

**BTPR 2053**

**FUNDAMENTALS OF IMAGE PROCESSING**

**Image Processing Assignment Report**

**Soo De Xiang (B180092B)**

**Lecturer: Dr. Pang Yee Yong**

**Department of Computer Science**

**Faculty of Engineering and Information Technology**

**Assignment 1**

import java.io.\*;

public class A1 {

public static String order = null;

public static void main(String[] args) {

try {

String fileName = "yoda.tif";

File file = new File(fileName);

FileInputStream z = new FileInputStream(file);

System.out.println("File Name: " + file.getName());

int value;

int q = 0;

int de = 0;

int allde = -1;

int ifd = -1;

int stripOffsets = -1;

String[][] dearr = null;

byte[] countbyte = null;

File outputfile = new File("assignment1.raw");

FileOutputStream output = new FileOutputStream(outputfile);

while ((value = z.read()) != -1) {

if (q == 0) {

int value2 = z.read();

String hex = changehex(value) + changehex(value2);

if (hex.equals("4949")) {

order = "LSB";

} else if (hex.equals("4D4D")) {

order = "MSB";

}

System.out.println("LSB / MSB: " + order);

q++;

} else if (q == 2) {

int value2 = z.read();

int[] nums = {value, value2};

String hexString = adddectohex(nums);

System.out.println("Version: " + hexString);

q++;

} else if (q == 4) {

int value2 = z.read();

int value3 = z.read();

int value4 = z.read();

int[] nums = {value, value2, value3, value4};

String hexString = adddectohex(nums);

System.out.println("First Offset IFD: " + Integer.parseInt(hexString, 16));

q += 3;

} else if (q == 8) {

int value2 = z.read();

int[] nums = {value, value2};

String hexString = adddectohex(nums);

allde = Integer.parseInt(hexString, 16);

System.out.println("Total DE: " + allde);

dearr = new String[allde][8];

q++;

} else if ((q >= 10 && q < (10 + 12 \* allde)) && ((stripOffsets > 0 && q <= stripOffsets && ((q + 12) <= stripOffsets)) || (stripOffsets == -1))) {

int tag1 = value;

int tag2 = z.read();

int type1 = z.read();

int type2 = z.read();

int length1 = z.read();

int length2 = z.read();

int length3 = z.read();

int length4 = z.read();

int value1 = z.read();

int value2 = z.read();

int value3 = z.read();

int value4 = z.read();

int[] tagNameNums = { tag1, tag2 };

int[] tagTypeNums = { type1, type2 };

String tagName = findtag(tagNameNums);

String tagTypeName = findtagtype(tagTypeNums);

if (tagName != null && tagTypeName != null) {

String tagValue = adddectohex(new int[] {value1, value2, value3, value4});

String tagLength = adddectohex(new int[] {length1, length2, length3, length4});

int tagValueDec = Integer.parseInt(tagValue, 16);

int tagLengthDec = Integer.parseInt(tagLength, 16);

int wordLength1 = 29 - tagName.length();

int wordLength2 = 9 - tagTypeName.length();

String wordSpaces1 = "";

String wordSpaces2 = "";

for (int i = 0; i < wordLength1; i++) {

wordSpaces1 += " ";

}

for (int i = 0; i < wordLength2; i++) {

wordSpaces2 += " ";

}

if (tagName == "StripOffsets") {

stripOffsets = tagValueDec;

System.out.println("Size of IFD: " + stripOffsets);

}

dearr[de][0] = Integer.toString(Integer.parseInt(adddectohex(tagNameNums), 16));

dearr[de][1] = tagName;

de++;

}

q += 11;

} else if (q == (10 + 12 \* allde)) {

int value2 = z.read();

int[] nums = {value, value2};

String offsetNextIFDHex = adddectohex(nums);

ifd = Integer.parseInt(offsetNextIFDHex, 16);

System.out.println("Consecutive Offset IFD (Offset of Next IFD): " + ifd);

q++;

} else if (q >= 10 && stripOffsets > 0 && q >= stripOffsets) {

if (q == (stripOffsets)) {

int allbyte = (int) (file.length() - stripOffsets);

countbyte = new byte[allbyte];

for (int i = 0; i < (dearr.length - 1); i++) {

if (dearr[i][0] != null) {

int tagDec = Integer.parseInt(dearr[i][0]);

String tagName = dearr[i][1];

System.out.println(tagDec + tagName );

}

}

}

}

q++;

}

output.close();

z.close();

} catch (IOException e) {

System.out.println("File is not exists");

}

}

private static String adddectohex(int[] decNums) {

String hexString = "";

for (int i = 0; i < decNums.length; i++) {

if (order == "LSB") {

hexString += changehex(decNums[decNums.length - (i + 1)]);

} else if (order == "MSB") {

hexString += changehex(decNums[i]);

}

}

return hexString;

}

private static String changehex(int decNum) {

if (decNum < 10) {

return "0" + decNum;

} else {

return String.format("%02X", decNum);

}

}

private static String findtag(int tagType) {

String tagName = null;

switch (tagType) {

case 254:

tagName = "NewSubfileType";

break;

case 256:

tagName = "ImageWidth";

break;

case 257:

tagName = "ImageLength";

break;

case 258:

tagName = "BitsPerSample";

break;

case 259:

tagName = "Compression";

break;

case 262:

tagName = "PhotometricInterpretation";

break;

case 273:

tagName = "StripOffsets";

break;

case 277:

tagName = "SamplesPerPixel";

break;

case 278:

tagName = "RowsPerStrip";

break;

case 279:

tagName = "StripByteCounts";

break;

case 282:

tagName = "XResolution";

break;

case 283:

tagName = "YResolution";

break;

case 296:

tagName = "ResolutionUnit";

break;

}

return tagName;

}

private static String findtagtype(int tagTypeDec) {

String tagTypeName = null;

switch (tagTypeDec) {

case 1:

tagTypeName = "byte";

break;

case 2:

tagTypeName = "ASCII";

break;

case 3:

tagTypeName = "short";

break;

case 4:

tagTypeName = "long";

break;

case 5:

tagTypeName = "rational";

break;

}

return tagTypeName;

}

private static String findtag(String tagHex) {

int tagType = Integer.parseInt(tagHex, 16);

return findtag(tagType);

}

private static String findtag(int[] tagNums) {

String tagHex = adddectohex(tagNums);

return findtag(tagHex);

}

private static String findtagtype(String tagTypeHex) {

int tagType = Integer.parseInt(tagTypeHex, 16);

return findtagtype(tagType);

}

private static String findtagtype(int[] tagTypeNums) {

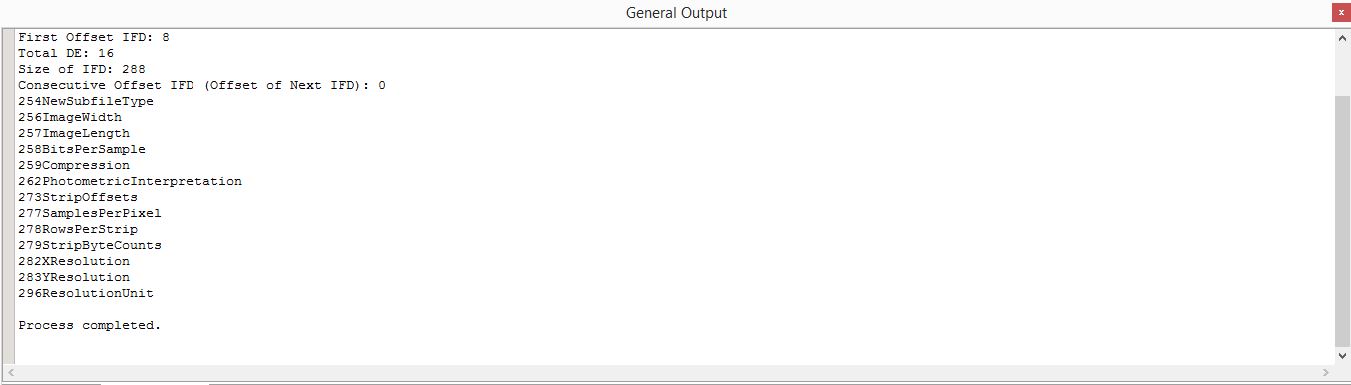
String tagTypeHex = adddectohex(tagTypeNums);

return findtagtype(tagTypeHex);

}

}

**Output:**

****

**Assignment 2**

import java.io.\*;

import java.nio.\*;

public class A2 {

public static void main(String[] args) {

String filename = "yoda.raw";

File file = new File(filename);

try {

FileInputStream z = new FileInputStream(file);

File outputFile = new File("a2\_patterning.raw");

FileOutputStream output = new FileOutputStream(outputFile);

String fileName = file.getName();

System.out.println("Source File Info");

System.out.println("File Name: " + fileName);

System.out.println("Patterning Data");

int height = 62;

int weight = 123;

int newweight = weight\*3;

int newheight = height\*3;

int totalPatternSeq = newweight \* newheight;

int value;

int[][] data = new int[newweight][newheight];

int rowCount = 0;

int colCount = 0;

int index = 0;

while ((value = z.read()) != -1) {

int[] patterntype = checkpattern(value);

int patternWriteIndex = 0;

int rowdata1 = rowCount \* 3;

int rowdata2 = rowdata1 + 1;

int rowdata3 = rowdata2 + 1;

int coldata1 = colCount \* 3;

int coldata2 = coldata1 + 1;

int coldata3 = coldata2 + 1;

writeData(data, rowdata1, coldata1, patterntype[0]);

writeData(data, rowdata1, coldata2, patterntype[1]);

writeData(data, rowdata1, coldata3, patterntype[2]);

writeData(data, rowdata2, coldata1, patterntype[3]);

writeData(data, rowdata2, coldata2, patterntype[4]);

writeData(data, rowdata2, coldata3, patterntype[5]);

writeData(data, rowdata3, coldata1, patterntype[6]);

writeData(data, rowdata3, coldata2, patterntype[7]);

writeData(data, rowdata3, coldata3, patterntype[8]);

if (rowCount < height && colCount == (weight - 1)) {

colCount = 0;

rowCount++;

} else if (rowCount < height && colCount < weight) {

colCount++;

}

index++;

}

int writeCount = 0;

for (int[] data1 : data) {

for (int j = 0; j < data1.length; j++) {

output.write(data1[j]);

writeCount++;

}

}

output.flush();

output.close();

z.close();

System.out.println("Dithering D1 Data");

FileInputStream z\_d1 = new FileInputStream(file);

File outputFiled1 = new File("a2\_dithering.raw");

FileOutputStream outputd1 = new FileOutputStream(outputFiled1);

int[] d1Model = {0, 128, 192, 64};

int[][] d1Data = new int[height][weight];

index = 0;

colCount = 0;

rowCount = 0;

while ((value = z\_d1.read()) != -1) {

if ((rowCount == 0 || rowCount % 2 == 0) && (colCount == 0 || colCount % 2 == 0)) {

d1Data[rowCount][colCount] = value > d1Model[0] ? 255 : 0;

} else if ((rowCount == 0 || rowCount % 2 == 0) && (colCount + 1) % 2 == 0) {

d1Data[rowCount][colCount] = value > d1Model[1] ? 255 : 0;

} else if (rowCount > 0 && (rowCount + 1) % 2 == 0 && (colCount == 0 || colCount % 2 == 0)) {

d1Data[rowCount][colCount] = value > d1Model[2] ? 255 : 0;

} else if (rowCount > 0 && (rowCount + 1) % 2 == 0) {

d1Data[rowCount][colCount] = value > d1Model[3] ? 255 : 0;

}

if (colCount == (weight - 1) && rowCount < height) {

colCount = 0;

rowCount++;

} else if (colCount < weight && rowCount < height) {

colCount++;

} else {

throw new Error("condition error");

}

index++;

}

for (int[] dataSet : d1Data) {

for (int i = 0; i < dataSet.length; i++) {

outputd1.write(dataSet[i]);

}

}

outputd1.flush();

outputd1.close();

z\_d1.close();

System.out.println("Dithering D1 Data");

FileInputStream z\_d2 = new FileInputStream(file);

File outputFile\_d2 = new File("assignment2\_d2.raw");

FileOutputStream output\_d2 = new FileOutputStream(outputFile\_d2);

int[][] d2Model = {

{ 0, 128, 32, 160 },

{ 192, 64, 224, 96 },

{ 48, 176, 16, 144 },

{ 240, 112, 208, 80 },

};

int[][] d2Data = new int[height][weight];

index = 0;

colCount = 0;

rowCount = 0;

while ((value = z\_d2.read()) != -1) {

int modelRowIndex;

int modelColIndex;

if ((rowCount + 1) % 4 == 0) {

modelRowIndex = 3;

} else if ((rowCount + 1) % 3 == 0) {

modelRowIndex = 2;

} else if ((rowCount + 1) % 2 == 0) {

modelRowIndex = 1;

} else {

modelRowIndex = 0;

}

if ((colCount + 1) % 4 == 0) {

modelColIndex = 3;

} else if ((colCount + 1) % 3 == 0) {

modelColIndex = 2;

} else if ((colCount + 1) % 2 == 0) {

modelColIndex = 1;

} else {

modelColIndex = 0;

}

d2Data[rowCount][colCount] = value > d2Model[modelRowIndex][modelColIndex] ? 255 : 0;

if (colCount == (weight - 1) && rowCount < height) {

colCount = 0;

rowCount++;

} else if (colCount < weight && rowCount < height) {

colCount++;

} else {

throw new Error("condition error");

}

index++;

}

for (int[] dataSet : d2Data) {

for (int i = 0; i < dataSet.length; i++) {

output\_d2.write(dataSet[i]);

}

}

output\_d2.flush();

output\_d2.close();

z\_d2.close();

} catch (IOException ex) {

System.out.println("File is not exists");

}

}

private static int[] checkpattern(int colorDec) {

int[] result = new int[9];

double baseNum = 25.5;

for (int i = 0; i < 10; i++) { // 10 type pattern

double start = i \* baseNum;

double end = start + baseNum;

if (start <= colorDec && colorDec <= end) {

result = getPatternArr(i);

break;

}

}

return result;

}

private static int[] getPatternArr(int pattern) {

int[] arr = new int[]{};

switch (pattern) {

case 0:

arr = new int[]{

0, 0, 0,

0, 0, 0,

0, 0, 0};

break;

case 1:

arr = new int[]{

0, 0, 0,

0, 0, 0,

0, 0, 255};

break;

case 2:

arr = new int[]{

255, 0, 0,

0, 0, 0,

0, 0, 255};

break;

case 3:

arr = new int[]{

255, 0, 255,

0, 0, 0,

0, 0, 255};

break;

case 4:

arr = new int[]{

255, 0, 255,

0, 0, 0,

255, 0, 255};

break;

case 5:

arr = new int[]{

255, 0, 255,

0, 0, 0,

255, 255, 255};

break;

case 6:

arr = new int[]{

255, 0, 255,

255, 0, 0,

255, 255, 255};

break;

case 7:

arr = new int[]{

255, 255, 255,

255, 0, 0,

255, 255, 255};

break;

case 8:

arr = new int[]{

255, 255, 255,

255, 0, 255,

255, 255, 255};

break;

case 9:

arr = new int[]{

255, 255, 255,

255, 255, 255,

255, 255, 255};

break;

}

return arr;

}

private static boolean isBetween(int x, int start, int end) {

return start <= x && x <= end;

}

private static void writeData(int[][] data, int x, int y, int value) {

if (data[x][y] == 0) {

data[x][y] = value;

} else {

throw new Error("the array position has value");

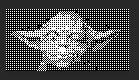
}

}

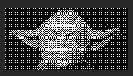
}

**Output**

**Dilthering**

****

**Halftoning**



Assignment 3

import java.io.\*;

import java.math.BigDecimal;

public class A3 {

public static void main(String[] args) {

String file1 = "yoda.raw";

File file = new File(file1);

try {

FileInputStream z = new FileInputStream(file);

String fileName = file.getName();

int height = 62;

int weight = 123;

int[] originalImg = new int[weight \* height];

int[] amountpixel = new int[256];

System.out.println("File Name: " + fileName);

int value;

int index = 0;

while((value = z.read()) != -1) {

originalImg[index] = value;

amountpixel[value]++;

index++;

}

int[] runsum = new int[256];

int totalRunningSum = 0;

for (int i = 0; i < amountpixel.length; i++) {

int acc = 0;

if (i > 0) {

acc = runsum[i - 1];

}

totalRunningSum = runsum[i] = acc + amountpixel[i];

}

System.out.println("Total Running Sum: " + totalRunningSum);

double[] nrunsum = new double[256];

for (int i = 0; i < runsum.length; i++) {

nrunsum[i] = (double) runsum[i] / totalRunningSum;

}

int[] timesrunsum = new int[256];

System.out.println(" Histogram Equalized Values ");

System.out.println("Gray-Level No of Pixel Run Sum Normalized Multiply 255");

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

for (int i = 0; i < nrunsum.length; i++) {

double multipliedNum = nrunsum[i] \* 255;

timesrunsum[i] = (int)Math.round(multipliedNum);

String space0 = " | ";

String space1 = " | ";

String space2 = " | ";

String space3 = " | ";

if (i > 9 && i <= 99) {

space0 = " ";

} else if (i > 99) {

space0 = " ";

}

for (int j = 0; j < Integer.toString(amountpixel[i]).length(); j++) {

space1 = space1.substring(0, space1.length() - 1);

}

for (int j = 0; j < Integer.toString(runsum[i]).length(); j++) {

space2 = space2.substring(0, space2.length() - 1);

}

for (int j = 0; j < Double.toString(nrunsum[i]).length(); j++) {

space3 = space3.substring(0, space3.length() - 1);

}

System.out.println(i + space0 + amountpixel[i] + space1 + runsum[i] + space2 + nrunsum[i] + space3 + timesrunsum[i]);

}

File outputFile = new File("a3.raw");

FileOutputStream fout = new FileOutputStream(outputFile);

for (int i = 0; i < originalImg.length; i++) {

fout.write(timesrunsum[originalImg[i]]);

}

fout.flush();

fout.close();

z.close();

} catch (Exception e) {

}

}

}

Assignment 4

import java.io.\*;

public class A4 {

public static void main(String[] args) {

String file1 = "yoda.raw";

File file = new File(file1);

try {

FileInputStream z = new FileInputStream(file);

String fileName = file.getName();

int weight = 123;

int height = 62;

int[][] imagearray = new int[height][weight];

int[][] convolutionarray = new int[height][weight];

System.out.println("File Name: " + fileName);

int value;

int row = 0;

int col = 0;

int[][] matrics = {

{-1, 0, 1},

{-2, 0, 2},

{-1, 0, 1},};

while ((value = z.read()) != -1) {

imagearray[row][col] = value;

if (col == weight - 1) {

col = 0;

row++;

} else if (row < height) {

col++;

}

}

File outputFile = new File("a4.raw");

FileOutputStream output = new FileOutputStream(outputFile);

for (int rowIndex = 1; rowIndex < imagearray.length - 2; rowIndex++) {

for (int colIndex = 1; colIndex < imagearray[rowIndex].length - 2; colIndex++) {

int k1 = matrics[0][0] \* imagearray[rowIndex + 1] [colIndex + 1];

int k2 = matrics[0][1] \* convolutionarray[rowIndex + 1] [colIndex];

int k3 = matrics[0][2] \* imagearray[rowIndex + 1] [colIndex - 1];

int k4 = matrics[1][0] \* imagearray[rowIndex] [colIndex + 1];

int k5 = matrics[1][1] \* convolutionarray[rowIndex] [colIndex];

int k6 = matrics[1][2] \* imagearray[rowIndex] [colIndex - 1];

int k7 = matrics[2][0] \* imagearray[rowIndex - 1] [colIndex + 1];

int k8 = matrics[2][1] \* convolutionarray[rowIndex - 1] [colIndex];

int k9 = matrics[2][2] \* imagearray[rowIndex - 1] [colIndex - 1];

// matrics(0,0)h(0,0) f (X + 1, Y + 1)

// matrics(0,1)h(1,0) f (X, Y + 1)

// matrics(0,2)h(2,0) f (X - 1, Y + 1)

// matrics(1,0)h(0,1) f(X+1,Y)

// matrics(1,1)h(1,1) f (X, Y)

// matrics(1,2)h(2,1) f (X - 1, Y)

// matrics(2,0)h(0,2) f (X + 1, Y - 1)

// matrics(2,1)h(1,2) f (X, Y - 1)

// matrics(2,)h(2,2) f (X - 1, Y - 1)

int sum = k1 + k3 + k4 + k6 + k7 + k9;

if (sum < 0) {

sum = 0;

} else if (sum > 255) {

sum = 255;

}

convolutionarray[rowIndex][colIndex] = sum;

}

}

for (int[] img : convolutionarray) {

for (int i = 0; i < img.length; i++) {

output.write(img[i]);

}

}

output.flush();

output.close(); // close file output stream

z.close(); // close file input stream

} catch (IOException e) {

System.out.println("Error: " + e.toString());

}

}

private static int findvalue(int value) {

if (value < 0) {

return 0;

} else if (value > 255) {

return 255;

}

return value;

}

}

Output:



Assignment 5

import java.io.\*;

public class A5 {

public static void main(String[] args) {

String file1 = "yoda.raw";

File file = new File(file1);

try {

FileInputStream z = new FileInputStream(file);

String fileName = file.getName();

System.out.println("File Name: " + fileName);

int weight = 123;

int height = 62;

int value;

int[][] imagearray = new int[height][weight];

int[][] passfilterarray = new int[height][weight];

int col = 0;

int row = 0;

int[][] matricsl = {

{1, 1, 1},

{1, 1, 1},

{1, 1, 1}};

int[][] matricsh = {

{-1, -1, -1},

{-1, 8, -1},

{-1, -1, -1}};

while ((value = z.read()) != -1) {

imagearray[row][col] = value;

if (col == weight - 1) {

col = 0;

row++;

} else if (row < height) {

col++;

}

}

File outputFile\_lowpass = new File("a5lowpass.raw");

File outputFile\_highpass = new File("a5highpass.raw");

FileOutputStream output\_lowpass = new FileOutputStream(outputFile\_lowpass);

FileOutputStream output\_highpass = new FileOutputStream(outputFile\_highpass);

for (int rowIndex = 1; rowIndex < imagearray.length - 2; rowIndex++) {

for (int colIndex = 1; colIndex < imagearray[rowIndex].length - 2; colIndex++) {

int k1 = calulateKernelNum(matricsl[0][0], imagearray[rowIndex + 1][colIndex + 1], true);

int k2 = calulateKernelNum(matricsl[0][1], imagearray[rowIndex + 1][colIndex], true);

int k3 = calulateKernelNum(matricsl[0][2], imagearray[rowIndex + 1][colIndex - 1], true);

int k4 = calulateKernelNum(matricsl[1][0], imagearray[rowIndex][colIndex + 1], true);

int k5 = calulateKernelNum(matricsl[1][1], imagearray[rowIndex][colIndex], true);

int k6 = calulateKernelNum(matricsl[1][2], imagearray[rowIndex][colIndex - 1], true);

int k7 = calulateKernelNum(matricsl[2][0], imagearray[rowIndex - 1][colIndex + 1], true);

int k8 = calulateKernelNum(matricsl[2][1], imagearray[rowIndex - 1][colIndex], true);

int k9 = calulateKernelNum(matricsl[2][2], imagearray[rowIndex - 1][colIndex - 1], true);

int sum = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9;

if (sum < 0) {

sum = 0;

} else if (sum > 255) {

sum = 255;

}

passfilterarray[rowIndex][colIndex] = sum;

}

}

for (int[] img : passfilterarray) {

for (int i = 0; i < img.length; i++) {

output\_lowpass.write(img[i]);

}

}

for (int rowIndex = 1; rowIndex < imagearray.length - 2; rowIndex++) {

for (int colIndex = 1; colIndex < imagearray[rowIndex].length - 2; colIndex++) {

int k1 = calulateKernelNum(matricsh[0][0], imagearray[rowIndex + 1][colIndex + 1], false);

int k2 = calulateKernelNum(matricsh[0][1], imagearray[rowIndex + 1][colIndex], false);

int k3 = calulateKernelNum(matricsh[0][2], imagearray[rowIndex + 1][colIndex - 1], false);

int k4 = calulateKernelNum(matricsh[1][0], imagearray[rowIndex][colIndex + 1], false);

int k5 = calulateKernelNum(matricsh[1][1], imagearray[rowIndex][colIndex], false);

int k6 = calulateKernelNum(matricsh[1][2], imagearray[rowIndex][colIndex - 1], false);

int k7 = calulateKernelNum(matricsh[2][0], imagearray[rowIndex - 1][colIndex + 1], false);

int k8 = calulateKernelNum(matricsh[2][1], imagearray[rowIndex - 1][colIndex], false);

int k9 = calulateKernelNum(matricsh[2][2], imagearray[rowIndex - 1][colIndex - 1], false);

int sum = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9;

if (sum < 0) {

sum = 0;

} else if (sum > 255) {

sum = 255;

}

passfilterarray[rowIndex][colIndex] = sum;

}

}

for (int[] img : passfilterarray) {

for (int i = 0; i < img.length; i++) {

output\_highpass.write(img[i]);

}

}

output\_lowpass.flush();

output\_lowpass.close(); // close file output stream

output\_highpass.flush();

output\_highpass.close(); // close file output stream

z.close(); // close file input stream

} catch (IOException e) {

System.out.println("Error: " + e.toString());

}

}

private static int checkKernelValue(int value) {

if (value < 0) {

return 0;

} else if (value > 255) {

return 255;

}

return value;

}

private static int calulateKernelNum(int hVal, int fValue, boolean dividable) {

if (dividable) {

return Math.round((hVal \* fValue) / 9);

} else {

return hVal \* fValue;

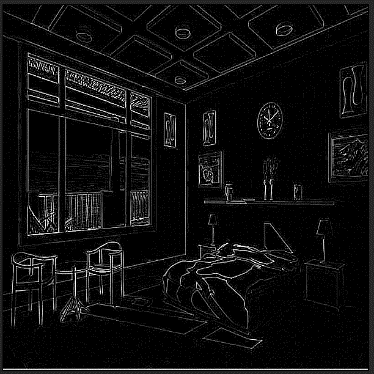
}

}

}

OutPut:

highpass



Lowpass

