CSCI 381 – Computer Vision (JAVA)
Program: Project 4.1: Morphology
Name: Isaac Gordon

Due Date:
Soft copy: 3/12/2019 Tuesday before midnight
Hard copy: 3/14/2019 Thursday in class

```
step -1: make sure all inputted arguments are valid

step 0:
  • Step 0.1:
      o imageInFile               ← arg[0]
      o structElementInFile       ← arg[1]
      o dilationResultOutFile     ← arg[2]
      o erosionResultOutFile      ← arg[3]
      o openingResultOutFile      ← arg[4]
      o closingResultOutFile      ← arg[5]
  • Step 0.2:
      o Load image from imageInFile
      o Load structuring element from structElementInFile

Step 1: Frame image

Step 2: Print Loaded Image and Structuring Element
  • Step 2.1: console ← framedImage
  • Step 2.2: console ← structElement

Step 3: Dilation
  • Step 3.1: Dilate the image
  • Step 3.2: console ← dilatedImage
  • Step 3.3: Unframe the dilatedImgae
  • Step 3.4: dilationResultOutfile  ← unframed dilatedImage

Step 4: Erosion
  • Step 4.1: Erode the image
  • Step 4.2: console ← erodedImage
  • Step 4.3: Unframe the erodedImgae
  • Step 4.4: erosionResultOutfile  ← unframed erodedImage

Step 5: Openeing
  • Step 5.1: Perform opening the image
  • Step 5.2: console ← openedImage
  • Step 5.3: Unframe the openedImgae
```

- Step 5.4: openingResultOutfile ← unframed openedImage

Step 6: Closing

- Step 6.1: Close the image
- Step 6.2: console ← closedImage
- Step 6.3: Unframe the closedImgae
- Step 6.4: closingResultOutfile ← unframed closedImage

Step ~7: Close all open streams (occurs on the method level everytime prettyPring is called.)

Code: MorphologyMain.java

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


public class MorphologyMain {

    public static void main(String args[]){
        File imageInFile, structElementInFile;
        File dilationResultOutFile, erosionResultOutFile;
        File openingResultOutFile, closingResultOutFile;

        final String ARG_ERROR_STRING = "Improper arguements. Correct syntax
is: \n>> ... <input1.txt> <input2.txt> <output1.txt> <output2.txt>
<output3.txt> <output4.txt>"
                        + " \n\twhere: \n\tinput1.txt is a grey-scale image
with header \n\tinput2.txt is a Structuring Element with a double header
\n\toutput1.txt is a file to "
                + " print the result of the a Dilation to \n\toutput2.txt is
a file to print the result of the Erosion.\n\toutput3.txt is a file to print
the result of the Opening."
                + "\n\toutput4.txt is a file to print the result of the
Closing.";

            //STEP -1: Check for bad arguements
            if(args.length != 6) {
                  System.out.println(ARG_ERROR_STRING);
                  System.exit(0);
            }//if

            for(String s: args) {
                  if(!s.endsWith("txt")) {
                        System.out.println(s + " is not a valid arguement.
Must be a .txt file.");
                        System.exit(0);
                  }//if
            }//for

        try{
            //Step 0
            imageInFile = new File(args[0]);
            structElementInFile = new File(args[1]);
            dilationResultOutFile = new File(args[2]);
            erosionResultOutFile = new File(args[3]);
            openingResultOutFile = new File(args[4]);
            closingResultOutFile = new File(args[5]);
            Image img = new Image(imageInFile);
            StructuringElement structuringElement = new
StructuringElement(structElementInFile);

            //Step 1
            Image framedImage = img.frame(structuringElement);

            //Step 2
            framedImage.prettyprint(true);
            structuringElement.prettyprint(true);

            //Step 3
            Image dilatedImage = Morpher.dilation(framedImage, 
structuringElement);
            dilatedImage.prettyprint(true);
            dilatedImage = dilatedImage.unframe(structuringElement);
            dilatedImage.prettyprint(dilationResultOutFile, true);

            //Step 4
            Image erodedImage = Morpher.erosion(framedImage, 
structuringElement);
            erodedImage.prettyprint(true);
            erodedImage = erodedImage.unframe(structuringElement);
            erodedImage.prettyprint(erosionResultOutFile, true);

            //Step 5
            Image openedImage = Morpher.opening(framedImage, 
structuringElement);
            openedImage.prettyprint(true);
            openedImage = openedImage.unframe(structuringElement);
            openedImage.prettyprint(openingResultOutFile, true);

            //Step 6
            Image closedImage = Morpher.closing(framedImage, 
structuringElement);
            closedImage.prettyprint(true);
            closedImage = closedImage.unframe(structuringElement);
            closedImage.prettyprint(closingResultOutFile, true);

        } catch(FileNotFoundException fnf){
            fnf.printStackTrace();
            System.exit(0);
        } catch(IllegalArgumentException iae){
            iae.printStackTrace();
            System.exit(0);
        }
    }//main
}//class
```

Code: Image.java

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

public class Image {
    int numRows;                //number of rows based on header
    int numCols;                //number of cols based on header
    int minVal;                 //lowest value based on header
    int maxVal;                 //largest value based on header

    int[][] imgAry;             //actual image data
    boolean isFramed;           //is this a framed image?

    public Image(){

    }//null contructor

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

    public Image(File imageFile) throws FileNotFoundException{
        Scanner inStream = new Scanner(new FileReader(imageFile));

        //get header values
        this.numRows = inStream.nextInt();
        this.numCols = inStream.nextInt();
        this.minVal = inStream.nextInt();
        this.maxVal = inStream.nextInt();
        initWithZeros();

        //get all image data
        for(int i = 0; i < this.numRows; i++){
            for(int j = 0; j < this.numCols; j++){
                imgAry[i][j] = inStream.nextInt();
            }//for
        }//for
        inStream.close();

        //set as unframed image
        isFramed = false;
    }//constrcutor from file


      /**
     * Initializs imgAry with a 2D array of zeros.
     */
    public void initWithZeros(){
        imgAry = new int[numRows][numCols];
        for(int r = 0; r < numRows; r++){
            for(int c = 0; c < numCols; c++){
                imgAry[r][c] = 0;
            }//for
        }//for
    }//initWithZeroes

    /**
     * @param imgAry the imgAry to set
     */
    public void copyImage(Image image) {
        this.numCols = image.getNumCols();
        this.numRows = image.getNumRows();
        this.minVal = image.getMinVal();
        this.maxVal = image.getMaxVal();
        this.isFramed = image.isFramed();
        this.imgAry = new int[this.numRows][this.numCols];

        int[][] copyAry = image.getImgAry();
        for(int i = 0; i < numRows; i++){
            for(int j=0; j < numCols; j++){
                  this.imgAry[i][j] = copyAry[i][j];
            }//for
        }//fpr

    }//copyImage

    /**
     * Creates a frame around this image based of the origin of the
structuring element
     * @param structuringElement
     * @return a copy of this image framed, null if the image is already
framed
     */
    public Image frame(StructuringElement structuringElement){
        Image framedImage = null;
        if(this.isFramed) return framedImage;

        //get all frame dims
        int[] frameDims = structuringElement.computeFrame();
        int top = frameDims[0];
        int bottom = frameDims[1];
        int left  = frameDims[2];
        int right = frameDims[3];

        //get new imgAry, new row count, new col count
        int newNumRows = this.numRows + top + bottom;
        int newNumCols = this.numCols + left + right;
        int[][] newImgAry = new int[newNumRows][newNumCols];

        //copy this.imgAry into the framed newImgAry
        for(int r = top; r < newNumRows - bottom; r++){
            for(int c = left; c < newNumCols - right; c++){
                newImgAry[r][c] = this.imgAry[r - top][c - left];
            }//for
        }//for

        //create new framedImage to return
        framedImage = new Image(newNumRows, newNumCols, this.minVal, 
this.maxVal);
        framedImage.setImgAry(newImgAry);
        framedImage.setFramed(true);
        return framedImage;
    }//frame()

    /**
     * Unframe an image and return the unframed image
     * @return
     */
    public Image unframe(StructuringElement structuringElement){
        if(!this.isFramed) return this;
        Image unframedImage = new Image();

        //find unframe image dims
        int[] frameDims = structuringElement.computeFrame();
        int top = frameDims[0];
        int bottom = frameDims[1];
        int left  = frameDims[2];
        int right = frameDims[3];

        int newNumRows = this.numRows - top - bottom;
        int newNumCols = this.numCols - left - right;
        int[][] newImgAry = new int[newNumRows][newNumCols];

        //copy this.imgAry into the framed newImgAry
        for(int r = 0; r < newNumRows; r++){
            for(int c = 0; c < newNumCols; c++){
                newImgAry[r][c] = this.imgAry[r + top][c + left];
            }//for
        }//for

        unframedImage.setNumCols(newNumCols);
        unframedImage.setNumRows(newNumRows);
        unframedImage.setImgAry(newImgAry);
        unframedImage.setFramed(false);
        return unframedImage;
    }//unframe

    /**
     * Prints the image to the console
     */
    public void prettyprint(boolean doBinary){
        //header
        System.out.println("\n"+ numRows + " " + numCols + " " + minVal + " " 
+ maxVal);

        //imgAry print -> if doBinary then print 0's else replace 0's with 
space
        for(int i = 0; i < numRows; i++){
            for(int j = 0; j < numCols; j++){
                if(this.imgAry[i][j] == 1)
                    System.out.print("1");
                else{
                    if(doBinary) System.out.print("0");
                    else System.out.print(" ");
                }//if-else
            }//for
            System.out.println();
        }//for
        System.out.println("\n");
    }//prettyprint to console

    /**
     * Prints the image to an outputfile
     * @param outputFile the file to send the image
     * @throws FileNotFoundException
     */
    public void prettyprint(File outputFile, boolean doBinary) throws
FileNotFoundException{
        PrintWriter outputstream = new PrintWriter(outputFile);
        outputstream.println(numRows + " " + numCols  + " " + minVal  + " " +
maxVal);
            for(int i = 0; i < numRows; i++){
            for(int j = 0; j < numCols; j++){
                if(this.imgAry[i][j] == 1)
                    outputstream.print("1");
                else{
                    if(doBinary) outputstream.print("0");
                    else outputstream.print(" ");
                }//if-else
            }//for
            outputstream.println();
        }//for
            outputstream.close();
    }//preetyprint to outputFile

    /**
     * @return the numRows
     */
    public int getNumRows() {
        return numRows;
    }//getNumRows

    /**
     * @param numRows the numRows to set
     */
    public void setNumRows(int numRows) {
        this.numRows = numRows;
    }

    /**
     * @return the numCols
     */
    public int getNumCols() {
        return numCols;
    }

    /**
     * @param numCols the numCols to set
     */
    public void setNumCols(int numCols) {
        this.numCols = numCols;
```

```
}

/**
 * @return the minVal
 */
public int getMinVal() {
    return minVal;
}

/**
 * @param minVal the minVal to set
 */
public void setMinVal(int minVal) {
    this.minVal = minVal;
}

/**
 * @return the maxVal
 */
public int getMaxVal() {
    return maxVal;
}

/**
 * @param maxVal the maxVal to set
 */
public void setMaxVal(int maxVal) {
    this.maxVal = maxVal;
}

/**
 * @return the imgAry
 */
public int[][] getImgAry() {
    return imgAry;
}

/**
 * @param imgAry the imgAry to set
 */
public void setImgAry(int[][] imgAry) {
  this.imgAry = new int[this.numRows][this.numCols];
    for(int i = 0; i < imgAry.length; i++){
        for(int j = 0; j < imgAry[i].length;j++){
              this.imgAry[i][j] = imgAry[i][j];
        }
    }
}

/**
 * @return the isFramed
 */
public boolean isFramed() {
    return isFramed;
}

/**
```

```
     * @param isFramed the isFramed to set
     */
    public void setFramed(boolean isFramed) {
        this.isFramed = isFramed;
    }

}//class
```

Code: StructuringElement.java

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

public class StructuringElement{

    int numRows;
    int numCols;
    int minVal;
    int maxVal;
    int originX;
    int originY;

    int[][] structImgArray;

    public StructuringElement(){

    }//null constructor

    public StructuringElement(int numRows, int numCols, int minVal, int
maxVal, int originX, int originY){
        this.numRows = numRows;
        this.numCols = numCols;
        this.minVal = minVal;
        this.maxVal = maxVal;
        this.originX = originX;
        this.originY = originY;
        initZeroArray();
    }//value contructor

    public StructuringElement(File structFile) throws FileNotFoundException{
        Scanner inStream = new Scanner(new FileReader(structFile));

        //get header values
        this.numRows = inStream.nextInt();
        this.numCols = inStream.nextInt();
        this.minVal = inStream.nextInt();
        this.maxVal = inStream.nextInt();
        this.originX = inStream.nextInt();
        this.originY = inStream.nextInt();
        this.structImgArray = new int[numRows][numCols];

        //get all image data
        for(int i = 0; i < this.numRows; i++){
            for(int j = 0; j < this.numCols; j++){
                  structImgArray[i][j] = inStream.nextInt();
                  //System.out.println("round ("+ i+","+j+")= "+
structImgArray[i][j]);
            }//for
        }//for
        inStream.close();
    }//main constructor
    public void initZeroArray(){
        for(int r = 0; r < this.numRows; r++){
            for(int c = 0; c < numCols; c++){
                structImgArray[r][c] = 0;
            }//for
        }//for
    }//initZeroArray

    public void prettyprint(boolean doBinary){
        //headers
        System.out.println("\n" + numRows + " " + numCols + " " + minVal + "
" + maxVal);
        System.out.println(originX + " " + originY);

        //structElement print -> if doBinary then print 0's else replace 0's
with space
        for(int i = 0; i < numRows; i++){
            for(int j = 0; j < numCols; j++){
                if(structImgArray[i][j] == 1)
                    System.out.print("1");
                else{
                    if(doBinary) System.out.print("0");
                    else System.out.print(" ");
                }//if-else
            }//for
            System.out.println();
        }//for
        System.out.println("\n");
    }//prettyPrint

        /**
     * Computes the framing dimesntions based on a given structuring element.
     * @return int[4] = [topFrameDim, bottomFrameDim, leftFrameDim,
rightFrameDim]
     */
    public int[] computeFrame(){
        int[] dims = new int[4];

        //find distance from origin to all sides, and add said dims to dims[]
        dims[0] = this.originY;
        dims[1] = (this.numRows - 1) - this.originY;
        dims[2] = this.originX;
        dims[3] = (this.numCols - 1) - this.originX;

        return dims;
    }//computeFrame

    /**
     * @return the numRows
     */
    public int getNumRows() {
        return numRows;
    }

    /**
     * @return the numCols
     */
    public int getNumCols() {
        return numCols;
    }

    /**
     * @return the minVal
     */
    public int getMinVal() {
        return minVal;
    }

    /**
     * @return the maxVal
     */
    public int getMaxVal() {
        return maxVal;
    }

    /**
     * @return the originX
     */
    public int getOriginX() {
        return originX;
    }

    /**
     * @return the originY
     */
    public int getOriginY() {
        return originY;
    }

    /**
     * @return the structImgArray
     */
    public int[][] getStructImgArray() {
        return structImgArray;
    }

}//class
```

Code: Morpher.java

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


public abstract class Morpher {

    /**
     * Dilates a framed image using a specified structuring element. Assumes
the image is framed
     * using the specified structElement.
     * @param originalImage the image to dilate
     * @param structElement the structuring element to use
     */
    public static Image dilation(Image originalImage, StructuringElement
structElement) throws IllegalArgumentException{
        //make sure the image inputted is a framed image
        if(!originalImage.isFramed()){
            throw new IllegalArgumentException("Inputted Image must be
framed.");
        }//if

        //copy original image
        Image morphedImage = new Image();
        morphedImage.copyImage(originalImage);
        int[] frameDims = structElement.computeFrame();

        int top = frameDims[0];
        int bottom = frameDims[1];
        int left = frameDims[2];
        int right = frameDims[3];
        int originVal =
structElement.structImgArray[structElement.getOriginX()][structElement.getOri
ginY()];
        int[][] oImg = originalImage.getImgAry();

        //the actual dilation ASSUMES FRAMED IMAGE
        for(int r = top; r < originalImage.getNumRows() - bottom; r++){
            for(int c = left; c < originalImage.getNumCols() - right; c++){

                //if the origin matches the current pixel, then dilate
                  if(oImg[r][c] == originVal){
                    //dilate using each element of the structElement
                    for(int i = 0; i < structElement.getNumRows(); i++){
                        for(int j = 0; j < structElement.getNumCols(); j++){
                            int x = r+i-top;
                            int y = c+j-right;

                            if(structElement.structImgArray[i][j] == 1)
                                morphedImage.imgAry[x][y] = 1;
                        }//for j
                    }//for i
                }//if
            }//for c
        }//for r
        return morphedImage;
    }//dilation

    /**
     * Erodes a framed image using a specified structuring element. Assumes 
the image is framed
     * using the specified structElement.
     * @param originalImage the image to erode
     * @param structElement the structuring element to use
     */
    public static Image erosion(Image originalImage, StructuringElement 
structElement){
        //make sure the image inputted is a framed image
        if(!originalImage.isFramed()){
            throw new IllegalArgumentException("Inputted Image must be 
framed.");
        }//if

        //copy original image
        Image morphedImage = new Image();
        morphedImage.copyImage(originalImage);
        int[] frameDims = structElement.computeFrame();

        int top = frameDims[0];
        int bottom = frameDims[1];
        int left = frameDims[2];
        int right = frameDims[3];

        int[][] newImgAry = new 
int[morphedImage.getNumRows()][morphedImage.getNumCols()];
        for(int i = 0; i < morphedImage.getNumRows(); i++){
            for(int j = 0; j < morphedImage.getNumCols(); j++){
                  newImgAry[i][j] = morphedImage.imgAry[i][j];
            }
        }

        //the actual erosion ASSUMES FRAMED IMAGE
        for(int r = top; r < originalImage.getNumRows() - bottom; r++){
            for(int c = left; c < originalImage.getNumCols() - right; c++){

                //if the structElement can "stand" at r,c then do nothing, 
otherwise wipe to zeros
                boolean matches = true;
                                                  int hit=0;
                for(int i = 0; i < structElement.getNumRows(); i++){
                    for(int j = 0; j < structElement.getNumCols(); j++){
                        int x = r-top+i;
                        int y = c-left+j;

                        //if structElem at this index is 1 but diesnt natch, 
set mayeches=false
                        if(!(structElement.structImgArray[i][j] == 
originalImage.imgAry[x][y]) && (structElement.structImgArray[i][j] == 1)){
                            matches = false;
                        }//if
                    }//for j
                }//for i
                //if there is no match here, wipe to zero
                if(!matches) newImgAry[r][c] = 0;
            }//for
        }//for
        morphedImage.setImgAry(newImgAry);
        return morphedImage;
    }//erosion

    public static Image opening(Image originalImage, StructuringElement
structElement){
        //copy origional image
        Image morphedImage = new Image();
        morphedImage.copyImage(originalImage);

        //run an erosion and then dialtion
        morphedImage = erosion(morphedImage, structElement);
        morphedImage = dilation(morphedImage, structElement);
        return morphedImage;
    }//opening

    public static Image closing(Image originalImage, StructuringElement
structElement){
        //copy origional image
        Image morphedImage = new Image();
        morphedImage.copyImage(originalImage);

        //run a dilation then erosion
        morphedImage = dilation(morphedImage, structElement);
        morphedImage = erosion(morphedImage, structElement);
        return morphedImage;
    }//closing
}//class
```

Output: Dilation Results

```
42 31 0 0
1100000000000001000000000000011
1110000000000011100000000000111
1100110000000111110000000000010
0001111000001111111000000110000
0011111000011111111100001111000
0111110000111111111110001110000
0111111001111111111111000111000
1111110011111111111111100111100
0111101011111111111111100111000
0011111111111111111111100010000
0011111111111111111111111111000
0111111111111111111111111111100
0011111111111111111111111111000
0001110011111111111111100000000
0011100001111111111111110000000
0001000011111111111110111000000
0000000111001111111100011100000
0000001110000111111110001110000
0000011100000011100111000100000
0001001000000011100010000000000
0011100000000011100000000000000
0111110000000011100000000100000
0111110000000111110000001110000
0011100000000111110000011100000
0001001000001111111000111000000
0000011100011111111101111110000
0000111110111111111111111111000
0011110111111111111111011111000
0111100011111111111111111111100
0011000011111111111111110111100
0000000011111111111111100011000
0011000011111111111111100000000
0111100011111111111111100000000
0111100011111111111111111000000
0011000011111111111111111100000
0000000111111111111111111110000
0000001110111111111110000111000
0011011100001111111000110111000
0111111000000111110001111010000
1111110000000111110000111100000
1110100000000011100000011000000
1100000000000001000000000000000


21 21 0 0
011111111101111111110
011111111101111111110
011111111101111111110
111111111111111111111
111111111111111111111
111111111111111111111
111111111111111111111
111111111111111111111
000001111101111100000
111111111111111111111
111111111111111111111
111111111111111111111
111111111111111111111
111111111111111111111
111111111101111111110
111111111101111111110
111111111101111111110
111111111111111111111
111111111111111111111
111111111111111111111
111111111111111111111
```

Output: Erosion Results

```
42 31 0 0
0000000000000000000000000000000
0000000000000000000000000000000
0000000000000001000000000000000
0000000000000011100000000000000
0000000000000110010000000000000
0000000000001100001000000000000
0000000000011100011100000000000
0000000000101100111010000000000
0000000000000000010000000000000
0000000000101000000010000000000
0000000000101100110110000000000
0000000001000111001111000000000
0000000000101010000110000000000
0000000000110001000110000000000
0000000000001000101000000000000
0000000000000000110000000000000
0000000000000001100000000000000
0000000000000001000000000000000
0000000000000000000000000000000
0000000000000000000000000000000
0000000000000000000000000000000
0001000000000000000000000000000
0001000000000001000000000000000
0000000000000001000000000000000
0000000000000011100000000000000
0000000000000110010000000000000
0000000000001100001000000000000
0000000000010010000000000000000
0000000000100001000000000000000
0000000000100001100000000000000
0000000000100011000010000000000
0000000000000110000110000000000
0000000000101110000010000000000
0000000000111001000000000000000
0000000000110000100010000000000
0000000000001001110000000000000
0000000000000111110000000000000
0000000000000011100000000000000
0000000000000001000000000000000
0000000000000001000000000000000
0000000000000000000000000000000
0000000000000000000000000000000


21 21 0 0
000000000000000000000
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
011111111111111111110
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
011111111111111111110
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
000000010000010000000
011111111111111111110
000000000000000000000
```

Output: Opening Results

```
42 31 0 0
0000000000000000000000000000000
0000000000000001000000000000000
0000000000000011100000000000000
0000000000000111110000000000000
0000000000001111111000000000000
0000000000011110011100000000000
0000000000111110111110000000000
0000000001111111111111000000000
0000000000101100111010000000000
0000000001111100110111000000000
0000000001111111111111000000000
0000000011101111111111100000000
0000000001111111001111000000000
0000000001111011101111000000000
0000000000111101111110000000000
0000000000001001111000000000000
0000000000000011110000000000000
0000000000000011100000000000000
0000000000000001000000000000000
0000000000000000000000000000000
0001000000000000000000000000000
0011100000000000100000000000000
0011100000000001110000000000000
0001000000000001110000000000000
0000000000000111100000000000000
0000000000001111110000000000000
0000000000011110011100000000000
0000000000111111001000000000000
0000000001110011100000000000000
0000000001110011110010000000000
0000000001110111100111000000000
0000000000101111001111000000000
0000000001111111000111000000000
0000000001111111100010000000000
0000000001111001110111000000000
0000000000111111111010000000000
0000000000001111111000000000000
0000000000000111110000000000000
0000000000000011100000000000000
0000000000000011100000000000000
0000000000000001000000000000000
0000000000000000000000000000000


21 21 0 0
000000111000111000000
000000111000111000000
000000111000111000000
000000111000111000000
111111111111111111111
111111111111111111111
111111111111111111111
000000111000111000000
000000111000111000000
000000111000111000000
111111111111111111111
111111111111111111111
111111111111111111111
000000111000111000000
000000111000111000000
000000111000111000000
000000111000111000000
000000111000111000000
111111111111111111111
111111111111111111111
111111111111111111111
```

Output: Closing Result

```
42 31 0 0
1000000000000000000000000000001
1100000000000001000000000000010
0000000000000011100000000000000
0000110000000111110000000000000
0001110000001111111000000110000
0011100000011111111100000100000
0011110000111111111110000010000
0111100001111111111111000011000
0011000001111111111111000010000
0001101011111111111111000000000
0001111111111111111111100010000
0011111111111111111111111111000
0001110011111111111111100000000
0000100001111111111111000000000
0001000000111111111110100000000
0000000001001111111100010000000
0000000010000111111000001000000
0000000100000011100100000100000
0000001000000000100001000000000
0000000000000001000000000000000
0001000000000001000000000000000
0011100000000001000000000000000
0011100000000011100000000100000
0001000000000011100000001000000
0000000000000111110000010000000
0000001000001111111000111000000
0000010100011111111101011110000
0000100010111111111110001110000
0011000001111111111111010111000
0000000001111111111111100011000
0000000001111111111111000000000
0000000001111111111111000000000
0011000001111111111111000000000
0011000001111111111111100000000
0000000001111111111111111000000
0000000010111111111110000100000
0000000100001111111000000010000
0000001000000111110000000010000
0011010000000011100000110000000
0110100000000011100000011000000
1100000000000001000000000000000
0000000000000000000000000000000


21 21 0 0
000000111000111000000
001111111000111111100
001111111000111111100
001111111000111111100
111111111111111111111
111111111111111111111
111111111111111111111
000000111000111000000
000000111000111000000
000000111000111000000
111111111111111111111
111111111111111111111
111111111111111111111
011111111000111111100
011111111000111111100
011111111000111111100
011111111000111111100
011111111000111111100
111111111111111111111
111111111111111111111
111111111111111111111
```