Computer Vision

Project 5: Image Compression via Distance Transform

Student: Andrew Alleyne

3/28/2021

Algorithms steps for Image Compression via Distance Transform

Distance transform Algorithm

Pass 1

1. Scan given image left to right and top to bottom.
2. If the current pixel is greater than 0. Look at current pixels surrounding neighbors.
3. Apply the given template and find the minimum of each neighbor.
4. Repeat until all pixels are processed.

Pass 2

1. Scan given image right to left and bottom to top.
2. If the current pixel is greater than 0. Look at all neighbors.
3. Calculate the min of each neighbor including the current pixel and apply the given template.

Local Maxima Operation

1. Scan given image left to right and top to bottom.
2. Check if current pixel is a local maximum.
    a. Scan each neighbor and if they are lesser than the current pixel replaces the current pixels value with 0.
    b. However, if they are greater than or equal write the pixel to the skeleton array.

Skeleton Image Compression

1. Using the Local Maxima Operation. Scan the image left to right and top the bottom.
2. If the current pixel is greater than 0.
3. Display the row, col, and value.

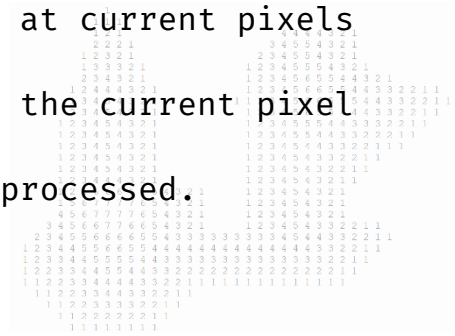
Results of Lossless compress

30 40 0 1

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

Expansion Pass 1

1. Scan given image left to right and top to bottom.
2. If the current pixel is equal to 0. Look at current pixels surrounding neighbors.
3. Calculate the max of each neighbor using the current pixel and apply the given template.
4. Repeat these steps until all pixels are processed.



Expansion Pass 2

1. Scan given image left to right and top to bottom.
2. Look at neighbors of the current pixel.
3. Calculate the max of each neighbor using the current pixel and apply the given template.
4. If the max is found for the given neighbors, *current Pixel = max − 1.*
5. Repeat these steps until all pixels are processed.

Source Code

Driver Code

```
/*
Andrew Alleyne
Project 5 - Image Compression via Distance Transform

*/

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        if (args.length < 3) {
            System.out.println("Provide more arguments");
        }

        String image = args[0];
        String outFile1 = args[1];
        String outFile2 = args[2];
        String skeletonFile = args[3] + "_skeleton.txt";
        String decompressFile = args[4] + "_decompressed.txt";

        int zeroFramedAry[][];
        int skeletonAry[][];

        int numRows = 0;
        int numCols = 0;
        int maxVal = 0;
        int minVal = 0;

        FileWriter of;
        FileWriter of2;
        FileWriter of3;
        FileWriter of4;

        Scanner scanner;
        Scanner sScanner;


        try {

            /*Does not work if you already have a named file in your
directory. These files must be created
             * at the same time. */
            File sf = new File(skeletonFile);
            File df = new File(decompressFile);


            if (sf.createNewFile() && df.createNewFile()) {
                System.out.println(sf.getName() + " has been created!");
```

```
                System.out.println(df.getName() + " has been created!");
            } else {
                System.out.println(sf.getName() + " may already exist!");
                System.out.println(df.getName() + " may already exist!");

            }


            File file = new File(image);
            File sFile = new File(skeletonFile);

            scanner = new Scanner(file);
            sScanner = new Scanner(sFile);

            if (scanner.hasNextInt()) numRows = scanner.nextInt();
            if (scanner.hasNextInt()) numCols = scanner.nextInt();
            if (scanner.hasNextInt()) minVal = scanner.nextInt();
            if (scanner.hasNextInt()) maxVal = scanner.nextInt();


            of = new FileWriter(outFile1);
            of2 = new FileWriter(outFile2);
            of3 = new FileWriter(skeletonFile);
            of4 = new FileWriter(decompressFile);


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


            imageProcessing ip = new imageProcessing(numRows, numCols,
minVal, maxVal);


            //Zero frame array to preform operations on binary image
            ip.setZero(zeroFramedAry);
            ip.setZero(skeletonAry);
            ip.loadImage(scanner, zeroFramedAry);

            ip.compute8DistancePass1(zeroFramedAry, of);
            ip.reformatPrettyPrint(zeroFramedAry, of);

            ip.compute8DistancePass2(zeroFramedAry, of);
            ip.reformatPrettyPrint(zeroFramedAry, of);


            ip.skeletonExtraction(zeroFramedAry, skeletonAry, of);
            ip.reformatPrettyPrint(skeletonAry, of);
            ip.extractLocalMaxima(skeletonAry, of3);


            ip.setZero(zeroFramedAry);
            ip.load(sScanner, zeroFramedAry);

            ip.skeletonExpansionPass1(zeroFramedAry, of2);
            ip.reformatPrettyPrint(zeroFramedAry, of2);
```

```
            ip.skeletonExpansionPass2(zeroFramedAry, of2);
            ip.reformatPrettyPrint(zeroFramedAry, of2);


            //ary2File (zeroFramedAry, decompressFile)
            ip.thresholdDecompression(zeroFramedAry, of4);

            of.close();
            of2.close();
            of3.close();
            of4.close();

        } catch (FileNotFoundException e) {
            System.out.println("Could not find the specified file!");
            e.printStackTrace();
        } catch (IOException e) {
            System.out.println("Failed I/O operation!");
            e.printStackTrace();
        }
    }
}
```

Image Processing Class

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class imageProcessing {

    int numRows;
    int numCols;
    int maxVal;
    int minVal;
    int newMinVal;
    int newMaxVal;
    int neigh[];


    imageProcessing(int numRows, int numCols, int minVal, int maxVal) {

        this.numRows = numRows;
        this.numCols = numCols;
        this.minVal = minVal;
        this.maxVal = maxVal;


        neigh = new int[10];


    }

    void setZero(int array[][]) {
        for (int i = 0; i < numRows + 2; i++) {
            for (int j = 0; j < numCols + 2; j++) {
                array[i][j] = 0;
            }
        }
    }

    void loadImage(Scanner scanner, int zeroFramedAry[][]) {
        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {
                if (scanner.hasNextInt()) zeroFramedAry[i][j] =
scanner.nextInt();
            }
        }
    }

    void reformatPrettyPrint(int array[][], FileWriter of) throws IOException
{
        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {
                int number = array[i][j];

                if (number == 0) {
                    of.write("  ");

                } else {
```

```
                    of.write(number + " ");

                }


            }
            of.write("\n");
        }
        of.flush();
        of.write("\n");
    }

    void compute8DistancePass1(int zeroFramedAry[][], FileWriter of) throws
IOException {
        of.write("Results of 1st Distance Transform Pass:  \n");


        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {

                if (zeroFramedAry[i][j] > 0) {
                    zeroFramedAry[i][j] = neighborMin(zeroFramedAry, i, j,
1);
                }
            }
        }
        of.flush();
    }

    void compute8DistancePass2(int zeroFramedAry[][], FileWriter of) throws
IOException {
        of.write("Results of 2nd Distance Transform Pass:  \n");

        for (int i = numRows; i >= 1; i--) {
            for (int j = numCols; j >= 1; j--) {

                if (zeroFramedAry[i][j] > 0) {
                    zeroFramedAry[i][j] = neighborMin(zeroFramedAry, i, j,
2);
                }
            }
        }
        of.flush();
    }

    int neighborMin(int zeroFramedAry[][], int i, int j, int pass) {
        int firstMin = 0;
        if (pass == 1) {
            neigh[0] = zeroFramedAry[i - 1][j - 1] + 1;
            neigh[1] = zeroFramedAry[i - 1][j] + 1;
            neigh[2] = zeroFramedAry[i - 1][j + 1] + 1;
            neigh[3] = zeroFramedAry[i][j - 1] + 1;
            firstMin = neigh[0];

            for (int k = 0; k < 4; k++) {

                if (firstMin > neigh[k]) {
```

```
                    firstMin = neigh[k];
                }
            }

        }

        if (pass == 2) {

            neigh[0] = zeroFramedAry[i + 1][j - 1] + 1;
            neigh[1] = zeroFramedAry[i + 1][j] + 1;
            neigh[2] = zeroFramedAry[i + 1][j + 1] + 1;
            neigh[3] = zeroFramedAry[i][j + 1] + 1;
            neigh[4] = zeroFramedAry[i][j];
            firstMin = neigh[0];
            for (int k = 0; k <= 4; k++) {
                if (firstMin > neigh[k]) firstMin = neigh[k];
            }


        }

        return firstMin;
    }

    void skeletonExtraction(int zeroFramedAry[][], int skeletonAry[][],
FileWriter of) throws IOException {
        of.write("Results of Local Maxima: \n");

        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {
                if (zeroFramedAry[i][j] > 0) {
                    computeLocalMaxima(zeroFramedAry, skeletonAry, i, j);
                }
            }
        }


        of.flush();
    }

    void computeLocalMaxima(int zeroFramedAry[][], int skeletonAry[][], int
i, int j) {


        neigh[0] = zeroFramedAry[i - 1][j - 1];
        neigh[1] = zeroFramedAry[i - 1][j];
        neigh[2] = zeroFramedAry[i - 1][j + 1];
        neigh[3] = zeroFramedAry[i][j - 1];
        neigh[4] = zeroFramedAry[i][j + 1];
        neigh[5] = zeroFramedAry[i + 1][j - 1];
        neigh[6] = zeroFramedAry[i + 1][j];
        neigh[7] = zeroFramedAry[i + 1][j + 1];

        int max = 0;
        for (int k = 0; k < 8; k++) {
            if (zeroFramedAry[i][j] >= neigh[k]) {
                max = zeroFramedAry[i][j];
```

```
            } else {
                max = 0;
                break;
            }

        }
        skeletonAry[i][j] = max;
    }

    void extractLocalMaxima(int skeletonAry[][], FileWriter of3) {
        try {


            of3.write("Results of Lossless compression: \n \n");
            of3.write(numRows + " " + numCols + " " + minVal + " " + maxVal +
"\n");
            of3.write("\n");

            int max = skeletonAry[0][0];

            for (int i = 1; i <= numRows; i++) {
                for (int j = 1; j <= numCols; j++) {
                    if (skeletonAry[i][j] > 0) {
                        of3.write((i) + " " + (j) + " " + skeletonAry[i][j] +
"\n");

                        if (skeletonAry[i][j] > max) {
                            max = skeletonAry[i][j];
                        }
                    }
                }
            }


            newMaxVal = max;


            of3.flush();
        } catch (IOException e) {
            System.out.println("Could not extract local maxima!");
            e.printStackTrace();

        }
    }

    void load(Scanner sScanner, int zeroFramedAry[][]) {
        if (sScanner.hasNextLine()) {
            sScanner.nextLine();
        }

        if (sScanner.hasNextInt()) numRows = sScanner.nextInt();
        if (sScanner.hasNextInt()) numCols = sScanner.nextInt();
        if (sScanner.hasNextInt()) minVal = sScanner.nextInt();
        if (sScanner.hasNextInt()) maxVal = sScanner.nextInt();

        int Lc_rows = 0;
```

```
        int Lc_cols = 0;
        int Lc_value = 0;


        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {

                if (sScanner.hasNextInt()) Lc_rows = sScanner.nextInt();
                if (sScanner.hasNextInt()) Lc_cols = sScanner.nextInt();
                if (sScanner.hasNextInt()) Lc_value = sScanner.nextInt();

                zeroFramedAry[Lc_rows][Lc_cols] = Lc_value;

            }
        }
    }

    void skeletonExpansionPass1(int zeroFramedAry[][], FileWriter of2) throws
IOException {
        of2.write("Results of 1st Expansion pass: \n");
        of2.write("\n");

        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {
                if (zeroFramedAry[i][j] == 0) {
                    getMax(zeroFramedAry, i, j);
                }
            }
        }
    }

    void getMax(int zeroFramedAry[][], int i, int j) {

        neigh[0] = zeroFramedAry[i - 1][j - 1] - 1;
        neigh[1] = zeroFramedAry[i - 1][j] - 1;
        neigh[2] = zeroFramedAry[i - 1][j + 1] - 1;
        neigh[3] = zeroFramedAry[i][j - 1] - 1;
        neigh[4] = zeroFramedAry[i][j + 1] - 1;
        neigh[5] = zeroFramedAry[i + 1][j - 1] - 1;
        neigh[6] = zeroFramedAry[i + 1][j] - 1;
        neigh[7] = zeroFramedAry[i + 1][j + 1] - 1;

        int max;

        for (int k = 0; k < 8; k++) {
            max = neigh[k];

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

    void skeletonExpansionPass2(int zeroFramedAry[][], FileWriter of2) throws
IOException {
        of2.write("Results of 2nd Expansion pass: \n");
```

```
        for (int i = numRows; i >= 1; i--) {
            for (int j = numCols; j >= 1; j--) {
                getMax2(zeroFramedAry, i, j);

            }
        }


        newMinVal = zeroFramedAry[0][0];
        newMaxVal = zeroFramedAry[0][0];

        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {


                if (zeroFramedAry[i][j] > newMaxVal) {
                    newMaxVal = zeroFramedAry[i][j];
                } else {
                    newMinVal = zeroFramedAry[i][j];
                }
            }
        }

        of2.write(numRows + " " + numCols + " " + newMinVal + " " + newMaxVal
+ "\n");


    }

    void getMax2(int zeroFramedAry[][], int i, int j) {

        neigh[0] = zeroFramedAry[i - 1][j - 1];
        neigh[1] = zeroFramedAry[i - 1][j];
        neigh[2] = zeroFramedAry[i - 1][j + 1];
        neigh[3] = zeroFramedAry[i][j - 1];
        neigh[4] = zeroFramedAry[i][j + 1];
        neigh[5] = zeroFramedAry[i + 1][j - 1];
        neigh[6] = zeroFramedAry[i + 1][j];
        neigh[7] = zeroFramedAry[i + 1][j + 1];

        int max;


        for (int k = 0; k < 8; k++) {
            max = neigh[k];

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

        }
    }

    void thresholdDecompression(int array[][], FileWriter of) throws
IOException {
        of.write("Results of Threshold Decompression: \n\n");
```

```
        of.write(numRows + " " + numCols + " " + minVal + " " + maxVal +
"\n");


        for (int i = 1; i <= numRows; i++) {
            for (int j = 1; j <= numCols; j++) {

                if (array[i][j] >= 1) {
                    of.write("1 ");
                } else {
                    of.write("0 ");
                }


            }
            of.write("\n");
        }

        of.write("\n");
        of.flush();
    }


}
```

Output for program

Data set 1.

Binary 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
```

⇓

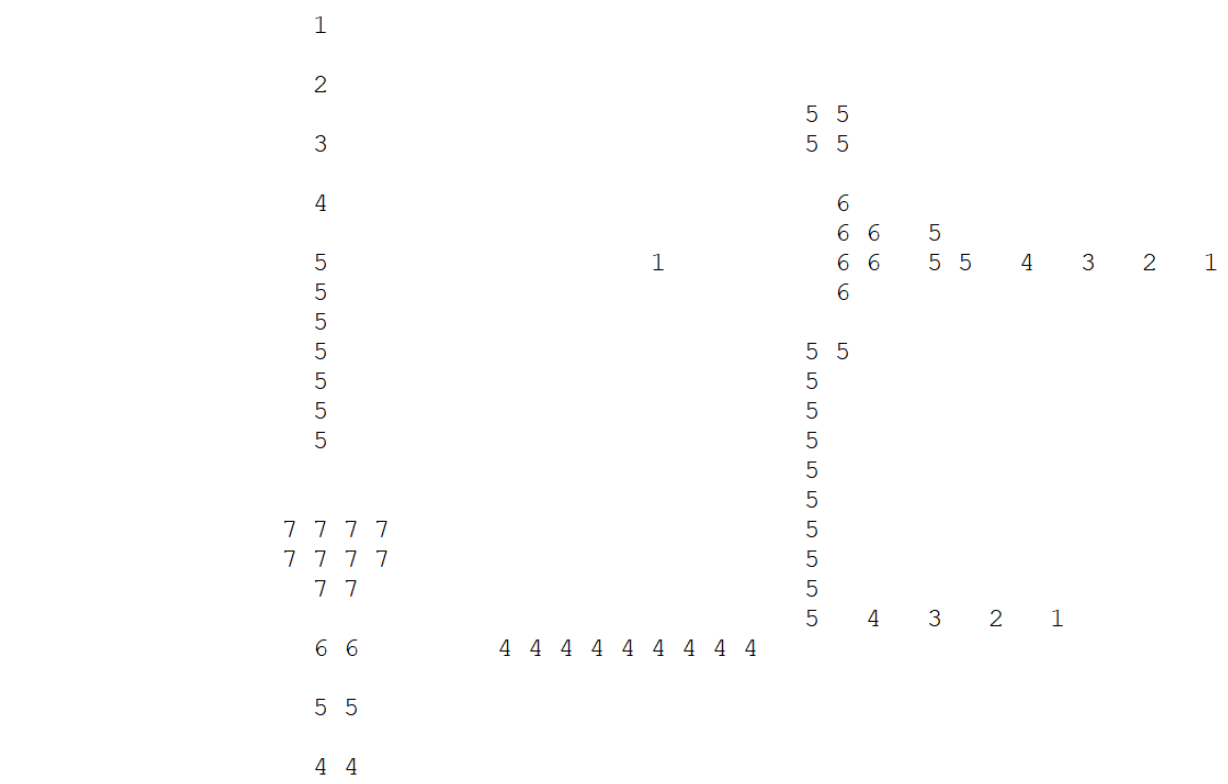
Distance Transform Pass 1

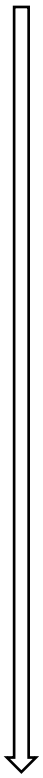
Results of 1st Distance Transform Pass:

```
                                                           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
        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
```

```
Results of 2nd Distance Transform Pass:
                                            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
```

Results of Local Maxima:

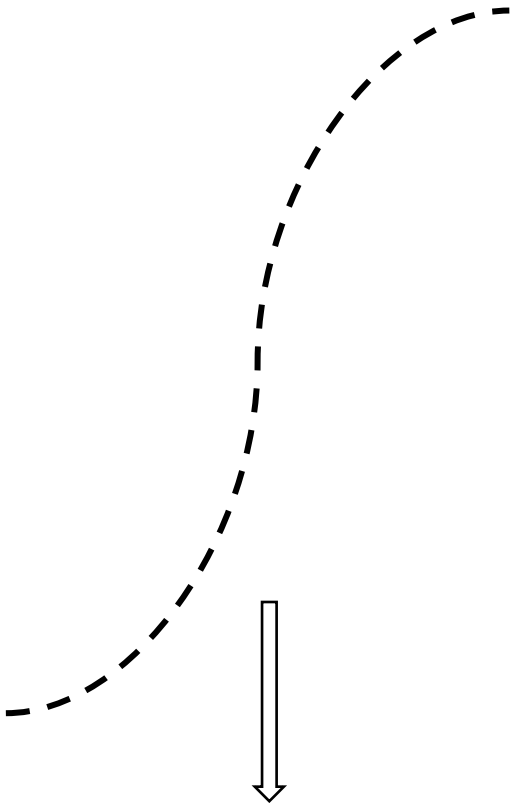




Results of Lossless compression:

30 40 0 1

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

Results of 1st Expansion pass:

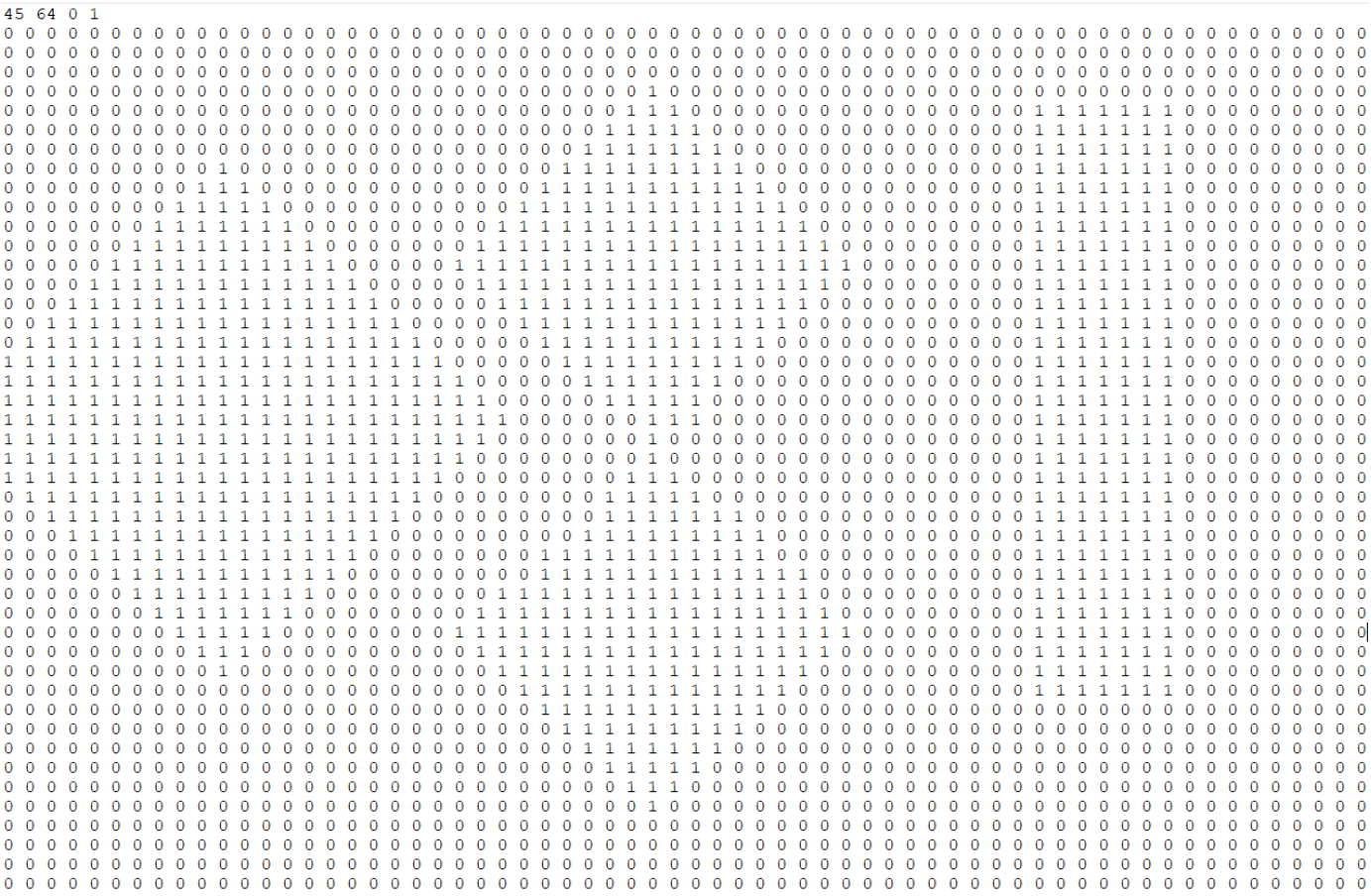
Results of 2nd Expansion pass:
30 40 0 7

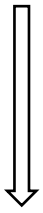
Results of Threshold Decompression:

```
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 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 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
```

Data set 2.

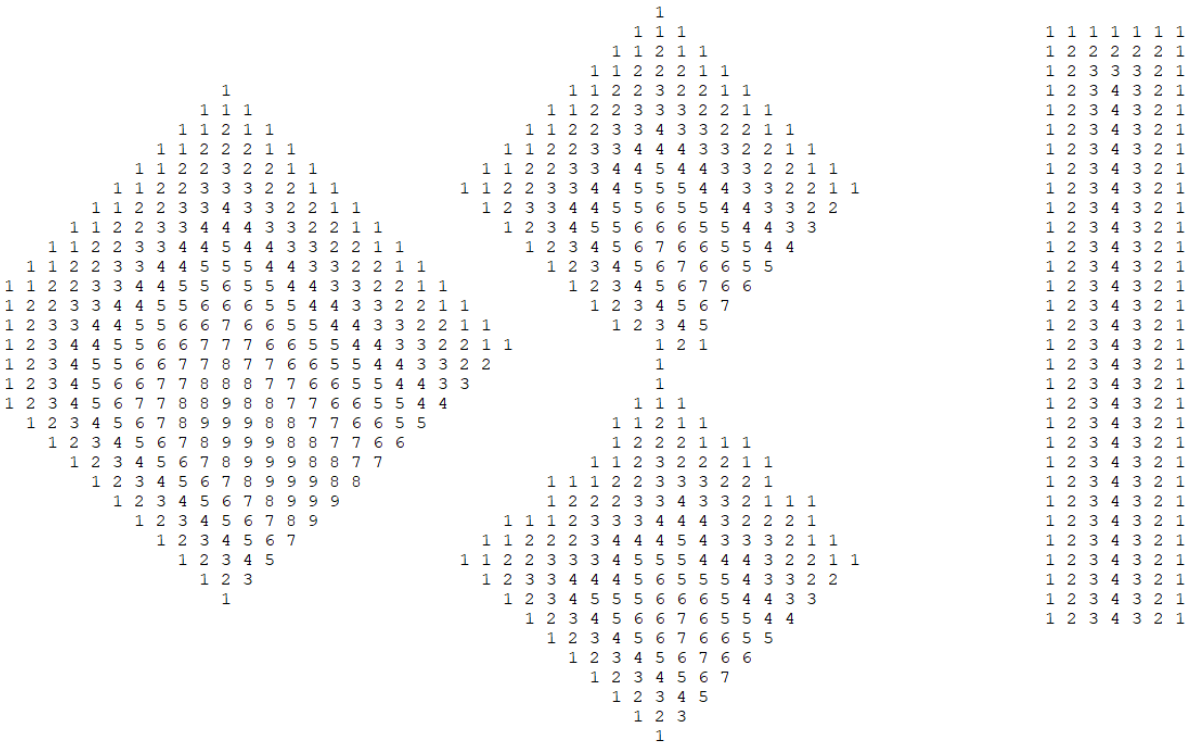
Binary Image



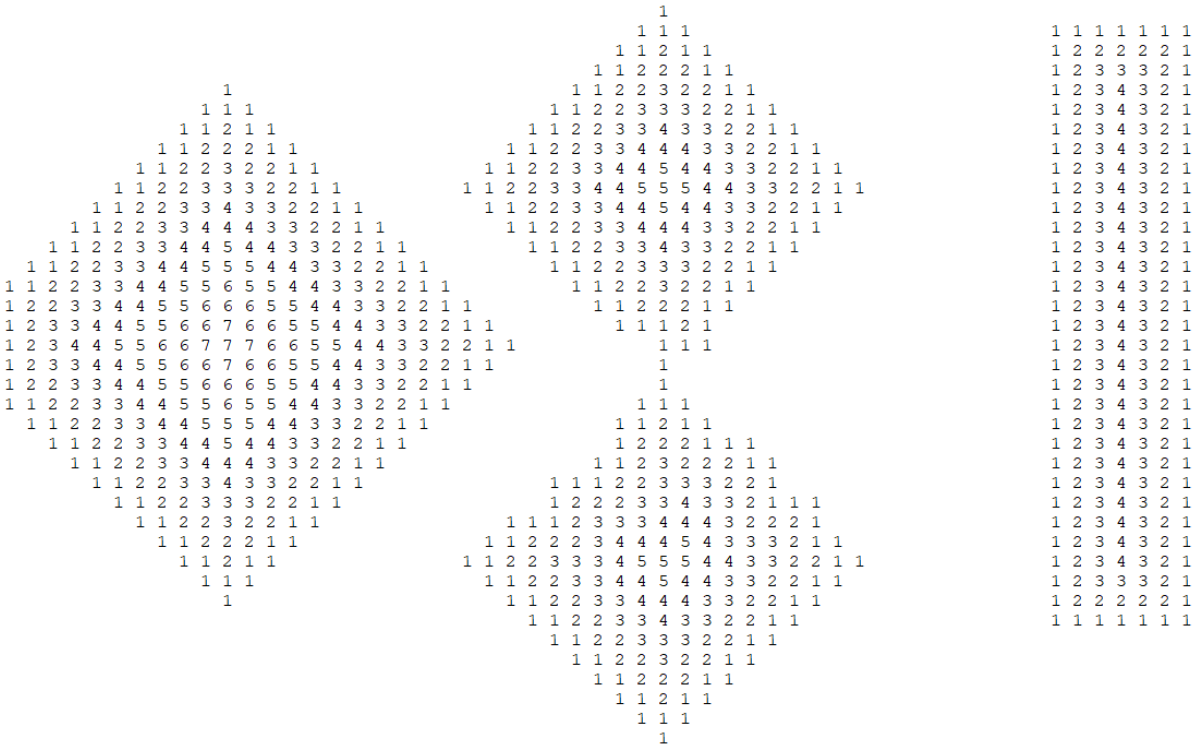


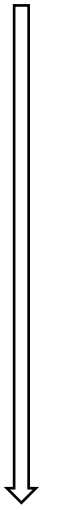
Distance Transform Pass 1

Results of 1st Distance Transform Pass:

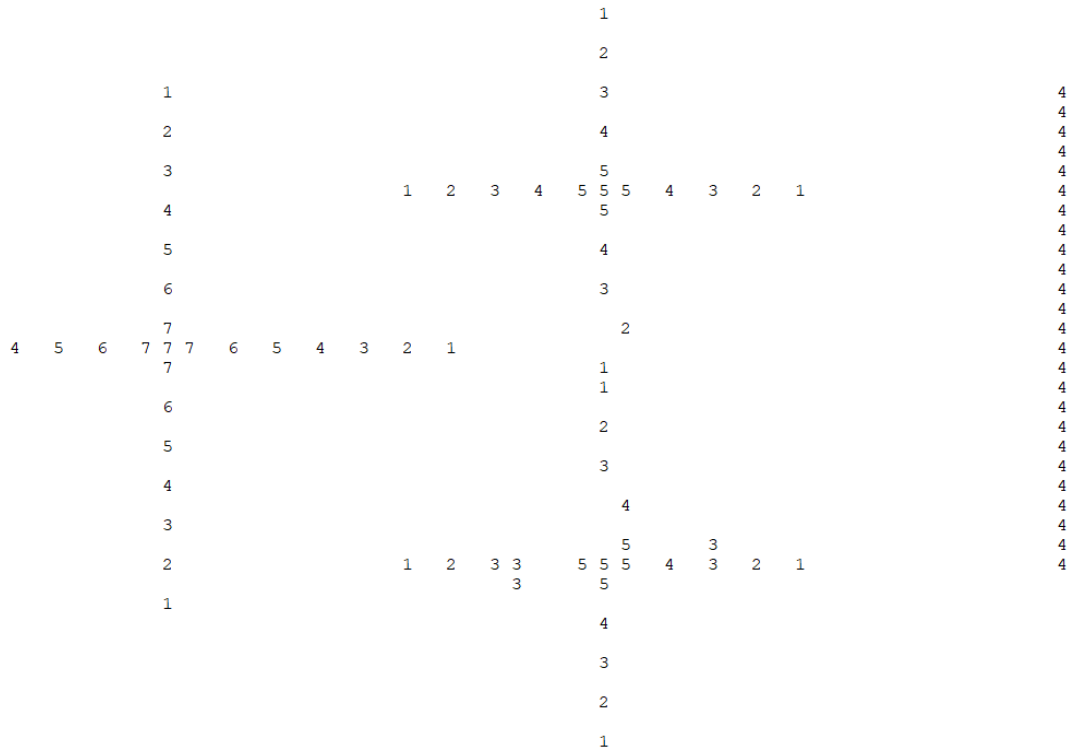


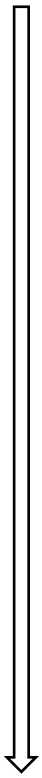
Results of 2nd Distance Transform Pass:





Results of Local Maxima:

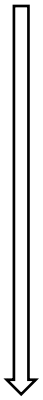




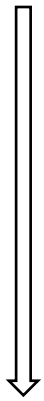
Results of Lossless compression:

45 64 0 1

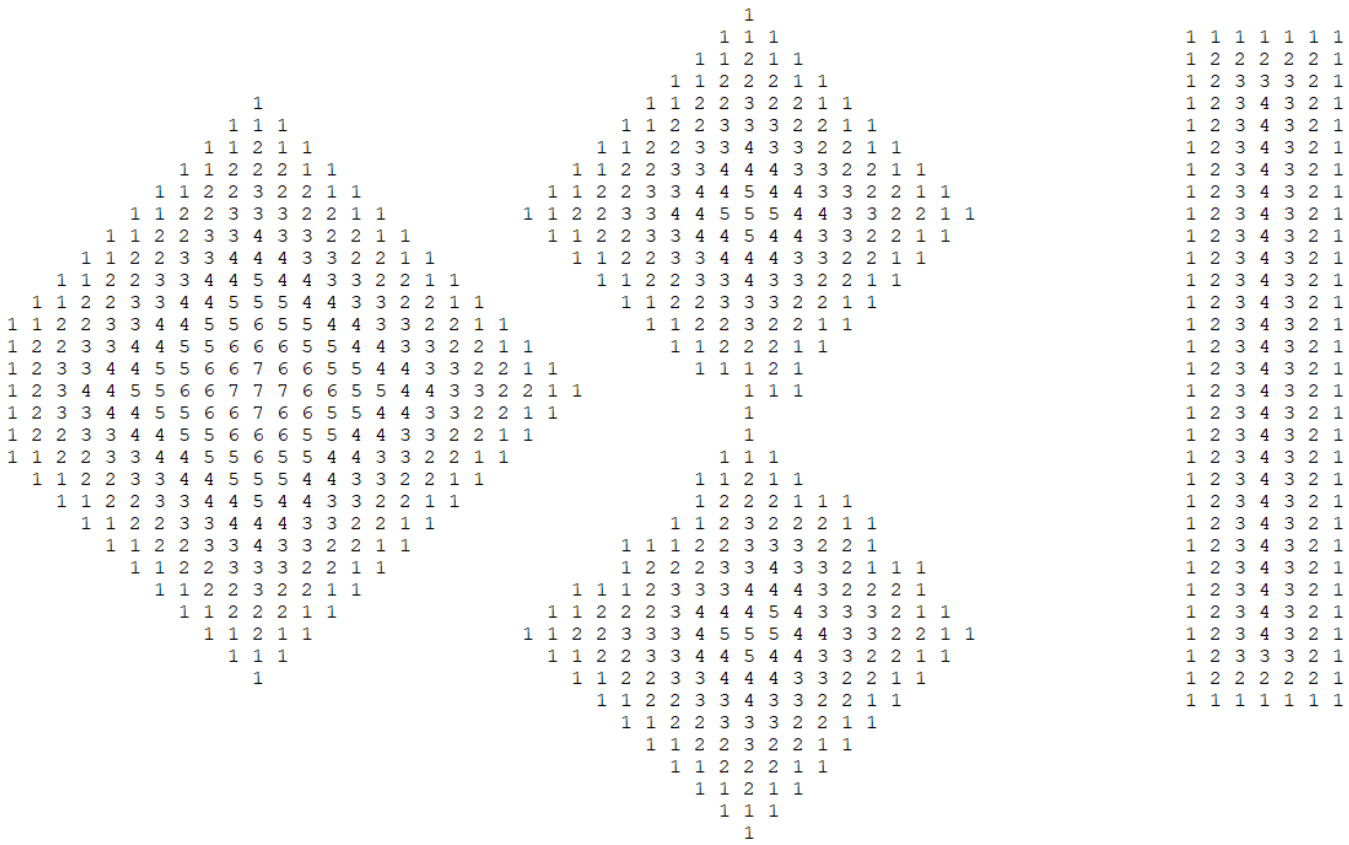
4 31 1
6 31 2
8 11 1
8 31 3
8 52 4
9 52 4
10 11 2
10 31 4
10 52 4
11 52 4
12 11 3
12 31 5
12 52 4
13 22 1
13 24 2
13 26 3
13 28 4
13 30 5
13 31 5
13 32 5
13 34 4
13 36 3
13 38 2
13 40 1
13 52 4
14 11 4
14 31 5
14 52 4
15 52 4
16 11 5
16 31 4
16 52 4
17 52 4
18 11 6
18 31 3
18 52 4
19 52 4
20 11 7
20 32 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



Results of 1st Expansion pass:



Results of 2nd Expansion pass:
45 64 0 7



Results of Threshold Decompression:

45 64 0 1