Student: Pawan Bhatta
Project Due Date: 03/30/2021

Distance Transform- Pass 1 Algorithm:
Step 0: image ← given Binary Image
Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel
Step 2: if P(i,j) > 0 //is an object pixel
look at neighbors: a, b, c, d
P(i,j) ← min(a, b, c, d) + 1
Step 3: repeat steps 1 to 2 until all pixels are processed

Distance Transform- Pass 2 Algorithm:
Step 0: image ← given the result of Pass-1
Step 1: Scan image Right-Left & Bottom-Top
P(i,j) ← next pixel
Step 2: if P(i,j) > 0 //is an object pixel
Look at neighbors: e, f, g, h and P(i,j)
P(i,j) ← min(e+1, f+1, g+1, h+1 , P(i,j))
Step 3: repeat steps 1 to 2 until all pixels are processed

Local Maxima Operation Algorithm:
Step 0: image ← given the result of Pass-2
skeleton ← array of same size at image
Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel
Step 2: check if P(i,j) is a Local Maxima:
P(i,j) is a Local Maxima:
iff P(i,j) >= a, b, c, d, e, f, g, h
skeleton(i,j) ← P(i,j) //retain the distance
else
skeleton(i,j) ← 0
Step 3: repeat steps 1 to 2 until all pixels are processed

Skeleton Image Compression Algorithm:
Step 0: image ← given the result of Local Maxima Operation
output ← file for result of compression
Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel
Step 2: if P(i,j) > 0 //is a Local Maxima
output ← i j P(i,j)
Step 3: repeat steps 1 to 2 until all pixels are processed

Expansion Pass 1:

Step 0: image ← load the File rendered by Skeleton Compression
Step 1: Scan image Left-Right & Top-Bottom
    P(i,j) ← next pixel
Step 2: if P(i,j) == 0
    look at neighbors of P(i,j): a, b, c, d, e, f, g, h
    max ← max(a-1, b-1, c-1, d-1, e-1, f-1, g-1, h-1)
    if P(i,j) < max
      P(i,j) ← max
Step 3: repeat steps 1 to 2 until all pixels are processed

Expansion Pass 2:

Step 0: image ← rendered by Expansion Pass 1
Step 1: Scan image Right-Left & Bottom-Top
    P(i,j) ← next pixel
Step 2: look at neighbors of P(i,j): a, b, c, d, e, f, g, h
    max ← max(a, b, c, d, e, f, g, h)
    if P(i,j) < max
      P(i,j) ← max-1
Step 3: repeat steps 1 to 2 until all pixels are processed

Threshold Decompression Algorithm:

Step 0: output ← open file for the decompressed image
    image ← result of Expansion Pass-2
Step 1: write the original image header to the de-compressed file
    output ← numRows , numCols , minVal , maxVal
Step 2: Threshold the image
    Scan image Left-Right & Top-Bottom
    P(i,j) ← next pixel
Step 3: if P(i,j) > 0
    output ← 1 and a blank space
    else
    output ← 0 and a blank space
Step 4: repeat steps 2 to 3 until all pixels are processed

Source Code:

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

class ImageProcessing {
    int numImgRows;
    int numImgCols;
    int minVal;
    int maxVal;
    int newMin;
    int newMax;
    int rowFrameSize;
    int colFrameSize;
    int extraRows;
    int extraCols;
    int[][] zeroFramedAry;
    int[][] skeletonAry;

    ImageProcessing(Scanner imgFile) {
        setTotalFrameSize();
        setFrameSize();
        loadHeader(imgFile);
        initializeArrays();

    }

    void initializeArrays() {
        zeroFramedAry = new int[numImgRows + extraRows][numImgCols + extraCols];
        skeletonAry = new int[numImgRows + extraRows][numImgCols + extraCols];
    }

    void setTotalFrameSize() {
        extraRows = 2;
        extraCols = 2;
    }

    void setFrameSize() {
        rowFrameSize = extraRows / 2;
        colFrameSize = extraCols / 2;
    }

    void loadHeader(Scanner imgFile) {
        numImgRows = imgFile.nextInt();
        numImgCols = imgFile.nextInt();
        minVal = imgFile.nextInt();
        maxVal = imgFile.nextInt();
    }

    void loadImg(Scanner imgFile) {
```

```
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
                zeroFramedAry[i][j] = imgFile.nextInt();
            }
        }
    }

    void loadTriplets(Scanner compressedImg) {
        int count = -1;
        while (compressedImg.hasNextLine()) {
            count++;
            String line = compressedImg.nextLine();
            if (count == 0) {
                continue;
            }

            String[] splitStr = line.trim().split("\\s+");
            int i = Integer.parseInt(splitStr[0]);
            int j = Integer.parseInt(splitStr[1]);
            zeroFramedAry[i + rowFrameSize][j + colFrameSize] =
Integer.parseInt(splitStr[2]);
        }
    }

    void writeHeader(BufferedWriter outFile) throws IOException {
        outFile.write(numImgRows + " " + numImgCols + " " + newMin + " " + newMax + "\n");
    }

    void prettyPrint(BufferedWriter outFile) throws IOException {
        writeHeader(outFile);
        int maxLength;
        maxLength = Integer.toString(newMax).length();
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
                if (zeroFramedAry[i][j] == 0) {
                    outFile.write(". ");
                } else {
                    outFile.write(Integer.toString(zeroFramedAry[i][j]) + " ");
                }
                int currentLength;
                currentLength = Integer.toString(zeroFramedAry[i][j]).length();

                while (currentLength < maxLength) {
                    outFile.write(" ");
                    currentLength++;
                }
            }
            outFile.write("\n");
        }
    }
```

```
void prettyPrint(BufferedWriter outFile, int[][] arrayToPrint) throws IOException {
    writeHeader(outFile);
    int maxLength;
    maxLength = Integer.toString(newMax).length();
    for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
        for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
            if (arrayToPrint[i][j] == 0) {
                outFile.write(". ");
            } else {
                outFile.write(Integer.toString(arrayToPrint[i][j]) + " ");
            }
            int currentLength;
            currentLength = Integer.toString(arrayToPrint[i][j]).length();

            while (currentLength < maxLength) {
                outFile.write(" ");
                currentLength++;
            }
        }
        outFile.write("\n");
    }
}

void print2DArray(int[][] array) {
    for (int i = 0; i < array.length; i++) {
        for (int j = 0; j < array[0].length; j++) {
            System.out.print(array[i][j]);
        }
        System.out.print("\n");
    }
    System.out.print("\n\n");
}

int findMin(int... array) {
    int minVal = 9999;
    for (int i : array) {
        if (i < minVal) {
            minVal = i;
        }
    }
    return minVal;
}

int findMax(int... array) {
    int maxVal = 0;
    for (int i : array) {
        if (i > maxVal) {
            maxVal = i;
        }
```

```
        }
        return maxVal;
    }

    void passOne8Connectedness() {
        newMin = 9999;
        newMax = 0;
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
                int pixelVal = zeroFramedAry[i][j];
                if (pixelVal > 0) {
                    int a = zeroFramedAry[i - 1][j - 1];
                    int b = zeroFramedAry[i - 1][j];
                    int c = zeroFramedAry[i - 1][j + 1];
                    int d = zeroFramedAry[i][j - 1];
                    zeroFramedAry[i][j] = findMin(a, b, c, d) + 1;
                }
                // Updating newMin and newMax
                if (zeroFramedAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (zeroFramedAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
        }
    }

    void passTwo8Connectedness() {
        newMin = 9999;
        newMax = 0;
        for (int i = numImgRows + rowFrameSize - 1; i >= rowFrameSize; i--) {
            for (int j = numImgCols + colFrameSize - 1; j >= colFrameSize; j--) {
                int pixelVal = zeroFramedAry[i][j];
                if (pixelVal > 0) {
                    int e = zeroFramedAry[i][j + 1];
                    int f = zeroFramedAry[i + 1][j - 1];
                    int g = zeroFramedAry[i + 1][j];
                    int h = zeroFramedAry[i + 1][j + 1];
                    zeroFramedAry[i][j] = findMin(e + 1, f + 1, g + 1, h + 1, pixelVal);
                }
                // Updating newMin and newMax
                if (zeroFramedAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (zeroFramedAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
        }
```

```
    }

    boolean isLocalMaxima(int p, int[] neighbours) {
        boolean returnVal = true;
        for (int neighbour : neighbours) {
            if (p < neighbour) {
                returnVal = false;
            }
        }
        return returnVal;
    }

    void localMaxima() {
        newMin = 9999;
        newMax = 0;
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
                int pixelVal = zeroFramedAry[i][j];
                int a = zeroFramedAry[i - 1][j - 1];
                int b = zeroFramedAry[i - 1][j];
                int c = zeroFramedAry[i - 1][j + 1];
                int d = zeroFramedAry[i][j - 1];
                int e = zeroFramedAry[i][j + 1];
                int f = zeroFramedAry[i + 1][j - 1];
                int g = zeroFramedAry[i + 1][j];
                int h = zeroFramedAry[i + 1][j + 1];
                int[] neighbours = { a, b, c, d, e, f, g, h };
                if (isLocalMaxima(pixelVal, neighbours)) {
                    skeletonAry[i][j] = zeroFramedAry[i][j];
                } else {
                    skeletonAry[i][j] = 0;
                }

                // Updating newMin and newMax
                if (skeletonAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (skeletonAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
        }
    }

    void skeletonImgCompression(BufferedWriter outFile) throws IOException {
        writeHeader(outFile);
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {
                int pixelVal = skeletonAry[i][j];
                if (pixelVal > 0) {
```

```
                    outFile.write((i - rowFrameSize) + " " + (j - colFrameSize) + " " +
pixelVal + "\n");
                }
            }
        }
    }

    void expansionPassOne() {
        newMin = 9999;
        newMax = 0;
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {

                if (zeroFramedAry[i][j] == 0) {
                    int a = zeroFramedAry[i - 1][j - 1];
                    int b = zeroFramedAry[i - 1][j];
                    int c = zeroFramedAry[i - 1][j + 1];
                    int d = zeroFramedAry[i][j - 1];
                    int e = zeroFramedAry[i][j + 1];
                    int f = zeroFramedAry[i + 1][j - 1];
                    int g = zeroFramedAry[i + 1][j];
                    int h = zeroFramedAry[i + 1][j + 1];

                    int max = findMax(a, b, c, d, e, f, g, h) - 1;

                    if (zeroFramedAry[i][j] < max) {
                        zeroFramedAry[i][j] = max;
                    }
                }
                // Updating newMin and newMax
                if (zeroFramedAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (zeroFramedAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
        }
    }

    void expansionPassTwo() {
        newMin = 9999;
        newMax = 0;
        for (int i = numImgRows + rowFrameSize - 1; i >= rowFrameSize; i--) {
            for (int j = numImgCols + colFrameSize - 1; j >= colFrameSize; j--) {
                int a = zeroFramedAry[i - 1][j - 1];
                int b = zeroFramedAry[i - 1][j];
                int c = zeroFramedAry[i - 1][j + 1];
                int d = zeroFramedAry[i][j - 1];
                int e = zeroFramedAry[i][j + 1];
```

```
                int f = zeroFramedAry[i + 1][j - 1];
                int g = zeroFramedAry[i + 1][j];
                int h = zeroFramedAry[i + 1][j + 1];

                int max = findMax(a, b, c, d, e, f, g, h, zeroFramedAry[i][j]);

                if (zeroFramedAry[i][j] < max) {
                    zeroFramedAry[i][j] = max - 1;
                }

                // Updating newMin and newMax
                if (zeroFramedAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (zeroFramedAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
        }
    }

    void threshold(int thrVal, BufferedWriter outFile) throws IOException {
        newMin = 9999;
        newMax = 0;
        outFile.write(numImgRows + " " + numImgCols + " " + 0 + " " + 1 + "\n");
        for (int i = rowFrameSize; i < numImgRows + rowFrameSize; i++) {
            for (int j = colFrameSize; j < numImgCols + colFrameSize; j++) {

                if (zeroFramedAry[i][j] >= thrVal) {
                    outFile.write(1 + " ");
                } else {
                    outFile.write(0 + " ");
                }
                // Updating newMin and newMax
                if (zeroFramedAry[i][j] > newMax) {
                    newMax = zeroFramedAry[i][j];
                }
                if (zeroFramedAry[i][j] < newMin) {
                    newMin = zeroFramedAry[i][j];
                }
            }
            outFile.write("\n");

        }
    }

    static String splitAndAddExtension(String originalString, String extension) {
        String[] parts = originalString.split("\\.");
        String returnVal = parts[0] + "_" + extension + "." + parts[1];
        return returnVal;
```

```
    }

    public static void main(String[] args) throws IOException {
        String inputName1 = args[0]+".txt";
        FileReader inputReader1 = null;
        BufferedReader buffInReader1 = null;
        Scanner imgFile = null;

        String outputName1 = args[1]+".txt";
        FileWriter outputWriter1 = null;
        BufferedWriter prettyPrintFile = null;

        String outputName2 = args[0]+"_skeleton.txt";
        FileWriter outputWriter2 = null;
        BufferedWriter compressedImg = null;

        String outputName3 = args[0]+"_decompressed.txt";
        FileWriter outputWriter3 = null;
        BufferedWriter expandedImg = null;

        String outputName4 = args[2]+".txt";
        FileWriter outputWriter4 = null;
        BufferedWriter expansionPrettyPrint = null;

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

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

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

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

            outputWriter4 = new FileWriter(outputName4);
            expansionPrettyPrint = new BufferedWriter(outputWriter4);

            // Compression steps begins
            ImageProcessing d = new ImageProcessing(imgFile);
            d.loadImg(imgFile);
            d.passOne8Connectedness();
            prettyPrintFile.write("Result of: Pass One 8 connectness Distance
Transform\n");
            d.prettyPrint(prettyPrintFile);

            d.passTwo8Connectedness();
```

```
            prettyPrintFile.write("\nResult of: Pass Two 8 connectness Distance
Transform\n");
            d.prettyPrint(prettyPrintFile);

            d.localMaxima();
            prettyPrintFile.write("\nResult of: Local Maxima Operation\n");
            d.prettyPrint(prettyPrintFile, d.skeletonAry);

            d.skeletonImgCompression(compressedImg);

            if (compressedImg != null)
                compressedImg.close();

            // Decompression Steps Begins
            FileReader inputReader2 = new FileReader(outputName2);
            BufferedReader buffInReader2 = new BufferedReader(inputReader2);
            Scanner compressedInputFile = new Scanner(buffInReader2);

            // Reinitializing everything
            ImageProcessing d2 = new ImageProcessing(compressedInputFile);
            d2.loadTriplets(compressedInputFile);
            d2.expansionPassOne();
            expansionPrettyPrint.write("\nResult of: Expansion Pass 1\n");
            d2.prettyPrint(expansionPrettyPrint);
            d2.expansionPassTwo();
            expansionPrettyPrint.write("\nResult of: Expansion Pass 2\n");
            d2.prettyPrint(expansionPrettyPrint);
            d2.threshold(1, expandedImg);

            if (compressedInputFile != null) {
                compressedInputFile.close();
            }
        } finally {
            if (imgFile != null)
                imgFile.close();
            if (prettyPrintFile != null)
                prettyPrintFile.close();
            if (expandedImg != null)
                expandedImg.close();
            if (expansionPrettyPrint != null)
                expansionPrettyPrint.close();

        }
    }
}
```

Outputs

Data_1:

Original Image

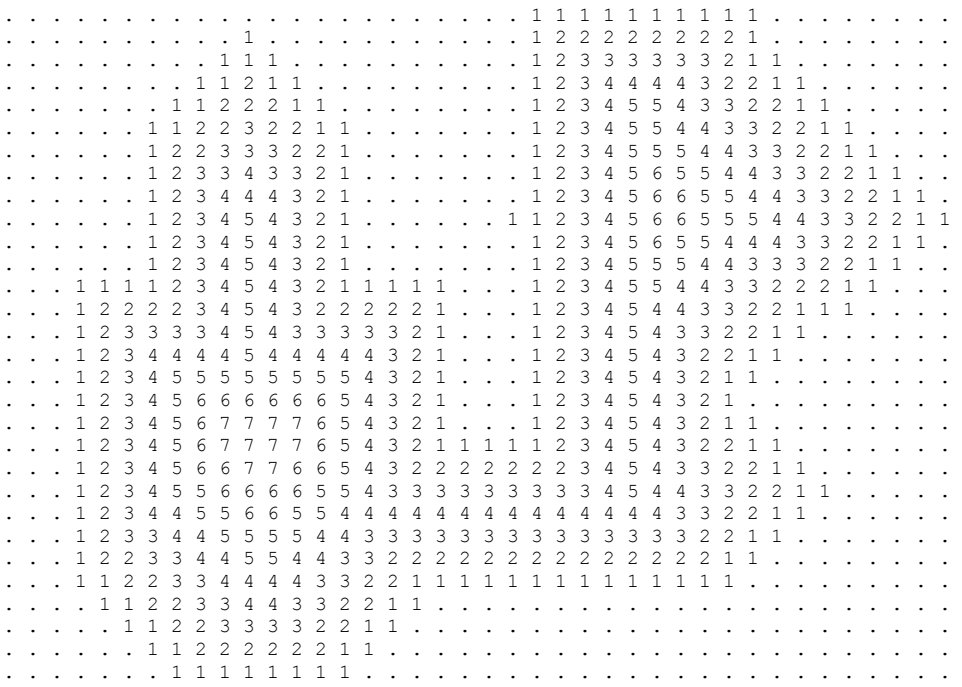
```
30 40 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Result of: Pass One 8 connectedness Distance Transform

```
30 40 0 10
. . . . . . . . . . . . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . . . . . . . . 1 . . . . . . . . . . . 1 2 2 2 2 2 2 2 2 1 . . . . . . . .
. . . . . . . . . 1 1 1 . . . . . . . . . . 1 2 3 3 3 3 3 3 2 1 1 . . . . . . .
. . . . . . . . 1 1 2 1 1 . . . . . . . . . 1 2 3 4 4 4 4 3 2 2 1 1 . . . . . .
. . . . . . . 1 1 2 2 2 1 1 . . . . . . . . 1 2 3 4 5 5 4 3 3 2 2 1 1 . . . . .
. . . . . . 1 1 2 2 3 2 2 1 1 . . . . . . . 1 2 3 4 5 5 4 4 3 3 2 2 1 1 . . . .
. . . . . . 1 2 2 3 3 3 2 2 1 . . . . . . . 1 2 3 4 5 5 5 4 4 3 3 2 2 1 1 . . .
. . . . . . 1 2 3 3 4 3 3 2 1 . . . . . . . 1 2 3 4 5 6 5 5 4 4 3 3 2 2 1 1 . .
. . . . . . 1 2 3 4 4 4 3 2 1 . . . . . . . 1 2 3 4 5 6 6 5 5 4 4 3 3 2 2 1 1 .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . 1 1 2 3 4 5 6 6 6 5 5 4 4 3 3 2 2 1 1
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 6 7 6 6 5 5 4 4 3 3 2 2 .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 6 7 7 6 6 5 5 4 4 3 3 . .
. . . 1 1 1 1 2 3 4 5 4 3 2 1 1 1 1 1 . . . 1 2 3 4 5 6 7 7 7 6 6 5 5 4 4 . . .
. . . 1 2 2 2 2 3 4 5 4 3 2 2 2 2 2 1 . . . 1 2 3 4 5 6 7 8 7 7 6 6 5 5 . . . .
. . . 1 2 3 3 3 3 4 5 4 3 3 3 3 3 2 1 . . . 1 2 3 4 5 6 7 8 8 7 7 6 . . . . . .
. . . 1 2 3 4 4 4 4 5 4 4 4 4 4 3 2 1 . . . 1 2 3 4 5 6 7 8 8 8 7 . . . . . . .
. . . 1 2 3 4 5 5 5 5 5 5 5 5 4 3 2 1 . . . 1 2 3 4 5 6 7 8 9 8 . . . . . . . .
. . . 1 2 3 4 5 6 6 6 6 6 6 5 4 3 2 1 . . . 1 2 3 4 5 6 7 8 9 . . . . . . . . .
. . . 1 2 3 4 5 6 7 7 7 7 6 5 4 3 2 1 . . . 1 2 3 4 5 6 7 8 1 1 . . . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1 1 1 1 1 2 3 4 5 6 7 2 2 1 1 . . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 2 2 2 2 2 2 3 4 5 6 3 3 2 2 1 1 . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 5 4 3 3 3 3 3 3 3 3 3 4 5 4 4 3 3 2 2 1 1 . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 5 4 4 4 4 4 4 4 4 4 4 4 5 5 4 4 3 3 2 2 . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 3 3 . . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 5 4 4 . . . . . . . .
. . . 1 2 3 4 5 6 7 8 8 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 6 6 5 5 . . . . . . . . .
. . . . 1 2 3 4 5 6 7 8 8 8 8 8 8 8 . . . . . . . . . . . . . . . . . . . . . .
. . . . . 1 2 3 4 5 6 7 8 9 9 9 9 . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . 1 2 3 4 5 6 7 8 9 10 . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 2 3 4 5 6 7 8 . . . . . . . . . . . . . . . . . . . . . . . . .
```

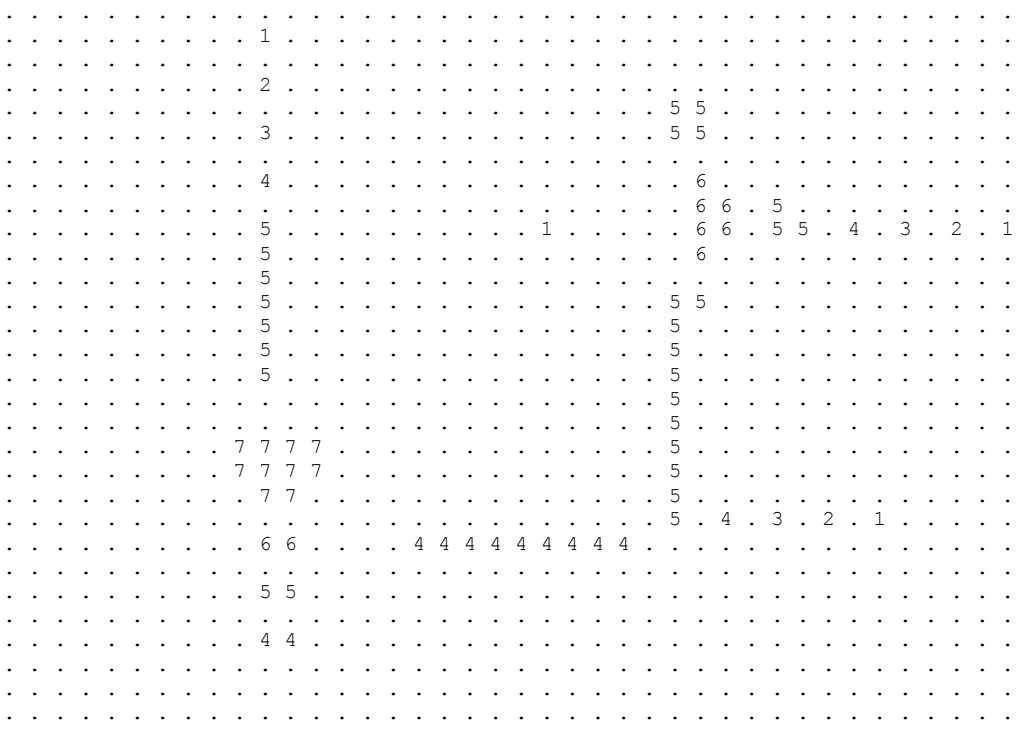
Result of: Pass Two 8 connectness Distance Transform

30 40 0 7



Result of: Local Maxima Operation

30 40 0 7



Skeleton/ Compressed Image via Distance Transform

```
30 40 0 7
1 10 1
3 10 2
4 26 5
4 27 5
5 10 3
5 26 5
5 27 5
7 10 4
7 27 6
8 27 6
8 28 6
8 30 5
9 10 5
9 21 1
9 27 6
9 28 6
9 30 5
9 31 5
9 33 4
9 35 3
9 37 2
9 39 1
10 10 5
10 27 6
11 10 5
12 10 5
12 26 5
12 27 5
13 10 5
13 26 5
14 10 5
14 26 5
15 10 5
15 26 5
16 26 5
17 26 5
18 9 7
18 10 7
18 11 7
18 12 7
18 26 5
19 9 7
19 10 7
19 11 7
19 12 7
19 26 5
20 10 7
20 11 7
20 26 5
21 26 5
21 28 4
21 30 3
21 32 2
21 34 1
22 10 6
22 11 6
22 16 4
22 17 4
22 18 4
22 19 4
22 20 4
22 21 4
22 22 4
22 23 4
22 24 4
24 10 5
24 11 5
26 10 4
26 11 4
```

Result of: Expansion Pass 1

```
30 40 0 7
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . 1 2 1 . . . . . . . . . . . . . 4 4 4 4 3 2 1 . . . . . . . .
. . . . . . . . . 2 2 2 1 . . . . . . . . . . . 3 4 5 5 4 3 2 1 . . . . . . . .
. . . . . . . . 1 2 3 2 1 . . . . . . . . . . 2 3 4 5 5 4 3 2 1 . . . . . . . .
. . . . . . . . 1 3 3 3 2 1 . . . . . . . . 1 2 3 4 5 5 5 4 3 2 1 . . . . . . .
. . . . . . . . 2 3 4 3 2 1 . . . . . . . . 1 2 3 4 5 6 5 5 4 4 3 2 1 . . . . .
. . . . . . . 1 2 4 4 4 3 2 1 . . . . . . . 1 2 3 4 5 6 6 5 5 4 4 3 3 2 2 1 1 .
. . . . . . . 1 3 4 5 4 3 2 1 . . . . . . 1 1 2 3 4 5 6 6 5 5 5 4 4 3 3 2 2 1 1
. . . . . . . 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 6 5 5 4 4 4 3 3 2 2 1 1 .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 5 5 4 4 3 3 3 2 2 1 1 . .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 5 4 4 3 3 2 2 2 1 1 . . .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 4 4 3 3 2 2 1 1 1 . . . .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 4 3 3 2 2 1 1 . . . . . .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 4 3 2 2 1 1 . . . . . . .
. . . . . . 1 2 3 4 4 4 3 2 1 . . . . . . . 1 2 3 4 5 4 3 2 1 1 . . . . . . . .
. . . . . . 1 2 6 6 6 6 6 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 . . . . . . . . .
. . . . . . 1 5 6 7 7 7 7 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 . . . . . . . . .
. . . . . . 4 5 6 7 7 7 7 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 . . . . . . . . .
. . . . . 3 4 5 6 6 7 7 6 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 3 2 2 1 1 . . . . . .
. . . . 2 3 4 5 5 6 6 6 6 5 5 4 3 3 3 3 3 3 3 3 3 4 5 4 4 3 3 2 2 1 1 . . . . .
. . . 1 2 3 4 4 5 5 6 6 5 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 2 2 1 1 . . . . . .
. . . 1 2 3 3 4 4 5 5 5 5 4 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 1 . . . . . . .
. . . 1 2 2 3 3 4 4 5 5 4 4 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 . . . . . . . .
. . . 1 1 2 2 3 3 4 4 4 4 3 3 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 2 2 3 3 4 4 3 3 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . .
. . . . . 1 1 2 2 3 3 3 3 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . 1 1 2 2 2 2 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . .
```

Result of: Expansion Pass 2

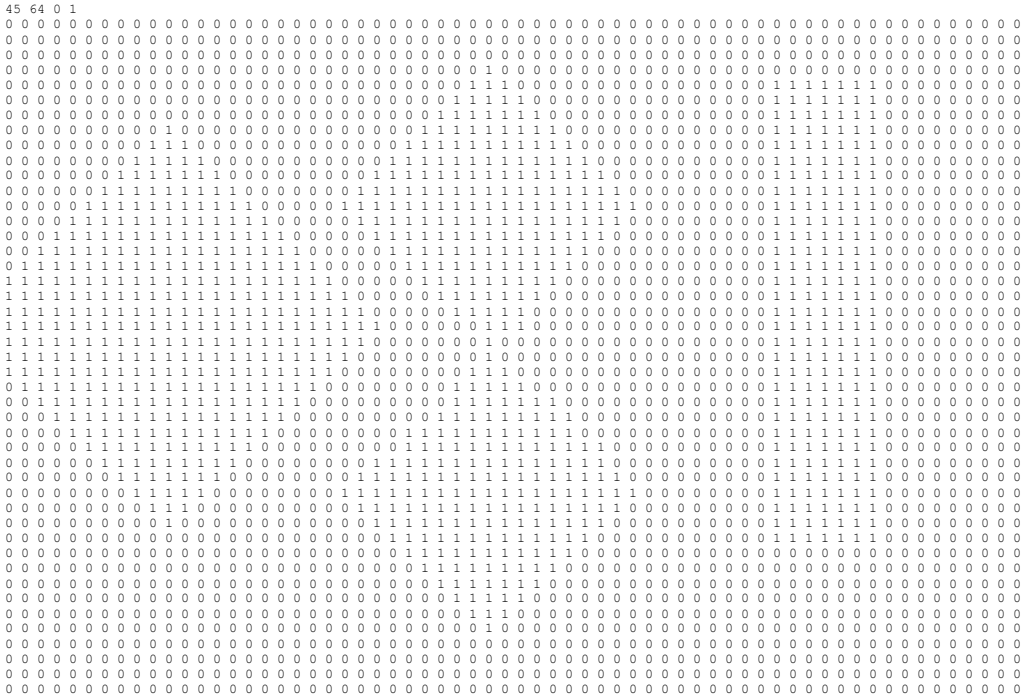
```
30 40 0 7
. . . . . . . . . . . . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . . . . . . . . 1 . . . . . . . . . . . 1 2 2 2 2 2 2 2 2 1 . . . . . . . .
. . . . . . . . . 1 1 1 . . . . . . . . . . 1 2 3 3 3 3 3 3 2 1 1 . . . . . . .
. . . . . . . . 1 1 2 1 1 . . . . . . . . . 1 2 3 4 4 4 4 3 2 2 1 1 . . . . . .
. . . . . . . 1 1 2 2 2 1 1 . . . . . . . . 1 2 3 4 5 5 4 3 3 2 2 1 1 . . . . .
. . . . . . 1 1 2 2 3 2 2 1 1 . . . . . . . 1 2 3 4 5 5 4 4 3 3 2 2 1 1 . . . .
. . . . . . 1 2 2 3 3 3 2 2 1 . . . . . . . 1 2 3 4 5 5 5 4 4 3 3 2 2 1 1 . . .
. . . . . . 1 2 3 3 4 3 3 2 1 . . . . . . . 1 2 3 4 5 6 5 5 4 4 3 3 2 2 1 1 . .
. . . . . . 1 2 3 4 4 4 3 2 1 . . . . . . . 1 2 3 4 5 6 6 5 5 4 4 3 3 2 2 1 1 .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . 1 1 2 3 4 5 6 6 5 5 5 4 4 3 3 2 2 1 1
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 6 5 5 4 4 4 3 3 2 2 1 1 .
. . . . . . 1 2 3 4 5 4 3 2 1 . . . . . . . 1 2 3 4 5 5 5 4 4 3 3 3 2 2 1 1 . .
. . . 1 1 1 1 2 3 4 5 4 3 2 1 1 1 1 1 . . . 1 2 3 4 5 5 4 4 3 3 2 2 2 1 1 . . .
. . . 1 2 2 2 2 3 4 5 4 3 2 2 2 2 2 1 . . . 1 2 3 4 5 4 4 3 3 2 2 1 1 1 . . . .
. . . 1 2 3 3 3 3 4 5 4 3 3 3 3 3 2 1 . . . 1 2 3 4 5 4 3 3 2 2 1 1 . . . . . .
. . . 1 2 3 4 4 4 4 5 4 4 4 4 4 3 2 1 . . . 1 2 3 4 5 4 3 2 2 1 1 . . . . . . .
. . . 1 2 3 4 5 5 5 5 5 5 5 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 1 . . . . . . . .
. . . 1 2 3 4 5 6 6 6 6 6 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 . . . . . . . . .
. . . 1 2 3 4 5 6 7 7 7 7 6 5 4 3 2 1 . . . 1 2 3 4 5 4 3 2 1 1 . . . . . . . .
. . . 1 2 3 4 5 6 7 7 7 7 6 5 4 3 2 1 1 1 1 1 2 3 4 5 4 3 2 2 1 1 . . . . . . .
. . . 1 2 3 4 5 6 6 7 7 6 6 5 4 3 2 2 2 2 2 2 2 3 4 5 4 3 3 2 2 1 1 . . . . . .
. . . 1 2 3 4 5 5 6 6 6 6 5 5 4 3 3 3 3 3 3 3 3 3 4 5 4 4 3 3 2 2 1 1 . . . . .
. . . 1 2 3 4 4 5 5 6 6 5 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 2 2 1 1 . . . . . .
. . . 1 2 3 3 4 4 5 5 5 5 4 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 1 . . . . . . .
. . . 1 2 2 3 3 4 4 5 5 4 4 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 . . . . . . . .
. . . 1 1 2 2 3 3 4 4 4 4 3 3 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 2 2 3 3 4 4 3 3 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . .
. . . . . 1 1 2 2 3 3 3 3 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . 1 1 2 2 2 2 2 2 1 1 . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . .
```

Decompressed Image

```
30 40 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0
```

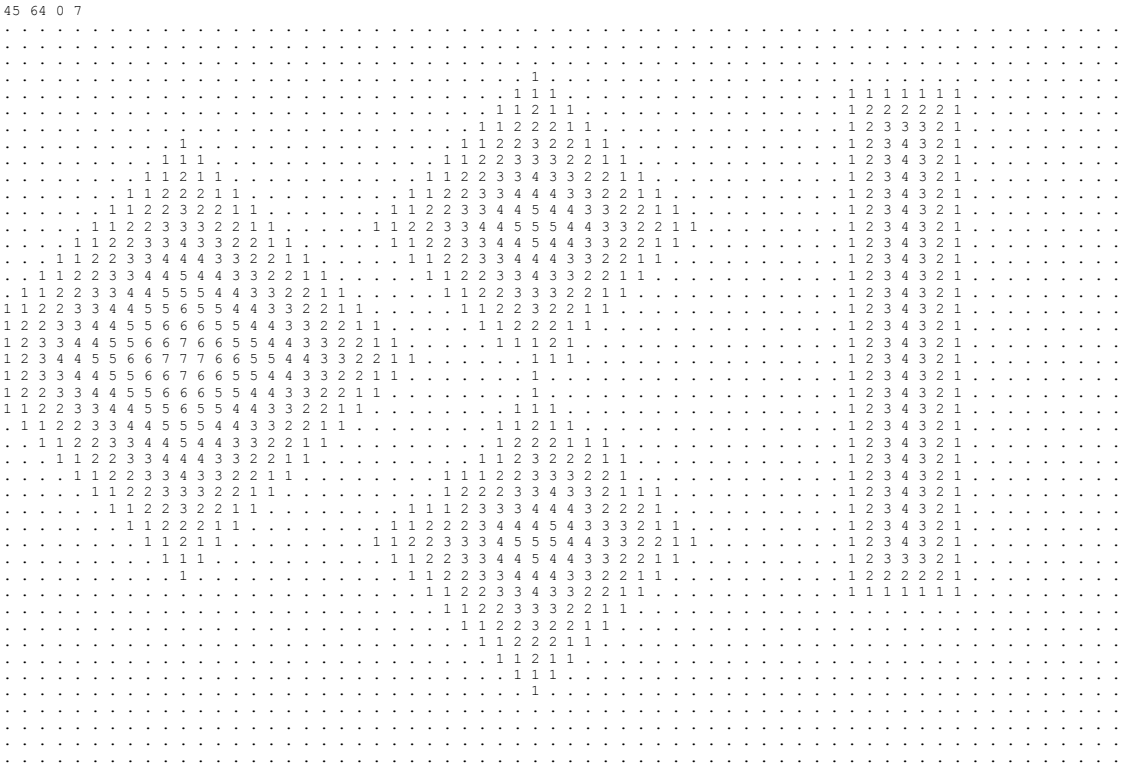
Data_2:

Original Image

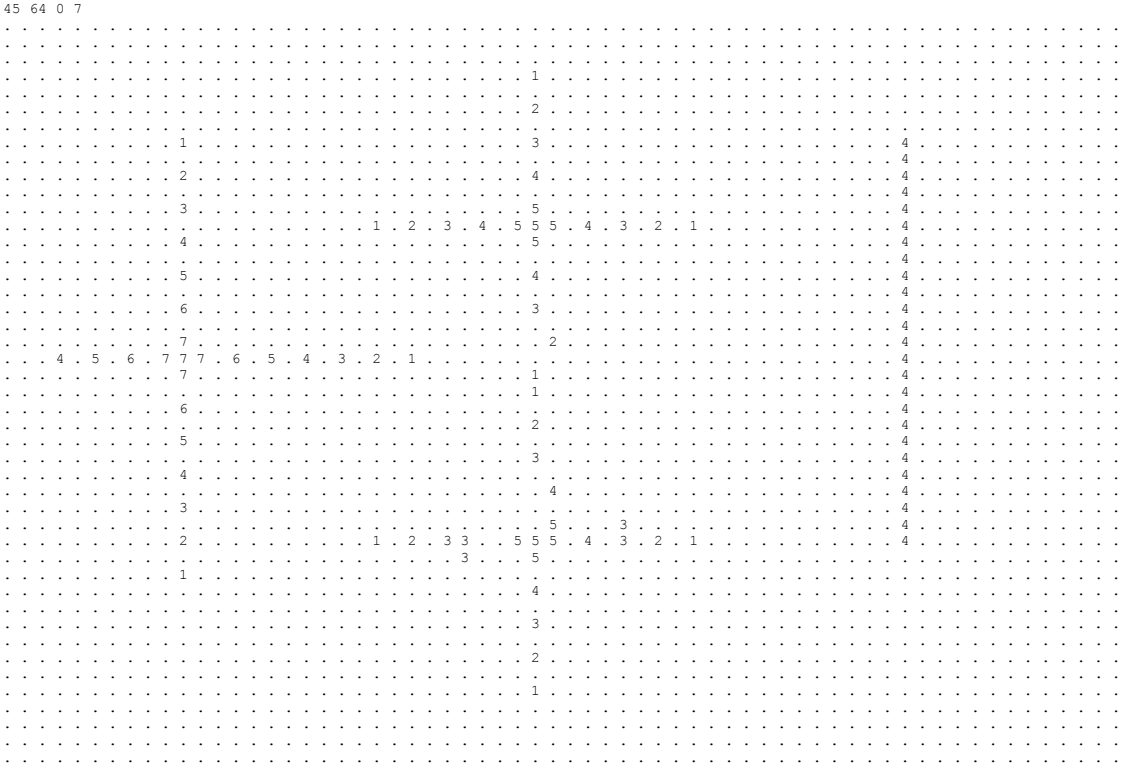


Result of: Pass One 8 connectedness Distance Transform

Result of: Pass Two 8 connectedness Distance Transform



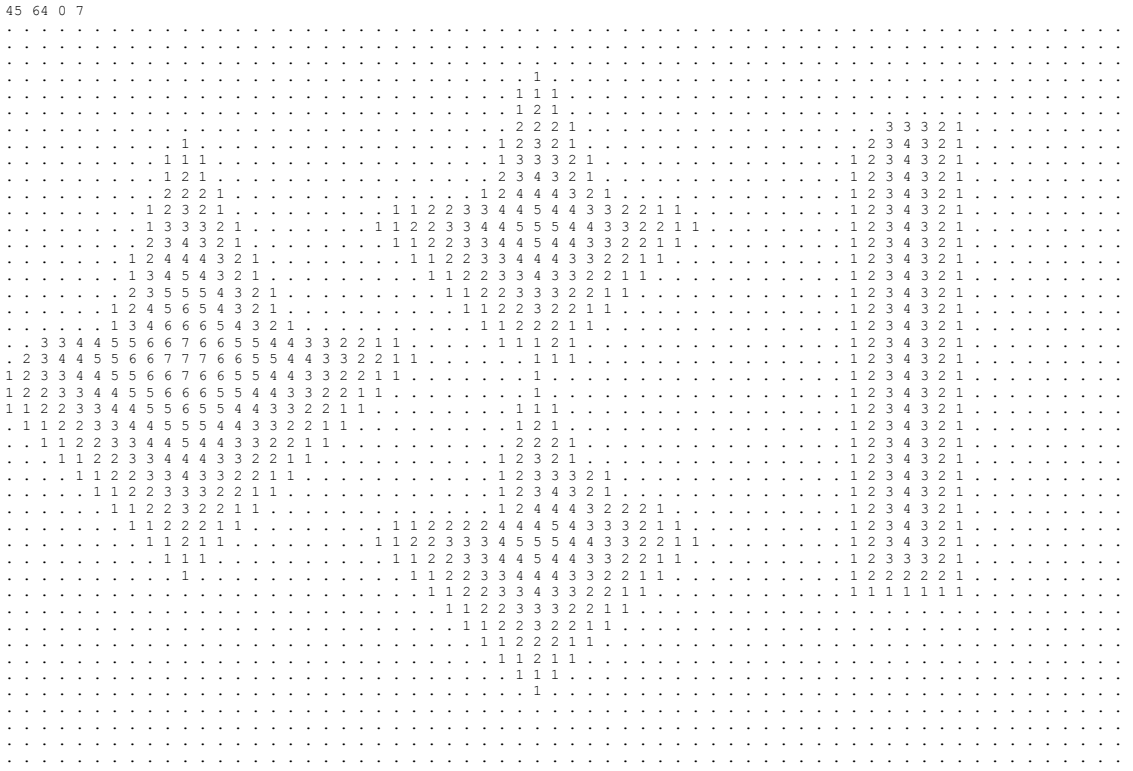
Result of: Local Maxima Operation



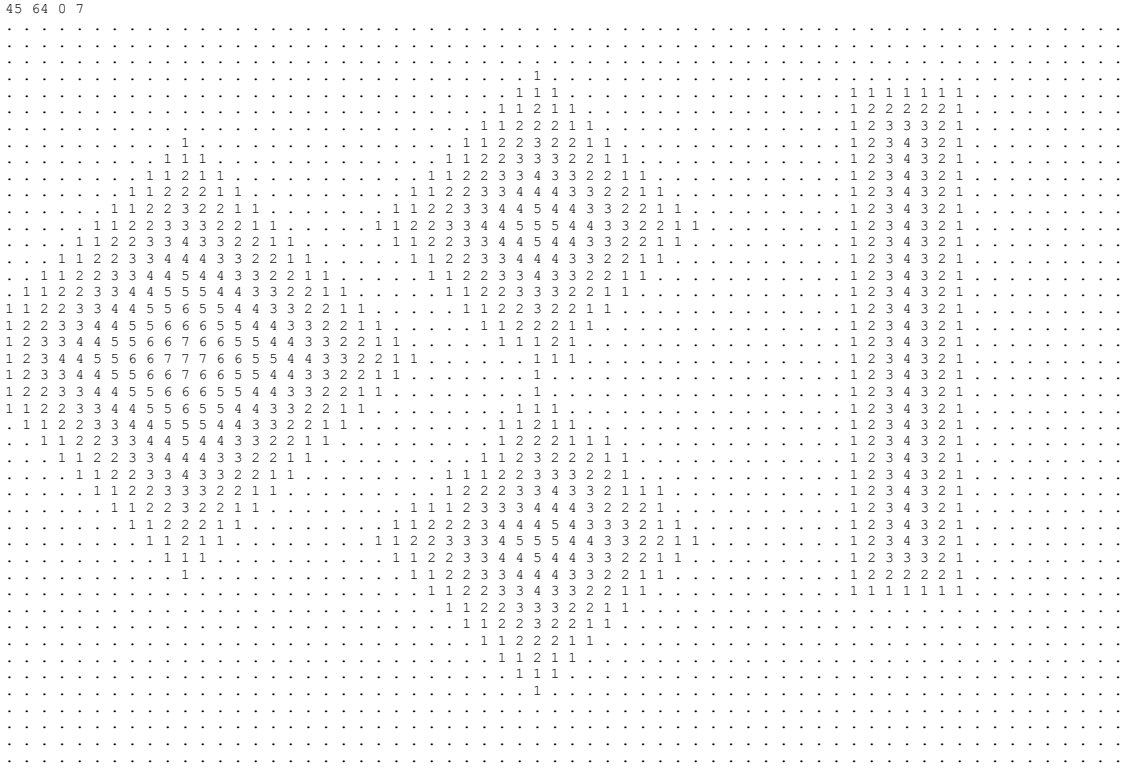
Skeleton/Compressed Image via Distance Transform

45 64 0 7
3 30 1
5 30 2
7 10 1
7 30 3
7 51 4
8 51 4
9 10 2
9 30 4
9 51 4
10 51 4
11 10 3
11 30 5
11 51 4
12 21 1
12 23 2
12 25 3
12 27 4
12 29 5
12 30 5
12 31 5
12 33 4
12 35 3
12 37 2
12 39 1
12 51 4
13 10 4
13 30 5
13 51 4
14 51 4
15 10 5
15 30 4
15 51 4
16 51 4
17 10 6
17 30 3
17 51 4
18 51 4
19 10 7
19 31 2
19 51 4
20 3 4
20 5 5
20 7 6
20 9 7
20 10 7
20 11 7
20 13 6
20 15 5
20 17 4
20 19 3
20 21 2
20 23 1
20 51 4
21 10 7
21 30 1
21 51 4
22 30 1
22 51 4
23 10 6
23 51 4
24 30 2
24 51 4
25 10 5
25 51 4
26 30 3
26 51 4
27 10 4
27 51 4
28 31 4
28 51 4
29 10 3
29 51 4
30 31 5
30 35 3
30 51 4
31 10 2
31 21 1
31 23 2
31 25 3
31 26 3
31 29 5
31 30 5
31 31 5
31 33 4
31 35 3
31 37 2
31 39 1
31 51 4
32 26 3
32 30 5
33 10 1
34 30 4
36 30 3
38 30 2
40 30 1

Result of: Expansion Pass 1



Result of: Expansion Pass 2



Decompressed Image