МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ
Кафедра интеллектуальных информационных технологий

Отчет
по дисциплине
«Современные методы защиты информации»
по лабораторной работе № 2
«Избыточное кодирование данных в информационных системах. Итеративные коды»

Выполнил:
студент 4 курса
группы ИИ-22
Полиенко В.Э.
Проверила:
Хацкевич А.С.

Брест 2024

Цель: приобретение практических навыков кодирования/декодирова- ния двоичных данных при использовании итеративных кодов.

Постановка задачи:

Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.

Вариант	k	k_1	k_2	z	Количество групп паритетов
1	16	4	4	-	2, 3
		8	2	-	2, 3
		4	2	2	2, 3, 4, 5
		2	4	2	2, 3, 4, 5

Ход работы

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class IterativeCode {
    private int sizeX, sizeY, sizeZ, length, num_parities;
    private ArrayList<List<Integer>> matrix = new ArrayList<>();
    private String word, wordParity;
    private List<Integer> oldRowParity = new ArrayList<>();
    private List<Integer> oldColumnParity = new ArrayList<>();

    IterativeCode(int sizeX, int sizeY, int sizeZ, int length, int num_parities) {
        this.sizeX = sizeX;
        this.sizeY = sizeY;
        this.sizeZ = sizeZ;
        this.length = length;
        this.num_parities = num_parities;
    }

    public void setMatrix(int num) {
        word = Integer.toBinaryString(num);
        wordParity = word;

        System.out.println(word + " " + word.length());
        int position = 0;
        for (int i = 0; i < sizeY; i++) {
            List<Integer> row = new ArrayList<>();
            for (int j = 0; j < sizeX; j++) {
                if (position < word.length()) {
                    row.add(Character.getNumericValue(word.charAt(position)));
                } else {
                    row.add(0);
                }
                position++;
            }
            matrix.add(row);
        }
        printMatrix(matrix);
        calculate_parities(matrix);
    }

    public void calculate_parities(ArrayList<List<Integer>> matrix) {
        List<Integer> listRow = new ArrayList<>();
        List<Integer> listColumn = new
ArrayList<>(Collections.nCopies(matrix.get(0).size(), 0));
```

```
        for (int i = 0; i < matrix.size(); i++) {
            int rowXor = 0;
            for (int j = 0; j < matrix.get(0).size(); j++) {
                rowXor ^= matrix.get(i).get(j);
                listColumn.set(j, listColumn.get(j) ^ matrix.get(i).get(j));
            }
            listRow.add(rowXor);
        }

        System.out.println("Строки:");
        printList(listRow);
        System.out.println("Колонки:");
        printList(listColumn);

        oldRowParity = new ArrayList<>(listRow);
        oldColumnParity = new ArrayList<>(listColumn);

        int controlBit = 0;

        for (int num : listRow) {
            wordParity += num;
            controlBit ^= num;
        }
        for (int num : listColumn) {
            wordParity += num;
            controlBit ^= num;
        }

        System.out.println("Диагональ вверх");
        calculate_diagonal_parity_up(matrix);
        System.out.println("Диагональ вниз");
        calculate_diagonal_parity_down(matrix);

        listColumn.add(controlBit);

        System.out.println(word + "\n новая матрица\n" + wordParity);
        printMatrix(matrix);
    }

    public void calculate_diagonal_parity_up(ArrayList<List<Integer>> matrix) {
        List<Integer> diagonal_party = new ArrayList<>();
        for (int i = matrix.size() - 1; i >= 0; i--) {
            int row = i, column = 0;
            int Xor = 0;
            while (row < matrix.size() && column < matrix.get(0).size()) {
                Xor ^= matrix.get(row).get(column);
                column++;
                row++;
            }
            diagonal_party.add(Xor);
        }
        printList(diagonal_party);
    }

    public void calculate_diagonal_parity_down(ArrayList<List<Integer>> matrix) {
        List<Integer> diagonal_party = new ArrayList<>();
        for (int i = 0; i < matrix.size(); i++) {
            int row = i, column = 0;
            int Xor = 0;
            while (row > 0 && column < matrix.get(0).size()) {
                Xor ^= matrix.get(row).get(column);
                column++;
                row--;
            }
            diagonal_party.add(Xor);
        }
        printList(diagonal_party);
    }
```

```
    public void introduceError() {
        Random rand = new Random();
        int row = rand.nextInt(sizeY);
        int col = rand.nextInt(sizeX);
        matrix.get(row).set(col, matrix.get(row).get(col) ^ 1);
        System.out.println("Внесена ошибка в позицию: (" + row + ", " + col +
")");
        printMatrix(matrix);
    }

    public void detectAndFixErrors() {
        List<Integer> newRowParity = new ArrayList<>();
        List<Integer> newColumnParity = new
ArrayList<>(Collections.nCopies(matrix.get(0).size(), 0));

        for (int i = 0; i < matrix.size(); i++) {
            int rowXor = 0;
            for (int j = 0; j < matrix.get(0).size(); j++) {
                rowXor ^= matrix.get(i).get(j);
                newColumnParity.set(j, newColumnParity.get(j) ^
matrix.get(i).get(j));
            }
            newRowParity.add(rowXor);
        }

        System.out.println("Новые паритеты строк: " + newRowParity);
        System.out.println("Новые паритеты столбцов: " + newColumnParity);
        System.out.println("Старые паритеты строк: " + oldRowParity);
        System.out.println("Старые паритеты столбцов: " + oldColumnParity);

        int errorRow = -1;
        int errorCol = -1;

        for (int i = 0; i < newRowParity.size(); i++) {
            if (newRowParity.get(i) != oldRowParity.get(i)) {
                errorRow = i;
                break;
            }
        }

        for (int j = 0; j < newColumnParity.size(); j++) {
            if (newColumnParity.get(j) != oldColumnParity.get(j)) {
                errorCol = j;
                break;
            }
        }
        if (errorRow != -1 && errorCol != -1) {
            System.out.println("Обнаружена ошибка в позиции: (" + errorRow + ", "
+ errorCol + ")");
            matrix.get(errorRow).set(errorCol, matrix.get(errorRow).get(errorCol)
^ 1);
            System.out.println("Ошибка исправлена!");
        } else {
            System.out.println("Ошибок не обнаружено.");
        }

        printMatrix(matrix);
    }

    public void printList(List<Integer> array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    public void printMatrix(ArrayList<List<Integer>> matrix) {
```

```
        for (List<Integer> row : matrix) {
            for (int num_row : row) {
                System.out.print(num_row + " ");
            }
            System.out.println();
        }
    }

}
```

Вывод программы:

```
1011010011100011 16
1 0 1 1
0 1 0 0
1 1 1 0
0 0 1 1
Строки:
1 1 1 0
Колонки:
0 0 1 0
Диагональ вверх
0 1 0 0
Диагональ вниз
0 0 0 1
1011010011100011
 новая матрица
10110100111000111110010
1 0 1 1
0 1 0 0
1 1 1 0
0 0 1 1
Внесена ошибка в позицию: (0, 3)
1 0 1 0
0 1 0 0
1 1 1 0
0 0 1 1
```

```
Новые паритеты строк: [0, 1, 1, 0]
Новые паритеты столбцов: [0, 0, 1, 1]
Старые паритеты строк: [1, 1, 1, 0]
Старые паритеты столбцов: [0, 0, 1, 0]
Обнаружена ошибка в позиции: (0, 3)
Ошибка исправлена!
1 0 1 1
0 1 0 0
1 1 1 0
0 0 1 1
```

Вывод: разработали приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.