Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ПМиК

РАСЧЁТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ

по дисциплине «Представление графической информации»

Тема: «Конвертор количества цветов в изображении.»

Вариант №2.

Выполнил: студент группы ИП-815

Сафонов Руслан Викторович

Проверил: доцент кафедры ПМиК   
Перцев Игорь Владимирович

Новосибирск 2022

**Оглавление**

1. Текст задания...................................................................................................3

2. Описание алгоритма работы программы......................................................4

3. Текст программы…….....................................................................................6

4. Результаты выполнения задания..................................................................16

**ТЕКСТ ЗАДАНИЯ**

Написать программу-конвертор количества цветов в изображении.

Предлагаемый алгоритм. Для уменьшения количества цветов выбираются наиболее часто встречаемые цвета в исходном изображении. Причем эти цвета не должны быть слишком похожими друг на друга. Для сравнения цветов вычисляются разности между RGB составляющими.

*Delta=(R1-R2)2 + (G1-G2)2+ (B1-B2)2*

После формирования новой палитры цвета заменяются на наиболее похожие из записанных в палитру.

Можно использовать любой другой алгоритм преобразования цветов (например медианного сечения) главное требование – алгоритм должен быть реализован самостоятельно.

Программа должны выводить изображение на экран до и после конвертирования.

Требования к оформлению отчета

Отчет должен включать в себя:

1. Титульный лист, вариант задания.
2. Текст задания
3. Описание алгоритма работы программы.
4. Текст программы
5. Скриншот с результатами.

Так же необходимо предоставить все файлы, относящиеся к проекту, в том числе EXE-файл

Номер варианта контрольной работы определяется по последней цифре номера по списку группы

1. Преобразовать 256-цветный BMP файл в 16-цветный BMP файл.
2. Преобразовать True Color BMP файл в 16-цветный BMP файл.
3. Преобразовать True Color BMP файл в 256-цветный BMP файл.
4. Преобразовать True Color PCX файл в 16-цветный PCX файл.
5. Преобразовать True Color PCX файл в 256-цветный PCX файл.
6. Преобразовать 256-цветный PCX файл в 16-цветный PCX файл.
7. Преобразовать 256-цветный BMP файл в 16-цветный PCX файл.
8. Преобразовать 256-цветный PCX файл в 16-цветный BMP файл.
9. Преобразовать True Color BMP файл в 256-цветный PCX файл.
10. Преобразовать True Color PCX файл в 256-цветный BMP файл.

**ОПИСАНИЕ АЛГОРИТМА РАБОТЫ ПРОГРАММЫ**

В данной работе был использован метод квантования цветов медианным сечением. Далее будет приведено описание данного метода по шагам:

1. Метод квантования цветов с помощью медианного сечения применяется при выборе 256 цветов, чтобы представить полноцветное изображение, содержащее несколько тысяч цветов. Чтобы понять, как работает метод медианного сечения, мы представим цветовое пространство как куб. Каждая ось соответствует одному из трех основных цветов и деления на оси нумеруются от 0 до 255; большему номеру соответствует большая интенсивность цвета. Цвета в изображении отмечаются внутри куба так же, как точки на трехмерном графике.

2. Первый шаг состоит в отсечении "краёв" куба, которые не содержат пикселов. К примеру, если у всех пикселов значения красной компоненты не меньше, чем 8 и не больше, чем 250, то отбрасываются части куба от К="0" до К="7" и от К="251" до К="255."

3. Второй шаг заключается в разрезании полученного параллелепипеда на два в срединной точке (медиане) самой длинной стороны. Если самая длинная сторона параллельна оси B (Синий цвет), то компьютер выбирает срединное синее значение из всех синих значений, представленных в параллелепипеде (к примеру, 50-ое вхождение в отсортированном списке из 100 синих значений) и разрезает в этой точке. Теперь параллелепипед разделен на два параллелепипеда меньшего размера, содержащих одинаковое количество пикселов.

4. Весь предыдущий процесс отсечение пустых "краев" и разрезание самой длинной стороны в срединной точке повторяется для двух меньших параллелепипедов. Теперь исходный куб разделен на четыре параллелепипеда, содержащих приблизительно одинаковое количество пикселов.

5. Медианное сечение повторно применяется для того, чтобы разделить куб на 8, 16, 32, 64, 128 и 256 параллелепипедов. Они содержат примерно одно и то же количество пикселов и их объемы обратно пропорциональны плотностям пикселов.

6.Имея пространство, разделенное таким образом, легко выбрать палитру. Каждый из 256 параллелепипедов содержит пикселы приблизительно одинакового цвета и центр каждого параллелепипеда представляет оптимальное значение цвета для палитры. Имея координаты вершин, очень просто вычислить координаты центральной точки (вместо того, чтобы вычислять центральную точку, усредняем значения всех пикселов, находящихся внутри параллелепипеда; на вычисления уйдет больше времени, но полученная палитра будет лучше). Вычислив R, G и B координаты для всех 256 центральных точек в параллелепипедах, получим 256 цветов, которые и будут составлять палитру.

**ТЕКСТ ПРОГРАММЫ**

**ColorCube.java**

public class ColorCube {

public int minR;

public int minG;

public int minB;

public int maxR;

public int maxG;

public int maxB;

public ColorCube(int minB, int maxB, int minG, int maxG, int minR, int maxR) {

this.minB = minB;

this.maxB = maxB;

this.minG = minG;

this.maxG = maxG;

this.minR = minR;

this.maxR = maxR;

}

}

**Main.java**

import javax.imageio.ImageIO;

import javax.swing.\*;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.TreeSet;

public class Main {

private static FileOutputStream outputStream;

private static final int PALETTE\_OFFSET\_SIZE = 1078;

public static int[] readInputFile(String inFilename, String outFilename) throws IOException {

FileInputStream inputStream = new FileInputStream(inFilename);

outputStream = new FileOutputStream(outFilename);

int[] inputFileBytes = new int[inputStream.available()];

int i = 0;

while (inputStream.available() > 0) {

inputFileBytes[i] = inputStream.read();

i++;

}

return inputFileBytes;

}

public static void writeHeaderInOutputFile(int[] inputFileBytes) throws IOException {

for (int j = 0; j <= 53; j++) {

outputStream.write(inputFileBytes[j]);

}

}

public static ColorCube findColorCubeBorders(int[] inputFileBytes, int[][][] colorsCoordinateSystem, int width, int height, int offSet) {

int minB = Integer.MAX\_VALUE, maxB = 0;

int minG = Integer.MAX\_VALUE, maxG = 0;

int minR = Integer.MAX\_VALUE, maxR = 0;

// Find borders of the cube

for (int j = 0; j < height \* width \* 3; j += 3) {

int tempB = inputFileBytes[j + offSet];

int tempG = inputFileBytes[j + 1 + offSet];

int tempR = inputFileBytes[j + 2 + offSet];

colorsCoordinateSystem[tempB][tempG][tempR]++;

if (tempB < minB) {

minB = tempB;

} else if (tempB > maxB) {

maxB = tempB;

}

if (tempG < minG) {

minG = tempG;

} else if (tempG > maxG) {

maxG = tempG;

}

if (tempR < minR) {

minR = tempR;

} else if (tempR > maxR) {

maxR = tempR;

}

}

return new ColorCube(minB, maxB, minG, maxG, minR, maxR);

}

public static int findMedianByColorInCube(int[][][] colorsCoordinateSystem, ColorCube someCube, int color) {

TreeSet<Integer> colors = new TreeSet<>();

for (int b = 0; b < colorsCoordinateSystem.length; b++) {

for (int g = 0; g < colorsCoordinateSystem[b].length; g++) {

for (int r = 0; r < colorsCoordinateSystem[b][g].length; r++) {

switch (color) {

case 1:

if (b >= someCube.minB && b <= someCube.maxB) {

colors.add(b);

}

break;

case 2:

if (g >= someCube.minG && g <= someCube.maxG) {

colors.add(g);

}

break;

case 3:

if (r >= someCube.minR && r <= someCube.maxR) {

colors.add(r);

}

break;

}

}

}

}

int colorsCount = colors.size();

int i = 0;

for (int tempColor : colors) {

if (i == colorsCount / 2) return tempColor;

i++;

}

return -1;

}

public static void main(String[] args) throws IOException {

String inputFile = "Fish.bmp";

String outputFile = "Out.bmp";

int[] inputFileBytes = readInputFile(inputFile, outputFile);

// Set new Size

System.out.println("Old Size = " + getValue(2, 5, inputFileBytes) + "byte");

int newSize = PALETTE\_OFFSET\_SIZE + getValue(18, 21, inputFileBytes) \* getValue(22, 25, inputFileBytes); //System.out.println("newSize: " + newSize);

setValue(2, 5, inputFileBytes, newSize);

System.out.println("New Size = " + getValue(2, 5, inputFileBytes) + "byte");

// Set new Offset

int offSet = getValue(10, 13, inputFileBytes);

setValue(10, 13, inputFileBytes, 1078);

System.out.println("Old offset " + offSet);

System.out.println("New offset: " + getValue(10, 13, inputFileBytes));

int width = getValue(18, 21, inputFileBytes);

int height = getValue(22, 25, inputFileBytes);

System.out.println("Width: " + width);

System.out.println("Height: " + height);

// Set new count of bits per pixel

int bitsPerPixel = 8;

System.out.println("Old BitsPerPix = " + getValue(28, 29, inputFileBytes));

setValue(28, 29, inputFileBytes, bitsPerPixel);

System.out.println("New BitsPerPix = " + getValue(28, 29, inputFileBytes));

// Set new image size

System.out.println("Old SizeImage in bytes = " + getValue(34, 37, inputFileBytes));

setValue(34, 37, inputFileBytes, width \* height);

System.out.println("New SizeImage in bytes = " + getValue(34, 37, inputFileBytes));

writeHeaderInOutputFile(inputFileBytes);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int[][][] colorsCoordinateSystem = new int[256][256][256];

ColorCube firstColorCube = findColorCubeBorders(inputFileBytes, colorsCoordinateSystem, width, height, offSet);

int minB = firstColorCube.minB;

int maxB = firstColorCube.maxB;

int minG = firstColorCube.minG;

int maxG = firstColorCube.maxG;

int minR = firstColorCube.minR;

int maxR = firstColorCube.maxR;

ColorCube[] prevCubes = new ColorCube[1];

prevCubes[0] = new ColorCube(minB, maxB, minG, maxG, minR, maxR);

for (int j = 1; j <= 8; j++) { // 2, 4, 8, 16, 32, 64, 128, 256

ColorCube[] nextCubes = new ColorCube[1 << j]; // 2, 4, 8, 16, 32, 64, 128, 256

for (int oldIndex = 0, newIndex = 0; oldIndex < 1 << (j - 1); oldIndex++, newIndex += 2) {

int modB = prevCubes[oldIndex].maxB - prevCubes[oldIndex].minB;

int modG = prevCubes[oldIndex].maxG - prevCubes[oldIndex].minG;

int modR = prevCubes[oldIndex].maxR - prevCubes[oldIndex].minR;

// Find most long component from one of the B, G, R and split it by 2 new cubes

if (modB >= modR && modB >= modG) {

int median = findMedianByColorInCube(colorsCoordinateSystem, prevCubes[oldIndex], 1);

nextCubes[newIndex] = new ColorCube(prevCubes[oldIndex].minB, median, prevCubes[oldIndex].minG, prevCubes[oldIndex].maxG, prevCubes[oldIndex].minR, prevCubes[oldIndex].maxR);

nextCubes[newIndex + 1] = new ColorCube(median + 1, prevCubes[oldIndex].maxB, prevCubes[oldIndex].minG, prevCubes[oldIndex].maxG, prevCubes[oldIndex].minR, prevCubes[oldIndex].maxR);

} else if (modR >= modB && modR >= modG) {

int median = findMedianByColorInCube(colorsCoordinateSystem, prevCubes[oldIndex], 3);

nextCubes[newIndex] = new ColorCube(prevCubes[oldIndex].minB, prevCubes[oldIndex].maxB, prevCubes[oldIndex].minG, prevCubes[oldIndex].maxG, prevCubes[oldIndex].minR, median);

nextCubes[newIndex + 1] = new ColorCube(prevCubes[oldIndex].minB, prevCubes[oldIndex].maxB, prevCubes[oldIndex].minG, prevCubes[oldIndex].maxG, median + 1, prevCubes[oldIndex].maxR);

} else {

int median = findMedianByColorInCube(colorsCoordinateSystem, prevCubes[oldIndex], 2);

nextCubes[newIndex] = new ColorCube(prevCubes[oldIndex].minB, prevCubes[oldIndex].maxB, prevCubes[oldIndex].minG, median, prevCubes[oldIndex].minR, prevCubes[oldIndex].maxR);

nextCubes[newIndex + 1] = new ColorCube(prevCubes[oldIndex].minB, prevCubes[oldIndex].maxB, median + 1, prevCubes[oldIndex].maxG, prevCubes[oldIndex].minR, prevCubes[oldIndex].maxR);

}

// Find borders of the new cubes again

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

boolean isOneColorFound = false;

minB = Integer.MAX\_VALUE; maxB = 0;

minG = Integer.MAX\_VALUE; maxG = 0;

minR = Integer.MAX\_VALUE; maxR = 0;

for (int b = nextCubes[newIndex + x].minB; b <= nextCubes[newIndex + x].maxB; b++) {

for (int g = nextCubes[newIndex + x].minG; g <= nextCubes[newIndex + x].maxG; g++) {

for (int r = nextCubes[newIndex + x].minR; r <= nextCubes[newIndex + x].maxR; r++) {

if (colorsCoordinateSystem[b][g][r] != 0) {

if (b < minB) {

minB = b;

} else if (b > maxB) {

maxB = b;

}

if (g < minG) {

minG = g;

} else if (g > maxG) {

maxG = g;

}

if (r < minR) {

minR = r;

} else if (r > maxR) {

maxR = r;

}

isOneColorFound = true;

}

}

}

}

if (isOneColorFound) {

nextCubes[newIndex + x] = new ColorCube(minB, maxB, minG, maxG, minR, maxR);

} else {

nextCubes[newIndex + x] = new ColorCube(0, 0, 0, 0, 0, 0);

}

}

}

prevCubes = nextCubes;

}

int[] newRasterData = new int[height \* width];

for (int j = offSet, r = 0; j < height \* width \* 3; j += 3, r++) {

for (int k = 0; k < 256; k++) {

if (inputFileBytes[j] <= prevCubes[k].maxB && inputFileBytes[j] >= prevCubes[k].minB

&& inputFileBytes[j + 1] <= prevCubes[k].maxG && inputFileBytes[j + 1] >= prevCubes[k].minG

&& inputFileBytes[j + 2] <= prevCubes[k].maxR && inputFileBytes[j + 2] >= prevCubes[k].minR) {

newRasterData[r] = k;

break;

}

}

}

// Create new palette

for (int j = 54, k = 0; j < 1078; j += 4, k++) {

int countOfPoints = 0;

int blueCount = 0, greenCount = 0, redCount = 0;

for (int b = prevCubes[k].minB; b <= prevCubes[k].maxB; b++) {

for (int g = prevCubes[k].minG; g <= prevCubes[k].maxG; g++) {

for (int r = prevCubes[k].minR; r <= prevCubes[k].maxR; r++) {

if (colorsCoordinateSystem[b][g][r] != 0) {

countOfPoints += colorsCoordinateSystem[b][g][r];

blueCount += colorsCoordinateSystem[b][g][r] \* b;

greenCount += colorsCoordinateSystem[b][g][r] \* g;

redCount += colorsCoordinateSystem[b][g][r] \* r;

}

}

}

}

if (countOfPoints != 0) {

outputStream.write(blueCount / countOfPoints);

outputStream.write(greenCount / countOfPoints);

outputStream.write(redCount / countOfPoints);

} else {

outputStream.write(blueCount);

outputStream.write(greenCount);

outputStream.write(redCount);

}

outputStream.write(inputFileBytes[j + 3]);

}

// Write new raster

for (int j = 0; j < height \* width; j++) {

outputStream.write(newRasterData[j]);

}

displayImages(inputFile, outputFile, width, height);

}

public static int getValue(int startAddress, int endAddress, int[] fileToBytes) {

int answer = fileToBytes[endAddress];

for (int i = endAddress; i > startAddress; i--) {

answer = (answer << 8) | fileToBytes[i - 1];

}

return answer;

}

public static void setValue(int startAddress, int endAddress, int[] fileToBytes, int value) {

for (int i = startAddress; i <= endAddress; i++) {

fileToBytes[i] = value & 255;

value >>= 8;

}

}

public static void displayImages(String firstImageFilename, String secondImageFilename, int width, int height) throws IOException {

BufferedImage img1 = ImageIO.read(new File(firstImageFilename));

ImageIcon icon1 = new ImageIcon(img1);

JLabel label1 = new JLabel();

label1.setIcon(icon1);

label1.setLocation(0, 0);

BufferedImage img2 = ImageIO.read(new File(secondImageFilename));

ImageIcon icon2 = new ImageIcon(img2);

JLabel label2 = new JLabel();

label2.setIcon(icon2);

label2.setLocation(300, 300);

JFrame frame = new JFrame();

frame.setLayout(new FlowLayout());

frame.setSize(width \* 2, height);

frame.add(label1);

frame.add(label2);

frame.setVisible(true);

frame.pack();

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void displayAdditionalHeaderInfo(int[] inputFileBytes) {

System.out.println("Reserved1 = " + getValue(6, 7, inputFileBytes));

System.out.println("Reserved2 = " + getValue(8, 9, inputFileBytes));

System.out.println("\n\n Info about image ");

System.out.println("HeaderSize = " + getValue(14, 17, inputFileBytes));

System.out.println("Width = " + getValue(18, 21, inputFileBytes));

System.out.println("Height = " + getValue(22, 25, inputFileBytes));

System.out.println("Planes = " + getValue(26, 27, inputFileBytes));

System.out.println("Compression = " + getValue(30, 33, inputFileBytes));

System.out.println("XPelsPerMeter = " + getValue(38, 41, inputFileBytes));

System.out.println("YPelsPerMeter = " + getValue(42, 45, inputFileBytes));

System.out.println("ClrUsed = " + getValue(46, 49, inputFileBytes));

System.out.println("ClrImportant = " + getValue(50, 53, inputFileBytes));

}}

**РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЗАДАНИЯ**

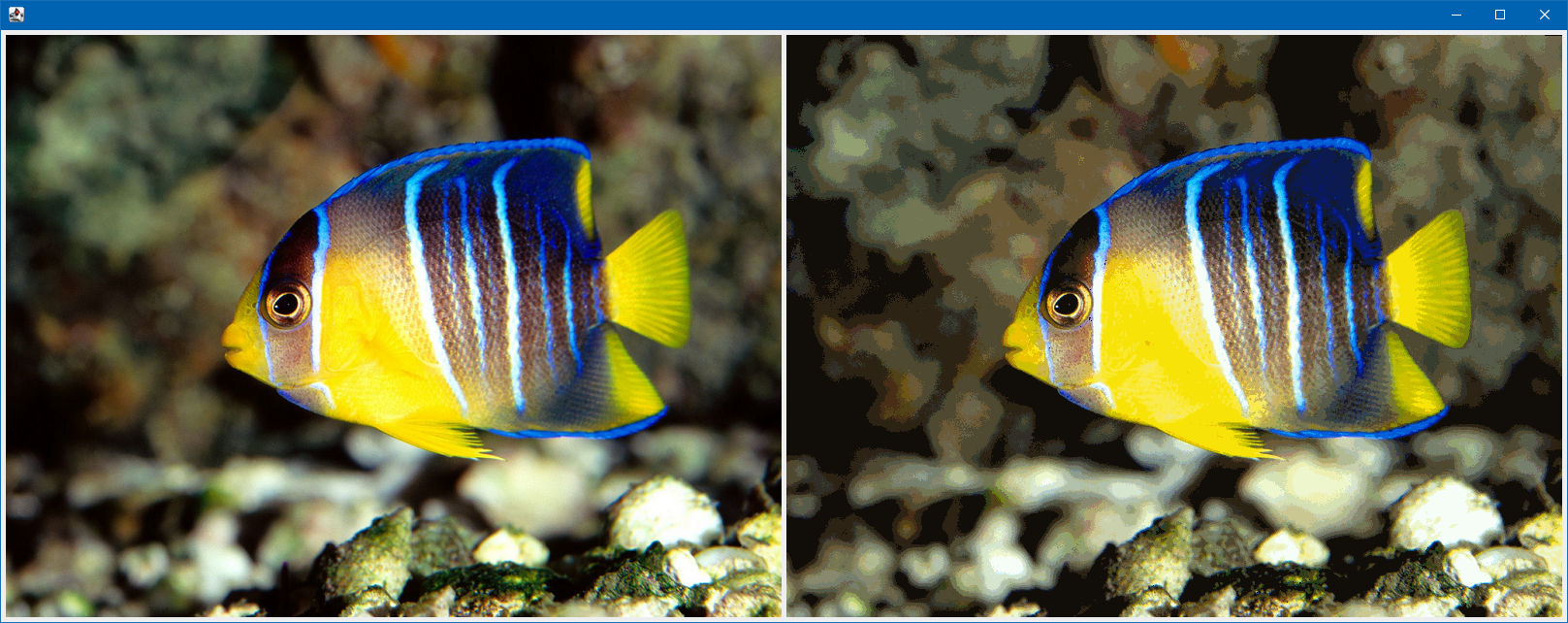


Рис 1. – Результат преобразования True Color изображение в 256-цветное изображение.