CSC 381-34: Proj1 (JAVA)

Swrajit Paul

Due date: Sept. 13, 2018

Algorithm steps:

```
III. Main( )
*****************************

step 0: - open the input file and output file
        - read the image header, the four numbers
        - dynamically allocate all 1-D and 2-D array

        - thr_value <-- read from argv[1]

step 1: loadImage (imgInAry, mirrorFramedAry)
           // read from input file and load onto imgInAry begins at [0][0]
           // and load onto mirrorFramedAry begin at [1,1]

step 2: ComputeHistogram(imgInAry, hist, maxVal)

step 3: printHist(hist) // see the format in the above.

step 4: mirrowFramed (mirrorFramedAry) // Use the algorithm given in class

step 5: computeAVG3X3 (mirrorFramedAry,  tempAry) // see algorithm below

Step 6: computThreshold (tempAry,imgOutAry)

Step 7: prettyPrint (imgOutAry)

step 8: output the image header (numRows, numCols, newMin, newMax)
     to Output2(argv[3]): the result of thresholded  image

step 9: output tempAry, begin at [1,1], within the frame, to Output2(argv[3])

step 10: close all files

*****************************
VI. computeHistogram(imgInAry, maxVal, hist )
*****************************

step 1: dynamically allocate the hist array size maxVal+1 and initialize to 0

step 2: // process imgInAryfrom left to right and top to bottom

     p(i,j) <- next pixel
     hist[p(i,j)]++

step 3: repeat step 2 until all pixels are processed.


************************************
IV. computeAVG3X3 (mirrorFramedAry,  tempAry)
```

```
************************************

step 1: process the MirrorframedAry, from left to right and top to bottom
        using i, and j, begin at (1, 1) until one before the last row.

       p(i,j) <-- next pixel

Step 2: loadNeighbors (neighborAry)
      // load the 3 x 3 neighbors of p(i,j)into neightAry

step 3: tempAry(i,j) <-- Avg3x3(neighborAry)
           // compute the averaging of neighborAry

        - keep tracking the newMin and newMax of tempAry

step 4: repeat step 1 - step3  until all pixels inside of the framed are
processed


*******************************
VI. computeThreshold(MirrorframedAry, imgOutAry, thr_value)
*******************************

step 1: process the MirrorframedAry, from left to right and top to bottom
        using i, and j, begin at (1, 1) until one before the last row.

       p(i,j) <-- next pixel

      if (p(i,j)>= thr_value
           imgOutAry(i,j) <--  1
      else
           imgOutAry(i,j) <--  0

step 2: repeat step 1 until all pixels are processed.


***********************************
III. prettyPrint (imgOutAry)
***********************************

step 1: outFile <-- open Output2(argv[3]

step 2: // process imgOutAry from left to right and top to bottom

      p(i,j) <- next pixel
      if p(i,j) > 0
         output p(i,j) to outFile
     else
         output '  ' // 2 blanks to outFile

step 3: repeat step 2 until all pixels are processed.
```

SOURCE CODE

```
/**
 * Author: Swrajit Paul
 */

import java.io.*;
import java.util.Scanner;

public class imageProcessing {

	static int numRows;
	static int numCols;
	static int minVal;
	static int maxVal;
	static int newMin;
	static int newMax;

	static int[][] imgInAry;
	static int[][] imgOutAry;
	static int[][] mirrorFramedAry;
	static int[][] tempAry;
   static int[] hist;
   static int[] neighborAry = new int[9];

   static FileInputStream fInput = null;
   static FileOutputStream fOutputone;
   static FileOutputStream fOutputtwo;
   static FileOutputStream fOutputthree;

   static Scanner inputfile;

   public imageProcessing() {

   }

   public static void loadImage(int[][] arrayA,int[][] arrayB) {

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

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

				arrayA[i][j] = inputfile.nextInt();
			}
		}

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

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

				arrayB[i+1][j+1] = arrayA[i][j];
			}
		}

	}
```

```
public static void ComputeHistogram(int[][] imgInAry, int[] hist, int mVal) {

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

		for(int j = 0; j < imgInAry[0].length; j++) {

			hist[imgInAry[i][j]] += 1;

		}
	}

}

public static void printHist(int[] hist) {

	PrintStream print = new PrintStream(fOutputone);
	print.println(numRows + " " + numCols + " " + minVal + " " + maxVal);

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

		print.println(j + " " + hist[j]);

	}
}

public static void mirrowFramed (int[][] mirrorFramedAry) {
	// java automatically initializes everything to zero
}

public static void computeAVG3X3 (int[][] mirrorFramedAry,int[][] tempAry) {// see algorithm below

	newMin = Integer.MAX_VALUE;
	newMax = 0;

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

		for(int j = 1; j < mirrorFramedAry[0].length-1; j++) {

			neighborAry[0] = mirrorFramedAry[i-1][j-1];
			neighborAry[1] = mirrorFramedAry[i-1][j];
			neighborAry[2] = mirrorFramedAry[i-1][j+1];
			neighborAry[3] = mirrorFramedAry[i][j-1];
			neighborAry[4] = mirrorFramedAry[i][j];
			neighborAry[5] = mirrorFramedAry[i][j+1];
			neighborAry[6] = mirrorFramedAry[i+1][j-1];
			neighborAry[7] = mirrorFramedAry[i+1][j];
			neighborAry[8] = mirrorFramedAry[i+1][j+1];

			int sum =0;

			for (int k =0; k < neighborAry.length; k++) {
				sum += neighborAry[k];
			}

			int avg = sum / 9;
```

```
                        tempAry[i][j] = avg;

                        if (avg <= newMin) {
                                newMin = avg;
                        }
                        if (avg >= newMax) {
                                newMax = avg;
                        }

                }
        }
}

public static void computThreshold (int[][] tempAry, int[][] imgOutAry, int thr_value) {

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

                for(int j = 1; j < tempAry[0].length-1; j++) {

                        int pixel = tempAry[i][j];

                        if(pixel >= thr_value) {
                                imgOutAry[i-1][j-1] = 1;
                        }
                        else {
                                imgOutAry[i-1][j-1]= 0;
                        }

                }
        }
}

public static void prettyPrint (int[][] imgOutAry) {

        PrintStream print = new PrintStream(fOutputthree);
        print.println(numRows + " " + numCols + " " + minVal + " " + maxVal);

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

                for(int j = 0; j < imgOutAry[0].length; j++) {

                        if(imgOutAry[i][j] > 0) {
                                print.print(imgOutAry[i][j]);
                        }
                        else {
                                print.print(" ");
                        }
                }
                print.println();
        }
}

public static void main(String[] args) {
```

```
int thr_Value = 0;
try {

	String inputone = args[0];
	thr_Value = Integer.parseInt(args[1]);
	String outputone = args[2];
	String outputtwo = args[3];
	String outputthree = args[4];

	fInput = new FileInputStream(inputone);
	fOutputone = new FileOutputStream(outputone);
	fOutputtwo = new FileOutputStream(outputtwo);
	fOutputthree = new FileOutputStream(outputthree);

} catch (IOException e) {
	System.out.println("one of the arguments in missing or wrong");
}


inputfile = new Scanner(fInput);

numRows = inputfile.nextInt();
numCols = inputfile.nextInt();
minVal = inputfile.nextInt();
maxVal = inputfile.nextInt();

imgInAry = new int[numRows][numCols];
imgOutAry = new int[numRows][numCols];
mirrorFramedAry = new int[numRows+2][numCols+2];;
tempAry = new int[numRows+2][numCols+2];
hist = new int[maxVal+1];

loadImage(imgInAry, mirrorFramedAry);
ComputeHistogram(imgInAry, hist, maxVal);
printHist(hist);
mirrowFramed (mirrorFramedAry);
computeAVG3X3 (mirrorFramedAry,  tempAry);
computThreshold (tempAry, imgOutAry, thr_Value);
prettyPrint (imgOutAry);

PrintStream print = new PrintStream(fOutputtwo);
print.println(numRows + " " + numCols + " " + minVal + " " + maxVal);

for(int i = 1; i < tempAry.length-1; i++) {
	for(int j = 1; j < tempAry[0].length-1; j++) {
		print.print(tempAry[i][j]);
	}
	print.println();
} // writing to outputfile

inputfile.close();
try {
	fInput.close();
} catch (IOException e) {e.printStackTrace();
} } }
```

INPUT

DATA 1

```
31 40 0 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 7 2 3 2 3 5 3 2 3 1 2 1 2 1 2 1 2 1 0
1 2 1 2 1 2 2 3 2 3 2 3 2 1 2 1 2 4 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
0 3 1 2 1 2 1 2 1 2 1 2 1 2 4 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 0 1
1 2 1 2 1 2 4 3 2 3 2 3 2 3 5 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 1 2 1 2 1 2 1 2 1 0
0 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 9 5 0 1
1 2 4 2 1 2 4 3 2 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 2 1 2 9 2 1 0
0 1 2 1 2 3 5 4 5 7 7 8 7 7 8 7 7 7 7 8 7 8 7 7 7 8 7 7 7 7 7 7 5 2 9 3 2 1 2 1
0 2 3 1 2 1 2 1 5 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 7 6 1 2 1 2 1 2 1
1 2 3 3 2 1 2 1 6 9 9 9 9 7 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 8 7 5 1 2 1 4 1 1 0
0 0 1 1 2 2 1 2 3 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 8 7 5 1 2 1 2 5 1 0
1 2 3 3 2 1 2 1 5 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 8 7 2 1 2 1 2 1 1 0
0 9 9 1 2 1 2 1 5 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 0 9 9 9 9 8 7 1 2 1 5 1 1 0 0
0 1 2 3 3 1 2 1 5 8 9 9 9 9 0 9 1 9 0 9 9 9 9 9 9 9 0 0 9 9 9 9 9 8 2 1 2 1 2 4 5 0
0 0 1 1 2 2 1 2 6 9 9 9 9 0 0 9 9 0 0 9 9 9 9 9 9 9 9 9 9 9 9 8 7 1 1 2 1 2 1 0 0
0 0 1 1 2 2 4 5 6 9 9 9 9 9 9 9 9 1 1 9 9 9 9 9 9 9 9 9 9 9 9 8 7 1 2 1 2 1 0 0 0
1 2 3 3 4 4 5 5 7 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 2 9 9 9 9 9 8 7 2 1 2 1 2 1 5 0
0 0 1 1 2 2 4 5 6 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 0 9 9 9 9 9 9 9 8 7 1 1 2 1 2 1 0 0
0 1 2 3 3 3 3 5 5 9 9 7 9 9 9 9 9 9 9 9 9 9 9 9 0 0 9 9 9 9 9 9 9 1 1 2 1 9 1 4 0 0
0 2 3 1 1 3 3 4 5 8 8 8 8 8 8 8 8 8 9 8 8 8 8 7 8 8 8 8 8 8 8 7 1 1 2 1 2 9 0 1
0 1 2 1 2 3 5 6 5 9 9 7 9 7 7 8 7 8 8 7 9 7 9 7 7 7 2 9 7 7 7 7 0 0 1 1 2 1 2 1
0 3 1 2 1 2 4 3 2 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 5 6 2 1 2 1 2 0 1
1 2 1 2 1 2 4 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 1 2 1 2 4 8 1 2 1 0
0 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 0 1
1 2 1 2 1 2 4 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 1 2 1 2 3 2 1 2 1 0
0 3 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 0 1
0 1 2 1 2 1 2 1 2 1 2 1 2 1 5 1 2 1 2 1 2 1 2 1 5 1 2 1 2 1 2 1 2 1 2 5 2 1 2 1
2 3 4 3 2 3 2 3 5 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 1 2 1 2 1 2 1 2 1 0
1 2 1 2 5 2 2 3 2 3 2 3 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

DATA 2

```
31 40 0 9
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2 3 1 3 2 3 1 1 2 3 1 1 2 1 1 1 2 1 0 1 1 0 0 0 0 0 0 0 0 0 2 3 1 3 2 3 1 1 2 3
0 1 1 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 8 1 2 1 1 1 2 1 1 1 2 1 1 9 9 1 1 1 2 1
2 3 2 3 2 3 2 1 2 3 2 1 2 3 2 3 2 3 1 7 9 9 1 1 2 3 2 1 2 3 2 3 1 9 9 3 1 1 2 3
0 0 2 3 1 1 2 3 0 2 3 1 2 3 1 1 2 3 9 8 8 7 9 2 3 1 1 2 3 2 3 1 1 2 2 3 1 1 2 3
1 0 1 2 0 2 2 0 3 0 3 0 1 0 2 0 1 7 7 9 9 8 8 7 0 1 2 0 2 2 0 3 0 3 0 1 0 2 0 2
0 0 2 1 1 1 2 1 1 1 2 1 1 1 2 1 9 8 8 7 9 0 0 7 9 0 0 0 1 1 2 1 1 1 2 1 1 1 2 1
2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 8 7 9 9 7 7 0 0 8 8 8 2 1 1 1 2 1 1 1 2 1 1 1 2 1
0 1 2 1 0 1 2 1 2 1 2 2 2 0 8 7 9 8 7 7 9 9 8 8 7 9 9 1 2 0 2 0 1 3 2 0 1 1 2 0
2 3 2 3 2 3 2 1 2 3 2 1 2 8 7 9 8 7 7 9 9 8 8 7 9 9 7 9 1 1 2 1 1 1 2 1 1 1 2 1
0 1 3 2 0 1 1 2 0 0 0 0 8 7 9 8 7 7 9 9 8 8 7 9 9 8 7 9 9 0 0 0 0 3 2 0 1 1 2 0
0 0 0 3 2 0 1 1 2 0 0 8 7 9 8 7 7 9 9 8 8 7 0 0 6 9 8 8 7 8 3 2 0 1 1 2 0 0 0 0
0 3 2 0 1 1 2 0 1 0 7 9 8 7 7 9 9 8 8 7 9 9 0 0 9 9 8 8 7 8 9 0 0 3 2 0 1 1 2 0
0 3 2 0 1 1 2 0 0 9 8 8 8 9 8 8 7 7 7 9 9 7 9 9 8 8 7 8 9 7 7 9 3 2 0 1 1 2 0 0
0 1 3 2 0 1 1 2 7 9 8 7 7 9 9 8 9 8 8 7 0 0 7 7 8 7 8 9 7 8 8 7 9 3 2 0 1 1 2 0
3 2 0 1 1 2 0 9 8 9 8 8 7 9 8 7 7 9 9 8 0 7 9 9 6 9 8 8 7 8 6 9 8 8 2 1 1 1 2 3
0 0 0 0 0 0 1 1 7 9 9 6 9 8 8 7 9 9 8 8 7 9 9 8 7 9 9 6 9 8 8 7 9 2 1 2 1 0 0 0
0 0 8 9 0 0 0 1 1 9 8 8 7 9 0 0 7 9 9 8 8 7 9 9 6 0 0 8 7 8 6 9 2 1 0 8 9 0 0 0
0 0 8 9 0 0 0 0 1 1 8 7 7 9 0 0 9 8 8 7 9 9 7 7 8 0 0 9 7 8 8 2 1 0 0 8 9 0 0 0
3 2 0 1 1 2 0 0 0 1 1 8 8 9 8 8 7 7 7 9 9 7 0 9 8 8 7 8 9 7 2 1 8 9 0 0 0 0 0 0
0 1 9 9 0 1 1 2 0 0 1 1 8 7 7 9 9 8 8 7 9 9 7 7 9 9 8 8 7 2 1 0 8 9 2 0 1 1 2 0
0 0 9 9 0 1 1 2 0 0 0 1 1 9 8 7 7 9 9 8 8 7 9 9 1 9 8 8 2 1 1 1 2 1 1 1 2 1 1 1
```

```
0 2 0 1 3 2 0 1 1 2 0 1 1 1 9 8 7 7 9 9 8 8 7 9 9 8 7 2 1 0 2 0 1 3 2 0 1 1 2 0
0 1 3 2 0 1 1 2 0 0 0 0 0 1 1 9 8 7 7 9 9 8 8 7 9 9 2 1 1 1 2 1 1 1 2 1 1 1 2 1
1 1 2 1 1 1 2 1 1 1 2 1 2 0 1 1 9 8 7 7 9 9 8 8 7 2 1 1 2 0 2 0 1 8 9 0 1 1 2 0
2 3 2 1 2 3 2 1 2 0 0 0 0 0 0 1 1 9 9 7 7 9 9 8 2 1 1 1 1 1 2 1 1 8 9 1 1 1 2 1
0 0 8 9 0 3 2 0 1 1 2 0 1 0 0 0 1 1 8 7 9 9 7 2 1 0 0 0 1 1 2 1 1 1 2 1 1 1 2 1
0 2 8 9 1 2 3 2 1 2 0 0 0 0 0 0 0 1 1 9 9 8 0 1 0 1 2 0 2 2 0 3 0 3 0 1 0 2 0 2
0 0 0 0 0 0 0 0 0 2 3 2 1 2 3 2 1 2 1 1 8 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 2 1 2 1 2 1 2 2 2 1 2 1 2 1 2 1 2 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

OUTPUT

OUTPUT FOR DATA 1

Histogram for data 1

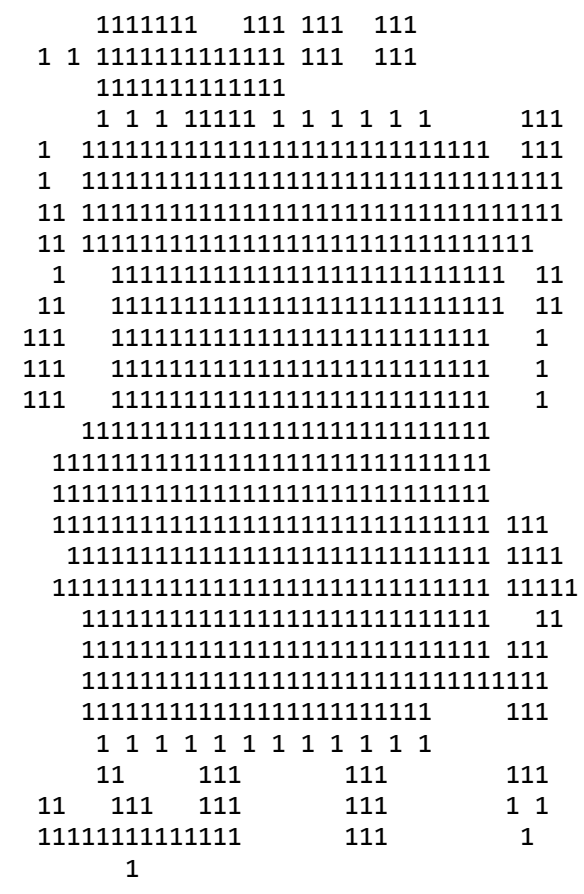
```
31 40 0 9
0 146
1 271
2 309
3 99
4 19
5 54
6 30
7 46
8 61
9 205
```

AVG FILTER IMG for Data 1

```
31 40 0 9
0000000000000000000000000000000000000000
0111111111111111111111111111111111111100
1111112222222111222122211222111111111110
1121212222222222222122211222111111111110
1111112222222222222111111111111111111110
1111112121212222212121212121211111122210
1121122222333333333333333333333221133310
0121122334444444444444444444444433243310
0122122345676767666777666766666643232210
0122122357877788888888888888877643221110
0112111247888888888888888888887642112210
0122111257988899999999998889988642112210
1232111247999999999999987678988531112110
1333111247899887878899986457988531112110
1232111257898666756799987567988531112110
0011122357898666634699998778988531111110
0012223468998778755799998889988531111100
0012234568999999877899887889988531111100
0112234568888999999997657899874212221110
0111223467888888888888655788887421223210
0112223467888888888888765677887421223220
0111123456777767777777676666766531112210
0111123344555555555555555444544321222110
1111122222333333333333333333333222222110
1111122222222222222222222222211111222110
1111112121212121212121212121211111111110
0111112211111222111111122211111111222110
1122111222111222111111122211111111212110
1122222222222222111111122211111111121110
0111111121111111111111111111111111111100
0000110000000000000000000000000000000000
```

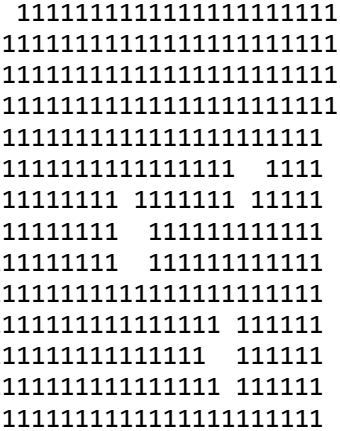
Threshold Value = 2

31 40 0 9



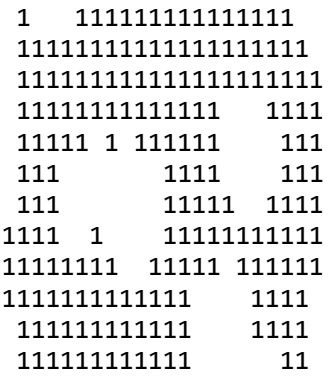
Threshold Value = 6

31 40 0 9



Threshold Value = 8

```
31 40 0 9
```



OUTPUT FOR DATA 2

Histogram for data 2

```
31 40 0 9
0 317
1 295
2 193
3 63
4 0
5 0
6 7
7 99
8 123
9 143
```

AVG FILTER IMG for Data 2

```
31 40 0 9
00000000000000000000000000000000000000
00111111111111010001110000000001122310010
11212211121111111134310111111134531111
01111111111112112235663211111122134531111
01111111211111123678753211112212332111 1
00111111111111123678765531011111111111111
00111111111101135778644553111111111111110
01111111111112357877644675310111111110110
11111111111123578777766787631111111110110
01211111111235787778888888763100111110110
01111111111357877788876678875311111110000
01111111012577777888864468877531111110000
01111101135787877788864468877753211000000
01110111357788888777655678877775421001 00
11111235787888777865578778777776421111 0
01100014688778888886557777877777642101 10
01221013678777657887678876667777632221 00
```

```
013320013577764468888887644677753124420 0
024420001356864467888777644676543223320 0
024420000135776677788766766775334432210 0
024420100013578777888776778753124420000 0
024421100001357778888887777531012321011 0
013321100000135777888887766311111111111 0
011111111000013577788888763211011321011 0
011111110000001357778887532111112442011 0
023221110100000135778875310001112442011 0
024432211100000013578753100011112221111 0
024321111111000001356531000000101000000 0
012211111111111111234310000000101000000 0
000000001111111110111000000000000000000 0
000000000000000000000000000000000000000 0
```

Threshold Value = 2

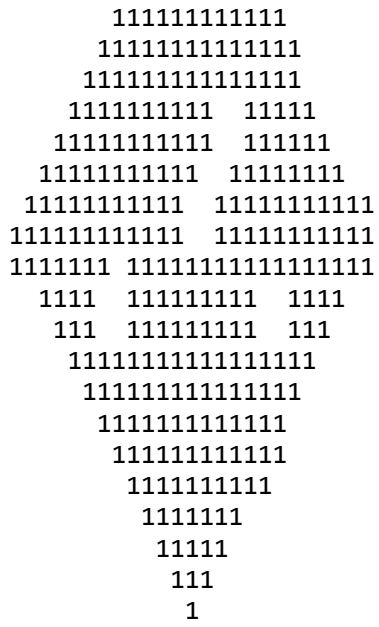
```
31 40 0 9

                                  111
  1 11   1         111            1111
             1  11111111      11 1111
         1      111111111     11 1111
               1111111111
               11111111111
             11111111111111
            1111111111111111
  1        111111111111111111
           111111111111111111
          1111111111111111111
         11111111111111111111111
       1111111111111111111111111
      111111111111111111111111111
       11111111111111111111111111
  11   111111111111111111111111111111
  111   1111111111111111111111111 1111
 1111    11111111111111111111111111111
 1111     111111111111111111111111111
 1111      111111111111111111 1111
 1111       11111111111111111   111
  111        111111111111111
              11111111111111     11
               111111111111     1111
 1111           111111111       1111
 111111          1111111        111
 1111             11111
  11              1111
```

Threshold Value = 6

```
31 40 0 9


                    11
                  1111
                 11111
                 1111
                11111  11
```



Threshold Value = 8

31 40 0 9