

Cover Sheet

CV

Project 3: MorphologyBasic4

Java

Student: Fengzhang Du

Project Due Date: 02/28/2021

Algorithm Steps for Compute Noise Filters:

step 0:

Open imgFile, structFile, dilateOutFile, erodeOutFile, openingOutFile, closingOutFile, prettyPrintFile.

step 1:

numRows, numCols, minVal, maxVal \leftarrow read from imgFile.

step 2:

struct.numRows, struct.numCols, struct.minVal, struct.maxVal \leftarrow read from structFile.

struct.rowOrigin, struct.colOrigin \leftarrow read from structFile.

// Struct Class are inherited from Image Class, which stores all the information related to the Structuring element.

step 3:

loadImg (imgFile);

zeroFramedAry, structAry, morphAry, tempAry \leftarrow dynamically allocate

step 4:

loadStruct (structFile);

step 5:

prettyPrint ("Original Image", zeroFramedAry, prettyPrintFile);

prettyPrint ("Structuring Element", struct.structAry, prettyPrintFile);

step 6:

computeDilation (zeroFramedAry, morphAry);

aryToFile (morphAry, dilateOutFile);

prettyPrint ("Dilation", morphAry, prettyPrintFile);

zero2DAry();

step 7:

computeErosion (zeroFramedAry, morphAry);

aryToFile (morphAry, erodeOutFile);

prettyPrint (morphAry, prettyPrintFile);

zero2DAry();

step 8:

computeClosing (zeroFramedAry, tempAry, morphAry);

aryToFile (morphAry, closingOutFile);

prettyPrint (morphAry, prettyPrintFile);

zero2DAry();

step 9:

computeOpening (zeroFramedAry, tempAry, morphAry);

aryToFile (morphAry, openingOutFile);

prettyPrint (morphAry, prettyPrintFile);

step 10:

close all files. (automatically in Java 7).

Source Code

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

public class Main{
    public static void main(String[] args) throws IOException{
        try (
            // input image
            Scanner input = new Scanner(new BufferedReader(new FileReader(args[0])));
            // input structure
            Scanner input_struct = new Scanner(new BufferedReader(new FileReader(args[1])));
            // output
            BufferedWriter dilateOutFile = new BufferedWriter(new FileWriter(args[2]));
            BufferedWriter erodeOutFile = new BufferedWriter(new FileWriter(args[3]));
            BufferedWriter closingOutFile = new BufferedWriter(new FileWriter(args[4]));
            BufferedWriter openingOutFile = new BufferedWriter(new FileWriter(args[5]));
            // - prettyPrintFile (args[6]): pretty print which are stated in the algorithm steps
            BufferedWriter prettyPrintFile = new BufferedWriter(new FileWriter(args[6], true));
        ){
            // Read and store image header info.
            int header[] = new int[4];
            for (int i=0; i<4; i++){
                if(input.hasNextInt()) header[i] = input.nextInt();
            }

            // Read and store Structure Element header info.
            int header_struct[] = new int[6];
            for (int i=0; i<6; i++){
                if(input_struct.hasNextInt()) header_struct[i] = input_struct.nextInt();
            }

            Image img = new Image(header[0], header[1], header[2], header[3]);
            img.struct = new Struct(header_struct[0], header_struct[1], header_struct[2],
header_struct[3], header_struct[4], header_struct[5]);

            img.loadImg(input);
            img.loadStruct(input_struct);
            // prettyPrint 1 -> original img
            img.prettyPrint("Original Image", img.zeroFramedAry, prettyPrintFile);
            // prettyPrint 2 -> structure element
            img.prettyPrint("Structuring Element", img.struct.structAry, prettyPrintFile);

            // - dilateOutFile (args[2]): the result of dilation image with header.
            img.computeDilation(img.zeroFramedAry, img.morphAry);
            img.aryToFile(img.morphAry, dilateOutFile);
            img.prettyPrint("Dilation", img.morphAry, prettyPrintFile); // prettyPrint 3 -> dilation
            img.zero2DAry();

            // - erodeOutFile (args[3]): the result of erosion image with headers.
            img.computeErosion(img.zeroFramedAry, img.morphAry);
            img.aryToFile(img.morphAry, erodeOutFile);
            img.prettyPrint("Erosion", img.morphAry, prettyPrintFile);
            img.zero2DAry();
        }
    }
}
```

```

        // - closingOutFile (args[4]): the result of closing image with header.
        img.computeClosing(img.zeroFramedAry, img.tempAry, img.morphAry);
        img.aryToFile(img.morphAry, closingOutFile);
        img.prettyPrint("Closing", img.morphAry, prettyPrintFile);
        img.zero2DAry();

        // - openingOutFile (args[5]): the result of opening image with header.
        img.computeOpening(img.zeroFramedAry, img.tempAry, img.morphAry);
        img.aryToFile(img.morphAry, openingOutFile);
        img.prettyPrint("Opening", img.morphAry, prettyPrintFile);
        // close input and output file automatically in Java 7.
    }
}

}

class Image {
    // fields
    int numRows=0, numCols=0, minVal=0, maxVal=0;
    // int[][] body;
    int[][] zeroFramedAry;
    int[][] morphAry;
    int[][] tempAry;
    Struct struct;

    // constructor
    public Image(int numRows, int numCols, int minVal, int maxVal) {
        this.numRows = numRows;
        this.numCols = numCols;
        this.minVal = minVal;
        this.maxVal = maxVal;
    }

    // methods
    void loadImg(Scanner input){
        this.zeroFramedAry = new int[this.numRows+ struct.extraRows][this.numCols + struct.extraCols];
        this.morphAry = new int[this.numRows+ struct.extraRows][this.numCols + struct.extraCols];
        this.tempAry = new int[this.numRows+ struct.extraRows][this.numCols + struct.extraCols];

        for(int i=struct.rowFrameSize; i<numRows + struct.rowFrameSize; i++){
            for(int j=struct.colFrameSize; j<numCols + struct.colFrameSize; j++){
                if(input.hasNextInt()) zeroFramedAry[i][j] = input.nextInt();
                else{
                    System.out.println( "Corrupted Image input data!");
                    System.exit(0);
                }
            }
        }
    }

    void loadStruct(Scanner input_struct){
        int index = 0;
        for(int i=0; i<struct.numRows; i++){
            for(int j=0; j<struct.numCols; j++){
                if(input_struct.hasNextInt()) {
                    int temp = input_struct.nextInt();
                    struct.structAry[i][j] = temp;
                    struct.stlD[index] = temp;
                }
            }
        }
    }
}

```

```

        index++;
    }
    else{
        System.out.println( "Corrupted struct input data!");
        System.exit(0);
    }
}
}

void zero2DAry(){ // set a 2D array to 0.
    this.morphAry = new int[this.numRows+ struct.extraRows][this.numCols + struct.extraCols];
    this.tempAry = new int[this.numRows+ struct.extraRows][this.numCols + struct.extraCols];
}

void computeDilation(int[][] zeroFramedAry, int[][] morphAry){
    for(int i=struct.rowFrameSize; i<numRows + struct.rowFrameSize; i++){
        for(int j=struct.colFrameSize; j<numCols + struct.colFrameSize; j++){
            if (zeroFramedAry[i][j] == 1){
                dilation(i, j, zeroFramedAry, morphAry);
            }
        }
    }
}

void dilation(int i, int j, int[][] inAry, int[][] outAry){
    int neighbors[] = new int[struct.stlD.length];
    int index = 0;
    // store input array's neighbors as 1d array.
    for (int k=i-struct.rowOrigin; k<i+(struct.numRows-struct.rowOrigin); k++){
        for (int d=j-struct.colOrigin; d<j+(struct.numCols-struct.colOrigin); d++){
            neighbors[index] = inAry[k][d];
            index++;
        }
    }
    // comput output array.
    for (int u=0; u<neighbors.length; u++){
        if(struct.stlD[u] == 1) neighbors[u] = 1;
    }

    index = 0;
    for (int k=i-struct.rowOrigin; k<i+(struct.numRows-struct.rowOrigin); k++){
        for (int d=j-struct.colOrigin; d<j+(struct.numCols-struct.colOrigin); d++){
            if(neighbors[index] == 1){
                outAry[k][d] = neighbors[index];
            }
            index++;
        }
    }
}

void computeErosion(int[][] zeroFramedAry, int[][] morphAry){
    for(int i=struct.rowFrameSize; i<numRows + struct.rowFrameSize; i++){
        for(int j=struct.colFrameSize; j<numCols + struct.colFrameSize; j++){
            if (zeroFramedAry[i][j] > 0){
                erosion(i, j, zeroFramedAry, morphAry);
            }
        }
    }
}

```

```

    }
}

void erosion(int i, int j, int[][] inAry, int[][] outAry){
    int neighbors[] = new int[struct.stlD.length];
    int index = 0;
    // store input array's neighbors as 1d array.
    for (int k=i-struct.rowOrigin; k<i+(struct.numRows-struct.rowOrigin); k++){
        for (int d=j-struct.colOrigin; d<j+(struct.numCols-struct.colOrigin); d++){
            neighbors[index] = inAry[k][d];
            index++;
        }
    }
    // comput output array.
    for (int u=0; u<neighbors.length; u++){
        // this pixel does not match the structing element, mark with 0.
        if(struct.stlD[u] == 1 && neighbors[u] == 0) {
            outAry[i][j] = 0;
            return;
        }
    }
    outAry[i][j] = 1;
}

void computeClosing(int[][] zeroFramedAry, int[][] tempAry, int[][] morphAry){
    this.computeDilation(zeroFramedAry, tempAry);
    this.computeErosion(tempAry, morphAry);
}

void computeOpening(int[][] zeroFramedAry, int[][] tempAry, int[][] morphAry){
    this.computeErosion(zeroFramedAry, tempAry);
    this.computeDilation(tempAry, morphAry);
}

void aryToFile(int [][] arr, BufferedWriter output) throws IOException {
    output.write(Integer.toString(numRows) + " " + Integer.toString(numCols) + " ");
    output.write(Integer.toString(minVal) + " " + Integer.toString(maxVal) + "\n");
    for(int i=struct.rowFrameSize; i<arr.length-struct.rowFrameSize; i++){
        for(int j=struct.colFrameSize; j<arr[0].length-struct.colFrameSize; j++){
            output.write(Integer.toString(arr[i][j]) + " ");
        }
        output.write("\n");
    }
}

void prettyPrint(String title, int [][] arr, BufferedWriter output) throws IOException {
    output.write(title + "\n");
    // print structing element.
    if(this.struct != null && arr == struct.structAry ) {
        output.write(Integer.toString(struct.numRows) + " " + Integer.toString(struct.numCols) + " ");
        output.write(Integer.toString(struct.minVal) + " " + Integer.toString(struct.maxVal) + "\n");
        output.write(Integer.toString(struct.rowOrigin) + " " +
Integer.toString(struct.colOrigin)+"\n");
        for(int i=0; i<arr.length; i++){
            for(int j=0; j<arr[i].length; j++){
                if(arr[i][j] == 0){
                    output.write("." + " ");
                }
            }
        }
    }
}

```

```

        }else{
            output.write(Integer.toString(arr[i][j]) + " ");
        }
    }
    output.write("\n");
}
}else{ // print image
    output.write(Integer.toString(numRows) + " " + Integer.toString(numCols) + " ");
    output.write(Integer.toString(minVal) + " " + Integer.toString(maxVal) + "\n");
    for(int i=struct.rowFrameSize; i<arr.length-struct.rowFrameSize; i++){
        for(int j=struct.colFrameSize; j<arr[0].length-struct.colFrameSize; j++){
            if(arr[i][j] == 0){
                output.write(".") + " ";
            }else{
                output.write(Integer.toString(arr[i][j]) + " ");
            }
        }
        output.write("\n");
    }
    output.write("\n");
}
}
}

class Struct extends Image{
    // fields
    int rowOrigin, colOrigin;
    int rowFrameSize, colFrameSize, extraRows, extraCols;
    int[][] structAry;
    int[] stlD;
    // constructor
    public Struct(int rows, int cols, int min, int max, int rowOrigin, int colOrigin){
        super(rows, cols, min, max);
        this.rowOrigin = rowOrigin;
        this.colOrigin = colOrigin;
        this.rowFrameSize = this.numRows/2;
        this.colFrameSize = this.numCols/2;
        this.extraRows = this.rowFrameSize * 2;
        this.extraCols = this.colFrameSize * 2;
        this.structAry = new int[this.numRows][this.numCols];
        this.stlD = new int[this.numRows * this.numCols];
    }
}
}

```

* Program Output on next page.

Program Output

Data_1

data1_dilateOutFile

[illegible]

```
data1_closingOutFile
```

[illegible]

data1_erodeOutFile

[illegible]

data1_openingOutFile

[illegible]

```
data1_Structuring Element prettyPrint
```

$$\begin{array}{cccc} 3 & 3 & 0 & 1 \\ 1 & 1 & & \\ . & 1 & . & \\ 1 & 1 & 1 & \\ . & 1 & . & \end{array}$$

```
data1_Closing prettyPrint
```

[illegible]

42 31 0 1

[illegible]

42 31 0 1

[illegible]

Data_2

32 60 0 1

[illegible]

[illegible][illegible]

[illegible][illegible]

```

3 3 1 1
1 1
1 1 1
1 1 1
1 1 1

```

32 60 0 1

```
data2 Erosion prettyPrint
```

32 60 0 1

This image shows a full page of dot grid paper. The background is white, and it is covered with a regular pattern of small, dark grey dots. The dots are arranged in straight horizontal and vertical rows, creating a grid-like appearance. There are no margins, text, or other markings on the page.

32 60 0 1

```
data2_Opening prettyPrint
```

32 60 0 1

This image shows a full page of dot grid paper. The background is white, and it is covered with a regular pattern of small, dark grey dots. The dots are arranged in a precise square grid, with equal spacing between them both horizontally and vertically. There are no margins, text, or other markings on the page.

Program Output

Data 3

data3_dilateOutFile

[illegible]

data3_erodeOutFile

[illegible]

data3_closingOutFile

[illegible]

[illegible][illegible]

```

2 2 1 1
0 0
1 1
1 1

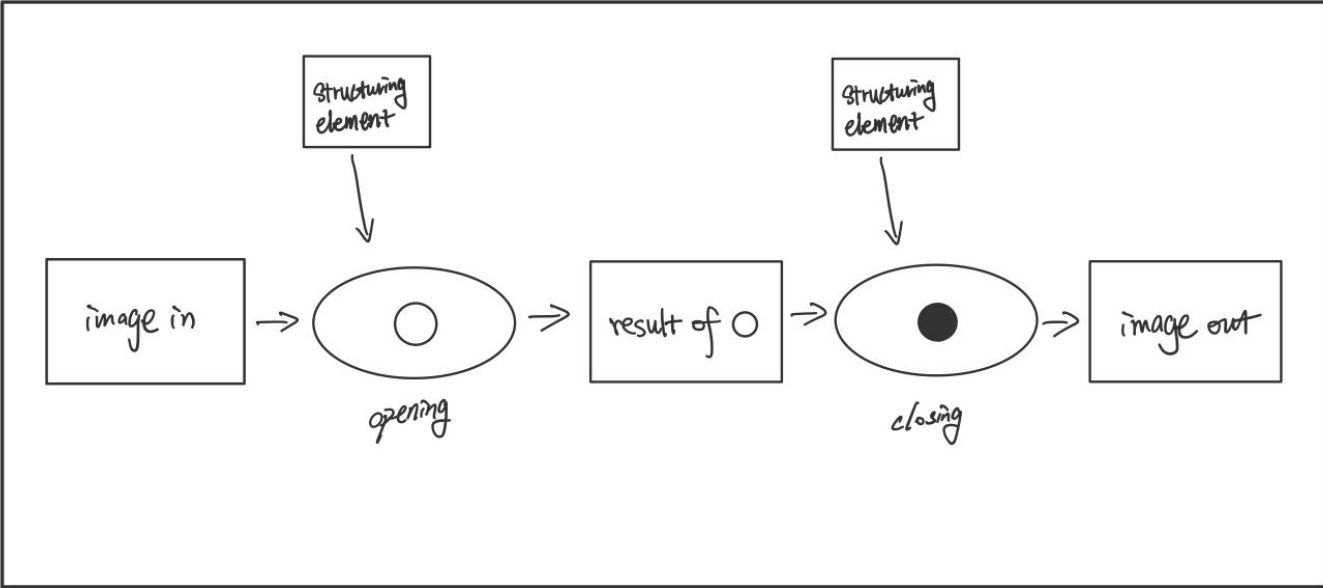
```

[illegible][illegible][illegible]


```
data3_Opening prettyPrint -> this is the result. Extract the cross object and remove background noise.
```

```
25 42 0 1
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
. . . . . 1 1 1 . . . 1 1 1 . . . . . 1 1 1 . . . 1 1 1 . . . . .
```

Object-process-diagram for Image 4



Program Output

Data 4 / 1st Run

```
data4_1_dilateOutFile
```

[illegible]

```
data4_1_closingOutFile
```

[illegible]

data4_1_erodeOutFile

[illegible]

data4_1_openingOutFile -> use this result for the 2nd run.

[illegible]

```
data4_1_Structuring Element prettyPrint
```

3 3 0 1

$$\begin{array}{ccc} 1 & 1 & \\ . & 1 & . \\ 1 & 1 & 1 \\ . & . & . \end{array}$$

```
data4_1_Closing prettyPrint
```

38 31 0 1

[illegible]

38 31 0 1

38 31 0 1

38 31 0 1

38 31 0 1

[illegible]

[illegible][illegible][illegible]

38 31 0 1

[illegible]

Extract the circular object in image 4, and fill in the holes.
38 31 0 1

[illegible]

38 31 0 1

[illegible]

38 31 0 1

[illegible]

- This is the end of the report.