

***** Cover Page *****

Class: CV
Name: Frank Youmet
Project: Project 4
Project Name: Morphology
Language: Java
Due Date: 10/12/2024 before 12:00AM
Submit Date: 10/12/2024 before 4:00PM

Top Level algorithm steps

III. Data structure:

- Morphology class
- (int) numImgRows
- (int) numImgCols
- (int) imgMin
- (int) imgMax
- (int) numStructRows
- (int) numStructCols
- (int) structMin
- (int) structMax
- (int) rowOrigin
- (int) colOrigin
- (int) rowFrameSize // set to (numStructRows / 2), integer division, i.e., 3/2 is 1; 4/2 is 2; 5/2 is 2.
- (int) colFrameSize // set to (numStructCols / 2).
- (int) extraRows // set to (rowFrameSize * 2)
- (int) extraCols // set to (colFrameSize * 2)
- (int) rowSize // set to (numImgRows + extraRows)
- (int) colSize // set to (numImgCols + extraCols)
- (int[][]) zeroFramedAry // a dynamically allocate 2D array, size of rowSize by colSize.
- (int[][]) morphAry // Same size as zeroFramedAry.
- (int[][]) tempAry // Same size as zeroFramedAry.

// tempAry is to be used as the intermediate result within opening and closing operations.

- (int[][]) structAry //a dynamically allocate 2D array of size numStructRows by numStructCols.

Methods:

- constructor (..) // may performs all allocations and initializations.
- zero2DAry (Ary, nRows, nCols) // Set the entire Ary (nRows by nCols) to zero.
- loadImg (...) // load imgFile to zeroFramedAry inside of frame, begins at (rowOrigin, colOrigin). On your own!
- loadstruct (...) // load structFile to structAry. On your own!
- ComputeDilation (inAry, outAry) // process every pixel in inAry, store result in outAry // see algorithm below.
- ComputeErosion (inAry, outAry) // process every pixel in inAry, store result in outAry // see algorithm below.
- ComputeOpening (inAry, outAry, tmp) // see algorithm below.
- ComputeClosing (inAry, outAry, tmp) // see algorithm below.
- onePixelDilation (i, j, inAry, outAry) // Perform dilation on pixel (i, j) with structAry. // See algorithm below.
- onePixelErosion (i, j, inAry, outAry) // Perform erosion on pixel (i, j) with structAry. // See algorithm below.
- AryToFile (inAry, fileOut) //

output the image header (same as input image header)

// output pixels inside of frame of inAry to fileOut

- binaryPrettyPrint (inAry, fileOut) // output all pixels in inAry, including pixels in the frame.

// if inAry [i, j] == 0 output ‘.’ // a period follows by a blank

// else output ‘1’ // 1 follows by a blank

IV. Main(...)

Step 0: inFile, structFile ← open via args [] for reading.

Step 1: numImgRows, numImgCols, imgMin, imgMax ← read from inFile.
numStructRows, numStructCols, structMin, structMax ← read from structFile.
rowOrigin, colOrigin ← read from structFile.

Step 2: zeroFramedAry, structAry, morphAry, tempAry ← dynamically allocate // see description in the above.
initialized all members of the class. // see description in the above.

Step 3: zero2DAry (zeroFramedAry, rowSize, colSize) // see description in the above.

Step 4: loadImg (inFile, zeroFramedAry) // see description in the above.
binaryPrettyPrint (zeroFramedAry, prettyPrintFile) // with caption.

Step 5: zero2DAry (structAry, numStructRows, numStructCols)

```
loadstruct (structFile, structAry)
binaryPrettyPrint (structAry, prettyPrintFile) // with captions.
```

Step 6: choice ← from args [2] // Use Integer.parseInt () method.

Step 7: if choice is 1
 process1 (prettyPrintFile)
if choice is 2
process2 (prettyPrintFile)
if choice is 3
 process3 (prettyPrintFile)
if choice is 4
 process4 (prettyPrintFile)
if choice is 5
 process5 (prettyPrintFile)

Step 8: close all files

V. process1 (prettyPrintFile)

Step 1: fileName ← “dilationOutFile.txt”
 outFile ← open (fileName)

Step 2: zero2DAry (morphAry, rowSize, colSize)
 ComputeDilation (zeroFramedAry, morphAry)
 AryToFile (morphAry, outFile)
 binaryPrettyPrint (morphAry, prettyPrintFile)

Step 3: close outFile

VI.
process2 (prettyPrintFile)

Step 1: fileName ← “erosionOutFile.txt”
 outFile ← open (fileName)

Step 2: zero2DAry (morphAry, rowSize, colSize)
 ComputeErosion (zeroFramedAry, morphAry)
 AryToFile (morphAry, outFile)
 binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.

Step 3: close outFile

VII. process3 (prettyPrintFile)

Step 1: fileName ← “openingOutFile.txt”
 outFile ← open (fileName)

Step 2: zero2DAry (morphAry, rowSize, colSize)
 ComputeOpening (zeroFramedAry, morphAry, tempAry)
 AryToFile (morphAry, outFile)
 binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.

Step 3: close outFile

VIII. process4 (prettyPrintFile)

Step 1: fileName ← “closingOutFile.txt”

```
outFile ← open (fileName)
```

```
Step 2: zero2DAry (morphAry, rowSize, colSize)
        ComputeClosing (zeroFramedAry, morphAry, tempAry)
        AryToFile (morphAry, outFile)
        binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.
```

```
Step 3: close outFile
```

```
*****
```

```
IX.
process5 (prettyPrintFile)
```

```
*****
```

```
Step 1: fileName ← “dilationOutFile.txt”
        outFile ← open (fileName)
        zero2DAry (morphAry, rowSize, colSize)
        ComputeDilation (zeroFramedAry, morphAry)
        AryToFile (morphAry, outFile)
        binaryPrettyPrint (morphAry, prettyPrintFile)
        close (outFile)
```

```
Step 2: fileName ← “erosionOutFile.txt”
        outFile ← open (fileName)
        zero2DAry (morphAry, rowSize, colSize)
        ComputeErosion (zeroFramedAry, morphAry)
        AryToFile (morphAry, outFile)
        binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.
        close (outFile)
```

```
Step 3: fileName ← “openingOutFile.txt”
        outFile ← open (fileName)
        zero2DAry (morphAry, rowSize, colSize)
        ComputeOpening (zeroFramedAry, morphAry, tempAry)
        AryToFile (morphAry, outFile)
        binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.
        close (outFile)
```

```
Step 4: fileName ← “closingOutFile.txt”
        outFile ← open (fileName)
        zero2DAry (morphAry, rowSize, colSize)
        ComputeClosing (zeroFramedAry, morphAry, tempAry)
        AryToFile (morphAry, outFile)
        binaryPrettyPrint (morphAry, prettyPrintFile) //with captions.
        close (outFile)
```

```
*****
```

```
X. ComputeDilation (inAry, outAry) // process dilation on each pixel inside of zeroFramedAry.
```

```
*****
```

```
step 1: i ← rowFrameSize
```

```
step 2: j ← colFrameSize
```

```
step 3: if inAry [i, j] > 0
        onePixelDilation (i, j, inAry, outAry) // only processing one pixel inAry[i,j]
```

```
step 4: j++
```

```
step 5: repeat step 3 to step 4 while j < (colSize)
```

```
step 6: i++
```

```
step 7: repeat step 2 to step 6 while i < (rowSize)
```

```
*****
```

```
XI. ComputeErosion (inAry, outAry) // process erosion on each pixel inside of zeroFramedAry
```

```
*****
```

```
step 1: i ← rowFrameSize
```

```
step 2: j ← colFrameSize
```

```
step 3: if inAry[i, j] > 0
```

```
onePixelErosion (i, j, inAry, outAry) // only processing one pixel inAry[i,j]
```

```
step 4:      j++
```

```
step 5:      repeat step 3 to step 4 while j < (colSize)
```

```
step 6:      i++
```

```
step 7:      repeat step 2 to step 6 while i < (rowSize)
```

```
*****
```

```
XII. onePixelDilation (i, j, inAry, outAry)
```

```
*****
```

```
step 0 :      iOffset ← i - rowOrigin  
              jOffset ← j - colOrigin // translation of image's coordinate (i, j) with respected to the origin of the structuring element
```

```
step 1:      rIndex ← 0
```

```
step 2:      cIndex ← 0
```

```
step 3:      if (structAry[rIndex][cIndex] > 0)  
              outAry[iOffset + rIndex][jOffset + cIndex] ← 1
```

```
step 4:      cIndex ++
```

```
step 5:      repeat step 3 to step 4 while cIndex < numStructCols
```

```
step 6:      rIndex ++
```

```
step 7:      repeat step 2 to step 6 while rIndex < numStructRows
```

```
*****
```

```
XIII. onePixelErosion (i, j, inAry, outAry)
```

```
*****
```

```
step 0 :      iOffset ← i - rowOrigin  
              jOffset ← j - colOrigin // translation of image's coordinate (i, j) with respected to the origin of the structuring element  
              matchFlag ← true
```

```
step 1:      rIndex ← 0
```

```
step 2:      cIndex ← 0
```

```
step 3:      if (structAry[rIndex][cIndex] > 0) and (inAry[iOffset + rIndex][jOffset + cIndex] ) <= 0)  
              matchFlag ← false
```

```
step 4:      cIndex ++
```

```
step 5:      repeat step 3 to step 4 while (matchFlag == true) and (cIndex < numStructCols )
```

```
step 6:      rIndex ++
```

```
step 7:      repeat step 2 to step 6 while (matchFlag == true) and (rIndex < numStructRows)
```

```
step 8:      if matchFlag == true  
              outAry[i][j] ← 1  
      else  
              outAry[i][j] ← 0
```

```
*****
```

```
XIV. ComputeClosing (zeroFramedAry, morphAry, tempAry)
```

```
*****
```

```
step 1: ComputeDilation (zeroFramedAry, tempAry)
```

```
step 2: ComputeErosion (tempAry, morphAry)
```

```
*****
```

```
XV. ComputeOpening (zeroFramedAry, morphAry, tempAry)
```

```
*****
```

```
step 1: Compute Erosion (zeroFramedAry, tempAry)
```

```
step 2: ComputeDilation (tempAry, morphAry)
```

***** Source Code *****

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

class Morphology{
    public int numImgRows;
    public int numImgCols;
    public int imgMin;
    public int imgMax;
    public int numStructRows;
    public int numStructCols;
    public int structMin;
    public int structMax;
    public static int rowOrigin;
    public static int colOrigin;
    public int rowFrameSize;
    public int colFrameSize;
    public int extraRows;
    public int extraCols;
    public int rowSize;
    public int colSize;
    public int[][] zeroFramedAry;
    public int[][] morphAry;
    public int[][] tempAry;
    public int [][] structAry;

    public Morphology(int numImgRows, int numImgCols, int imgMin, int imgMax, int numStructRows, int numStructCols, int structMin, int structMax, int rowOrigin, int colOrigin){
        this.numImgRows = numImgRows;
        this.numImgCols = numImgCols;
        this.imgMin = imgMin;
        this.imgMax = imgMax;
        this.numStructRows = numStructRows;
        this.numStructCols = numStructCols;
        this.structMin = structMin;
        this.structMax = structMax;
        this.rowOrigin = rowOrigin;
        this.colOrigin = colOrigin;

        this.rowFrameSize = numStructRows / 2;
        this.colFrameSize = numStructCols / 2;

        this.extraRows = rowFrameSize * 2;
        this.extraCols = colFrameSize * 2;

        this.rowSize = numImgRows + extraRows;
        this.colSize = numImgCols + extraCols;

        this.zeroFramedAry = new int[rowSize][colSize];
        this.morphAry = new int[rowSize][colSize];
        this.tempAry = new int[rowSize][colSize];

        this.structAry = new int[numStructRows][numStructCols];

        zero2DAry(zeroFramedAry, rowSize, colSize);
        zero2DAry(morphAry, rowSize, colSize);
        zero2DAry(tempAry, rowSize, colSize);
        zero2DAry(structAry, numStructRows, numStructCols);
    }

    public void zero2DAry(int[][]Ary, int nRows, int nCols){
        for (int i = 0; i < nRows; i++){
            for (int j = 0; j < nCols; j++){
                Ary[i][j] = 0;
            }
        }
    }
}
```

```

    }
}

public void loadImg(BufferedReader inFileReader, int[][] zeroFramedAry) throws IOException {
    for (int i = rowOrigin; i < numImgRows + rowOrigin; i++) {
        String currentLine = inFileReader.readLine();
        StringTokenizer currentLineTokenizer = new StringTokenizer(currentLine);
        for (int j = colOrigin; j < numImgCols + colOrigin; j++) {
            if (currentLineTokenizer.hasMoreTokens()) {
                zeroFramedAry[i][j] = Integer.parseInt(currentLineTokenizer.nextToken());
            }
        }
    }
}
}
}

```

```

public void binaryPrettyPrint(int[][] inAry, BufferedWriter prettyPrintFile ) throws IOException {
    prettyPrintFile.write(inAry.length + " " + inAry[0].length + " 0" + " 1" + "\n");
    for(int i = 0; i < inAry.length;i++){
        for(int j = 0; j < inAry[i].length; j++){
            if(inAry[i][j] == 0) prettyPrintFile.write(" ");
            else prettyPrintFile.write("1 ");
        }
        prettyPrintFile.write("\n");
    }
}
}

```

```

public void loadStruct(BufferedReader structFile, int[][] structAry) throws IOException {
    for (int i = 0; i < structAry.length; i++){
        StringTokenizer structTokenizer = new StringTokenizer(structFile.readLine());
        for (int j = 0; j < structAry[i].length; j++){
            structAry[i][j] = Integer.parseInt(structTokenizer.nextToken());
        }
    }
}
}

```

```

public void process1(BufferedWriter prettyPrintFile) throws IOException {
    String filename = "dilationOutFile.txt";
    BufferedWriter outfile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeDilation(zeroFramedAry, morphAry);
    aryToFile(morphAry, outfile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outfile.close();
}

```

```

public void computeDilation(int[][] inAry, int[][] outAry){
    int i = rowFrameSize;

    while(i < rowSize){
        int j = colFrameSize;
        while (j < colSize){
            if(inAry[i][j] > 0){
                onePixelDilation(i, j, inAry, outAry);
            }
            j++;
        }
        i++;
    }
}
}

```

```

public void onePixelDilation(int i, int j, int[][] inAry, int[][] outAry){
    int iOffset = i - rowOrigin;
    int jOffset = j - colOrigin;

    int rIndex = 0;

```

```

while (rIndex < numStructRows){
    int cIndex = 0;
    while(cIndex < numStructCols){
        if(structArray[rIndex][cIndex] > 0){
            outAry[iOffset + rIndex][jOffset + cIndex] = 1;
        }
        cIndex++;
    }
    rIndex++;
}
}

public void aryToFile(int[][] inAry, BufferedWriter outFile) throws IOException {
    outFile.write( numImgRows + " " + numImgCols + " " + imgMin + " " + imgMax + "\n");
    for(int i = 0; i < inAry.length; i++){
        for (int j = 0; j < inAry[i].length; j++){
            outFile.write(inAry[i][j] + " ");
        }
        outFile.write("\n");
    }
}

public void process2(BufferedWriter prettyPrintFile) throws IOException {
    String filename = "erosionOutFile.txt";
    BufferedWriter outfile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeErosion(zeroFramedAry, morphAry);
    aryToFile(morphAry, outfile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outfile.close();
}

public void computeErosion(int[][] inAry, int[][] outAry){
    int i = rowFrameSize;
    while (i < rowSize){
        int j = colFrameSize;
        while(j < colSize){
            if(inAry[i][j] > 0){
                onePixelErosion(i, j, inAry, outAry);
            }
            j++;
        }
        i++;
    }
}

public void onePixelErosion(int i, int j, int[][] inAry, int[][] outAry){
    int iOffset = i - rowOrigin;
    int jOffset = j - colOrigin;

    boolean matchFlag = true;

    int rIndex = 0;

    while(matchFlag && (rIndex < numStructRows)){
        int cIndex = 0;

        while (matchFlag && (cIndex < numStructCols)){
            if((structArray[rIndex][cIndex] > 0) && (inAry[iOffset+rIndex][jOffset+cIndex] <= 0)){
                matchFlag = false;
            }
            cIndex++;
        }
        rIndex++;
    }

    if (matchFlag){
        outAry[i][j] = 1;
    }
}

```

```

    }
    else {
        outAry[i][j] = 0;
    }
}

```

```

public void process3(BufferedWriter prettyPrintFile) throws IOException {
    String filename = "openingOutFile.txt";
    BufferedWriter outfile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeOpening(zeroFramedAry, morphAry, tempAry);
    aryToFile(morphAry, outfile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outfile.close();
}

```

```

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

```

```

public void process4(BufferedWriter prettyPrintFile) throws IOException {
    String filename = "closingOutFile.txt";
    BufferedWriter outFile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeClosing(zeroFramedAry, morphAry, tempAry);
    aryToFile(morphAry, outFile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outFile.close();
}

```

```

public void process5(BufferedWriter prettyPrintFile) throws IOException {
    String filename = "dilationOutFile.txt";
    BufferedWriter outFile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeDilation(zeroFramedAry, morphAry);
    aryToFile(morphAry, outFile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outFile.close();

    filename = "erosionOutFile.txt";
    outFile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeErosion(zeroFramedAry, morphAry);
    aryToFile(morphAry, outFile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outFile.close();

    filename = "openingOutFile.txt";
    outFile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
    computeOpening(zeroFramedAry, morphAry, tempAry);
    aryToFile(morphAry, outFile);
    binaryPrettyPrint(morphAry, prettyPrintFile);

    outFile.close();

    filename = "closingOutFile.txt";
    outFile = new BufferedWriter(new FileWriter(filename));

    zero2DAry(morphAry, rowSize, colSize);
}

```



```

computeClosing(zeroFramedAry, morphAry, tempAry);
aryToFile(morphAry, outFile);
binaryPrettyPrint(morphAry, prettyPrintFile);

outFile.close();
}

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

}

public class YournetF_Project4_Main {
    public static void main(String[] args) throws IOException {

        //Checks to see if the inFile can be read.
        BufferedReader inFileReader = null;
        try{
            inFileReader = new BufferedReader(new FileReader(args[0]));
        } catch (FileNotFoundException e) {
            System.out.println("Unable to open file " + args[0] + "");
        }

        //Checks to see if the structFile can be read.
        BufferedReader structFileReader = null;
        try{
            structFileReader = new BufferedReader(new FileReader(args[1]));
        } catch (FileNotFoundException e) {
            System.out.println("Unable to open file " + args[1] + "");
        }

        //Checks to see if the prettyPrintFile can be opened.
        BufferedWriter prettyPrintFile = null;
        try{
            prettyPrintFile = new BufferedWriter(new FileWriter(args[3]));
        } catch (FileNotFoundException e) {
            System.out.println("Unable to open file " + args[3] + "");
        }

        //Attempts to read the header of the inFile.
        String inFileHeader = null;
        try {
            assert inFileReader != null;
            inFileHeader = inFileReader.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        //Checks the header and assigns the proper values to the Morphology class.
        StringTokenizer inFileTokenizer = new StringTokenizer(inFileHeader);
        int numImgRows = Integer.parseInt(inFileTokenizer.nextToken());
        int numImgCols = Integer.parseInt(inFileTokenizer.nextToken());
        int imgMin = Integer.parseInt(inFileTokenizer.nextToken());
        int imgMax = Integer.parseInt(inFileTokenizer.nextToken());

        //Attempts to read the header of the structFile.
        String structHeader = null;
        try {
            assert structFileReader != null;
            structHeader = structFileReader.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }

        StringTokenizer structTokenizer = new StringTokenizer(structHeader);
        int numStructRows = Integer.parseInt(structTokenizer.nextToken());

```

```

int numStructCols = Integer.parseInt(structTokenizer.nextToken());
int structMin = Integer.parseInt(structTokenizer.nextToken());
int structMax = Integer.parseInt(structTokenizer.nextToken());

String structOriginsLine = null;
try{
    assert structFileReader != null;
    structOriginsLine = structFileReader.readLine();
} catch (IOException e) {
    throw new RuntimeException(e);
}

structTokenizer = new StringTokenizer(structOriginsLine);
int rowOrigin = Integer.parseInt(structTokenizer.nextToken());
int colOrigin = Integer.parseInt(structTokenizer.nextToken());

//Creates an instance of morphology and initializes with the proper constructor values.
Morphology morphology = new Morphology(numImgRows, numImgCols, imgMin, imgMax, numStructRows, numStructCols, structMin, structMax, rowOrigin,
colOrigin);

morphology.zero2DAry(morphology.zeroFramedAry, morphology.rowSize, morphology.colSize);

morphology.loadImg(inFileReader, morphology.zeroFramedAry);
assert prettyPrintFile != null;
morphology.binaryPrettyPrint(morphology.zeroFramedAry, prettyPrintFile);

morphology.zero2DAry(morphology.structAry, morphology.numStructRows, morphology.numStructCols);
morphology.loadStruct(structFileReader, morphology.structAry);
morphology.binaryPrettyPrint(morphology.structAry, prettyPrintFile);

int choice = Integer.parseInt(args[2]);

switch (choice){
    case 1:
        morphology.process1(prettyPrintFile);
        break;
    case 2:
        morphology.process2(prettyPrintFile);
        break;
    case 3:
        morphology.process3(prettyPrintFile);
        break;
    case 4:
        morphology.process4(prettyPrintFile);
        break;
    case 5:
        morphology.process5(prettyPrintFile);
        break;
}

inFileReader.close();
structFileReader.close();
prettyPrintFile.close();
}
}

```


[illegible][illegible]

[illegible]

[illegible]

[illegible]

openingOutFile:

[illegible]

[illegible]

[illegible]

closingOutFile and prettyPrintFile from 3rd run

[illegible]

[illegible]

[illegible]

closingOutFile and prettyPrintFile from 4th run

[illegible][illegible]

[illegible][illegible]

