Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

«Защита информации и надёжность информационных систем»

Отчёт по лабораторной работе №5

ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. ЦИКЛИЧЕСКИЕ КОДЫ

Выполнил: Лешук Д. И.

ФИТ 3 курс 7 группа

Преподаватель: Николайчук А.Н

Минск 2024

**Цель:** приобретение практических навыков кодирования/ декодирования двоичных данных при использовании циклических кодов.

**Практическое задание:**

Задание выполняется по указанию преподавателя в соответствии с вариантом 10, порождающий полином ЦК – , из которого выбирается порождающий полином ЦК, а по значению соответствующего ему значения r = 6 – длина k информационного слова Xk. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные ошибки в кодовых словах. Определить параметры (n, k)-кода для своего варианта. Основой задания является разработка приложения.

2. Составить порождающую матрицу (n, k)-кода трансформировать ее в каноническую форму и далее – в проверочную матрицу канонической формы.

3. Используя порождающую матрицу ЦК, вычислить избыточные символы (слово Xr) кодового слова Xn и сформировать это кодовое слово.

4. Принять кодовое слово Yn со следующим числом ошибок: 0; 1; 2. Позиция ошибки определяется (генерируется) случайным образом.

5. Для полученного слова Yn вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки Еn = е1, е2, …, еn и исправить одиночную ошибку; проанализировать ситуацию при возникновении ошибки в 2 битах.

6. Результаты оформить в виде отчета по установленным правилам.

**Выполнение работы:**

Первым делом проанализируем имеющиеся у нас входные данные. Количество избыточных бит равно шести. Мы знаем, что r вычисляется по формуле . Отсюда получаем, что k = . Однако, мы можем взять k меньше чем , но больше чем , значение по формуле останется действительным. В итоге имеем: k = 26, r= 6, . Наш полином в двоичном виде имеет вид: 1001001. Определим сообщение из k символов, пусть оно будет таким: 00001011101000010000101110. Далее перейдём к созданию порождающей матрицы G циклического кода – рисунок 1. Определяем размерность нашей матрицы, далее циклами проходим по её строкам и заполняем ячейки коэффициентами полинома.

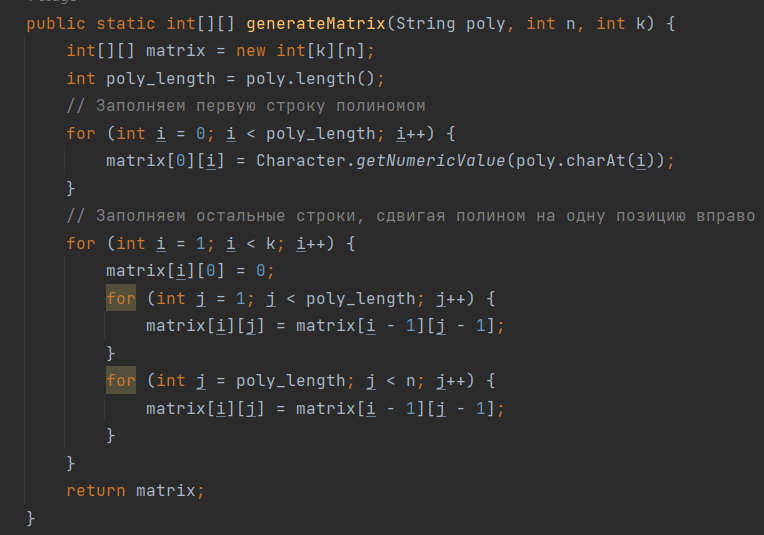


Рисунок 1 – Код функции, создающий порождающую матрицу ЦК

В результате выполнения матрица имеет такой вид – рисунок 2.

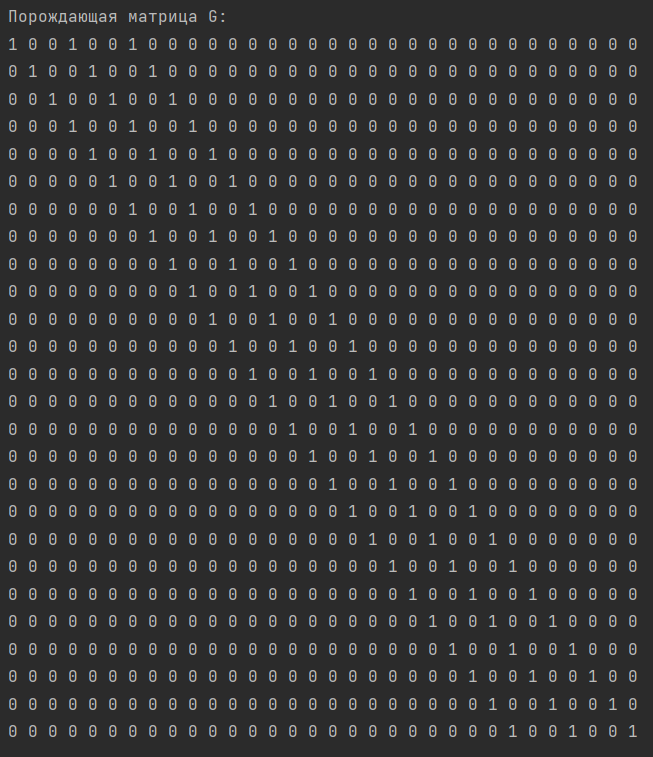


Рисунок 2 – Порождающая матрица G

Далее необходимо привести её к канонической форме – чтобы в левой части матрицы сформировалась подматрица I – диагональная матрица. Имеем следующую функцию для выполнения данной операции – рисунок 3.

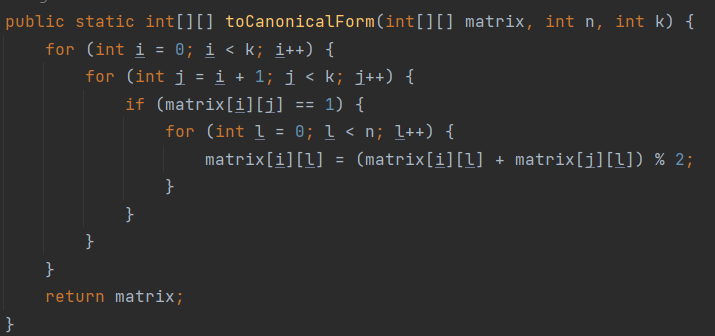


Рисунок 3 – Функция приведения порождающей матрицы к канонической форме

После преобразования имеем следующий вид (в канонической форме) – рисунок 4.



Рисунок 4 – Каноническая порождающая матрица

После записываем подматрицу A данной канонической матрицы размера 26 на 6, где столбцы – последние 6 столбцов матрицы Gk. И под ними записываем диагональную матрицу размера 6 на 6. В результате этих действий имеем проверочную матрицу. Для получения канонической проверочной матрицы осталось её транспонировать. Детальный листинг данных операций находится в приложении А, листинг 1. В результате этих действий имеем матрицу размера r на n, по которой уже можем находить избыточные биты – функция для их вычисления представлена на рисунке 5. Однако отметим, что для данных параметров и заданного порождающего полинома не все столбцы матрицы имеют вес больше двух (аналогичная ситуация получается и при k = 32), несмотря на правильную реализацию алгоритма (похожая ситуация встречалась и в домашнем задании на одной из лекций). Из таблицы в методическом пособии можем увидеть, что параметры соответствующие получаются следующие: .

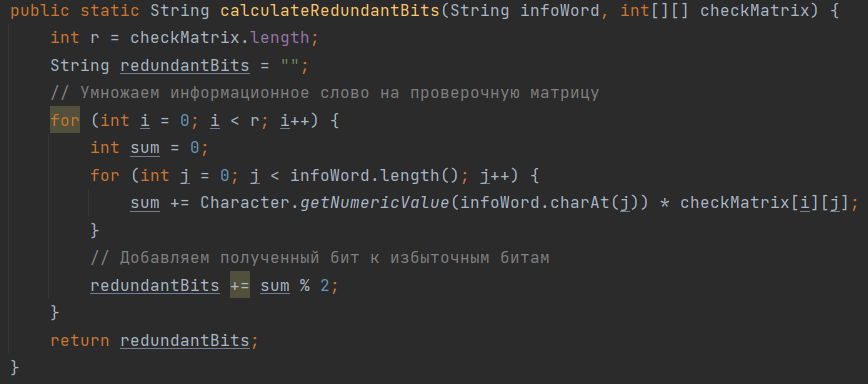


Рисунок 5 – Функция вычисления избыточных битов

В результате имеем такое значение r = 101001. Функция для исправления ошибок – на рисунке 6. Функцию для генерации ошибки на случайной позиции входного сообщения возьмём из прошлых лабораторных работ. Последовательно сгенерируем 1 и 2 ошибки во входном слове и исправим их с помощью разработанной для этого функции correctErrors. Результат представлен на рисунке 7.



Рисунок 6 – Функция исправления ошибок

Как мы можем видеть по рисунку 7, в случае одиночной ошибки она была успешно исправлена. В случае же двух ошибок, они исправлены не были. Следовательно, как и было сказано в условии, данным способом мы можем исправлять лишь одиночную ошибку.

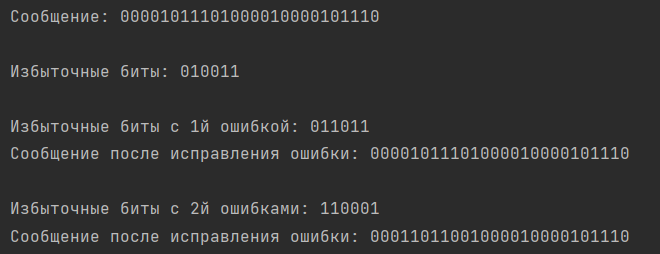


Рисунок 7 – Результат попытки исправления ошибок

**Вывод:** В результате лабораторной работы были приобретены теоретические знания о циклических кодах, проверен на практике способ исправления ошибок с помощью циклического кода, в основе которого лежало построение необходимых матриц по порождающему полиному, в результате которого была успешно исправлена единичная ошибка во входном слове.

**Приложение А**

import java.util.Random;

public class InfoMetrics {

public static int[][] generateMatrix(String poly, int n, int k) {

int[][] matrix = new int[k][n];

int poly\_length = poly.length();

// Заполняем первую строку полиномом

for (int i = 0; i < poly\_length; i++) {

matrix[0][i] = Character.getNumericValue(poly.charAt(i));

}

// Заполняем остальные строки, сдвигая полином на одну позицию вправо

for (int i = 1; i < k; i++) {

matrix[i][0] = 0;

for (int j = 1; j < poly\_length; j++) {

matrix[i][j] = matrix[i - 1][j - 1];

}

for (int j = poly\_length; j < n; j++) {

matrix[i][j] = matrix[i - 1][j - 1];

}

}

return matrix;

}

public static int[][] toCanonicalForm(int[][] matrix, int n, int k) {

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

for (int j = i + 1; j < k; j++) {

if (matrix[i][j] == 1) {

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

matrix[i][l] = (matrix[i][l] + matrix[j][l]) % 2;

}

}

}

}

return matrix;

}

public static int[][] generateCheckMatrix(int[][] generatorMatrix, int n, int k) {

int r = n - k;

int[][] checkMatrix = new int[n][r];

// "Выпиливаем" из генераторной матрицы матрицу, состоящую из последних r столбцов и k строк

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

System.arraycopy(generatorMatrix[i], k, checkMatrix[i], 0, r);

}

// Добавляем снизу диагональную матрицу размера r на r

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

checkMatrix[k + i][i] = 1;

}

return checkMatrix;

}

public static int[][] transposeMatrix(int[][] matrix) {

int rows = matrix.length;

int cols = matrix[0].length;

int[][] transposedMatrix = new int[cols][rows];

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

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

transposedMatrix[j][i] = matrix[i][j];

}

}

return transposedMatrix;

}

public static String calculateRedundantBits(String infoWord, int[][] checkMatrix) {

int r = checkMatrix.length;

String redundantBits = "";

// Умножаем информационное слово на проверочную матрицу

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

int sum = 0;

for (int j = 0; j < infoWord.length(); j++) {

sum += Character.getNumericValue(infoWord.charAt(j)) \* checkMatrix[i][j];

}

// Добавляем полученный бит к избыточным битам

redundantBits += sum % 2;

}

return redundantBits;

}

public static String correctError(String codeWord, String correctRedundantBits, int[][] checkMatrix) {

int n = checkMatrix[0].length;

int r = checkMatrix.length;

// Вычисляем избыточные биты с ошибкой

String errorRedBits = calculateRedundantBits(codeWord, checkMatrix);

// Вычисляем вектор ошибок

String errorVector = "";

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

errorVector += (Character.getNumericValue(errorRedBits.charAt(i)) +

Character.getNumericValue(correctRedundantBits.charAt(i))) % 2;

}

// Ищем столбец в проверочной матрице, который соответствует вектору ошибок

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

String column = "";

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

column += checkMatrix[j][i];

}

// Если нашли соответствующий столбец, исправляем ошибку

if (column.equals(errorVector)) {

// Инвертируем бит в позиции ошибки

char[] codeWordArray = codeWord.toCharArray();

codeWordArray[i] = (codeWordArray[i] == '0') ? '1' : '0';

codeWord = new String(codeWordArray);

break;

}

}

return codeWord;

}

public static String introduceErrors(String word, int numErrors) {

Random rand = new Random();

char[] wordArray = word.toCharArray();

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

int errorPos = rand.nextInt(word.length());

wordArray[errorPos] = (wordArray[errorPos] == '0') ? '1' : '0';

}

return new String(wordArray);

}

}

Листинг 1 – Полный код вспомогательног о класса

public class Main {

public static void main(String[] args) {

final String message = "00001011101000011";

String poly = "1001001"; // Порождающий полином G(x) = x^6 + x^3+ 1 в бинарной форме

int k = 17; // Длина сообщения

int r = 6; // Количество избыточных битов

int n = k + r; // Длина кодового слова

int[][] gMatrix = InfoMetrics.generateMatrix(poly, n, k);

System.out.println("Порождающая матрица G:");

for (int[] row : gMatrix) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

int[][] canonicalGMatrix = InfoMetrics.toCanonicalForm(gMatrix, n, k);

System.out.println("\n\nКаноническая порождающая матрица Gk:");

for (int[] row : canonicalGMatrix) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

int[][] checkMatrix = InfoMetrics.generateCheckMatrix(canonicalGMatrix, n, k);

System.out.println("\n\nПроверочная матрица H:");

for (int[] row : checkMatrix) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

int[][] canonicalCheckMatrix = InfoMetrics.transposeMatrix(checkMatrix);

System.out.println("\n\nКаноническая проверочная матрица Hk:");

for (int[] row : canonicalCheckMatrix) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

System.out.println("Сообщение: " + message);

String redBits = InfoMetrics.calculateRedundantBits(message, canonicalCheckMatrix);

System.out.println("\nИзбыточные биты: " + redBits);

String messageW1Err = InfoMetrics.introduceErrors(message, 1);

String redBits1Err = InfoMetrics.calculateRedundantBits(messageW1Err, canonicalCheckMatrix);

System.out.println("\nИзбыточные биты c 1й ошибкой: " + redBits1Err);

String correctedWord = InfoMetrics.correctError(messageW1Err, redBits, canonicalCheckMatrix);

System.out.println("Сообщение после исправления ошибки: " + correctedWord);

String messageW2Err = InfoMetrics.introduceErrors(message, 2);

String redBits2Err = InfoMetrics.calculateRedundantBits(messageW2Err, canonicalCheckMatrix);

System.out.println("\nИзбыточные биты c 2й ошибками: " + redBits2Err);

String correctedWord2 = InfoMetrics.correctError(messageW2Err, redBits, canonicalCheckMatrix);

System.out.println("Сообщение после исправления ошибки: " + correctedWord2);

}}

Листинг 2 – Код функции выполнения программы