Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп’ютерних наук

Кафедра безпеки інформаційних систем і технологій

Лабораторна робота №1

*з навчальної дисципліни*

«Компоненти складних комп’ютерних мереж»

Виконав:

Студент групи КБ-41

Кравченко Є.М.

Перевірив:

доцент

Малахов С.В.

Харків – 2020 р.

**Тема:** изучение метода кодирования цифровых потоков с частичной потерей информации на примере базового алгоритма сжатия изображений jpeg

**Мета:** практическое занятие предполагает самостоятельную разработку документа MathCAD, реализующего модель процесса кодирования/декодирования (сжатия/восстановления) файлов статических монохромных изображений с различной вероятностью перепада яркости элементов в соответствии с базовыми процедурами алгоритма JPEG. В тексте данного руководства излагается только последовательность действий при реализации необходимых преобразований, а также сформулированы рекомендации по использованию некоторых встроенных функций MathCAD. Выбор конкретного способа реализации указанных действий может быть произвольным.

**Ход работы**

1.Реализация алгоритма JPEG на языке программирования Java:

**import** javax.imageio.ImageIO;  
**import** java.awt.\*;  
**import** java.awt.image.BufferedImage;  
**import** java.io.File;  
**import** java.io.IOException;  
  
**public class** DCT {  
 **public static int** *N* = 8;  
 **public static int** *mk* = 10;  
 **public static int** *P* = 8;  
 **protected double**[] **c**;  
  
 **public** DCT() {  
 initCoefficients();  
 }  
  
 **protected void** initCoefficients() {  
 **c** = **new double**[*N*];  
 **for** (**int** i = 1; i < *N*; i++) {  
 **c**[i] = 1;  
 }  
 **c**[0] = 1. / Math.*sqrt*(2.);  
 }  
  
 **protected double**[][] forwardDCT(**double**[][] input) {  
 **double**[][] output = **new double**[*N*][*N*];  
  
 **for** (**int** u = 0; u < *N*; u++) {  
 **for** (**int** v = 0; v < *N*; v++) {  
 **double** sum = 0.0;  
 **for** (**int** x = 0; x < *N*; x++) {  
 **for** (**int** y = 0; y < *N*; y++) {  
 sum += input[x][y] \* Math.*cos*(((2 \* x + 1) / (2.0 \* *N*)) \* u \* Math.***PI***) \* Math.*cos*(((2 \* y + 1) / (2.0 \* *N*)) \* v \* Math.***PI***);  
 }  
 }  
 sum \*= **c**[u] \* **c**[v] / Math.*sqrt*(2 \* *N*);  
 output[u][v] = sum;  
 }  
 }  
 **return** output;  
 }  
  
 **protected double**[][] inverseDCT(**double**[][] input) {  
 **double**[][] output = **new double**[*N*][*N*];  
  
 **for** (**int** x = 0; x < *N*; x++) {  
 **for** (**int** y = 0; y < *N*; y++) {  
 **double** sum = 0.0;  
 **for** (**int** u = 0; u < *N*; u++) {  
 **for** (**int** v = 0; v < *N*; v++) {  
 sum += **c**[u] \* **c**[v] \* input[u][v] \* Math.*cos*(((2 \* x + 1) / (2.0 \* *N*)) \* u \* Math.***PI***) \* Math.*cos*(((2 \* y + 1) / (2.0 \* *N*)) \* v \* Math.***PI***);  
 }  
 }  
 sum /= Math.*sqrt*(2 \* *N*);  
 output[x][y] = sum;  
 }  
 }  
 **return** output;  
 }  
  
 **public static void** main(String[] args) **throws** IOException {  
 BufferedImage bufferedImage = ImageIO.*read*(**new** File(**"D:\\Season7\\Components of complex computer networks\\Lab1\\src\\main\\resources\\12.bmp"**));  
 **int** height = (**int**) (Math.*ceil*((**double**) bufferedImage.getHeight() / *N*) \* *N*);  
 **int** width = (**int**) (Math.*ceil*((**double**) bufferedImage.getWidth() / *N*) \* *N*);  
  
 **double**[][] R = **new double**[height][width];  
 **double**[][] G = **new double**[height][width];  
 **double**[][] B = **new double**[height][width];  
  
 **double**[][] dctImageR;  
 **double**[][] dctImageG;  
 **double**[][] dctImageB;  
 **for** (**int** i = 0; i < bufferedImage.getHeight(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getWidth(); j++) {  
 Color color = **new** Color(bufferedImage.getRGB(j, i));  
 R[i][j] = color.getRed();  
 G[i][j] = color.getGreen();  
 B[i][j] = color.getBlue();  
 }  
 }  
  
 dctImageR = *colorToDCTColor*(R);  
 dctImageG = *colorToDCTColor*(G);  
 dctImageB = *colorToDCTColor*(B);  
  
 R = *DCTColotToColor*(dctImageR);  
 G = *DCTColotToColor*(dctImageG);  
 B = *DCTColotToColor*(dctImageB);  
 **for** (**int** i = 0; i < bufferedImage.getHeight(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getWidth(); j++) {  
 bufferedImage.setRGB(j, i, **new** Color(  
 Math.*min*(Math.*max*((**int**) R[i][j], 0), 255),  
 Math.*min*(Math.*max*((**int**) G[i][j], 0), 255),  
 Math.*min*(Math.*max*((**int**) B[i][j], 0), 255)).getRGB());  
  
 }  
 }  
 ImageIO.*write*(bufferedImage, **"BMP"**, **new** File(**"D:\\Season7\\Components of complex computer networks\\Lab1\\src\\main\\resources\\12\_1.bmp"**));  
 }  
  
 **private static double**[][] DCTColotToColor(**double**[][] dctImage) {  
 **double**[][] image = **new double**[dctImage.**length**][dctImage[0].**length**];  
 DCT dct = **new** DCT();  
 **for** (**int** i = 0; i < dctImage.**length**; i += *N*) {  
 **for** (**int** j = 0; j < dctImage[i].**length**; j += *N*) {  
 **double**[][] d = **new double**[*N*][*N*];  
 **for** (**int** x = i, k = 0; x < i + *N*; x++, k++) {  
 **for** (**int** y = j, l = 0; y < j + *N*; y++, l++) {  
 d[k][l] = dctImage[x][y];  
 }  
 }  
 **double**[][] array;  
 array = dct.inverseDCT(d);  
 **for** (**int** x = i, k = 0; x < i + *N*; x++, k++) {  
 **for** (**int** y = j, l = 0; y < j + *N*; y++, l++) {  
 image[x][y] = array[k][l] \* *mk*;  
 }  
 }  
 }  
 }  
 **return** image;  
 }  
  
 **private static double**[][] colorToDCTColor(**double**[][] imageColor) {  
 DCT dct = **new** DCT();  
 **double**[][] dctImage = **new double**[imageColor.**length**][imageColor[0].**length**];  
 **for** (**int** i = 0; i < imageColor.**length**; i += *N*) {  
 **for** (**int** j = 0; j < imageColor[i].**length**; j += *N*) {  
 **double**[][] d = **new double**[*N*][*N*];  
 **for** (**int** x = i, k = 0; x < i + *N*; x++, k++) {  
 **for** (**int** y = j, l = 0; y < j + *N*; y++, l++) {  
 d[k][l] = imageColor[x][y];  
 }  
 }  
 **double**[][] dctArray;  
 dctArray = dct.forwardDCT(d);  
 */\*\*/* **for** (**int** l = 0; l < *N*; l++) {  
 **for** (**int** k = *N* - 1; k >= 0 && k > *P* - l - 1; k--) {  
 dctArray[l][k] = 0;  
 }  
 }  
 */\*\*/* **for** (**int** x = i, k = 0; x < i + *N*; x++, k++) {  
 **for** (**int** y = j, l = 0; y < j + *N*; y++, l++) {  
 dctImage[x][y] = Math.*round*(dctArray[k][l] / *mk*);  
 }  
 }  
 }  
 }  
 **return** dctImage;  
 }  
}

Исходное изображение:



**Выводы**

На практическом занятим было разработано приложение на языке Java реализующего модель процесса кодирования/декодирования (сжатия/восстановления) файлов статических монохромных изображений с различной вероятностью перепада яркости элементов в соответствии с базовыми процедурами алгоритма JPEG.

В ходе експериментов были выявлены следующие зависимости:

1.При увеличении N картинка завечивается;

2.При изменении Mk ничего не меняеться;

3.При слишком маленьком значении P=(1-2) изображение становится пиксельным, при P=(3-4) теряются детализация изображения. Оптимальным знаем является Р=N.