = rotorRightTotalOffsets / length;

rotorMiddleTotalOffsets = rotorRightFullRotations \* rotorMiddleStep;

rotorMiddleCurrentPosition = rotorMiddleTotalOffsets % length;

}

if (rotorMiddleTotalOffsets / length > 0) {

rotorMiddleFullRotations = rotorMiddleTotalOffsets / length;

rotorLeftTotalOffsets = rotorMiddleFullRotations \* rotorLeftStep;

rotorLeftCurrentPosition = rotorLeftTotalOffsets % length;

}

if (rotorLeftTotalOffsets / length > 0) {

rotorLeftFullRotations = rotorLeftTotalOffsets / length;

}

}

Листинг 2.10 – Код метода для управления изменением позиций роторов в процессе шифрования

Метод Encrypt реализует процесс шифрования текста, обрабатывая только буквы и оставляя другие символы без изменений. Это важно для обеспечения корректного шифрования текста. Метод принимает массив символов, представляющий открытый текст, который нужно зашифровать, возвращает зашифрованный текст в виде массива символов. Код метода представлен в листинге 2.11.

public char[] Encrypt(char[] openText) {

StringBuilder sb = new StringBuilder();

for (char letter : openText) {

if (isLetterInAlphabet(letter)) {

char encryptedLetter = processLetter(letter, Operation.ENCRYPT);

sb.append(encryptedLetter);

} else {

sb.append(letter);

}

}

return sb.toString().toCharArray();

}

Листинг 2.11 – Код метода для шифрования текста

Метод Decrypt реализует процесс расшифровки текста, аналогично методу шифрования. Он обрабатывает только буквы и оставляет другие символы без изменений, что позволяет восстановить исходный текст. Метод принимает массив символов, представляющий зашифрованный текст, который нужно расшифровать и возвращает в виде массива символов расшифрованный текст. Для расшифрования необходимо установить роторы в такие же позиции, как и при зашифровании. Алгоритм аналогичен, кроме функции расшифрования ротором. Код метода представлен в листинге 2.12.

public char[] Decrypt(char[] openText) {

StringBuilder sb = new StringBuilder();

for (char letter : openText) {

if (isLetterInAlphabet(letter)) {

char decryptedLetter = processLetter(letter, Operation.DECRYPT);

sb.append(decryptedLetter);

} else {

sb.append(letter);

}

}

return sb.toString().toCharArray();

}

Листинг 2.12 – Код метода для расшифрования текста

Метод calculateProbabilities принимает строку, которая содержит символы. Предназначен для вычисления вероятностей появления каждого уникального символа в заданной строке. Метод возвращает отображение, где ключами являются уникальные символы из строки, а значениями - соответствующие вероятности этих символов. Вероятность символа вычисляется как отношение количества его вхождений к общему количеству символов в строке. Код метода представлен в листинге 2.13.

public Map<Character, Double> calculateProbabilities(String text) {

Map<Character, Integer> frequencyMap = new HashMap<>();

int totalCount = 0;

for (char c : text.toCharArray()) {

frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);

totalCount++;

}

Map<Character, Double> probabilities = new HashMap<>();

for (Map.Entry<Character, Integer> entry : frequencyMap.entrySet()) {

char character = entry.getKey();

int frequency = entry.getValue();

double probability = (double) frequency / totalCount;

probabilities.put(character, probability);

}

return probabilities;

}

Листинг 2.13 – Метод для вычисления вероятностей появления каждого уникального символа в строке

Метод calculateEntropy нужен для вычисления энтропии текста. Метод принимает отображение, где ключами являются уникальные символы из строки, а значениями - соответствующие вероятности этих символов. Этотпараметр должен быть получен с помощью метода calculateProbabilities. Код метода представлен в листинге 2.14.

public double calculateEntropy(Map<Character, Double> charProbabilities) {

double entropy = 0.0;

for (double charProbability : charProbabilities.values()) {

if (charProbability > 0) {

entropy -= charProbability \* (Math.log(charProbability) / Math.log(2));

}

}

return entropy;

}

Листинг 2.14 – Метод для вычисления вероятностей появления каждого уникального символа в строке

Метод saveFrequenciesToExcel предназначен для сохранения частот символов в Excel файл. Принимает отображение, где ключами являются уникальные символы, а значениями - соответствующие частоты этих символов, и имя файла (включая путь), в который будет сохранён Excel файл с частотами символов. Код метода представлен в листинге 2.15.

public void saveFrequenciesToExcel(Map<Character, Double> frequencies, String filename) {

Workbook workbook = new XSSFWorkbook();

Sheet sheet = workbook.createSheet("Frequencies");

Row headerRow = sheet.createRow(0);

headerRow.createCell(0).setCellValue("Символ");

headerRow.createCell(1).setCellValue("Частота");

int rowNum = 1;

for (Map.Entry<Character, Double> entry : frequencies.entrySet()) {

Row row = sheet.createRow(rowNum++);

row.createCell(0).setCellValue(entry.getKey().toString());

row.createCell(1).setCellValue(entry.getValue());

}

try (FileOutputStream fileOut = new FileOutputStream(filename)) {

workbook.write(fileOut);

} catch (Exception e) {

e.printStackTrace();

}

System.out.println("Частота символов сохранена в " + filename);

}

Листинг 2.15 – Метод для сохранения частот символов в Excel файл

Метод createDataset предназначен для создания набора данных, который будет использоваться для построения гистограммы с помощью библиотеки JFreeChart. Принимает отображение, которое вычисляется с помощью метода calculateProbabilities. Метод возвращает заполненный объект DefaultCategoryDataset. Код метода представлен в листинге 2.16.

private CategoryDataset createDataset(Map<Character, Double> frequencies) {

DefaultCategoryDataset dataset = new DefaultCategoryDataset();

for (Map.Entry<Character, Double> entry : frequencies.entrySet()) {

dataset.addValue(entry.getValue(), "Частота", entry.getKey().toString());

}

return dataset;

}

Листинг 2.16 – Метод для создания набора данных

Метод saveHistogramToExcel предназначен для создания и сохранения гистограммы частот символов в Excel файл. Метод принимает отображение и путь к файлу Excel, ничего не возвращает. Код метода представлен в листинге 2.17.

void saveHistogramToExcel(Map<Character, Double> frequencies, String excelFilePath) {

CategoryDataset dataset = createDataset(frequencies);

JFreeChart histogram = ChartFactory.createBarChart(

"Частоты символов",

"Символ",

"Частота",

dataset

);

String imagePath = "histogram.png";

try {

ChartUtils.saveChartAsPNG(new File(imagePath), histogram, 800, 600);

}

catch (IOException e) {

e.printStackTrace();

}

insertImageToExcel(imagePath, excelFilePath);

}

Листинг 2.17 – Метод для создания и сохранения гистограммы частот символов в Excel файл

Метод processMessage является точкой входа для шифрования или расшифровки текста, обеспечивая создание экземпляра шифровальной машины и вывод информации о времени выполнения операции. Принимает как параметры начальную позицию правого ротора, начальную позицию среднего ротора, начальную позицию левого ротора, массив символов, представляющий текст для обработки (шифрования или расшифровки), операцию, которую нужно выполнить (шифрование или расшифровка). Возвращает результат в виде массива символов. Код метода представлен в листинге 2.18.

public static char[] processMessage(int rightRotorPosition, int middleRotorPosition, int leftRotorPosition, char[] text, Operation operation) {

Enigma enigmaObj = new Enigma(rightRotorPosition, middleRotorPosition, leftRotorPosition);

long startTime = System.nanoTime();

char[] resultMessage;

if (operation == Operation.ENCRYPT) {

resultMessage = enigmaObj.Encrypt(text);

System.out.println("Время шифрования: " + (System.nanoTime() - startTime) + " нс");

System.out.println("Зашифрованный текст: " + valueOf(resultMessage));

}

else {

resultMessage = enigmaObj.Decrypt(text);

System.out.println("Время расшифровки: " + (System.nanoTime() - startTime) + " нс");

System.out.println("Расшифрованный текст: " + valueOf(resultMessage));

}

return resultMessage;

}

Листинг 2.18 – Точка входа для шифрования или расшифровки текста

Метод saveFrequenciesAndHistograms отвечает за анализ и сохранение статистики по зашифрованному и расшифрованному текстам, принимает как параметры массив символов, представляющий зашифрованный текст, массив символов, представляющий расшифрованный текст, индекс варианта, который используется для формирования названий файлов. Результаты сохраняются в файлы формата .xlsx с помощью методов saveFrequenciesToExcel и saveHistogramToExcel. Код метода представлен в листинге 2.19.

static void saveFrequenciesAndHistograms(char[] encryptedText, char[] decryptedText, int variantIndex) {

String encryptedFilePath = "schemes/encryptedTextVariant" + variantIndex + ".xlsx";

String decryptedFilePath = "schemes/decryptedTextVariant" + variantIndex + ".xlsx";

frequences.saveFrequenciesToExcel(frequences.calculateProbabilities(encryptedText), encryptedFilePath);

frequences.saveHistogramToExcel(frequences.calculateProbabilities(encryptedText), encryptedFilePath);

frequences.saveFrequenciesToExcel(frequences.calculateProbabilities(decryptedText), decryptedFilePath);

frequences.saveHistogramToExcel(frequences.calculateProbabilities(decryptedText), decryptedFilePath);

}

Листинг 2.19 – Точка входа для шифрования или расшифровки текста

Пример зашифрования одного символа, если L – Gamma, M – III, R – Beta, Re – C, Li-Mi-Ri = 0-0-4:

1. Далее для зашифрования сообщения таблица считывается по строкам. Зашифрованное сообщение: «Karaaomomnc goihprwe».
2. Шифруемый символ: R, исходный алфавит: ABCDEFGHIJKLMNOPQRSTUVWXYZ.
3. Ротор R в положении 4: JVCNIXWPBQMDRTAKZGFUHOSLEY. Положение символа R в исходном алфавите: 17. Производим замену R 🡪 G;
4. Ротор M в положении 0: BDFHJLCPRTXVZNYEIWGAKMUSQO. Положение символа G в исходном алфавите: 6. Производим замену G 🡪 L;
5. Ротор L в положении 0: FSOKANUERHMBTIYCWLQPZXVGJD. Положение символа L в исходном алфавите: 11. Производим замену L 🡪 B;
6. Далее символ B проходит через рефлектор. Производим замену: B 🡪 V;
7. Обратный ход через ротор L. Положение символа V в алфавите ротора L: 22. Производим замену V 🡪 W;
8. Обратный ход через ротор M. Положение символа W в алфавите ротора M: 17. Производим замену W 🡪 Q;
9. Обратный ход через ротор R. Положение символа Q в алфавите ротора R: 9. Производим последнюю замену R 🡪 J.
10. Далее роторы смещаются и начинается шифрование следующего символа.

Результат работы приложения для роторов в позиции 0-0-4 приведен на рисунке 1.1.

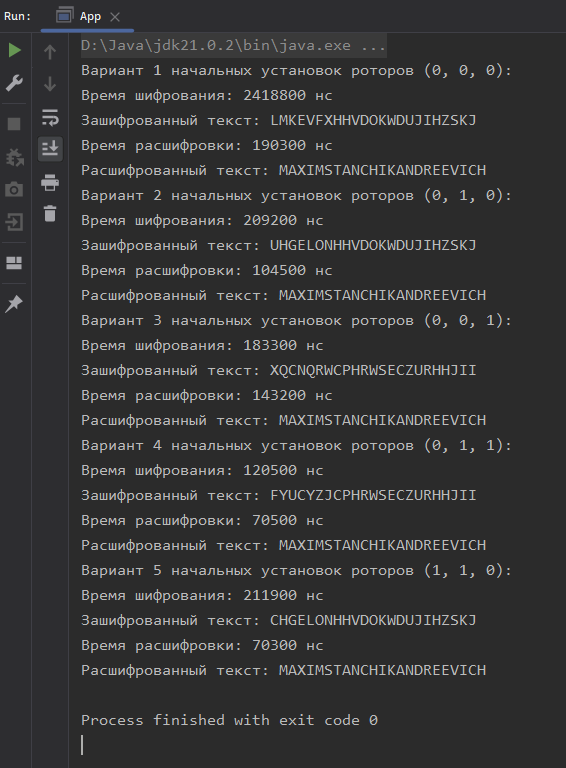


Рисунок 2.1 – Результат шифрования и дешифрования текста

Как видно из рисунка 1.1, расшифрование текста привело к исходному тексту, а это означает то, что машина «Энигма» работает корректно.

# Гистограммы частот появления символов

Необходимо было оценить частотные свойства символов в шифртекстах и сравнить этот параметр с частотными свойствами символов для исходного текста.

На рисунках 3.1 и 3.2 представлены гистограммы частот появления символов в текстах.

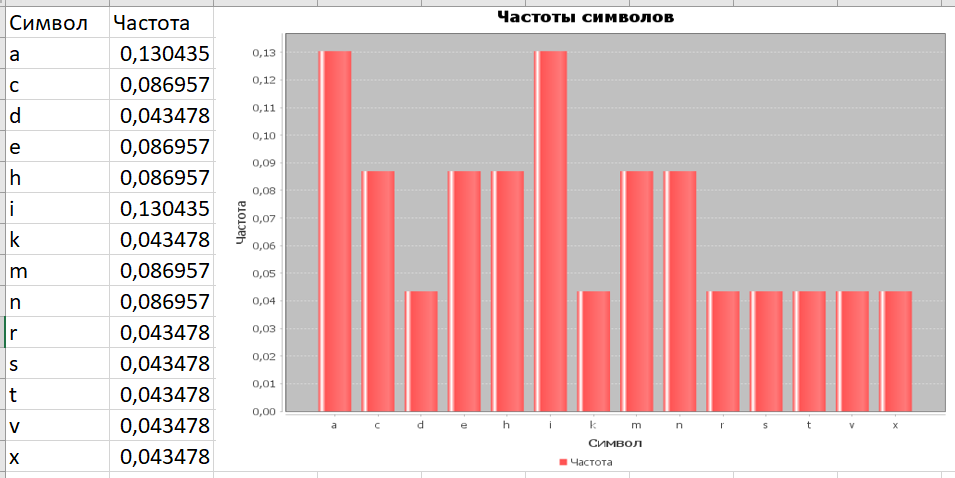


Рисунок 3.1 – Гистограмма частот появления символов исходного сообщения

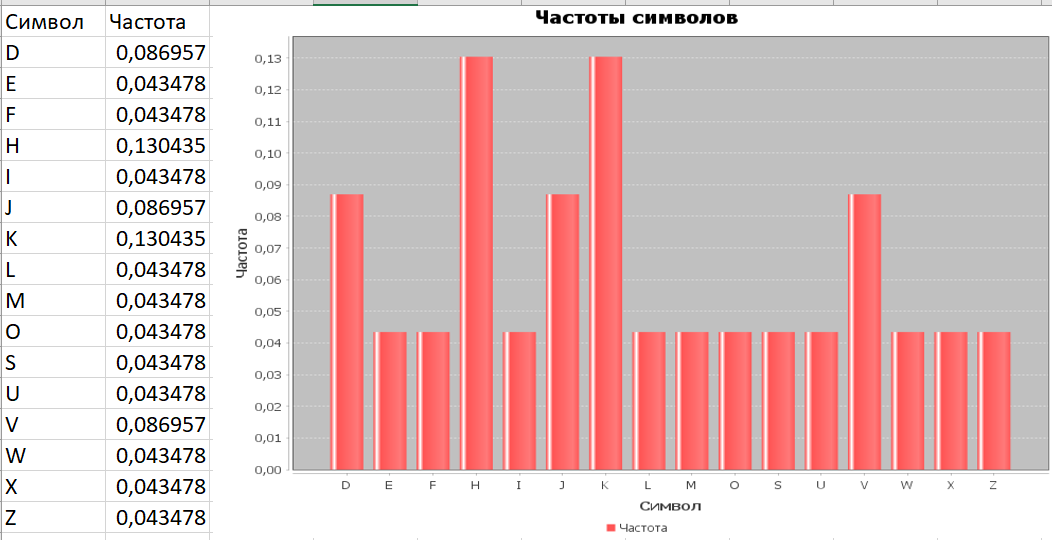


Рисунок 3.2 – Гистограмма частот появления символов зашифрованного текста

По гистограммам на рисунках 3.1 и 3.2 можно сделать вывод, что частоты появления не совпадают, что значительно увеличивает криптостойкость шифра машины «Энигма».

# Оценка криптостойкости «Энигмы» (10 вариант)

Количество различных перестановок роторов: всего имеется 10 роторов, из которых одномоментно может использоваться только 3, поэтому количество перестановок = 10 × 9 × 8 = 720.

На каждом роторе имеется 26 символов, поэтому общее количество перестановок букв – 26 × 26 × 26 = 17576.

Количество различных перестановок коммутаторов: всего имеется 26 букв, нам не важно, в каком порядке идут эти 13 пар, а также для каждой пары обратная пара будет той же парой («AB» = «BA»), поэтому число всех возможных перестановок 26 букв в коммутаторе будет равно:

26! ÷ (13! ×213) = 64764752532480000 ÷ 8192 = 7905853580625.

Таким образом, количество всевозможных состояний равно:

720 × 17576 × 7905853580625 = 100046363423806800000 ≈ 1 × 1020.

Модель «Энигма» в приложении демонстрирует хорошую теоретическую криптостойкость, но реальная «Энигма» обладала более сложной архитектурой и значительно большим числом возможных состояний, что делало её более стойкой к криптоанализу. Реальная модель «Энигма» могла достигать числа состояний порядка 10114 и более, реализация же приложения в данной работе имеет примерно 1020 состояний. В реальной «Энигме» количество роторов могло варьироваться, и использовались разные конфигурации, также использовались коммутаторы, которые обеспечивали дополнительные уровни сложности.

# Оценка времени выполнения операций зашифрования и расшифрования

Для оценки времени, затраченного на выполнение операций зашифрования и расшифрования, используется объект Nanotime, выводящий время выполнения операций в наносекундах.

График выполнения программы 15 раз приведен на рисунке 5.1.

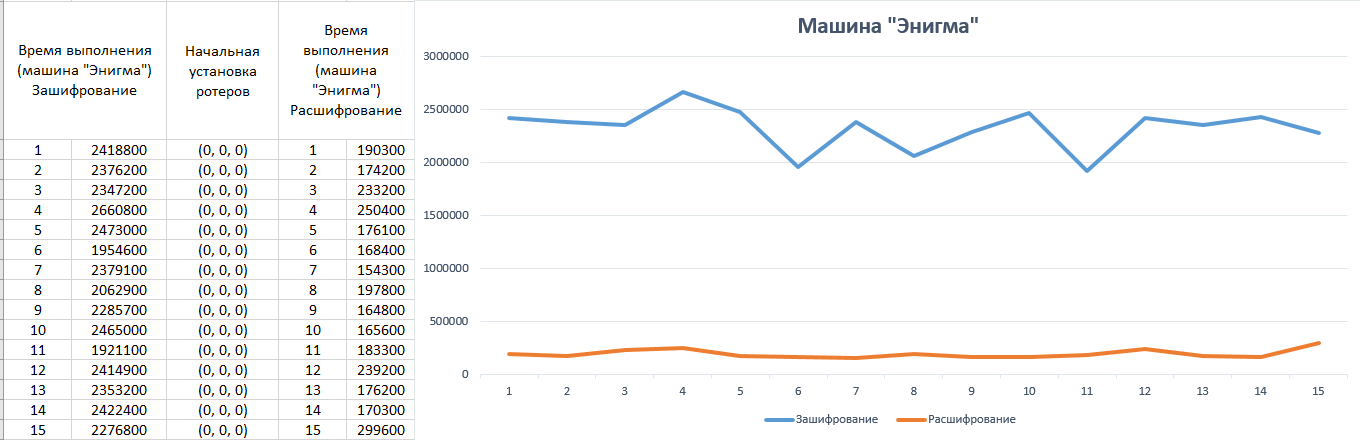


Рисунок 5.1 – График выполнения программы

Как видно из результатов запуска программы на рисунке 5.1, расшифрование происходит примерно в 11.91 раз быстрее, чем шифрование.

**Вывод:** в ходе выполнения лабораторной работы были изучены приницпы функционирования машин семейства «Энигма». Также было разработано приложение для реализации зашифрования и расшифрования с помощью симулятора машины «Энигма». Было выполнено исследование криптостойкости шифров на основе статистических данных о частотах появления символов в исходном и зашифрованном сообщениях. Шифровальная машина «Энигма» является в разы более криптостойкой, чем рассмотренные ранее шифры. Мой вариант машины значительно менее криптостойкий, чем реальный вариант машины «Энигма».