**CSC 381Image Processing (Java)**

**Project: 8 Connected Component**

**Student: Jeffrey Pulinat**

**Due date: Java soft copy: 2/20/2018 Tuesday before Midnight**

**early submission (+1) deadline 2/18/2018 before Midnight**

**Due date: Java hard copy: 2/22/2017 Thursday in class**

step 0: read the image header

dynamically allocate zeroFramedAry and all other arrays

step 1: load the input image onto zeroFramedAry

step 2: - 8ConnectCC\_Pass1 // as taught in class

- prettyprint the result of pass1// with proper caption

- print EQAry // with index up to newLable with proper caption

step 3: - 8ConnectCC\_Pass2 // as taught in class

- prettyprint the result of pass2// with caption

- print EQAry // with index up to newLable with caption

step 4: - manageEQAry // as taught in class.

- print EQAry // with index up to newLable with caption

step 5: - 8ConnectCC\_Pass3 // In the pass3, you will use the EQAry to relabel the components;

// keep track the newMin newMax

// as well as compute the property of each c.c.

// and store the c. c. properties

- prettyprint the result of pass3 of the connected c.c. // with caption

- Output the result of pass3 to outFile2 with updated image header

- print the propertis of the connected c.c. // with proper caption

**B) Source Code:**

**EightConnectComp.java**

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.util.Arrays;

import java.util.Scanner;

public class EightConnectedComp {

static int numRows;

static int numCols;

static int minVal;

static int maxVal;

static int newMin;

static int newMax;

static int newLabel = 0;

static int[][] zeroFramedAry;

static int []neighborAry = new int [5];

static int []EQAry;

static int[] Property;

static int totalComp;

public static void main(String[] argv) {

// TODO Auto-generated method stub

loadImage(argv[0]);

EQAry = new int [(numRows\*numCols)/2];

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

EQAry[i]=i;

}

try {

File output1 = new File(argv[1]);

BufferedWriter out = new BufferedWriter(new FileWriter(output1));

out.newLine();

ConnectCC\_Pass1();

out.write("PASS 1");

out.newLine();

prettyPrint(out);

out.newLine();

EQPrint(out);

out.newLine();

out.write("---------------");

out.newLine();

out.write("PASS 2");

out.newLine();

ConnectCC\_Pass2();

prettyPrint(out);

out.newLine();

EQPrint(out);

out.newLine();

out.write("---------------");

out.newLine();

manageEQAry();

out.write("After Managed ");

EQPrint(out);

out.newLine();

out.write("PASS 3");

out.newLine();

File output3 = new File(argv[3]);

BufferedWriter out3 = new BufferedWriter(new FileWriter(output3));

out3.write(numRows+" "+numCols+" "+newMin+" "+newMax);

out3.newLine();

ConnectCC\_Pass3(out3);

out3.close();

prettyPrint(out);

EQPrint(out);

out.newLine();

out.close();

File output2 = new File(argv[2]);

BufferedWriter out2 = new BufferedWriter(new FileWriter(output2));

prettyPrint(out2);

out2.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private static void printCCProperty(BufferedWriter out, ConComp[] comp) {

try{

int tcomp= totalComp-1;

out.write("Total Components: " +tcomp);

out.newLine();

for(int i=1; i<totalComp;i++){

out.write("Label: "+comp[i].getLabel());

out.newLine();

out.write("Number of Pixels: "+comp[i].getNumbpixels());

out.newLine();

out.write("Upper Left pixel of Bounding Box: "+comp[i].getMinRow()+" "+comp[i].getMinCol());

out.newLine();

out.write("Lower Right pixel of Bounding Box: "+comp[i].getMaxRow()+" "+comp[i].getMaxCol());

out.newLine();

out.write("\_\_\_\_\_\_");

out.newLine();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void EQPrint(BufferedWriter out) {

try{

out.write("EQAry:");

out.newLine();

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

out.write(EQAry[i] + ", ");

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static void manageEQAry() {

// TODO Auto-generated method stub

int []unique = new int [newLabel];

totalComp=0;

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

int j;

for (j = 0; j < i; j++){

if (EQAry[i] == EQAry[j])break;

}

//fill array with unique values

if (i == j){

unique[totalComp]=EQAry[i];

totalComp++;

}

}

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

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

if(EQAry[j]==unique[i]){

EQAry[j]=i;

}

}

}

newMin=0;

newMax=totalComp-1;

}

private static void prettyPrint(BufferedWriter out) {

try {

out.write(numRows+" "+numCols+" "+newMin+" "+newMax);

out.newLine();

for(int row = 1; row<=numRows; row++){ //print out from [1][1] w/o border

for(int col = 1; col<=numCols; col++){

int value = zeroFramedAry[row][col];

//int numSpaces = String.valueOf(value).length();

if(value!=0){

if(value>=10){

out.write(value+" ");

}else out.write(value+" ");

}else{ //when skipping 0's

if(value>=10){

out.write(" ");

}else out.write(" ");

}

}

out.newLine();

}

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private static void ConnectCC\_Pass3(BufferedWriter out) {

ConComp[] comp = new ConComp[totalComp];

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

comp[i]= new ConComp();

comp[i].setLabel(i);

}

for(int row = 1; row<=numRows; row++){

for(int col = 1; col<=numCols; col++){

int EQindex = zeroFramedAry[row][col];

int EQvalue = EQAry[EQindex];

zeroFramedAry[row][col]=EQvalue;

if(zeroFramedAry[row][col]==comp[EQvalue].getLabel()){

comp[EQvalue].incNumbpixels();

if(row<comp[EQvalue].getMinRow())comp[EQvalue].setMinRow(row);

if(col<comp[EQvalue].getMinCol())comp[EQvalue].setMinCol(col);

if(row>comp[EQvalue].getMaxRow())comp[EQvalue].setMaxRow(row);

if(col>comp[EQvalue].getMaxCol())comp[EQvalue].setMaxCol(col);

}

}

}

printCCProperty(out,comp);

}

private static void ConnectCC\_Pass2() {

int min;

boolean anyzero = false;

for(int row = numRows; row>=1; row--){

for(int col = numCols; col>=1; col--){

if(zeroFramedAry[row][col]>0){ //p(i,j) is not 0

loadNeighbors(row,col,2);

if(zeroNeighbors()==true){ //CASE 1: all neighbors are 0

//do nothing

}

else if(equalNeighbors(4)!= -1){ //CASE 2: neighbors are equal including (i,j)

//do nothing

}else { //CASE 3 neighbors are diff values

min = Integer.MAX\_VALUE;

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

if(neighborAry[i] > 0 && neighborAry[i]<min) {

min = neighborAry[i];

}

}

int index = zeroFramedAry[row][col];

zeroFramedAry[row][col]=min;

updateEQAry(index,min); //(index,value)

}

}else anyzero = true;

}

}

if(anyzero==false)newMin=1;

else newMin=0;

newMax=newLabel;

}

private static void ConnectCC\_Pass1() {

int min;

boolean anyzero = false;

for(int row = 1; row<=numRows; row++){

for(int col = 1; col<=numCols; col++){

if(zeroFramedAry[row][col]>0){ //p(i,j) is not 0

loadNeighbors(row,col,1);

if(zeroNeighbors()==true){ //CASE 1: all neighbors are 0

newLabel++;

zeroFramedAry[row][col]=newLabel;

}

else if(equalNeighbors(3)!= -1){ //CASE 2: neighbors are equal

zeroFramedAry[row][col]=equalNeighbors(3);

}else { //CASE 3 neighbors are diff values

min = Integer.MAX\_VALUE;

int max = 0;

for(int i=0; i<=3; i++) { //doesnt compare p(i,j)

if(neighborAry[i] > 0 && neighborAry[i]<min) {

min = neighborAry[i];

}

if(neighborAry[i] > 0 && neighborAry[i]>max) {

max = neighborAry[i];

}

}

zeroFramedAry[row][col]=min;

updateEQAry(max,min); //(index,value)

}

}else anyzero = true;

}

}

if(anyzero==false)newMin=1;

else newMin=0;

newMax=newLabel;

}

private static void updateEQAry(int index, int value) {

EQAry[index] = value;

}

private static boolean zeroNeighbors(){ //case1

boolean allZero = true;

for(int i=0; i<=3; i++){ //0 - 3 excludes p(i,j)

if (neighborAry[i] != 0){

allZero = false;

break;

}

}

return allZero;

}

private static int equalNeighbors(int bound){ //case 2

boolean equal = true;

int value = 0;

int count = 1; // keeps track of first non zero element

for(int i=0; i<=bound; i++){ //0-3 excludes p(i,j) 0-4 includes p(i,J)

if (neighborAry[i] > 0 && count == 1 ){

value = neighborAry[i];

count++;

}

if(count==2 && neighborAry[i] != 0 && neighborAry[i] != value){

equal = false;

value = -1;

break;

}

}

//return equal;

return value;

}

private static void loadImage(String argv) {

File input = new File(argv);

Scanner sc;

try {

sc = new Scanner(input);

//assign params

numRows=sc.nextInt();

numCols=sc.nextInt();

minVal=sc.nextInt();

maxVal=sc.nextInt();

zeroFramed(sc);

}catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

}

private static void zeroFramed(Scanner sc){

zeroFramedAry = new int [numRows+2][numCols+2];

for(int row = 1; row<=numRows; row++){

for(int col = 1; col<=numCols; col++){

int value =sc.nextInt();

zeroFramedAry[row][col] = value;

if(row == 1) zeroFramedAry[0][col] = 0; //zeros top row

if(row == numRows) zeroFramedAry[row+1][col] = 0; //zeros bottom row

}

}

for(int side=0; side<=numRows+1; side++){ //zero sides

zeroFramedAry[side][0] = 0; //left side

zeroFramedAry[side][numCols+1] = 0; //right side

}

sc.close();

}

private static void loadNeighbors(int row, int col, int pass) {

/\* a|b|c

\* d|e|f e = p(i,j)

\* g|h|i

\*/

if(pass==1){

neighborAry[0]=zeroFramedAry[row-1][col-1]; //a

neighborAry[1]=zeroFramedAry[row-1][col]; //b

neighborAry[2]=zeroFramedAry[row-1][col+1]; //c

neighborAry[3]=zeroFramedAry[row][col-1]; //d

neighborAry[4]=zeroFramedAry[row][col]; //e p(i,j)

}

if(pass==2){

neighborAry[4]=zeroFramedAry[row][col]; //e p(i,j)

neighborAry[0]=zeroFramedAry[row][col+1]; //f

neighborAry[1]=zeroFramedAry[row+1][col-1]; //g

neighborAry[2]=zeroFramedAry[row+1][col]; //h

neighborAry[3]=zeroFramedAry[row+1][col+1]; //i

}

}

}

**ConComp.java**

public class ConComp {

int label;

int numbpixels = 0;

int minRow = Integer.MAX\_VALUE;

int minCol = Integer.MAX\_VALUE;;

int maxRow = 0;

int maxCol = 0;

public int getLabel() {

return label;

}

public void setLabel(int label) {

this.label = label;

}

public int getNumbpixels() {

return numbpixels;

}

public void incNumbpixels() {

numbpixels++;

}

public int getMinRow() {

return minRow;

}

public void setMinRow(int minRow) {

this.minRow = minRow;

}

public int getMinCol() {

return minCol;

}

public void setMinCol(int minCol) {

this.minCol = minCol;

}

public int getMaxRow() {

return maxRow;

}

public void setMaxRow(int maxRow) {

this.maxRow = maxRow;

}

public int getMaxCol() {

return maxCol;

}

public void setMaxCol(int maxCol) {

this.maxCol = maxCol;

}

}

**OUTPUT FILES**

**outFile1.txt (all passes and EQ Ary)**

PASS 1

20 25 0 22

1 1 2

3 3 4 1 1 5 5 5 2

3 3 4 1 1 6 5 5 5 7 7 7 2

3 3 3 3 3 1 1 6 8 5 5 7 2

3 3 3 1 1 6 8 5 5 5 9 7 7 2

3 3 3 3 1 1 6 8 5 5 5 7 7 7

3 3 3 3 1 1 6 8 5 5 5 7

3 3 3 1 1 6 8 5 5 5 7 10 11

3 3 3 1 1 6 8 5 5 5 7 10 11

3 3 3 1 1 6 8 5 5 5 5 7 7 7 11

3 5 11 11

3 3 12 12 12 12 12 5 5 5 13 14 11

3 3 12 12 12 12 12 5 5 5 13 13 11 15

3 3 12 12 12 12 12 5 5 5 11 15

3 3 12 12 12 16 17 5 5 5 18 11 11 11

12 12 16 11

19 19 20 12 12 16 21 21 21 11

19 19 20 12 12 16 16 21 21 21 22 22 22 11 11

19 19 19 19 12 12 16 16 21 21 22 22 11

19 19 19 19 12 12 12 16 16 16 16 16 16 22 22 22

EQAry:

0, 1, 2, 3, 3, 5, 6, 7, 5, 7, 7, 11, 12, 11, 13, 11, 16, 17, 18, 12, 19, 16, 22,

---------------

PASS 2

20 25 0 22

1 1 2

3 3 3 1 1 5 5 5 2

3 3 3 1 1 6 5 5 5 7 7 7 2

3 3 3 3 3 1 1 6 5 5 5 7 2

3 3 3 1 1 6 5 5 5 5 7 7 7 2

3 3 3 3 1 1 6 5 5 5 5 7 7 7

3 3 3 3 1 1 6 5 5 5 5 7

3 3 3 1 1 6 5 5 5 5 7 7 11

3 3 3 1 1 6 5 5 5 5 7 7 11

3 3 3 1 1 6 5 5 5 5 5 7 7 7 11

3 5 11 11

3 3 12 12 12 12 12 5 5 5 11 11 11

3 3 12 12 12 12 12 5 5 5 11 11 11 11

3 3 12 12 12 12 12 5 5 5 11 11

3 3 12 12 12 16 17 5 5 5 18 11 11 11

12 12 16 11

12 12 12 12 12 16 16 16 16 11

12 12 12 12 12 16 16 16 16 16 22 22 22 11 11

12 12 12 12 12 12 16 16 16 16 22 22 11

19 19 12 12 12 12 12 16 16 16 16 16 16 22 22 22

EQAry:

0, 1, 2, 3, 3, 5, 6, 7, 5, 7, 7, 11, 12, 11, 11, 11, 16, 17, 18, 12, 12, 16, 22,

---------------

After Managed EQAry:

0, 1, 2, 3, 3, 4, 5, 6, 4, 6, 6, 7, 8, 7, 7, 7, 9, 10, 11, 8, 8, 9, 12,

PASS 3

20 25 0 12

1 1 2

3 3 3 1 1 4 4 4 2

3 3 3 1 1 5 4 4 4 6 6 6 2

3 3 3 3 3 1 1 5 4 4 4 6 2

3 3 3 1 1 5 4 4 4 4 6 6 6 2

3 3 3 3 1 1 5 4 4 4 4 6 6 6

3 3 3 3 1 1 5 4 4 4 4 6

3 3 3 1 1 5 4 4 4 4 6 6 7

3 3 3 1 1 5 4 4 4 4 6 6 7

3 3 3 1 1 5 4 4 4 4 4 6 6 6 7

3 4 7 7

3 3 8 8 8 8 8 4 4 4 7 7 7

3 3 8 8 8 8 8 4 4 4 7 7 7 7

3 3 8 8 8 8 8 4 4 4 7 7

3 3 8 8 8 9 10 4 4 4 11 7 7 7

8 8 9 7

8 8 8 8 8 9 9 9 9 7

8 8 8 8 8 9 9 9 9 9 12 12 12 7 7

8 8 8 8 8 8 9 9 9 9 12 12 7

8 8 8 8 8 8 8 9 9 9 9 9 9 12 12 12

EQAry:

0, 1, 2, 3, 3, 4, 5, 6, 4, 6, 6, 7, 8, 7, 7, 7, 9, 10, 11, 8, 8, 9, 12,

**outFile2.txt (Final Pass)**

20 25 0 12

1 1 2

3 3 3 1 1 4 4 4 2

3 3 3 1 1 5 4 4 4 6 6 6 2

3 3 3 3 3 1 1 5 4 4 4 6 2

3 3 3 1 1 5 4 4 4 4 6 6 6 2

3 3 3 3 1 1 5 4 4 4 4 6 6 6

3 3 3 3 1 1 5 4 4 4 4 6

3 3 3 1 1 5 4 4 4 4 6 6 7

3 3 3 1 1 5 4 4 4 4 6 6 7

3 3 3 1 1 5 4 4 4 4 4 6 6 6 7

3 4 7 7

3 3 8 8 8 8 8 4 4 4 7 7 7

3 3 8 8 8 8 8 4 4 4 7 7 7 7

3 3 8 8 8 8 8 4 4 4 7 7

3 3 8 8 8 9 10 4 4 4 11 7 7 7

8 8 9 7

8 8 8 8 8 9 9 9 9 7

8 8 8 8 8 9 9 9 9 9 12 12 12 7 7

8 8 8 8 8 8 9 9 9 9 12 12 7

8 8 8 8 8 8 8 9 9 9 9 9 9 12 12 12

**outFile3.txt (Final Pass Image Header + Connected Components details)**

20 25 0 12

Total Components: 12

Label: 1

Number of Pixels: 20

Upper Left pixel of Bounding Box: 1 8

Lower Right pixel of Bounding Box: 10 9

\_\_\_\_\_\_

Label: 2

Number of Pixels: 5

Upper Left pixel of Bounding Box: 1 25

Lower Right pixel of Bounding Box: 5 25

\_\_\_\_\_\_

Label: 3

Number of Pixels: 40

Upper Left pixel of Bounding Box: 2 1

Lower Right pixel of Bounding Box: 15 5

\_\_\_\_\_\_

Label: 4

Number of Pixels: 47

Upper Left pixel of Bounding Box: 2 13

Lower Right pixel of Bounding Box: 15 17

\_\_\_\_\_\_

Label: 5

Number of Pixels: 8

Upper Left pixel of Bounding Box: 3 11

Lower Right pixel of Bounding Box: 10 11

\_\_\_\_\_\_

Label: 6

Number of Pixels: 18

Upper Left pixel of Bounding Box: 3 20

Lower Right pixel of Bounding Box: 10 22

\_\_\_\_\_\_

Label: 7

Number of Pixels: 22

Upper Left pixel of Bounding Box: 8 19

Lower Right pixel of Bounding Box: 19 25

\_\_\_\_\_\_

Label: 8

Number of Pixels: 43

Upper Left pixel of Bounding Box: 12 2

Lower Right pixel of Bounding Box: 20 9

\_\_\_\_\_\_

Label: 9

Number of Pixels: 21

Upper Left pixel of Bounding Box: 15 11

Lower Right pixel of Bounding Box: 20 17

\_\_\_\_\_\_

Label: 10

Number of Pixels: 1

Upper Left pixel of Bounding Box: 15 13

Lower Right pixel of Bounding Box: 15 13

\_\_\_\_\_\_

Label: 11

Number of Pixels: 1

Upper Left pixel of Bounding Box: 15 20

Lower Right pixel of Bounding Box: 15 20

\_\_\_\_\_\_

Label: 12

Number of Pixels: 8

Upper Left pixel of Bounding Box: 18 20

Lower Right pixel of Bounding Box: 20 22

\_\_\_\_\_\_