Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Рязанский государственный радиотехнический

университет имени В.Ф. Уткина»

Кафедра «ЭВМ»

Отчет о лабораторной работе №4

«Шифры многобуквенной замены на примере шифра Хилла»

по дисциплине

«Защита информации»

Выполнили:

Студенты группы 045

Анохин В.А.

Вашкулатов Н.А.

Проверили:

доц. Крошилина С.В.

доц. Тишкина В.В.

**Цель работы**: выполнить шифрование и дешифрование сообщений.

**Ход работы**

На рисунках 1 – 6 показаны схемы алгоритмов данной программы.

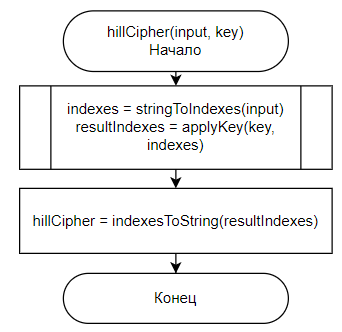


Рисунок 1 – Схема подпрограммы hillCipher

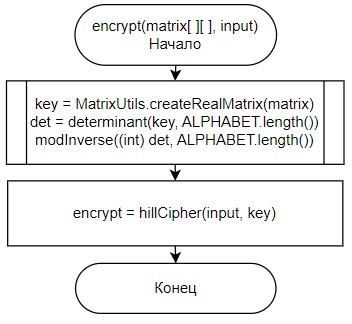


Рисунок 2 – Схема подпрограммы en crypt

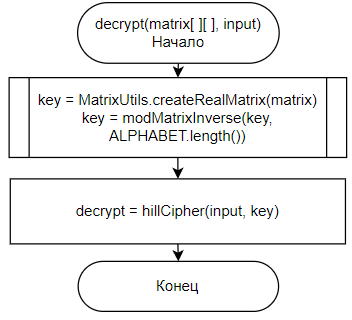


Рисунок 3 – Схема подпрограммы decrypt

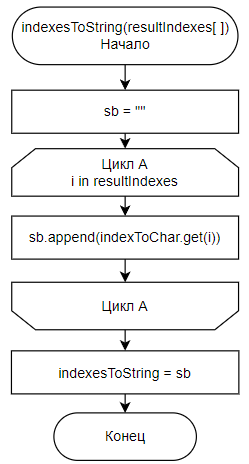


Рисунок 4 – Схема подпрограммы indexesToString

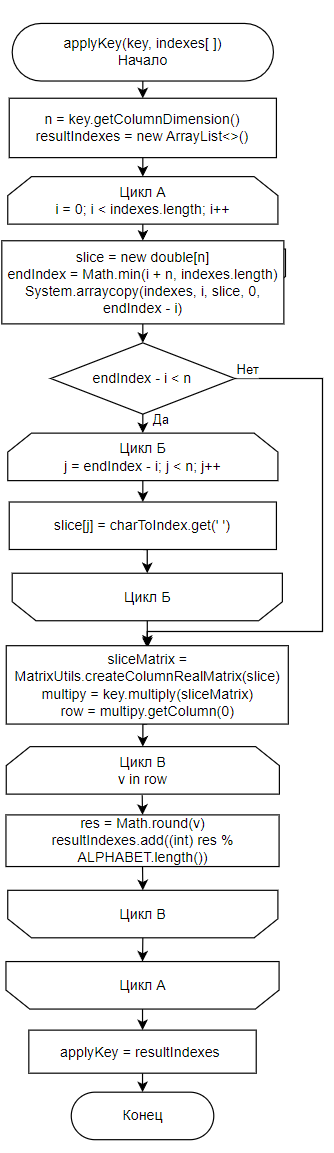


Рисунок 5 – Схема подпрограммы applyKey

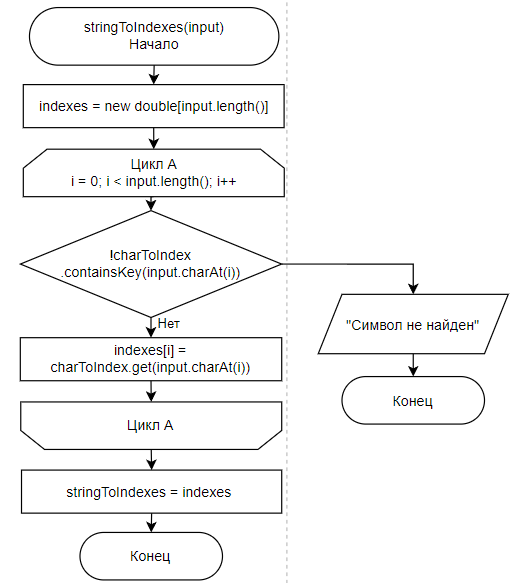


Рисунок 6 – Схема подпрограммы stringToIndexes

**Код программы:**

public class Hill {

private static final Map<Character, Integer> charToIndex = new HashMap<>();

private static final Map<Integer, Character> indexToChar = new HashMap<>();

static String ALPHABET = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя .?!";

static {

for (int i = 0; i < ALPHABET.length(); i++) {

charToIndex.put(ALPHABET.charAt(i), i);

indexToChar.put(i, ALPHABET.charAt(i));

}

}

private static String hillCipher(String input, RealMatrix key) {

double[] indexes = stringToIndexes(input);

List<Integer> resultIndexes = applyKey(key, indexes);

return indexesToString(resultIndexes);

}

public static String encrypt(double[][] matrix, String input) {

var key = MatrixUtils.createRealMatrix(matrix);

// для валидации ключа

var det = determinant(key, ALPHABET.length());

modInverse((int) det, ALPHABET.length());

return hillCipher(input, key);

}

public static String decrypt(double[][] matrix, String input) {

var key = MatrixUtils.createRealMatrix(matrix);

key = modMatrixInverse(key, ALPHABET.length());

return hillCipher(input, key);

}

private static List<Integer> applyKey(RealMatrix key, double[] indexes) {

int n = key.getColumnDimension();

List<Integer> resultIndexes = new ArrayList<>();

for (int i = 0; i < indexes.length; i += n) {

double[] slice = new double[n];

int endIndex = Math.min(i + n, indexes.length);

System.arraycopy(indexes, i, slice, 0, endIndex - i);

if (endIndex - i < n) {

for (int j = endIndex - i; j < n; j++) {

slice[j] = charToIndex.get(' ');

}

}

var sliceMatrix = MatrixUtils.createColumnRealMatrix(slice);

var multipy = key.multiply(sliceMatrix);

double[] row = multipy.getColumn(0);

for (double v : row) {

var res = Math.round(v);

resultIndexes.add((int) res % ALPHABET.length());

}

}

return resultIndexes;

}

private static String indexesToString(List<Integer> resultIndexes) {

var sb = new StringBuilder();

for (int i : resultIndexes) {

sb.append(indexToChar.get(i));

}

return sb.toString();

}

private static double[] stringToIndexes(String input) {

double[] indexes = new double[input.length()];

for (int i = 0; i < input.length(); i++) {

if (!charToIndex.containsKey(input.charAt(i))) throw new IllegalArgumentException("Символ не найден " + input.charAt(i));

indexes[i] = charToIndex.get(input.charAt(i));

}

return indexes;

}

public static int determinant(RealMatrix matrix, int mod) {

int n = matrix.getRowDimension();

if (n != matrix.getColumnDimension()) {

throw new IllegalArgumentException("Матрица не квадратная");

}

if (n == 1) {

return (int) (matrix.getEntry(0, 0) % mod);

}

int det = 0;

for (int i = 0; i < n; ++i) {

RealMatrix minor = getMinor(matrix, 0, i);

int sign = (i % 2 == 0) ? 1 : -1;

det = (det + sign \* (int) (matrix.getEntry(0, i) \* determinant(minor, mod))) % mod;

}

return det;

}

private static RealMatrix getMinor(RealMatrix matrix, int row, int col) {

int n = matrix.getRowDimension();

RealMatrix minor = MatrixUtils.createRealMatrix(n - 1, n - 1);

int minorRow = 0;

for (int i = 0; i < n; ++i) {

if (i == row) continue;

int minorCol = 0;

for (int j = 0; j < n; ++j) {

if (j == col) continue;

minor.setEntry(minorRow, minorCol, matrix.getEntry(i, j));

++minorCol;

}

++minorRow;

}

return minor;

}

public static RealMatrix modMatrixInverse(RealMatrix matrix, int mod) {

int det = determinant(matrix, mod);

if (det < 0) {

det = (int) (Math.floorMod(det + mod, mod));

}

int detInverse = modInverse(det, mod);

if (det == 0 || detInverse == -1) {

throw new IllegalArgumentException("Для данной матрицы модульной обратной матрицы не существует");

}

int n = matrix.getRowDimension();

RealMatrix inverseMatrix = MatrixUtils.createRealMatrix(n, n);

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

RealMatrix minor = getMinor(matrix, i, j).copy();

int sign = ((i + j) % 2 == 0) ? 1 : -1;

int detMinor = determinant(minor, mod);

int inverseEntry = (sign \* detMinor \* detInverse) % mod;

if (inverseEntry < 0) {

inverseEntry += mod;

}

inverseMatrix.setEntry(j, i, inverseEntry);

}

}

return inverseMatrix;

}

// Метод для вычисления модулярного мультипликативного обратного числа

static int modInverse(int base, int mod) {

if (base < 0) base = Math.abs(base);

return new InverseModulo().modInverse(base, mod);

}

}

**Вывод**: в ходе выполнения работы было выполнено шифрование и дешифрование сообщений шифром Хилла.