Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

«Защита информации и надёжность информационных систем»

Отчёт по лабораторной работе №9

СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ ЛЕМПЕЛЯ−ЗИВА

Выполнил: Лешук Д. И.

ФИТ 3 курс 7 группа

Преподаватель: Николайчук А.Н

Минск 2024

**Цель:** приобретение практических навыков использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных.

**Практическое задание:**

1. Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n1, n2).

2. С помощью приложения выполнить прямое и обратное преобразования произвольного текста длиной несколько килобайт.

Формат представления параметров p и q выбрать по указанию преподавателя.

3. Изменяя размеры окон, оценить скорость и эффективность (используя соотношения на с. 76) выполнения операций сжатия/распаковки.

4. Результаты оформить в виде отчета по установленным правилам.

**Выполнение работы:**

Определим класс LZ77 который будет функционально реализовывать сжатие и распаковку данных методом Лемпеля-Зива. Определим два константных поля: MAX\_WINDOW\_SIZE и MAX\_BUFFER\_SIZE, которые будут определять размеры словаря и буфера – при необходимости их значения можно менять перед запуском программы. Далее нам нужны основные функции с которыми придётся взаимодействовать – сжатия, распаковки. Функция сжатия будет вызывать ещё одну функцию – определения триады на данном шаге. Полный код данного класса размещён в приложении А. Для случайно взятой бинарной последовательности длиной 64 бита процесс и результат кодирования имеет следующий вид – рисунок 1.

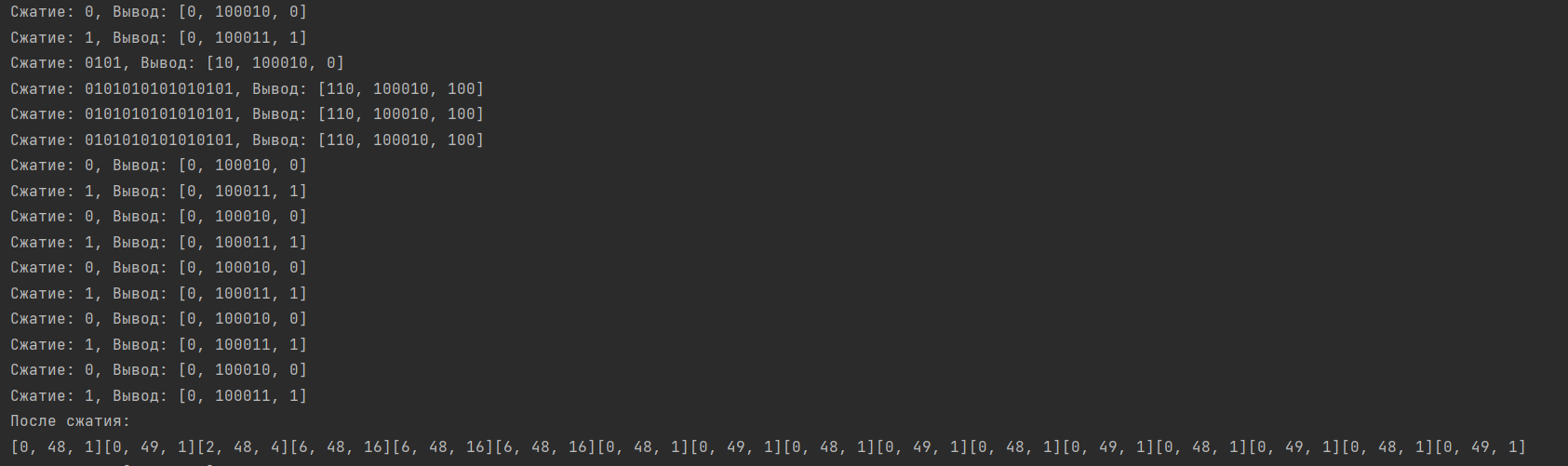


Рисунок 1 – Сжатие сообщения

Для удобства в итоговом представлении в триадах бинарный формат заменён на десятичный. Далее проведём процесс декодирования – рисунок 2.

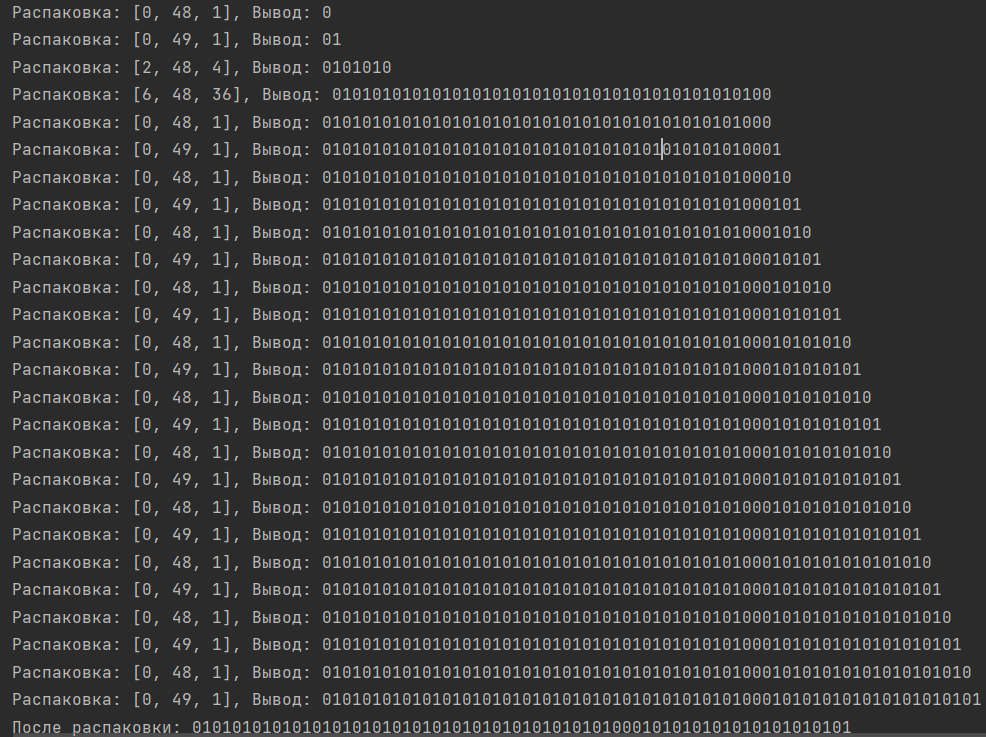


Рисунок 2 – Функция кодирования исходного сообщения

После распаковки можем сделать вывод, что сообщение успешно прошло данный процесс.

Теперь предположим как будут меняться значения R1 и R2 для разных наборов n1 и n2 и длины сообщения k – с увеличением n1 и n2 будет увеличиваться количество символов в каждой триаде ,соответственно будет увеличиваться количество символов в сжатом виде. При n1=n2=64 : ; : . При n1=n2=16: : ; :

**Вывод:** В процессе выполнения данной лабораторной работы были применён на практике метод сжатия Лемпеля-Зива к входной бинарной последовательности с разными значениями длины буфера и словаря, а также вычислены степень сжатия и качество сжатия.

**Приложение А**

import java.util.\*;

public class LZ77 {

private static final int MAX\_WINDOW\_SIZE = 16;

private static final int MAX\_BUFFER\_SIZE = 16;

public static List<int[]> compress(String input) {

List<int[]> output = new ArrayList<>();

int i = 0;

while (i < input.length()) {

int[] longestMatch = findLongestMatch(input, i);

output.add(longestMatch);

System.out.println("Сжатие: " + input.substring(i, i + longestMatch[2]) + ", Вывод: [" +

Integer.toBinaryString(longestMatch[0]) + ", " +

Integer.toBinaryString(longestMatch[1]) + ", " +

Integer.toBinaryString(longestMatch[2]) + "]");

i += longestMatch[2];

}

return output;

}

public static String decompress(List<int[]> compressedInput) {

StringBuilder output = new StringBuilder();

for (int[] part : compressedInput) {

int nextCharIndex = output.length() - part[0];

for (int i = 0; i < part[2] && nextCharIndex + i < output.length(); i++) {

output.append(output.charAt(nextCharIndex + i));

}

output.append((char) part[1]);

System.out.println("Распаковка: " + Arrays.toString(part) + ", Вывод: " + output);

}

return output.toString();

}

private static int[] findLongestMatch(String data, int currentPos) {

int endOfBuffer = Math.min(currentPos + MAX\_BUFFER\_SIZE, data.length());

int bestMatchDistance = -1;

int bestMatchLength = -1;

for (int j = currentPos + 2; j <= endOfBuffer; j++) {

int startOfWindow = Math.max(0, currentPos - MAX\_WINDOW\_SIZE);

String currentString = data.substring(currentPos, Math.min(j, data.length()));

for (int i = startOfWindow; i < currentPos; i++) {

int repetitions = currentPos - i;

String repeatedString = new String(new char[repetitions]).replace("\0", data.substring(i, currentPos));

if (repeatedString.length() > bestMatchLength && repeatedString.contains(currentString)) {

bestMatchDistance = currentPos - i;

bestMatchLength = currentString.length();

}

}

}

if (bestMatchDistance == -1 || currentPos + bestMatchLength >= data.length()) {

return new int[] {0, data.charAt(currentPos), 1};

} else {

return new int[] {bestMatchDistance, data.charAt(currentPos + bestMatchLength), bestMatchLength};

}

}

Листинг 1 – Полный код вспомогательног о класса

import java.util.Arrays;

import java.util.List;

public class Main {

public static void main(String[] args) {

String testString = "0101010101010101010101010101010101010101010101010101010101010101";

List<int[]> compressed = LZ77.compress(testString);

System.out.println("После сжатия: ");

for (int[] array : compressed) {

System.out.print(Arrays.toString(array));

}

System.out.println();

String decompressed = LZ77.decompress(compressed);

System.out.println("После распаковки: " + decompressed);

System.out.println(decompressed != testString);

}

}

Листинг 2 – Код функции выполнения программы