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 работф**

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;  
  
**class** DCT {  
 **private static double** *mk* = 10;  
 **private static int** *N* = 8;  
 **private static int** *P* = 1;  
 *// coefficients* **protected double**[][] **c**;  
 *// zig zag matrix* **protected int**[][] **zigzag**;  
  
 **public** DCT() {  
 initCoefficients();  
 **zigzag** = makeZigZagMatrix();  
 }  
  
 */\* initialize coefficient matrix \*/* **protected void** initCoefficients() {  
 **c** = **new double**[*N*][*N*];  
  
 **for** (**int** i = 1; i < *N*; i++) {  
 **for** (**int** j = 1; j < *N*; j++) {  
 **c**[i][j] = 1;  
 }  
 }  
  
 **for** (**int** i = 0; i < *N*; i++) {  
 **c**[i][0] = 1 / Math.*sqrt*(2.0);  
 **c**[0][i] = 1 / Math.*sqrt*(2.0);  
 }  
 **c**[0][0] = 0.5;  
 }  
  
 **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][v] / 4.0;  
 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][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 /= 4.0;  
 output[x][y] = sum;  
 }  
 }  
 **return** output;  
 }  
  
 */\* write dct coefficient matrix into 1D array in zig zag order \*/* **public double**[] zigZag(**double**[][] m) {  
 **double**[] zz = **new double**[*N* \* *N*];  
 **for** (**int** i = 0; i < *N*; i++) {  
 **for** (**int** j = 0; j < *N*; j++) {  
 zz[**zigzag**[i][j]] = m[i][j];  
 }  
 }  
 **return** zz;  
 }  
  
 */\* write zig zag ordered coefficients into matrix \*/* **public double**[][] unZigZag(**double**[] zz) {  
 **double**[][] m = **new double**[*N*][*N*];  
 **for** (**int** i = 0; i < *N*; i++) {  
 **for** (**int** j = 0; j < *N*; j++) {  
 m[i][j] = zz[**zigzag**[i][j]];  
 }  
 }  
 **return** m;  
 }  
  
 */\* generate zig zag matrix \*/* **private int**[][] makeZigZagMatrix() {  
 **int**[][] zz = **new int**[*N*][*N*];  
 **int** zval = 0;  
 **int** zval2 = *N* \* (*N* - 1) / 2;  
 **int** i, j;  
 **for** (**int** k = 0; k < *N*; k++) {  
 **if** (k % 2 == 0) {  
 i = 0;  
 j = k;  
 **while** (j > -1) {  
 zz[i][j] = zval;  
 zval++;  
 i++;  
 j--;  
 }  
 i = *N* - 1;  
 j = k;  
 **while** (j < *N*) {  
 zz[i][j] = zval2;  
 zval2++;  
 i--;  
 j++;  
 }  
 } **else** {  
 i = k;  
 j = 0;  
 **while** (i > -1) {  
 zz[i][j] = zval;  
 zval++;  
 j++;  
 i--;  
 }  
 i = k;  
 j = *N* - 1;  
 **while** (i < *N*) {  
 zz[i][j] = zval2;  
 zval2++;  
 i++;  
 j--;  
 }  
 }  
 }  
 **return** zz;  
 }  
  
 **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"**));  
 **double**[][] imageRED = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
 **double**[][] imageGREN = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
 **double**[][] imageBLUE = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
  
 **double**[][] dctImageRED = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
 **double**[][] dctImageGREEN = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
 **double**[][] dctImageBLUE = **new double**[bufferedImage.getWidth()][bufferedImage.getHeight()];  
 **for** (**int** i = 0; i < bufferedImage.getWidth(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 imageRED[i][j] = **new** Color(bufferedImage.getRGB(i, j)).getRed();  
 }  
 }  
 **for** (**int** i = 0; i < bufferedImage.getWidth(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 imageGREN[i][j] = **new** Color(bufferedImage.getRGB(i, j)).getGreen();  
 }  
 }  
 **for** (**int** i = 0; i < bufferedImage.getWidth(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 imageBLUE[i][j] = **new** Color(bufferedImage.getRGB(i, j)).getBlue();  
 }  
 }  
  
 dctImageRED = *colorToDCTColor*(imageRED, *P*);  
 dctImageGREEN = *colorToDCTColor*(imageGREN, *P*);  
 dctImageBLUE = *colorToDCTColor*(imageBLUE, *P*);  
  
 imageRED = *DCTColotToColor*(dctImageRED);  
 imageGREN = *DCTColotToColor*(dctImageGREEN);  
 imageBLUE = *DCTColotToColor*(dctImageBLUE);  
 **for** (**int** i = 0; i < bufferedImage.getWidth(); i++) {  
 **for** (**int** j = 0; j < bufferedImage.getHeight(); j++) {  
 **if** (imageRED[i][j] < 0) {  
 imageRED[i][j] = 0;  
 }  
 **if** (imageGREN[i][j] < 0) {  
 imageGREN[i][j] = 0;  
 }  
 **if** (imageBLUE[i][j] < 0) {  
 imageBLUE[i][j] = 0;  
 }  
 **if** (imageRED[i][j] > 255) {  
 imageRED[i][j] = 255;  
 }  
 **if** (imageGREN[i][j] > 255) {  
 imageGREN[i][j] = 255;  
 }  
 **if** (imageBLUE[i][j] > 255) {  
 imageBLUE[i][j] = 255;  
 }  
 bufferedImage.setRGB(i, j, **new** Color((**int**) imageRED[i][j], (**int**) imageGREN[i][j], (**int**) imageBLUE[i][j]).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, **int** p) {  
 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 k = 0; k < N; k++) {  
 for (int l = 0; l < N; l++) {  
 System.out.printf("%7.2f ", dctArray[k][l]);  
 }  
 System.out.println();  
 }\*/* **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;  
 }  
}

**Вопросы контроля**

1. Кратко изложить физическую основу процессов, используемых алгоритмом сжатия JPEG. Для каких целей он применяется?

2. Перечислить и пояснить суть основные этапов преобразования матриц

при сжатии изображений посредством алгоритмов, аналогичных JPEG.

3. Какие способы отбора значимых коэффициентов преобразования используются на практике?

4. Как влияют параметры исследуемого алгоритма Mk и P на величину коэффициента сжатия «КСЖ» и качество восстанавливаемых изображений?

5. Какие дополнительные действия можно было бы включить в рассмотренный алгоритм для повышения его эффективности?

6. Какие свойства изображений обеспечивают возможность использования

стеганографии (встраивания скрытого контента/данных)?

7. Привести примеры настроечных характеристик (N, Mk и P) алгоритма

сжатия в зависимости от типа и сложности структуры обрабатываемых изображений (вероятности перепада яркости между соседними элементами изображений).



**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.