МИНОБРНАУКИ РОССИИ САНКТ-ПЕТЕРБУРГКСКИЙ
ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра информационной безопасности

КУРСОВАЯ РАБОТА

По дисциплине «Алгоритмы и структуры данных»

Тема: Алгоритмы повышенной сложности

Студент гр 1361 _____ Горбунова Д.А.
Преподаватель _____ Беляев А.В.

Санкт-Петербург

2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Вариант R2. Сжатие изображений алгоритмом RLE оптимальным фиксированным кодом (L2).

Реализовать программу, осуществляющую сжатие черно-белого (глубина цвета – 1 бит) изображения и его восстановление по алгоритму RLE (поскольку кодируются только черные и белые пикселы нет необходимости передавать цвет пиксела – они считаются строго чередующимися, записывается только длина серии). Для сжатия рекомендуется выбрать отсканированный документ формата A4, изменить его размер до 600-1000 пикселей в высоту, преобразовать его глубину цвета в 1 бит, хранить в формате BMP. Рекомендуемое кодирование непрерывной последовательности пикселей: 4 бита (длина – 0...15 пикселей) или 8 бит (длина – 0...255 пикселей) – на усмотрение разработчика.

Модифицировать базовый вариант кода RLE следующим образом:

- на этапе анализа изображения вычислять длину файла результата в зависимости от длины кода (перебрать длины от 3 до 24 бит);
- выбрать для кодирования длину кода, дающего минимальный размер файла; первым байтом файла записать выбранную длину кода;
- на этапе декодирования считать первый байт и применить данную длину кода для декодирования.

Для разработки программы был выбран язык программирования Java.

Содержание пояснительной записки:

Введение, описание реализуемого алгоритма, результаты тестирования программы, заключение, список использованной литературы, приложение 1 – руководство пользователя, приложение 2 – исходный код программы.

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 05.11.2022

Дата сдачи реферата: 28.12.2022

Дата защиты реферата: 29.12.2022

Студент гр 1361 _____ Горбунова Д.А.

Преподаватель _____ Беляев А.В.

АННОТАЦИЯ

В работе представлена программа, производящая сжатие и восстановление изображение определенным алгоритмом. Программа реализована на языке java. Для сжатия изображения необходимо запустить программу и ввести в консоль номер команды.

SUMMARY

This paper presents a program that performs image compression and restoration using a specific algorithm. The program is implemented in java language. To compress an image, it is necessary to run the program and enter the command number into the console.

ВВЕДЕНИЕ

Целью курсовой работы является написание программы, которая будет оптимально сжимать изображение, заданное пользователем, и выдавать в консоль результируемый размер файла.

1. ОПИСАНИЕ РЕАЛИЗУЕМЫХ АЛГОРИТМОВ

Алгоритм RLE (run-length-encoding) — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов. Серией называется последовательность, состоящая из нескольких одинаковых символов. При кодировании (упаковке, сжатии) строка одинаковых символов, составляющих серию, заменяется строкой, содержащей сам повторяющийся символ и количество его повторов.

Другими словами, алгоритм берёт последовательности одинаковых элементов, и представляет их в виде пар «количество/значение». Например, строка вида «AAAAAABCCC» может быть преобразована в запись вроде «6×A, B, 3×C».

Длина кода в RLE — это длина последовательности бит, которая кодирует длину непрерывной последовательности пикселей одного цвета. Например, если пиксели были расположены в порядке ЧЧЧБББББЧЧ (где Ч – пиксель черного цвета, Б – белого), то в сжатом формате алгоритм передает последовательность 3-4-2.

Если длина кода равна трём (то есть алгоритм может кодировать длины кода от нуля до семи), до закодированный вид будет: 011-100-010. Если же длина кода равна пяти (алгоритм может кодировать длины кода от нуля до тридцати одного), то закодированный вид будет: 00011-00100-00010.

В зависимости от длины кода будет меняться и размер закодированного файла. Если длина кода для файла будет оптимальной, то размер будет минимальным.

2. ОПИСАНИЕ РЕАЛИЗУЕМОЙ ПРОГРАММЫ

BitIterator.java нужен для последовательного чтения нулей и единиц из массива байт.

BitRowConstructor.java нужен для последовательной записи нулей и единиц в массив байт

SeriaIterator.java нужен для чтения битов из массива байт наборами заданной длины. В своей работе использует BitIterator.java.

Главным классом программы является Compressor.java. В зависимости от выбора пользователя класс запускает метод сжатия или метод декомпрессии изображения.

При сжатии изображения, графическая часть выделяется в массив байт. Далее производится подсчет длин серий одинаковых битов, не привязанный к длине кода. Длины серий анализируются, на основе анализа выделяется оптимальная длина кода. Затем длины серий кодируются и записываются в файл в формате lra. Первый байт файла содержит длину кода, байты номер 2, 3, 4, 5 содержат ширину изображения, 6, 7, 8, и 9-й байты содержат высоту изображения. Байты, хранящие ширину и высоту, располагаются в соответствии с порядком байтов little endian.

При декомпрессии изображения считывается длина кода, ширина и высота изображения, а также часть, содержащая сжатые графические данные. На основе сжатых графических данных формируются массив длин серий. Этот массив используется для формирования непрерывной строки с восстановленными графическими данными. Далее строка разбивается на подстроки и форматируется таким образом, чтобы её длина была кратна 4-м байтам (недостающие биты дописываются в конец строк). Рассчитывается размер графической части и размер файла, после чего эти размеры, а также высота, ширина и графические данные чередуются с предзаданными наборами байт, характерными для bmp-файлов с глубиной цвета 1bit, располагаются в определенном порядке и записываются в bmp файл.

3. РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

Исходный файл в формате BMP изображен на рисунке 1.

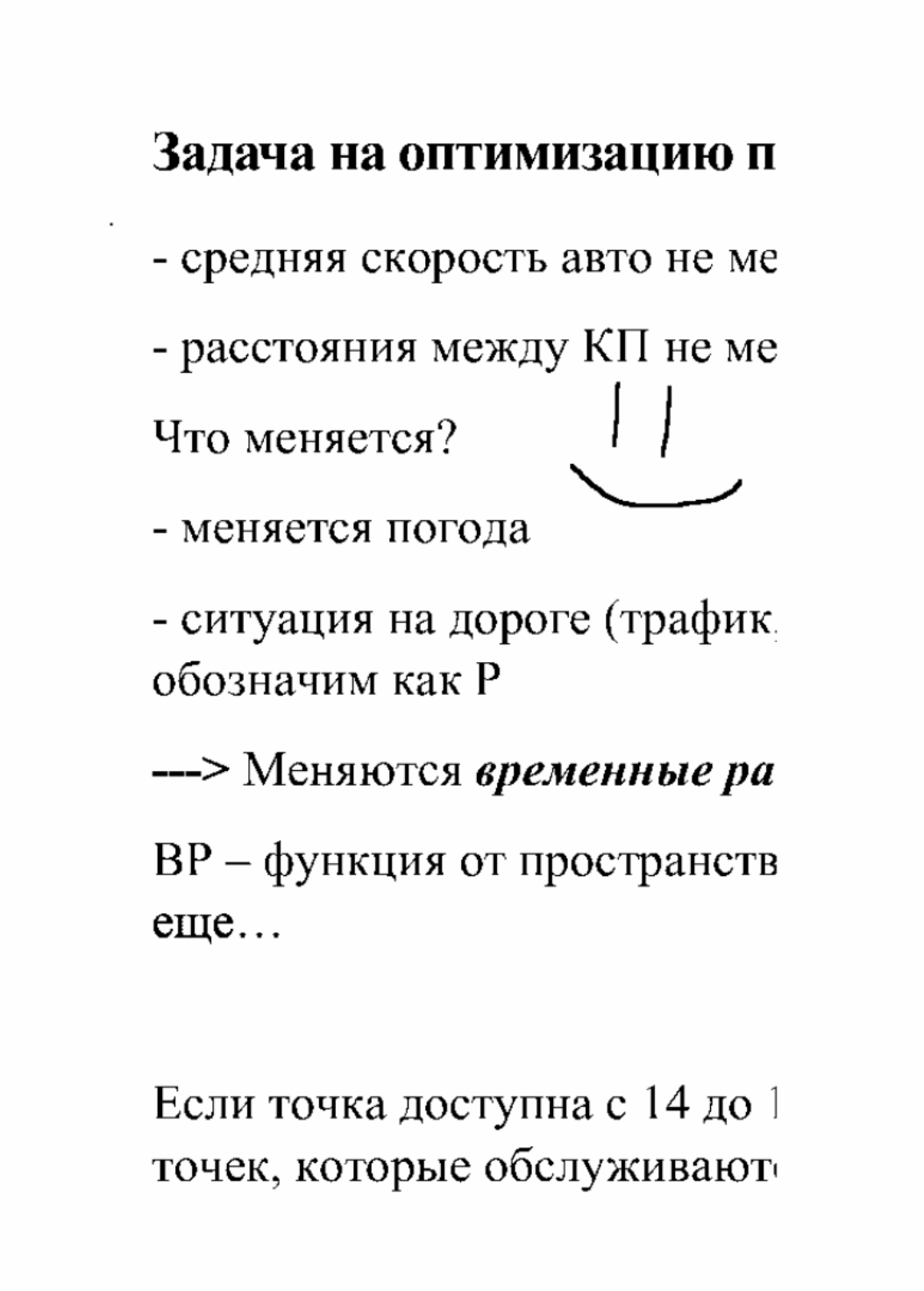

Рисунок 1 – исходное изображение

Вывод программы об оптимальной длине кода и о количестве символов. (рисунок 2)

```
Минимальный размер кода 157041bit при длине кода 9bit
```

Рисунок 2 – Вывод программы

Восстановленный в результате работы алгоритма декодирования файл изображен на рисунке 3.

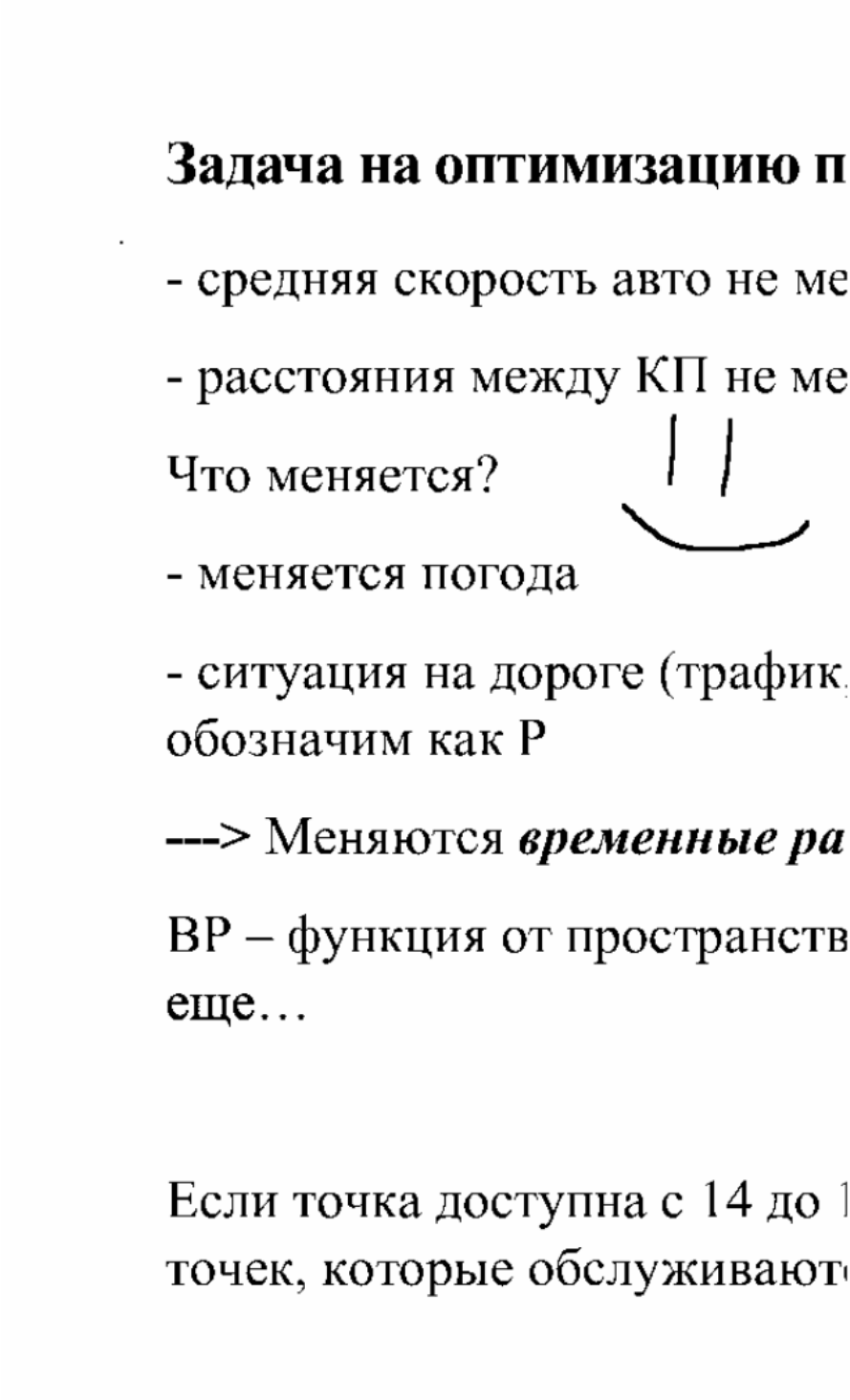

Рисунок 3 – восстановленный файл

Восстановленный файл идентичен исходному, что говорит о правильной работе алгоритма.

ЗАКЛЮЧЕНИЕ

В ходе подготовки к написанию курсовой работы были изучены алгоритм RLE и структура построение изображений типа bmp.

Результатом курсовой работы является программа на языке java, реализующая сжатие изображение оптимально на столько, на сколько это возможно.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- ✓ https://ru.bmstu.wiki/BMP_(Bitmap_Picture) (Дата обращение 15.12.2022)
- ✓ http://kunegin.com/ref1/code/6.htm (Дата обращения 20.12.2022)
- ✓ https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%8F%D0%B4%D0%BE%D0%BA_%D0%B1%D0%B0%D0%B9%D1%82%D0%BE%D0%B2 (Дата обращение 22.12.2022)

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Compressor.java:

Элемент кода, реализующий сжатие файла:

```
private static void compressAndWrite(int[][] arr, long bitRowWidth, long height){
    ArrayList<Long> unsizedSeriasLengths = new ArrayList<>();
    boolean currentSymbol = true;
    long currentSeriaLength = 0;
    boolean isFirstWhite = ((arr[0][0] >>> 7) & 1) == 1;

    for (int i = 0; i < arr.length; i++) {
        int j = 0;
        var iterator = new BitIterator(arr[i]);
        while (iterator.hasNext() && (bitRowWidth > j++)) {
            boolean bit = iterator.next();
            if (bit == currentSymbol) {
                currentSeriaLength++;
            } else {
                unsizedSeriasLengths.add(currentSeriaLength);
                currentSeriaLength = 1;
                currentSymbol = currentSymbol ? false : true;
            }
        }
    }
    unsizedSeriasLengths.add(currentSeriaLength);

    System.out.println("Длины серий:");
    System.out.println(unsizedSeriasLengths);
    int[] possibleCodeSizes = new int[25];
    for (int i = 3; i <= 24; i++) {
        possibleCodeSizes[i] = getEncodingBitLengthBy(i, unsizedSeriasLengths, isFirstWhite);
    }
    int minIndex = 3;
```

```
        int min = possibleCodeSizes[3];
        for (int i = 4; i <= 24; i++) {
            if (possibleCodeSizes[i] < min) {
                minIndex = i;
                min = possibleCodeSizes[i];
            }
        }
        int codeLength = minIndex;
        System.out.println("Первый пиксель " + (isFirstWhite ? "" : "не ") + "белый");
        System.out.println("Минимальный размер кода " + min + "bit при длине кода " + codeLength + "bit");
        long maxValue =
                getMaxValueForCodeLength(codeLength);
        var brc = new BitRowConstructor(min, codeLength);
        for (int i = 0; i < unsizedSeriasLengths.size(); i++) {
            int seriaSize = unsizedSeriasLengths.get(i).intValue();
            if (seriaSize > 0) {
                seriaSize -= maxValue;
                while (seriaSize > 0) {      //пока серия не доразбита пишем две серии (maxValue и 0)
                    brc.write((int) maxValue); //
                    brc.write(0);
                    seriaSize -= maxValue;
                }
                if (seriaSize != 0) {
                    brc.write((int) (seriaSize + maxValue));
                }
            } else {
                brc.write(0);
            }
        }
        writeCompressed(codeLength, bitRowWidth, height, brc.getBytesAndFinish());
    }
```

Элемент кода, реализующий восстановление файла:

```
    private static void decompressAndWrite(int[] compressedSerias, int codeLength, long width, long height) {
```

```
BitIterator bi = new BitIterator(compressedSerias);
SeriaIterator si = new SeriaIterator(bi, codeLength);
ArrayList<Integer> decompressedSerias = new ArrayList<>();
while (si.hasNext()) {
    decompressedSerias.add(si.next());
}
System.out.println("Восстановленные длины серий (чередование белый-черный-белый...):");
System.out.println(decompressedSerias);

BitRowConstructor brc1 = new BitRowConstructor((int) (width * height), 1);
int color = 1;
for (Integer ser : decompressedSerias) { //пишем цвет в строку нужное кол-во раз
    for (int i = 0; i < ser.intValue(); i++) {
        brc1.write(color);
    }
    color = color == 1 ? 0 : 1; //переключаем цвет
}
var decompressedMatrix = brc1.getBytesAndFinish();

System.out.println("Восстановленные данные:");
BitIterator bi1 = new BitIterator(decompressedMatrix);
int fWidth = formatedLengthOf((int) width);
int[][] splitedMatrix = new int[(int) height][fWidth];
for (int i = 0; i < height; i++) {
    BitRowConstructor rc = new BitRowConstructor(splitedMatrix[i], 1);
    for (int j = 0; j < width; j++) {
        rc.write(bi1.next());
    }
    var row = rc.getBytesAndFinish();
}
long rastrsize = height * fWidth;
int[] rastrsizeArr = longToByteArrayLittleEndian(rastrsize);
int[] wArr = longToByteArrayLittleEndian(width);
int[] hArr = longToByteArrayLittleEndian(height);
```

```
            int[] filesizeArr = longToByteArrayLittleEndian(rastrsize + 62);
            writeDecompressed(filesizeArr, wArr, hArr, rastrsizeArr, splitedMatrix);
        }
```

Файл SeriaIterator.java:

```
    public class SeriaIterator {
        BitIterator bi;
        int cl;
        private boolean hasNext = true;
        public SeriaIterator(final BitIterator bi, int codeLength) {
            this.bi = bi;
            this.cl = codeLength;
        }
        public boolean hasNext() {
            return hasNext;
        }

        public int next() { //getSeriaLength
            int val = 0;
            for (int i = 0; i < cl; i++) {
                val = val << 1;
                if (!bi.hasNext()) {
                    hasNext = false;
                    return 0;
                }
                int bit = bi.next() ? 1 : 0;
                val = val | bit;
            }
            return val;
        }
}
```

Файл BitRowConstructor.java

```
    public class BitRowConstructor {
        private int[] bytes;
```

```
private int byteIndex = 0;
private int bitIndexToRecord = 0;

boolean workIsDone = false;

private int codeLength;
public BitRowConstructor(int sizeBits, int codeLength) {
    bytes = new int[sizeBits/8 + ((sizeBits%8)>0?1:0)];
    this.codeLength = codeLength;
}

public BitRowConstructor(int[] arr, int codeLength) {
    bytes = arr;
    this.codeLength = codeLength;
}
public void write(int value){
    for (int i = 1; i <= codeLength; i++) {
        int bit = (value>>(codeLength-i))&1;
        bytes[byteIndex] = bytes[byteIndex]|bit;
        bitIndexToRecord++;
        if(bitIndexToRecord > 7){
            byteIndex++;
            bitIndexToRecord =0;
        }else{
            bytes[byteIndex] = bytes[byteIndex]<<1;
        }
    }
}
public void write(boolean value){
    for (int i = 1; i <= codeLength; i++) {
        int bit = value?1:0;
        bytes[byteIndex] = bytes[byteIndex]|bit;
        bitIndexToRecord++;
        if(bitIndexToRecord > 7){
```

```
                    byteIndex++;
                    bitIndexToRecord =0;
                }else{
                    bytes[byteIndex] = bytes[byteIndex]<<1;
                }
            }
        }
        public int[] getBytesAndFinish(){
            if(!workIsDone){
                if (byteIndex<bytes.length){
                    bytes[byteIndex]=bytes[byteIndex]<<(7 - bitIndexToRecord); //заполняет пустые биты
                }
                workIsDone=true;
                return bytes;
            }else return null;
        }
}
```

Файл BitIterator.java:

```
import java.util.Iterator;

public class BitIterator implements Iterator<Boolean> {

    private int byteLength;
    private byte bitIndex;
    private int byteIndex;
    private static final byte b00000001 = 1;

    private int[] arr;

    public BitIterator(int... bytes) {
        byteLength = bytes.length;
        bitIndex = 0;
        byteIndex = 0;
```

```
            arr = new int[byteLength];
            for (int i = 0; i < byteLength; i++) {
                arr[i] = bytes[i];
            }
        }

        @Override
        public boolean hasNext() {
            return (byteIndex<byteLength) && bitIndex<8;
        }

        @Override
        public Boolean next() {
            int value = (arr[byteIndex]>>>(7-bitIndex))&b00000001;
            if(bitIndex==7){
                 byteIndex++;
                 bitIndex=0;
            }else bitIndex++;
            return value==1;
        }
}
```