

[illegible]

.....1111111111.....
.....1.....122222221.....
.....111.....1233333211.....
.....11211.....123444432211.....
.....1122211.....1234554332211.....
.....112232211.....12345544332211.....
.....122333221.....123455544332211.....
.....123343321.....1234565544332211.....
.....123444321.....12345665544332211.....
.....123454321.....1123456665544332211.....
.....123454321.....12345676655443322.....
.....123454321.....1234567766554433.....
.....1111234543211111.....123456777665544.....
.....1222234543222221.....12345678776655.....
.....123333454333321.....123456788776.....
.....123444454444321.....12345678887.....
.....123455555554321.....123456789.....
.....123456666654321.....123456789.....
.....1234567777654321.....1234567811.....
.....123456788765432111112345672211.....
.....123456788765432222223456332211.....
.....1234567887654333333334544332211.....
.....12345678876544444444455443322.....
.....123456788765555555555555554433.....
.....1234567887666666666666665544.....
.....12345678877777777777777776655.....
.....12345678888888.....
.....123456789999.....
.....12345678910.....
.....12345678.....

Printing Table After Dist 8 pass 2
30 40 0 1

.....1111111111.....
.....1.....122222221.....
.....111.....1233333211.....
.....11211.....123444432211.....
.....1122211.....1234554332211.....
.....112232211.....12345544332211.....
.....122333221.....123455544332211.....
.....123343321.....1234565544332211.....
.....123444321.....12345665544332211.....
.....123454321.....112345665544332211.....
.....123454321.....12345655444332211.....
.....123454321.....1234555443332211.....
.....1111234543211111.....123455443322211.....
.....1222234543222221.....1234544332211.....
.....123333454333321.....123454332211.....
.....1234444544444321.....12345432211.....
.....123455555554321.....1234543211.....
.....123456666654321.....123454321.....
.....1234567777654321.....1234543211.....
.....123456777765432111112345432211.....
.....123456677665432222223454332211.....
.....123455666554333333334544332211.....
.....123445566554444444444444444332211.....
.....123344555544333333333333332211.....
.....12233445544332222222222222211.....
.....1122334444332211111111111111.....
.....11223344332211.....
.....112233332211.....
.....1122222211.....
.....11111111.....

Running Local Maxima 30 40 0 1

.....
.....1.....
.....
.....2.....
.....55.....
.....3.....55.....
.....
.....4.....6.....
.....66.5.....
.....5.....1.....66.55.4.3.2.1.....
.....5.....6.....
.....5.....55.....
.....5.....5.....
.....5.....5.....
.....5.....5.....
.....5.....
.....7777.....5.....
.....7777.....5.....
.....77.....5.....
.....66.....44444444.....
.....
.....55.....
.....44.....
.....
.....

Output2.txt

Zero fram arr

30 40 0 1

```
.....
.....1.....
.....
.....2.....
.....55.....
.....3.....55.....
.....
.....4.....6.....
.....66.5.....
.....5.....1.....66.55.4.3.2.1
.....5.....6.....
.....5.....
.....5.....55.....
.....5.....5.....
.....5.....5.....
.....5.....5.....
.....5.....
.....5.....
.....7777.....5.....
.....7777.....5.....
.....77.....5.....
.....5.4.3.2.1.....
.....66...44444444.....
.....
.....55.....
.....
.....44.....
.....
.....
.....
```

After Expansion Pass 1

30 40 0 1

```
.....
.....1.....
.....111.....
.....121.....4444321.....
.....2221.....34554321.....
.....12321.....234554321.....
.....133321.....12345554321.....
.....234321.....1234565544321.....
.....12444321.....12345665544332211.
.....13454321.....1123456655544332211
.....23454321.....12345655444332211.
.....123454321.....1234555443332211..
.....123454321.....123455443322211...
.....123454321.....12345443322111....
.....123454321.....123454332211.....
.....123454321.....12345432211.....
.....123444321.....1234543211.....
.....1266666654321...123454321.....
.....1567777654321...123454321.....
.....4567777654321...123454321.....
.....34566776654321...123454332211.....
```

```

....23455666655433333333334544332211.....
...12344556655444444444444444332211.....
...123344555544333333333333332211.....
...12233445544332222222222222211.....
...1122334444332211111111111111.....
....11223344332211.....
.....112233332211.....
.....1122222211.....
.....11111111.....

```

After Expansion Pass 2

30 40 0 1

```

.....1111111111.....
.....1.....122222221.....
.....111.....12333333211.....
.....11211.....123444432211.....
.....1122211.....1234554332211.....
.....112232211.....12345544332211....
.....122333221.....123455544332211...
.....123343321.....1234565544332211..
.....123444321.....12345665544332211.
.....123454321.....1123456655544332211
.....123454321.....12345655444332211.
.....123454321.....1234555443332211..
...1111234543211111...123455443322211...
...1222234543222221...12345443322111....
...1233334543333321...123454332211.....
...1234444544444321...12345432211.....
...123455555554321...1234543211.....
...123456666654321...123454321.....
...1234567777654321...1234543211.....
...123456777765432111112345432211.....
...1234566776654322222223454332211.....
...12345566665543333333334544332211.....
...12344556655444444444444444332211.....
...123344555544333333333333332211.....
...12233445544332222222222222211.....
...1122334444332211111111111111.....
....11223344332211.....
.....112233332211.....
.....1122222211.....
.....11111111.....

```

Skeleton.txt

```

2 11 1
4 11 2
5 27 5
5 28 5
6 11 3
6 27 5
6 28 5
8 11 4
8 28 6
9 28 6
9 29 6
9 31 5

```

10 11 5
10 22 1
10 28 6
10 29 6
10 31 5
10 32 5
10 34 4
10 36 3
10 38 2
10 40 1
11 11 5
11 28 6
12 11 5
13 11 5
13 27 5
13 28 5
14 11 5
14 27 5
15 11 5
15 27 5
16 11 5
16 27 5
17 27 5
18 27 5
19 10 7
19 11 7
19 12 7
19 13 7
19 27 5
20 10 7
20 11 7
20 12 7
20 13 7
20 27 5
21 11 7
21 12 7
21 27 5
22 27 5
22 29 4
22 31 3
22 33 2
22 35 1
23 11 6
23 12 6
23 17 4
23 18 4
23 19 4
23 20 4
23 21 4
23 22 4

23 23 4
23 24 4
23 25 4
25 11 5
25 12 5
27 11 4
27 12 4

Debug.txt

Entering Distance8

30 40 0 1

```
.....111111111.....
.....1.....122222221.....
.....111.....1233333211.....
.....11211.....123444432211.....
.....1122211.....1234554332211.....
.....112232211.....12345544332211...
.....122333221.....123455544332211...
.....123343321.....1234565544332211..
.....123444321.....12345665544332211.
.....123454321.....1123456665544332211
.....123454321.....12345676655443322.
.....123454321.....1234567766554433..
...1111234543211111...123456777665544...
...1222234543222221...12345678776655....
...1233334543333321...123456788776.....
...1234444544444321...12345678887.....
...123455555554321...1234567898.....
...123456666654321...123456789.....
...1234567777654321...1234567811.....
...123456788765432111112345672211.....
...123456788765432222223456332211.....
...1234567887654333333334544332211.....
...12345678876544444444455443322.....
...1234567887655555555555554433.....
...123456788766666666666665544.....
...123456788777777777777776655.....
....1234567888888.....
.....123456789999.....
.....12345678910.....
.....12345678.....
```

30 40 0 1

```
.....111111111.....
.....1.....122222221.....
```

```

.....111.....12333333211.....
.....11211.....123444432211.....
.....1122211.....1234554332211.....
.....112232211.....12345544332211....
.....122333221.....123455544332211...
.....123343321.....1234565544332211..
.....123444321.....12345665544332211.
.....123454321.....1123456655544332211
.....123454321.....12345655444332211.
.....123454321.....1234555443332211..
...1111234543211111...123455443322211...
...1222234543222221...12345443322111....
...1233334543333321...123454332211.....
...1234444544444321...12345432211.....
...123455555554321...1234543211.....
...123456666654321...123454321.....
...1234567777654321...1234543211.....
...123456777765432111112345432211.....
...1234566776654322222223454332211.....
...123455666554333333334544332211.....
...12344556655444444444444332211.....
...1233445554433333333333332211.....
...122334455443322222222222211.....
...11223344443322111111111111.....
...11223344332211.....
...112233332211.....
.....112222211.....
.....11111111.....

```

Leaving Distance8

Entering skeletonExtraction

Leaving skeletonExtraction

Entering deCompression method

Leaving deCompression method

Decompress.txt

30 40 0 1

```
00000000000000000000000000111111111100000000
00000000000010000000000000111111111100000000
00000000000111000000000000111111111110000000
00000000011111000000000001111111111111000000
0000000011111110000000001111111111111100000
0000000111111111000000001111111111111110000
0000000111111111100000001111111111111111000
```

[illegible]

Image2 Outputs

Output1.txt

[illegible]

[illegible]

Skeleton.txt

4	3	1	1
6	3	1	2
8	1	1	1
8	3	1	3
8	5	2	4
9	5	2	4
10	1	1	2
10	3	1	4
10	5	2	4
11	5	2	4
12	1	1	3
12	3	1	5
12	5	2	4
13	2	2	1
13	2	4	2
13	2	6	3
13	2	8	4
13	3	0	5
13	3	1	5
13	3	2	5
13	3	4	4
13	3	6	3
13	3	8	2
13	4	0	1
13	5	2	4
14	1	1	4
14	3	1	5
14	5	2	4
15	5	2	4
16	1	1	5
16	3	1	4
16	5	2	4
17	5	2	4
18	1	1	6
18	3	1	3
18	5	2	4
19	5	2	4
20	1	1	7
20	3	2	2

20 52 4
21 4 4
21 6 5
21 8 6
21 10 7
21 11 7
21 12 7
21 14 6
21 16 5
21 18 4
21 20 3
21 22 2
21 24 1
21 52 4
22 11 7
22 31 1
22 52 4
23 31 1
23 52 4
24 11 6
24 52 4
25 31 2
25 52 4
26 11 5
26 52 4
27 31 3
27 52 4
28 11 4
28 52 4
29 32 4
29 52 4
30 11 3
30 52 4
31 32 5
31 36 3
31 52 4
32 11 2
32 22 1
32 24 2
32 26 3
32 27 3
32 30 5
32 31 5
32 32 5
32 34 4
32 36 3
32 38 2
32 40 1
32 52 4
33 27 3
33 31 5
34 11 1
35 31 4

37 31 3
39 31 2
41 31 1
45 11 3
45 12 3
45 13 3
45 14 3
45 15 3
45 16 3
45 17 3
45 18 3
45 19 3
45 20 3
45 21 3
45 22 3
45 23 3
45 24 3
45 25 3
45 26 3
45 27 3
45 28 3
45 29 3
45 30 3
45 31 3
45 32 3
45 33 3
45 34 3
45 35 3
45 36 3
45 37 3
45 38 3
45 39 3
45 40 3
45 41 3
45 42 3
45 43 3
45 44 3
45 45 3
45 46 3
45 47 3
45 48 3
45 49 3
45 50 3
45 51 3
45 52 3
45 53 3
46 11 3
46 12 3
46 13 3
46 14 3
46 15 3
46 16 3
46 17 3

46 18 3
46 19 3
46 20 3
46 21 3
46 22 3
46 23 3
46 24 3
46 25 3
46 26 3
46 27 3
46 28 3
46 29 3
46 30 3
46 31 3
46 32 3
46 33 3
46 34 3
46 35 3
46 36 3
46 37 3
46 38 3
46 39 3
46 40 3
46 41 3
46 42 3
46 43 3
46 44 3
46 45 3
46 46 3
46 47 3
46 48 3
46 49 3
46 50 3
46 51 3
46 52 3
46 53 3

Debug.txt

Entering Distance8

50 64 0 1

```
.....  
.....  
.....  
.....1.....  
.....111.....111111.....  
.....11211.....1222221.....  
.....1122211.....1233321.....  
.....1.....112232211.....1234321.....  
.....111.....11223332211.....1234321.....  
.....11211.....1122334332211.....1234321.....  
.....1122211.....112233444332211.....1234321.....  
.....112232211.....11223344544332211.....1234321.....  
.....11223332211.....112233445544332211.....1234321.....  
.....1122334332211.....12334455655443322.....1234321.....  
.....112233444332211.....123455666554433.....1234321.....  
.....11223344544332211.....1234567665544.....1234321.....  
.....1122334455544332211.....12345676655.....1234321.....  
112233445565544332211.....123456766.....1234321.....
```



```
Leaving Distance8
Entering skeletonExtraction
Leaving skeletonExtraction
Entering deCompression method
Leaving deCompression method
```

[illegible]

Source Code

```
import java.io.*;

public class Main {

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

        //step 0
        File in = new File(args[0]),
            outFile1 = new File(args[1]),
            outFile2 = new File(args[2]),
            de = new File(args[3]);

        FileReader inFile = new FileReader(in);

        FileWriter out1 = new FileWriter(outFile1),
            out2 = new FileWriter(outFile2),
            debug = new FileWriter(de);

        BufferedReader br = new BufferedReader(inFile);
        String line = br.readLine();
        String [] header = line.split(" ");
        int numRows = Integer.parseInt(header[0]),
            numCol = Integer.parseInt(header[1]),
            minVal = Integer.parseInt(header[2]),
            maxVal = Integer.parseInt(header[3]);

        ImageProcessing proj = new ImageProcessing(numRow, numCol, minVal, maxVal);

        //step 1 + 2
        String fileName = args[0];
        fileName = fileName.substring(0, fileName.length()-4);
        FileWriter skeletonFile = new FileWriter(fileName + "_skeleton.txt");

        //step 3+4
        FileWriter decompressedFile = new FileWriter(fileName + "_decompressed.txt");

        //step 5
        proj.setZero(proj.zeroFramedAry);
        proj.setZero(proj.skeletonAry);

        //step 6
        proj.loadImage(inFile, br);

        //step 7
        proj.Distance8(out1, debug);
        // proj.printZero();

        //step 8
        proj.skeletonExtract(skeletonFile, out1, debug);

        //step 9
        FileReader skeletonFileRead = new FileReader(fileName + " skeleton.txt");
```

```

        //step 10
        proj.deCompression(skeletonFileRead, out2, debug);

        //step 11
        proj.binThr(proj.zeroFramedAry);

        //step 12
        //
        decompressedFile.write(numRow+" " +numCol +" " + minVal +" " + maxVal);

        //step 13
        decompressedFile.write(numRow + " " + numCol + " " + minVal + " " + maxVal +
"\n");
        for (int row = 1; row < numRow + 1; row++) {
            for (int col = 1; col < numCol + 1; col++) {
                decompressedFile.write(proj.zeroFramedAry[row][col] + " ");
            }
            decompressedFile.write("\n");
        }

        //step 14
        inFile.close();
        out1.close();
        out2.close();
        debug.close();
        decompressedFile.close();
        skeletonFileRead.close();

    }
}

```

ImageProcessing.java

```

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.Buffer;
import java.util.Arrays;
import java.util.zip.ZipError;

public class ImageProcessing {

    public int numRows,
        numCols,
        minVal,
        maxVal,
        newMinVal,
        newMaxVal;

    public int[][] zeroFramedAry,
        skeletonAry;

    public ImageProcessing(int row, int col, int min, int max){

        numRows = row;

```

```

        numCols = col;
        minVal = min;
        maxVal = max;

        zeroFramedAry = new int[numRows+2][numCols+2];
        skeletonAry = new int[numRows+2][numCols+2];
    }

    public void setZero(int[][] arr) {
        // Set all elements of Ary to 0
        for (int[] ints : arr) {
            Arrays.fill(ints, 0);
        }
    }

    public void loadImage(FileReader file, BufferedReader br) throws IOException {
        // Read image data from file and store it in the frame of zeroFramedAry
        // Implementation not shown
        String line;
        //      System.out.println(line)

        int count = 1;

        while((line=br.readLine()) != null){
            String[] data = line.split(" ");
            //      for (int i = count; i < numRows+1; i++){
            //          for (int j = 1; j< numCols+1; j++){
            //              zeroFramedAry[count][j] = Integer.parseInt(data[j-1]);
            //          }
            //      }
            count++;
        }

        printZero();
    }

    public void printZero(){
        for (int i = 0; i < numRows+2; i++){
            for (int j = 1; j< numCols+2; j++){
                if(zeroFramedAry[i][j] == 0){
                    //      System.out.print(". ");
                }else{
                    //      System.out.print(zeroFramedAry[i][j] + " ");
                }
            }
            //      System.out.println();
        }
    }

    public void Distance8(FileWriter out1, FileWriter debug) throws IOException {
        // Compute the 8-neighbor distance of pixel (i, j) in zeroFramedAry
        // Implementation not shown
        //step 0
        debug.write("Entering Distance8\n");
        //step 1
        out1.write("Printing OG Table\n");
        reformatPrettyPrint(zeroFramedAry, out1);
        distance8pass1(zeroFramedAry);
    }

```

```

        //step 2
        out1.write("Printing Table After Dist 8 pass 1\n");
        reformatPrettyPrint(zeroFramedAry, out1);
        reformatPrettyPrint(zeroFramedAry, debug);
        //step 3
        out1.write("Printing Table After Dist 8 pass 2\n");
        distance8pass2(zeroFramedAry);
        //step 4
        reformatPrettyPrint(zeroFramedAry, out1);
        reformatPrettyPrint(zeroFramedAry, debug);
        //step 5
        debug.write("Leaving Distance8\n");
    }

    private void distance8pass2(int[][] arr) {

        for (int r = numRows; r > 0 ; r--) {
            for (int c = numCols; c > 0 ; c--) {

                if(zeroFramedAry[r][c] > 0){
                    int e = zeroFramedAry[r][c + 1];
                    int f = zeroFramedAry[r + 1][c - 1];
                    int g = zeroFramedAry[r + 1][c];
                    int h = zeroFramedAry[r+1][c +1];

                    zeroFramedAry[r][c] = Math.min(Math.min(e+1, Math.min(f+1,
Math.min(g+1,h+1))), zeroFramedAry[r][c]);
                    newMinVal = Math.min(newMinVal , zeroFramedAry[r][c]);
                    newMaxVal = Math.max(newMaxVal , zeroFramedAry[r][c]);
                }

            }
        }
    }

    private void reformatPrettyPrint(int[][] arr, FileWriter out1) throws IOException
    {

        out1.write(numRows + " " + numCols + " " + minVal + " " + maxVal + "\n");
        String str = Integer.toString(newMaxVal);
        int width = str.length();

        int ww;
        int r = 1;

        while (r < numRows+1) {
            int c = 1;
            while (c < numCols+1) {
                if (arr[r][c] == 0) {
                    out1.write(".");
                } else {
                    out1.write(arr[r][c] + "");
                }
                str = Integer.toString(arr[r][c]);
                ww = str.length();
                while (ww <= width) {
                    out1.write(" ");
                    ww++;
                }
                c++;
            }
            r++;
            out1.write("\n");
        }
    }
}

```

```

    }
    out1.write("\n");

}

private void distance8pass1(int[][] arr) {

    for (int r = 1; r < numRows + 1; r++) {
        for (int c = 1; c < numCols + 1; c++) {

            if (arr[r][c] > 0) {
                int a = arr[r - 1][c - 1];
                int b = arr[r - 1][c];
                int c1 = arr[r - 1][c + 1];
                int d = arr[r][c - 1];

                zeroFramedArry[r][c] = Math.min(a, Math.min(b, Math.min(c1,d))) +
1;

            }

        }

    }

}

public void skeletonExtract(FileWriter skeletonFile, FileWriter out1, FileWriter
debug) throws IOException {
//    Step 0: debugFile//\nEntering skeletonExtraction"
    debug.write("Entering skeletonExtraction\n");
//    Step 1: localMaxima (zeroFramedArry, skeletonArry)
    localMax();
//    Step 2: reformatPrettyPrint (skeletonArry, outFile1)
    out1.write(" Running Local Maxima ");
    reformatPrettyPrint(skeletonArry, out1);
//    Step 3: compression (skeletonArry, skeletonFile)
    compression(skeletonArry, skeletonFile);
//    Step 4: close skeletonFile
    skeletonFile.close();
//    Step 5: debugFile//\nLeaving skeletonExtraction"
    debug.write("Leaving skeletonExtraction\n");
}

private void compression(int[][] arr, FileWriter skeletonFile) throws IOException
{

    for (int r = 1; r < numRows + 1; r++) {
        for (int c = 1; c < numCols + 1; c++) {

            if ( arr[r][c] > 0) {
                skeletonFile.write(r + " " + c + " " + arr[r][c] + "\n");
            }

        }

    }

}

private void localMax() {
    for (int r = 1 ; r<numRows+1; r++){

```

```

        for(int c=1; c < numCols+1; c++){

            //step 2
            if(isLocalMax(r,c)){

                skeletonAry[r][c] = zeroFramedAry[r][c];
            }else{
                skeletonAry[r][c] = 0;
            }
        }
    }

    private boolean isLocalMax(int r, int c) {

        int a = zeroFramedAry[r - 1][c - 1],
            b = zeroFramedAry[r - 1][c],
            c1 = zeroFramedAry[r - 1][c + 1],
            d = zeroFramedAry[r][c - 1],
            e = zeroFramedAry[r][c + 1],
            f = zeroFramedAry[r + 1][c - 1],
            g = zeroFramedAry[r + 1][c],
            h = zeroFramedAry[r + 1][c + 1];

        return zeroFramedAry[r][c] >= a &&
            zeroFramedAry[r][c] >= b &&
            zeroFramedAry[r][c] >= c1 &&
            zeroFramedAry[r][c] >= d &&
            zeroFramedAry[r][c] >= e &&
            zeroFramedAry[r][c] >= f &&
            zeroFramedAry[r][c] >= g &&
            zeroFramedAry[r][c] >= h;
    }

    public void deCompression( FileReader skeletonFile, FileWriter out2, FileWriter
debug) throws IOException {
    //    Step 0: debugFile // "Entering deCompression method"
    debug.write("Entering deCompression method\n");
    //    Step 1: setZero (zeroFramedAry)
    setZero(zeroFramedAry);
    //    step 2: load (skeletonFile, zeroFramedAry)
    load(skeletonFile);
    out2.write("Zero fram arr\n");
    reformatPrettyPrint(zeroFramedAry, out2);
    //    step 3: expansion8Pass1 (zeroFramedAry)
    expansion8Pass1();
    //    step 4: reformatPrettyPrint (zeroFramedAry, outFile2)
    out2.write("After Expansion Pass 1\n");
    reformatPrettyPrint(zeroFramedAry, out2);
    //    step 5: expansion8Pass2 (zeroFramedAry)
    expansion8Pass2();
    //    step 6
    out2.write("After Expansion Pass 2\n");
    reformatPrettyPrint(zeroFramedAry, out2);
    //step 7
    debug.write("Leaving deCompression method\n");

}

    private void load(FileReader skeletonFile) throws IOException {
        int r, c, val;
        String line;

```



```

BufferedReader br = new BufferedReader(skeletonFile);

while((line=br.readLine()) != null){
    String[] comp = line.split(" ");
    r = Integer.parseInt(comp[0]);
    c = Integer.parseInt(comp[1]);
    val = Integer.parseInt(comp[2]);
    zeroFramedAry[r][c] = val;
}

}

private void expansion8Pass1() {

    for (int r = 1; r < numRows + 1; r++) {
        for (int c = 1; c < numCols + 1; c++) {

            int a = zeroFramedAry[r - 1][c - 1] - 1,
                b = zeroFramedAry[r - 1][c] - 1,
                cl = zeroFramedAry[r - 1][c + 1] - 1,
                d = zeroFramedAry[r][c - 1] - 1,
                e = zeroFramedAry[r][c + 1] - 1,
                f = zeroFramedAry[r + 1][c - 1] - 1,
                g = zeroFramedAry[r + 1][c] - 1,
                h = zeroFramedAry[r + 1][c + 1] - 1;

            //step 2
            if (zeroFramedAry[r][c] == 0) {

                zeroFramedAry[r][c] = Math.max( zeroFramedAry[r][c],
Math.max(a, Math.max(b, Math.max(cl, Math.max(d, Math.max(e, Math.max(f, Math.max(g,
h))))))));

            }

        }
    }

}

private void expansion8Pass2() {

    for (int r = numRows; r > 0; r--) {
        for (int c = numCols; c > 0; c--) {

            //step 2
            int a = zeroFramedAry[r - 1][c - 1] - 1,
                b = zeroFramedAry[r - 1][c] - 1,
                cl = zeroFramedAry[r - 1][c + 1] - 1,
                d = zeroFramedAry[r][c - 1] - 1,
                e = zeroFramedAry[r][c + 1] - 1,
                f = zeroFramedAry[r + 1][c - 1] - 1,
                g = zeroFramedAry[r + 1][c] - 1,
                h = zeroFramedAry[r + 1][c + 1] - 1,
                max = zeroFramedAry[r][c];

            max = zeroFramedAry[r][c] = Math.max( zeroFramedAry[r][c],
Math.max(a, Math.max(b, Math.max(cl, Math.max(d, Math.max(e, Math.max(f, Math.max(g,
h))))))));


```

```
        if (zeroFramedAry[r][c] < max) {  
            zeroFramedAry[r][c] = max;  
        }  
    }  
}  
  
public void binThr(int[][] zeroFramedAry) {  
    for(int r = 0; r < numRows + 2; r++) {  
        for(int c = 0; c < numCols + 2; c++) {  
            if(zeroFramedAry[r][c]>0){  
                zeroFramedAry[r][c] = 1;  
            }else{  
                zeroFramedAry[r][c] = 0;  
            }  
        }  
    }  
}
```