

Student: Simon Kong

Project Due date: 9/24/2021

Algorithm Steps:

Main(...)

step 0: imgFile, structFile, dilateOutFile, erodeOutFile, openingOutFile, closingOutFile, prettyPrintFile \leftarrow open

step 1: numImgRows, numImgCols, imgMin, imgMax \leftarrow read from imgFile

numStructRows, numStructCols, structMin, structMax \leftarrow read from structFile

rowOrigin, colOrigin \square read from strucFile

step 2: zeroFramedAry, structAry, morphAry, tempAry \leftarrow dynamically allocate // see description in the above

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

step 4: loadImg (imgFile, zeroFramedAry) // see description in the above

prettyPrint (zeroFramedAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 5: zero2DAry(structAry, numStructRows, numStructCols)

loadstruct (structFile, structAry) // see description in the above

prettyPrint (structAry, prettyPrintFile) // see description in the above

step 6: zero2DAry(morphAry, rowSize, colSize)

ComputeDilation (zeroFramedAry, morphAry) // see algorithm below

AryToFile (morphAry, dilateOutFile) // see description in the above

prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 7: zero2DAry(morphAry, rowSize, colSize)

ComputeErosion (zeroFramedAry, morphAry) // see algorithm below

AryToFile (morphAry, erodeOutFile)

prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint

step 8: zero2DAry(morphAry, rowSize, colSize)

ComputeOpening (zeroFramedAry, morphAry, tempAry) // see algorithm below

```
AryToFile (morphAry, openingOutFile)
prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint
step 9: zero2DAry(morphAry, rowSize, colSize)
ComputeClosing (zeroFramedAry, morphAry, tempAry) // see algorithm below
AryToFile (morphAry, closingOutFile)
prettyPrint (morphAry, prettyPrintFile) // write a meaningful caption before prettyPrint
step 10: close all files
```

ComputeDilation (inAry, outAry)

```
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)
```

ComputeErosion(inAry, outAry)

```
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)
```

onePixelErosion(i, j, inAry, outAry)

step 0 : iOffset \leftarrow i - rowOrigin

jOffset \leftarrow j - colOrigin

// translation of image's coordinate (i, j) with respected of the origin of the structuring element

matchFlag \leftarrow true

step 1: rIndex \leftarrow 0

step 2: cIndex \leftarrow 0

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

matchFlag \leftarrow 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] \leftarrow 1

else

outAry[i][j] \leftarrow 0

ComputeClosing(inAry, outAry, tempAry)

step 1: ComputeDilation (zeroFramedAry, tempAry)

step 2: ComputeErosion (tempAry, morphAry)

ComputeOpening(inAry, outAry, tempAry)

step 1: Compute Erosion (zeroFramedAry, tempAry)

step 2: ComputeDilation (tempAry, morphAry)

onePixelDilation(inAry, outAry)

- 1) Offset x and y by -1
- 2) Replace value at offset position with structure element
- 3) Increment x
- 4) Repeat step 2 until end of row
- 5) Increment y
- 6) Repeat until end of columns

Source Code

Main Class

```
int main( int argc, const char * argv[] ) {  
  
    if(argc != 8){  
        cout << "Invalid amount of arguments";  
        exit(1);  
    }  
  
    string imgFile = argv[ 1 ] ;  
    string structFile = argv[ 2 ];  
    ifstream imgStream, structStream ;  
  
    imgStream.open( imgFile ) ;  
    structStream.open( structFile );  
  
    string dilateOutFile = argv[ 3 ] ;  
    string erodeOutFile = argv[ 4 ];  
    string closingOutFile = argv[ 5 ];  
    string openingOutFile = argv[ 6 ];  
    string prettyPrintFile = argv[ 7 ];  
  
    ofstream dilateStream, erodeStream, closingStream, openingStream, prettyPrintStream ;  
    dilateStream.open( dilateOutFile ) ;  
    erodeStream.open( erodeOutFile );  
    closingStream.open( closingOutFile );  
    openingStream.open( openingOutFile );  
    prettyPrintStream.open( prettyPrintFile );  
}
```

```
if( imgStream.is_open() && structStream.is_open() ){  
    // step 1 & 2  
    Morphology* morphObj = new Morphology( imgStream, structStream );  
  
    // step 3  
    morphObj -> zero2DAry(morphObj -> zeroFramedAry, morphObj -> rowSize, morphObj -> colSize );  
  
    // step 4  
    morphObj -> loadImg( imgStream, morphObj -> zeroFramedAry );  
    morphObj -> prettyPrint( morphObj -> zeroFramedAry, prettyPrintStream );  
  
    // step 5  
    morphObj -> zero2DAry( morphObj -> structAry, morphObj -> numStructRows, morphObj -> numStructCols );  
    morphObj -> loadStruct( structStream, morphObj -> structAry );  
    morphObj -> prettyPrint( morphObj -> structAry, prettyPrintStream );  
  
    // step 6  
    morphObj -> zero2DAry( morphObj -> morphAry, morphObj -> rowSize, morphObj -> colSize );  
    morphObj -> computeDilation( morphObj -> zeroFramedAry, morphObj -> morphAry );  
    morphObj -> aryToFile( morphObj -> morphAry, dilateStream );  
    morphObj -> prettyPrint( morphObj -> morphAry, prettyPrintStream );  
  
    // step 7  
    morphObj -> zero2DAry( morphObj -> morphAry, morphObj -> rowSize, morphObj -> colSize );  
    morphObj -> computeErosion( morphObj -> zeroFramedAry, morphObj -> morphAry );  
    morphObj -> aryToFile( morphObj -> morphAry, erodeStream );  
    morphObj -> prettyPrint( morphObj -> morphAry, prettyPrintStream );  
  
    // step 9  
    morphObj -> zero2DAry( morphObj -> morphAry, morphObj -> rowSize, morphObj -> colSize );  
    morphObj -> computeClosing( morphObj -> zeroFramedAry, morphObj -> morphAry, morphObj -> tempAry );  
    morphObj -> aryToFile( morphObj -> morphAry, openingStream );  
    morphObj -> prettyPrint( morphObj -> morphAry, prettyPrintStream );  
  
    // step 10  
    imgStream.close() ;  
    structStream.close();  
    dilateStream.close() ;  
    erodeStream.close();  
    closingStream.close();  
    openingStream.close();  
    delete morphObj ;  
}  
else{  
    cout << "Error: Input" << endl ;  
}  
  
return 0;  
}
```

Morphology Class

```
class Morphology{
public:
    int numImgRows,
        numImgCols,
        imgMin,
        imgMax,
        numStructRows,
        numStructCols,
        structMin,
        structMax,
        rowOrigin,
        colOrigin,
        rowFrameSize,
        colFrameSize,
        extraRows,
        extraCols,
        rowSize,
        colSize,
        msgCounter;

    int **zeroFramedAry, **morphAry, **tempAry, **structAry;
```

Constructor

```
public:
    Morphology( ifstream &imgFile, ifstream &structFile ){
        read_img_header( imgFile );
        read_struct_header( structFile );
        read_origin( structFile );

        this -> rowFrameSize = this -> numStructRows / 2;
        this -> colFrameSize = this -> numStructCols / 2;

        this -> extraRows = this -> rowFrameSize * 2;
        this -> extraCols = this -> colFrameSize * 2;
        this -> rowSize = this -> numImgRows + this -> extraRows;
        this -> colSize = this -> numImgCols + this -> extraCols;

        this -> zeroFramedAry = new int*[ rowSize ];
        this -> morphAry = new int*[ rowSize ];
        this -> tempAry = new int*[ rowSize ];

        for(int i = 0; i < this -> rowSize; i++){
            zeroFramedAry[i] = new int[ this -> colSize ];
            morphAry[i] = new int[ this -> colSize ];
            tempAry[i] = new int[ this -> colSize ];
        }

        this -> structAry = new int*[ this -> numStructRows ];

        for(int i = 0; i < this -> numStructRows; i++){
            structAry[i] = new int[ this -> numStructCols ];
        }

        this -> msgCounter = 0;
    }
```

Read Headers and origin

```
void read_img_header( ifstream &inFile ){
    inFile >> this -> numImgRows >> this -> numImgCols >> this -> imgMin >> this -> imgMax ;
}

void read_struct_header( ifstream &inFile ){
    inFile >> this -> numStructRows >> this -> numStructCols >> this -> structMin >> this -> structMax ;
}

void read_origin( ifstream &inFile ){
    inFile >> this -> rowOrigin >> this -> colOrigin;
}
```


Zero2DAry

```
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;
        }
    }
}
```

loadImg

```
void loadImg( ifstream &inFile, int **ary){
    for( int i = this -> rowOrigin; i < this -> numImgRows; i++){
        for( int j = this -> colOrigin; j < this -> numImgCols; j++){
            inFile >> ary[i][j];
        }
    }
}
```

loadStruct

```
void loadStruct( ifstream &inFile, int **ary){
    for( int i = 0; i < this -> numStructRows; i++ ){
        for( int j = 0; j < this -> numStructCols; j++){
            inFile >> ary[i][j];
        }
    }
}
```

computeDilation

```
void computeDilation( int **inAry, int **outAry){
    for( int i = this -> rowFrameSize; i < this -> rowSize; i++ ){
        for(int j = this -> colFrameSize; j < this -> colSize; j++ ){
            if( inAry[i][j] > 0 ) this -> onePixelDilation(i,j, inAry, outAry);
        }
    }
}
```

computeErosion

```
void computeErosion( int **inAry, int **outAry ){
    for( int i = this -> rowFrameSize; i < this -> rowSize; i++ ){
        for(int j = this -> colFrameSize; j < this -> colSize; j++){
            if( inAry[i][j] > 0 ) this -> onePixelErosion(i,j, inAry, outAry);
        }
    }
}
```

computeOpening

```
void computeOpening( int **inAry, int **outAry, int **tmp){
    this -> computeErosion(inAry, tmp);
    this -> computeDilation(tmp, outAry);
}
```

computeClosing

```
void computeClosing( int **inAry, int **outAry, int **tmp){
    this -> computeDilation(inAry, tmp);
    this -> computeErosion(tmp, outAry);
}
```

onePixelDilation

```
void onePixelDilation(int i, int j, int **inAry, int **outAry){
    int structRow = 0;
    int structCol = 0;

    for( int r = i - 1; r <= i + 1; r++ ){
        for( int c = j - 1; c <= j + 1; c++ ){
            if(this -> structAry[ structRow ][ structCol ] == 1){
                outAry[r][c] = this -> structAry[ structRow ][ structCol ];
            }
            structCol++;
        }
        structCol = 0;
        structRow++;
    }
}
```

onePixelErosion

```
void onePixelDilation(int i, int j, int **inAry, int **outAry){
    int structRow = 0;
    int structCol = 0;

    for( int r = i - 1; r <= i + 1; r++ ){
        for( int c = j - 1; c <= j + 1; c++ ){
            outAry[r][c] = this -> structAry[ structRow ][ structCol ];
            structCol++;
        }
        structCol = 0;
        structRow++;
    }
}
```

aryToFile

```
void aryToFile(int **ary, ofstream &outFile ){
    outFile << this -> numImgRows << " " << this -> numImgCols << " " << this -> imgMin << " " << this -> imgMax << endl;
    for( int i = this -> rowFrameSize; i < this -> rowSize; i++ ){
        for( int j = this -> colFrameSize; j < this -> colSize; j++ ){
            outFile << ary[i][j] << " ";
        }
        outFile << endl;
    }
}
```

prettyPrint

```
void prettyPrint( int **ary, ofstream &outFile){
    if(ary == this -> structAry){
        outFile << "Structure Element" << endl;
        for( int i = 0; i < this -> numStructRows; i++){
            for( int j = 0; j < this -> numStructCols; j++){
                if(ary[i][j] == 0) outFile << ". ";
                else outFile << ary[i][j] << " ";
            }
            outFile << endl;
        }
    }else{
        string msg[5] = {"Zero Framed", "Dilation", "Erosion", "Opening", "Closing"};
        outFile << msg[this -> msgCounter] << endl;
        this -> msgCounter++;

        for( int i = 0; i < this -> rowSize; i++){
            for( int j = 0; j < this -> colSize; j++){
                if(ary[i][j] == 0) outFile << ". ";
                else outFile << ary[i][j] << " ";
            }
            outFile << endl;
        }
    }
}
```

Test 1

dilateOutFile

[illegible]

erodeOutFile

[illegible]

openingOutFile

[illegible]

closingOutFile[illegible]

[illegible]

Opening

A 20x20 grid of dots with some dots highlighted in black. The pattern is symmetric about the main diagonal. The highlighted dots form a sparse, fractal-like structure. The main diagonal is entirely highlighted. There are several small clusters of dots off the diagonal, including a small 2x2 square in the top-left corner and a small 2x2 square in the bottom-right corner. The pattern is more complex than a simple diagonal line, with some dots appearing at regular intervals along the diagonal and in specific off-diagonal positions.

Erosion

Closing

[illegible]

Test 2

closingOutfile

[illegible]

dilateOutFile

[illegible]

erodeOutFile

[illegible]

openingOutFile

[illegible]

prettyPrintFile

[illegible]

Structure Element

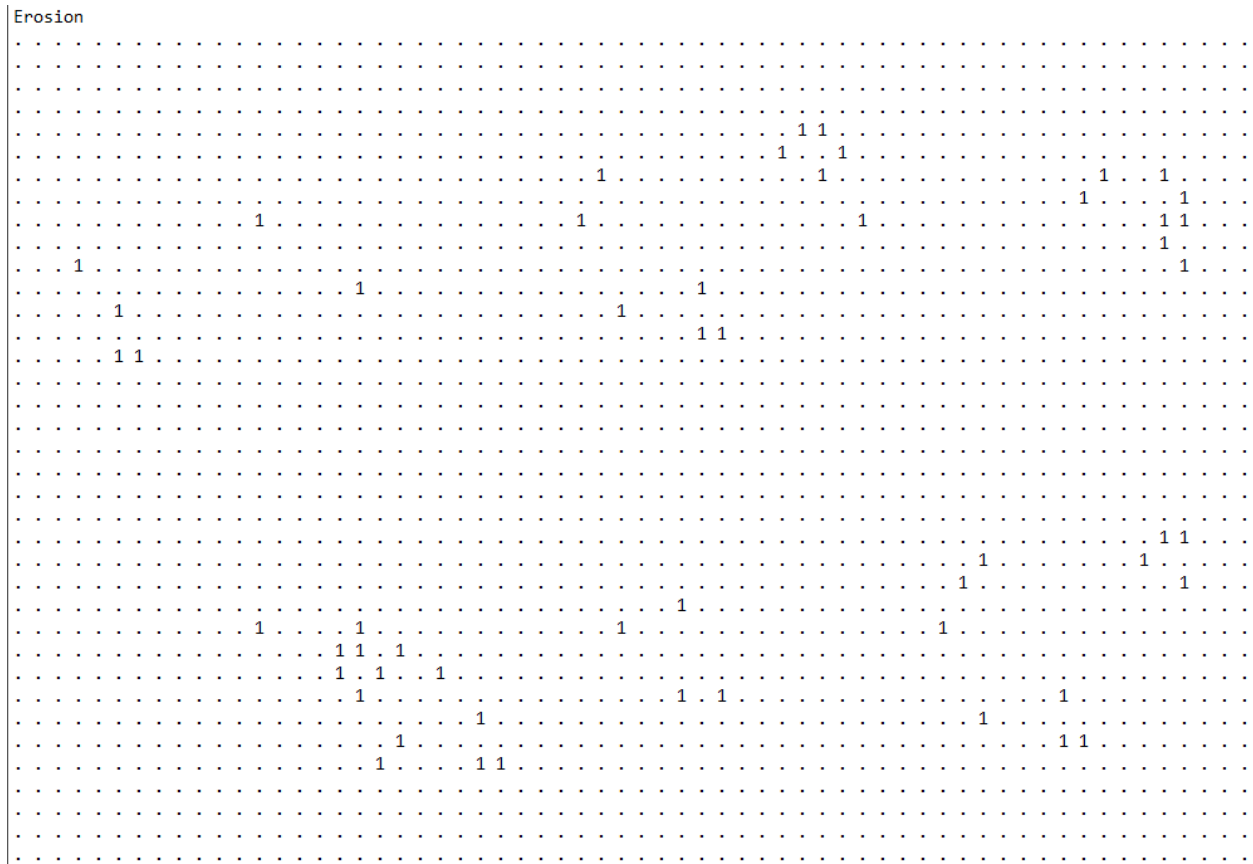
. . 1

. 1 .

1 . .

Dilation

[illegible]



Opening

A 20x20 grid of dots with some dots highlighted in black to form a sparse pattern. The pattern is symmetric about the main diagonal and includes several small clusters of dots.

Closing

[illegible]

Test 3

closingOutFile[illegible]

dilateOutFile

[illegible]

erodeOutFile

[illegible]

openingOutFile

[illegible]

prettyPrintFile

Zero Framed

[illegible]

Structure Element

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

Dilation

[illegible]

Erosion

Opening

[illegible]

Closing

[illegible]

Test 4

closingOutFile

[illegible]

dilateOutFile

[illegible]

erodeOutFile

[illegible]

openingOutFile

[illegible]

prettyPrintfile

Zero Framed

```
.....
.....
.....1.....1.....11.....
...111...111.....1...11...11111...11.....1.....1.....
.....1...11111.....1.....1.....1.....1.....1.....1.....
.....11111...11...11.....111...1...1.....11.....11.....1.....
.....1...11...11111...11.....11111...1.....1...1...1.....11.....1.....
...1.....11111111.....11111111...1.....1...1...1.....11.....1.....
.1.....1...111111.....11111111.....111111.....111111.....111111.....
.11...1.....1...111.....11111111...1.....11111111.....111111.....
.11...1.....11111111.....11111111.....11111111.....111111.....
.111.....1...1...1.....1...1...1...1...1...11.....111111.....1111.....
.1111.....111.....1...1...1...1...1...1...1...1...11.....1111.....
.11111.....1.....1.....1.....111.....1.....1.....1.....1.....1.....
...111.....1.....1.....1.....1.....1.....111.....111.....1.....1.....
.....1.....1.....1.....1.....1.....111.....111.....111.....111.....
.1.....1.....1.....1.....1.....111.....111.....111.....111.....
.....1.....1.....1.....1.....1.....111111.....11111111.....
...1.....1.....1.....1.....1.....111111.....11111111.....
.1.....1.....1.....1.....1.....111.....111.....111.....111.....
.1.....1.....111111.....111.....1.....1.....1.....1.....1.....1.....
.1.....1.....11111111.....11111111.....111111.....11111111.....1111.....
.11.....11111111.....11111111.....111111.....11111111.....1111.....
.11111.....1.....11111111.....11111111.....11111111.....11111111.....
.11111111.....1.....11111111.....11111111.....11111111.....11111111.....
.11111111.....11111111.....11111111.....11111111.....11111111.....
.11111111.....11111111.....11111111.....11111111.....11111111.....
.....111111.....1.....1.....1111.....1.....111111.....1111.....1.....
.....111.....1.....1.....1111.....1.....111111.....1111.....1.....
.....111111.....1.....111111.....111111.....111111.....111111.....
.....
.....
.....
.....
.....
```

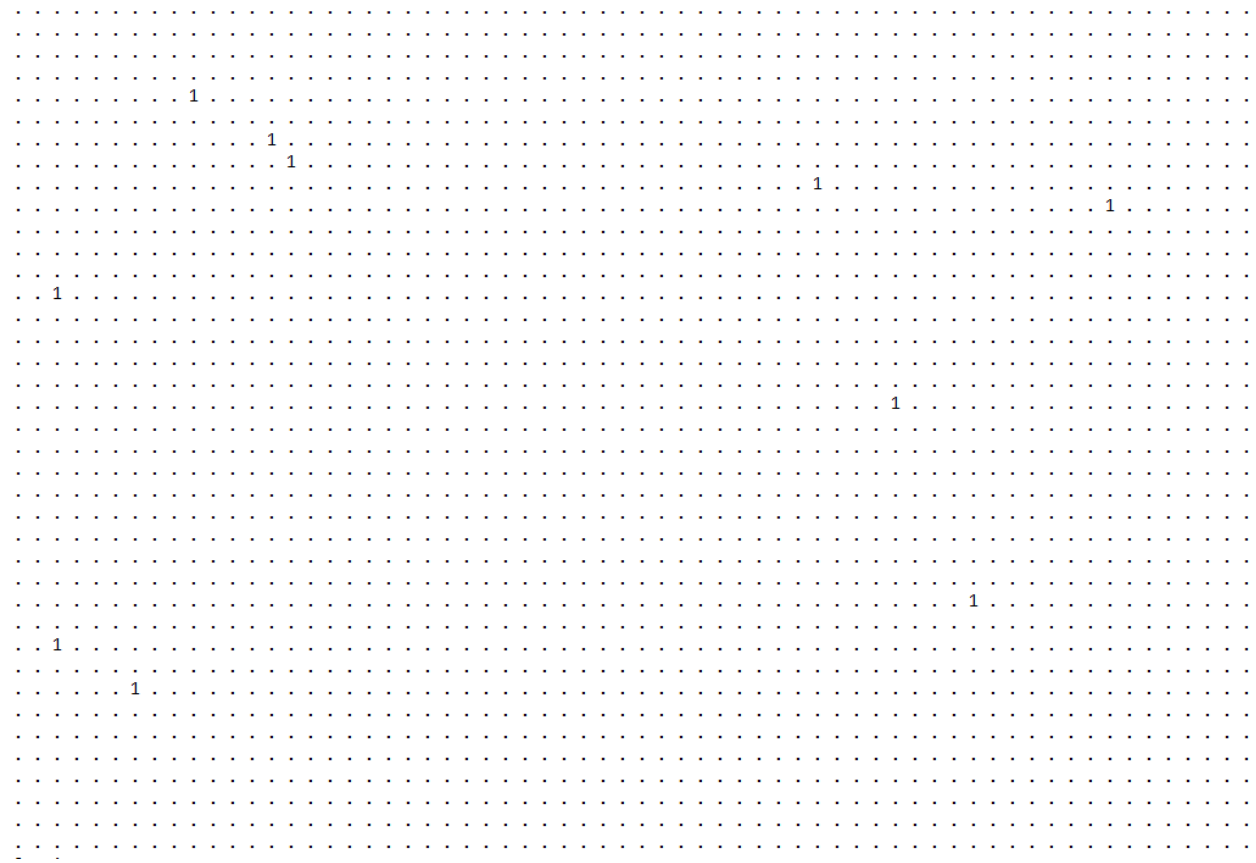
Structure Element

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

Dilation

[illegible]

Erosion



Opening

[illegible]

Closing

[illegible]