Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 4

По дисциплине СМЗКС

Тема: «Избыточное кодирование данных в информационных системах. Итеративные коды»

Выполнил:

Студент 4 курса

Группы ИИ-16

Журавлёв В.А.

Проверил:

Хацкевич М.В.

Брест, 2022

**Цель работы:** приобрести практические навыки кодирования/декодирования двоичных данных при использовании итеративных кодов.

**Задание:**

K= 24

K1=6

K2=2

Z=2

**Код программы:**

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

System.out.print("Введите текст: ");

Scanner scanner = new Scanner(System.in);

String text = scanner.nextLine();

int[][][] matrices = parseText(text);

System.out.println("Введенный текст представлен ввиде: ");

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 6; j++) {

System.out.println(Arrays.toString(matrices[i][j]));

}

System.out.println("---------");

}

//вертикальные паритеты для каждой из матриц

int xv[][] = getVerticalParitets(matrices);

System.out.println("Вертикальные паритеты: ");

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

System.out.println(Arrays.toString(xv[i]));

}

//горизонтальные паритеты для каждой из матриц

int xh[][] = getHorizontalParitets(matrices);

System.out.println("Горизонтальные паритеты: ");

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

System.out.println(Arrays.toString(xh[i]));

}

//z-паритеты

int xz[][] = getZParitets(matrices);

System.out.println("z-паритеты: ");

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

System.out.println(Arrays.toString(xz[i]));

}

System.out.print("Введите ошибки (z1,y1,x1;z2,y2,x2...): ");

String errorsLine = scanner.nextLine();

int[][][] matricesWithErrors = addErrors(matrices, errorsLine);

int newxv[][] = getVerticalParitets(matricesWithErrors);

int newxh[][] = getHorizontalParitets(matricesWithErrors);

int newxz[][] = getZParitets(matricesWithErrors);

List<VerticalError> verticalErrors = findVerticalErrors(xv, newxv);

List<HorizontalError> horizontalErrors = findHorizonralErrors(xh, newxh);

List<ZError> zErrors = findZErrors(xz, newxz);

System.out.println("--------------------");

System.out.println(verticalErrors);

System.out.println(horizontalErrors);

System.out.println(zErrors);

System.out.println("--------------------");

List<Error> errors = tryToFindErrors(verticalErrors, horizontalErrors, zErrors);

System.out.println("Ошибки найдены в координатах: " + errors);

restoreText(matricesWithErrors, errors);

}

private static String restoreText(int[][][] matricesWithErrors, List<Error> errors) {

String textBeforeRestore = matrixToText(matricesWithErrors);

System.out.println("Текст до восстановления: " + textBeforeRestore);

for (int i = 0; i < errors.size(); i++) {

Error error = errors.get(i);

matricesWithErrors[error.getZ()][error.getY()][error.getX()] ^= 1;

}

String textAfterRestore = matrixToText(matricesWithErrors);

System.out.println("Текст после восстановления: " + textAfterRestore);

return textAfterRestore;

}

private static String matrixToText(int[][][] matricesWithErrors) {

String binary = "";

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

for (int j = 0; j < matricesWithErrors[i].length; j++) {

for (int k = 0; k < matricesWithErrors[i][j].length; k++) {

binary += matricesWithErrors[i][j][k];

}

}

}

String result = "";

for (int i = 0; i < binary.length()/8; i++) {

char c = 0;

for (int j = 0; j < 8; j++) {

int binaryInt = Integer.parseInt(String.valueOf(binary.charAt(j + i \* 8)));

c += Math.pow(2, 7 - j) \* binaryInt;

}

result += c;

}

return result;

}

private static List<Error> tryToFindErrors(List<VerticalError> verticalErrors, List<HorizontalError> horizontalErrors, List<ZError> zErrors) {

List<Error> errors = new ArrayList<>();

List<ErrorCandidate> errorCandidates = new ArrayList<>();

for (int i = 0; i < verticalErrors.size(); i++) {

for (int j = 0; j < horizontalErrors.size(); j++) {

VerticalError verticalError = verticalErrors.get(i);

HorizontalError horizontalError = horizontalErrors.get(j);

if(verticalError.getZ() == horizontalError.getZ()){

ErrorCandidate errorCandidate = new ErrorCandidate();

errorCandidate.setX(verticalError.getX());

errorCandidate.setY(horizontalError.getY());

errorCandidate.setZ(verticalError.getZ());

errorCandidates.add(errorCandidate);

}

}

}

System.out.println("Кандидаты для ошибок: ");

System.out.println(errorCandidates);

System.out.println("-----------------");

for (int i = 0; i < errorCandidates.size(); i++) {

for (int j = 0; j < zErrors.size(); j++) {

ErrorCandidate errorCandidate = errorCandidates.get(i);

ZError zError = zErrors.get(j);

if(errorCandidate.getX() == zError.getX() && errorCandidate.getY() == zError.getY()){

Error error = new Error();

error.setX(errorCandidate.getX());

error.setY(errorCandidate.getY());

error.setZ(errorCandidate.getZ());

errors.add(error);

}

}

}

return errors;

}

private static List<ZError> findZErrors(int[][] xz, int[][] newxz) {

List<ZError> result = new ArrayList<>();

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

for (int j = 0; j < xz[i].length; j++) {

if(xz[i][j] != newxz[i][j]){

ZError zError = new ZError();

zError.setX(j);

zError.setY(i);

result.add(zError);

}

}

}

return result;

}

private static List<HorizontalError> findHorizonralErrors(int[][] xh, int[][] newxh) {

List<HorizontalError> result = new ArrayList<>();

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

for (int j = 0; j < xh[i].length; j++) {

if(xh[i][j] != newxh[i][j]){

HorizontalError horizontalError = new HorizontalError();

horizontalError.setZ(i);

horizontalError.setY(j);

result.add(horizontalError);

}

}

}

return result;

}

private static List<VerticalError> findVerticalErrors(int[][] xv, int[][] newxv) {

List<VerticalError> result = new ArrayList<>();

for (int i = 0; i < 2; i++) {

for (int j = 0; j < xv[i].length; j++) {

if(xv[i][j] != newxv[i][j]){

VerticalError verticalError = new VerticalError();

verticalError.setZ(i);

verticalError.setX(j);

result.add(verticalError);

}

}

}

return result;

}

private static int[][][] addErrors(int[][][] matrices, String errors) {

int[][][] result = new int[2][6][2];

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 6; j++) {

for (int k = 0; k < 2; k++) {

result[i][j][k] = matrices[i][j][k];

}

}

}

String[] coordinates = errors.split(";");

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

String[] coordinate = coordinates[i].split(",");

int z = Integer.parseInt(coordinate[0]);

int y = Integer.parseInt(coordinate[1]);

int x = Integer.parseInt(coordinate[2]);

result[z][y][x] ^= 1;

}

return result;

}

private static int[][] getZParitets(int[][][] matrices) {

int[][] result = new int[6][2];

for (int i = 0; i < 6; i++) {

for (int j = 0; j < 2; j++) {

result[i][j] = matrices[0][i][j]^matrices[1][i][j];

}

}

return result;

}

private static int[][] getHorizontalParitets(int[][][] matrices) {

int[][] result = new int[2][6];

for (int i = 0; i < 2; i++) {

int[][] matrix = matrices[i];

for (int j = 0; j < 6; j++) {

result[i][j] = matrix[j][0] ^ matrix[j][1];

}

}

return result;

}

private static int[][] getVerticalParitets(int[][][] matrices) {

int[][] result = new int[2][2];

for (int i = 0; i < 2; i++) {

int[][] matrix = matrices[i];

for (int j = 0; j < 2; j++) {

int xor = matrix[0][j];

for (int k = 1; k < 6; k++) {

xor ^= matrix[k][j];

}

result[i][j] = xor;

}

}

return result;

}

// 0 0

// 1 1

// 1 1

// 1 0

// 1 0

// 0 1

public static int[][][] parseText(String text) {

int[][][] result = new int[2][6][2];

String binaryText = "";

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

char c = text.charAt(i);

String binaryChar = Integer.toBinaryString(c);

while (binaryChar.length() < 8) {

binaryChar = "0".concat(binaryChar);

}

// System.out.println(c + " - " + binaryChar);

binaryText += binaryChar;

}

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 6; j++) {

for (int k = 0; k < 2; k++) {

result[i][j][k] = Integer.parseInt(String.valueOf(binaryText.charAt(i \* 12 + j \* 2 + k)));

}

}

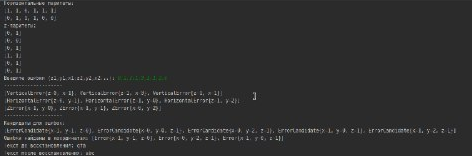
}

return result;

}

}

**Результаты работы программы:**

****

**Вывод:** приобрел практические навыки кодирования/декодирования двоичных данных при использовании итеративных кодов.