

Student: Pawan Bhatta

Project Due Date: 02/28/2021

Algorithm for Computing Dilation:

Step 1: $i \leftarrow \text{rowFrameSize}$
Step 2: $j \leftarrow \text{colFrameSize}$
Step 3: if $\text{inAry}[i,j] > 0$
 $\text{dilation}(i, j, \text{inAry}, \text{outAry})$ // only processing one pixel inAry[i,j]
Step 4: $j++$
Step 5: repeat step 3 to step 4 while $j < (\text{numImgCols} + \text{colFrameSize})$
Step 6: $i++$
Step 7: repeat step 2 to step 6 while $i < (\text{numImgRows} + \text{rowFrameSize})$

Algorithm for Computing Erosion:

Step 1: $i \leftarrow \text{rowFrameSize}$
Step 2: $j \leftarrow \text{colFrameSize}$
Step 3: if $\text{inAry}[i,j] > 0$
 $\text{erosion}(i, j, \text{inAry}, \text{outAry})$ // only processing one pixel inAry[i,j]
Step 4: $j++$
Step 5: repeat step 3 to step 4 while $j < (\text{numImgCols} + \text{colFrameSize})$
Step 6: $i++$
Step 7: repeat step 2 to step 6 while $i < (\text{numImgRows} + \text{rowFrameSize})$

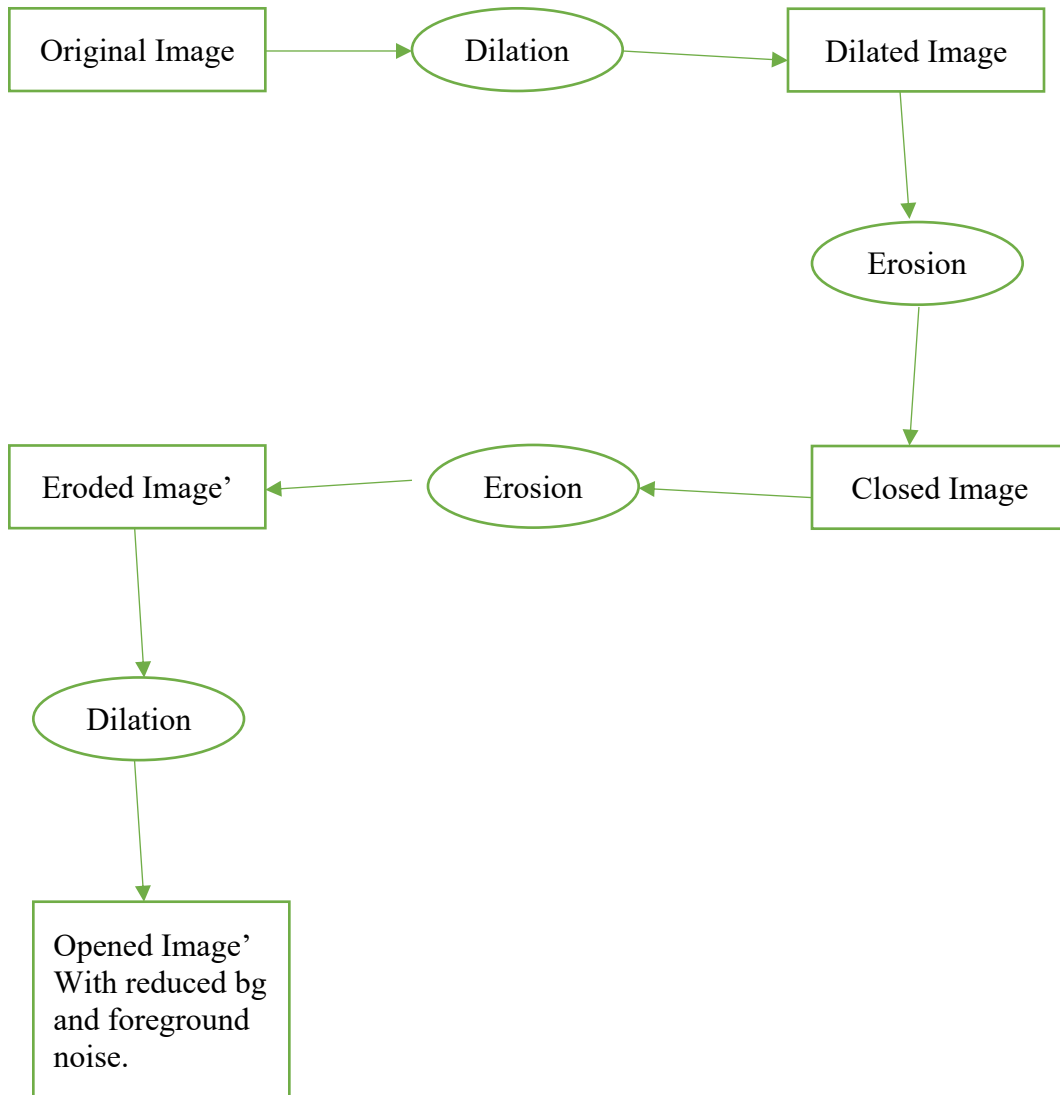
Algorithm for Computing Closing:

Step 1: $\text{ComputeDilation}(\text{zeroFramedAry}, \text{tempAry})$
Step 2: $\text{ComputeErosion}(\text{tempAry}, \text{morphAry})$

Algorithm for Computing Opening:

Step 1: $\text{ComputeErosion}(\text{zeroFramedAry}, \text{tempAry})$
Step 2: $\text{ComputeDilation}(\text{tempAry}, \text{morphAry})$

Object Process Diagram for extracting objects and removing background noise:



Source Code:

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

public class MorphologicalOperations {
    int numImgRows;
    int numImgCols;
    int imgMin;
    int imgMax;
    int newMin;
    int newMax;

    int numStructRows;
    int numStructCols;
    int structMin;
    int structMax;
    int rowOrigin;
    int colOrigin;

    int rowFrameSize;
    int colFrameSize;
    int extraRows;
    int extraCols;

    int [] [] zeroFramedAry;
    int [] [] morphAry;
    int [] [] tempAry;
    int [] [] structAry;
    int [] [] arrayWithoutFrame;

    void setFrameSize(){
        rowFrameSize=numStructRows/2;
        colFrameSize=numStructCols/2;
    }

    void setTotalFrameSize(){
        extraRows =rowFrameSize*2;
        extraCols =colFrameSize*2;
    }

    void allocateArrays(){
        zeroFramedAry=new int [numImgRows+extraRows][numImgCols+extraCols];
        morphAry=new int [numImgRows+extraRows][numImgCols+extraCols];
        tempAry=new int [numImgRows+extraRows][numImgCols+extraCols];
        structAry=new int [numStructRows][numStructCols];
        arrayWithoutFrame=new int [numImgRows][numImgCols];
    }
}
```

```

void zero2DAry(int [][] zeroFramedAry, int numImgRows, int numImgCols){
    for (int i=0;i<numImgRows;i++){
        for(int j=0;j<numImgCols;j++){
            zeroFramedAry[i][j]=0;
        }
    }
}

void loadImg(Scanner imgFile, int [][] zeroFramedAry){
    for (int i=rowFrameSize;i<numImgRows+rowFrameSize;i++) {
        System.out.print("\n");
        for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
            zeroFramedAry[i][j] = imgFile.nextInt();
            System.out.print(zeroFramedAry[i][j]);
        }
    }
    System.out.print("\n");
    System.out.print("\n");
}

void loadStruct(Scanner structFile, int [][] structArray){
    for (int i=0;i<numStructRows;i++){
        for(int j=0;j<numStructCols;j++){
            structArray[i][j]=structFile.nextInt();
        }
    }
}

void dilation(int i, int j, int [][] inAry, int [][] outAry){
    newMin=0;
    newMax=0;
    for(int k=0;k<numStructRows;k++){
        for(int l=0;l<numStructCols;l++){
            if(structAry[k][l]!=0){
                int rowDiff=k-rowOrigin;
                int colDiff=l-colOrigin;
                outAry[i+rowDiff][j+colDiff]=1;
                if(newMax==0){
                    newMax=1;
                }
            }
        }
    }
}

void erosion(int i, int j , int [][] inAry, int [][]outAry){
    newMin=0;
    newMax=0;
    boolean keep=true;
    for(int k=0;k<numStructRows;k++){

```

```

        for(int l=0;l<numStructCols;l++){
            if(structAry[k][l]!=0){
                int rowDiff=k-rowOrigin;
                int colDiff=l-colOrigin;
                if(inAry[i+rowDiff][j+colDiff]!=structAry[k][l]){
                    keep=false;
                };
            }
        }
    }
    if(keep==true){
        outAry[i][j]=inAry[i][j];
        if(newMax==0){
            newMax=1;
        }
    }
    else{
        outAry[i][j]=0;
    }
}

void computeDilation(int [][]inAry, int [][] outAry){
    int i=rowFrameSize;
    while(i<(numImgRows+rowFrameSize)){
        int j=colFrameSize;
        while(j<(numImgCols+colFrameSize)) {
            if (inAry[i][j] > 0) {
                dilation(i, j, inAry, outAry);
            }
            j++;
        }
        i++;
    }
}

void computeErosion(int [][]inAry, int [][] outAry){
    int i=rowFrameSize;
    while(i<(numImgRows+rowFrameSize)){
        int j=colFrameSize;
        while(j<(numImgCols+colFrameSize)) {
            if (inAry[i][j] > 0) {
                erosion(i, j, inAry, outAry);
            }
            j++;
        }
        i++;
    }
}

```

```

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

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

void removeFrame(int [][]arrayWithFrame, int [][]arrayWithoutFrame){
    zero2DAry(arrayWithoutFrame,numImgRows,numImgCols);
    for (int i=rowFrameSize;i<numImgRows+rowFrameSize;i++){
        for(int j=colFrameSize;j<numImgCols+colFrameSize;j++){
            arrayWithoutFrame[i-rowFrameSize][j-colFrameSize]=arrayWithFrame[i][j];
        }
    }
}

void prettyPrint(int [][] ary,BufferedWriter outFile ) throws IOException {
    for (int i=0;i<ary.length;i++){
        for(int j=0;j< ary[0].length;j++){
            if(ary[i][j]==0){
                outFile.write(". ");
            }
            else{
                outFile.write(Integer.toString(ary[i][j])+" ");
            }
        }
        outFile.write("\n");
    }
}

void writeImgHeader(BufferedWriter outFile) throws IOException {
    outFile.write(numImgRows+" "+ numImgCols+" "+imgMin+" "+imgMax+"\n");
}

void aryToFile(int [][] ary, BufferedWriter outFile) throws IOException {
    outFile.write(numImgRows+" "+ numImgCols+" "+imgMin+" "+imgMax+"\n");
    for (int i=rowFrameSize;i<(numImgRows+rowFrameSize);i++){
        for (int j =colFrameSize;j<(numImgCols+colFrameSize);j++){
            outFile.write(Integer.toString(ary[i][j])+" ");
        }
        outFile.write("\n");
    }
}

public static void main(String[] args) throws IOException {
    String inputName1 = args[0];
    FileReader inputReader1 = null;
}

```

```
BufferedReader buffInReader1 = null;
Scanner imgFile = null;

String inputName2 = args[1];
FileReader inputReader2 = null;
BufferedReader buffInReader2 = null;
Scanner structFile = null;

String outputName1 = args[2];
FileWriter outputWriter1 = null;
BufferedWriter dilateOutFile = null;

String outputName2 = args[3];
FileWriter outputWriter2 = null;
BufferedWriter erodeOutFile = null;

String outputName3 = args[4];
FileWriter outputWriter3 = null;
BufferedWriter closingOutFile = null;

String outputName4 = args[5];
FileWriter outputWriter4 = null;
BufferedWriter openingOutFile = null;

String outputName5 = args[6];
FileWriter outputWriter5 = null;
BufferedWriter prettyPrintFile = null;

try {
    inputReader1 = new FileReader(inputName1);
    buffInReader1 = new BufferedReader(inputReader1);
    imgFile = new Scanner(buffInReader1);

    inputReader2 = new FileReader(inputName2);
    buffInReader2 = new BufferedReader(inputReader2);
    structFile = new Scanner(buffInReader2);

    outputWriter1 = new FileWriter(outputName1);
    dilateOutFile = new BufferedWriter(outputWriter1);

    outputWriter2 = new FileWriter(outputName2);
    erodeOutFile = new BufferedWriter(outputWriter2);

    outputWriter3 = new FileWriter(outputName3);
    openingOutFile = new BufferedWriter(outputWriter3);

    outputWriter4 = new FileWriter(outputName4);
    closingOutFile = new BufferedWriter(outputWriter4);
```

```

outputWriter5 = new FileWriter(outputName5);
prettyPrintFile = new BufferedWriter(outputWriter5);

MorphologicalOperations morpOperations = new MorphologicalOperations();
if (imgFile.hasNextInt()) morpOperations.numImgRows = imgFile.nextInt();
if (imgFile.hasNextInt()) morpOperations.numImgCols = imgFile.nextInt();
if (imgFile.hasNextInt()) morpOperations.imgMin = imgFile.nextInt();
if (imgFile.hasNextInt()) morpOperations.imgMax = imgFile.nextInt();

if (structFile.hasNextInt()) morpOperations.numStructRows = structFile.nextInt();
if (structFile.hasNextInt()) morpOperations.numStructCols = structFile.nextInt();
if (structFile.hasNextInt()) morpOperations.structMin = structFile.nextInt();
if (structFile.hasNextInt()) morpOperations.structMax = structFile.nextInt();
if (structFile.hasNextInt()) morpOperations.rowOrigin = structFile.nextInt();
if (structFile.hasNextInt()) morpOperations.colOrigin = structFile.nextInt();

//setting up dynamic class members for morphologicalOperations
morpOperations.setFrameSize();
morpOperations.setTotalFrameSize();
morpOperations.allocateArrays();

morpOperations.zero2DAry(morpOperations.zeroFramedAry, morpOperations.numImgRows+morpOperations.extraRows, morpOperations.numImgCols+ morpOperations.extraCols);

//loading Input Image
morpOperations.loadImg(imgFile, morpOperations.zeroFramedAry);

//pretty printing input image
prettyPrintFile.write("Original Image\n");
prettyPrintFile.write(morpOperations.numImgRows+" "+
morpOperations.numImgCols+" "+morpOperations.imgMin+" "+morpOperations.imgMax+"\n");
morpOperations.prettyPrint(morpOperations.zeroFramedAry, prettyPrintFile);
morpOperations.zero2DAry(morpOperations.structAry, morpOperations.numStructRows,
morpOperations.numStructCols);

//loading structural element
morpOperations.loadStruct(structFile, morpOperations.structAry);

//pretty printing Structural element
prettyPrintFile.write("\nStructuring Element\n");
prettyPrintFile.write(morpOperations.numStructRows+"
"+morpOperations.numStructCols+" "+morpOperations.structMin+"
"+morpOperations.structMax+"\n");
prettyPrintFile.write(morpOperations.rowOrigin+"
"+morpOperations.colOrigin+"\n");
morpOperations.prettyPrint(morpOperations.structAry, prettyPrintFile);

//Dilation

```



```

morpOperations.zero2DAry(morpOperations.morphAry,morpOperations.numImgRows+morpOperations.extraRows,morpOperations.numImgCols+ morpOperations.extraCols );

morpOperations.computeDilation(morpOperations.zeroFramedAry,morpOperations.morphAry);
    dilateOutFile.write("Dilation\n");
    morpOperations.aryToFile(morpOperations.morphAry,dilateOutFile);
    prettyPrintFile.write("\nDilation\n");
    morpOperations.writeImgHeader(prettyPrintFile);

morpOperations.removeFrame(morpOperations.morphAry,morpOperations.arrayWithoutFrame);
    morpOperations.prettyPrint(morpOperations.arrayWithoutFrame, prettyPrintFile);

    //Erosion

morpOperations.zero2DAry(morpOperations.morphAry,morpOperations.numImgRows+morpOperations.extraRows,morpOperations.numImgCols+ morpOperations.extraCols );
    morpOperations.computeErosion(morpOperations.zeroFramedAry,
morpOperations.morphAry);
    erodeOutFile.write("Erosion\n");
    morpOperations.aryToFile(morpOperations.morphAry, erodeOutFile);
    prettyPrintFile.write("\nErosion\n");
    morpOperations.writeImgHeader(prettyPrintFile);

morpOperations.removeFrame(morpOperations.morphAry,morpOperations.arrayWithoutFrame);
    morpOperations.prettyPrint(morpOperations.arrayWithoutFrame, prettyPrintFile);

    //Opening

morpOperations.zero2DAry(morpOperations.morphAry,morpOperations.numImgRows+morpOperations.extraRows,morpOperations.numImgCols+ morpOperations.extraCols );
    morpOperations.computeOpening(morpOperations.zeroFramedAry,
morpOperations.morphAry, morpOperations.tempAry);
    openingOutFile.write("\nOpening\n");
    morpOperations.aryToFile(morpOperations.morphAry, openingOutFile);
    prettyPrintFile.write("\nOpening\n");
    morpOperations.writeImgHeader(prettyPrintFile);

morpOperations.removeFrame(morpOperations.morphAry,morpOperations.arrayWithoutFrame);
    morpOperations.prettyPrint(morpOperations.arrayWithoutFrame,prettyPrintFile );

    //Closing

morpOperations.zero2DAry(morpOperations.morphAry,morpOperations.numImgRows+morpOperations.extraRows,morpOperations.numImgCols+ morpOperations.extraCols );
    morpOperations.computeClosing(morpOperations.zeroFramedAry,
morpOperations.morphAry, morpOperations.tempAry);
    closingOutFile.write("\nClosing\n");
    morpOperations.aryToFile(morpOperations.morphAry, closingOutFile);
    prettyPrintFile.write("\nClosing\n");

```

```
morpOperations.removeFrame(morpOperations.morphAry,morpOperations.arrayWithoutFrame);
    morpOperations.writeImgHeader(prettyPrintFile);
    morpOperations.prettyPrint(morpOperations.arrayWithoutFrame,prettyPrintFile );
} finally {
    if (imgFile != null) imgFile.close();
    if (structFile!=null) structFile.close();
    if (dilateOutFile != null) dilateOutFile.close();
    if (erodeOutFile != null) erodeOutFile.close();
    if (closingOutFile != null) closingOutFile.close();
    if (openingOutFile != null) openingOutFile.close();
    if (prettyPrintFile != null) prettyPrintFile.close();
}
}
```

Outputs

Data_1:

Original Image

42 31 0 1

[illegible]

Structuring Element

3 3 0 1

1 1

. 1 .

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

. 1 .

42 31 0 1

42 31 0 1

[illegible]

42 31 0 1

42 31 0 1

[illegible]

42 31 0 1

42 31 0 1

[illegible]

42 31 0 1

42 31 0 1

[illegible]

Data_2:

```
Original Image
32 60 0 1
```

[illegible]

```
Structuring Element
3 3 1 1
1 1
1 1 1
1 1 1
1 1 1
```


32 60 0 1

[illegible]

32 60 0 1

[illegible]

32 60 0 0

[illegible]

32 60 0 0

[illegible]

32 60 0 0

[illegible]

```

opening
32 60 0 0

```

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. This type of paper is commonly used for sketching, journaling, and organizing notes.

[illegible][illegible]

Data_3:

Original Image

25 42 0 1

[illegible]

Structuring Element

3 3 0 1

$$\begin{array}{cc} 1 & 1 \end{array}$$

1 1 .

1 1 .

• • •

[illegible][illegible]

[illegible][illegible]

[illegible][illegible]

[illegible][illegible]

Data_4:

Original Image

38 31 0 1

[illegible]

Structuring Element

3 3 0 1

1 1

. 1 .

1 1 1

. 1 .

38 31 0 1

[illegible]

38 31 0 1

[illegible]

[illegible][illegible]

[illegible][illegible]

38 31 0 1

Closing

38 31 0 1

[illegible]

Data_4_Reprocessing:

Original Image

38 31 0 1

[illegible]

Structuring Element

3 3 0 1

1 1

. 1 .

1 1 1

. 1 .

[illegible][illegible]

[illegible][illegible]

[illegible][illegible]

