Name: Alif Rahi
Section: CSCI 381 - Computer Vision / Tues-Thurs 1:40-2:55pm
Project: 5
Due: March 27, 2023

Main algorithm steps:

```
IV. main(...)
*****************************
step 0:  inFile ← open the input file from argv [1]
         Connectness ← argv [2]
         option ← argv [3]
         RFprettyPrintFile, labelFile, propertyFile, deBugFile ← open from argv []
          numRows, numCols, minVal, maxVal ← read from inFile
         zeroFramedAry ← dynamically allocate.
         newLabel ← 0
step 1:  zero2D (zeroFramedAry)
step 2: loadImage (inFile, zeroFramedAry)
step 3: if option == 'y' or 'Y'
         conversion (zeroFramedAry)
step 4: if connectness == 4
           connected4 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
step 5: if connectness == 8
          connected4 (zeroFramedAry, newLabel, EQAry, RFprettyPrintFile, deBugFile)
step 6: labelFile ← output numRows, numCols, newMin, newMax to labelFile
step 7: printImg (zeroFramedAry, labelFile) // Output the result of pass3 inside of zeroFramedAry
```

Source code:

```
package RahiA_Project5;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.Scanner;
public class RahiA_Project5_Main {
        static Scanner inFile;
        static BufferedWriter prettyPrintFile1;
        static BufferedWriter prettyPrintFile2;
        static BufferedWriter decompressedFile;
        static BufferedWriter skeletonFile;
        static BufferedWriter debugFile;


        static class Imageprocessing{
                int numImgRows, numImgCols,imgMin,imgMax;
                int[][] zeroFramedAry, skeletonAry;
           int newMinVal = 9999;
           int newMaxVal = -1;
                public Imageprocessing(int numImgRows, int numImgCols,int imgMin,int imgMax) {
                        this.numImgCols = numImgCols;
                        this.numImgRows = numImgRows;
                        this.numImgCols = numImgCols;
                        this.imgMin = imgMin;
```

```
                this.imgMax = imgMax;
                this.zeroFramedAry = new int[numImgRows+2][numImgCols+2];
                this.skeletonAry = new int[numImgRows+2][numImgCols+2];

        }

        public void setZero(int[][] arr, int rSize, int cSize) {
                for(int i=0; i<rSize; i++) {
                        for(int j=0; j<cSize; j++) {
                                Arrays.fill(arr[i], 0);
                        }
                }
        }

        public void loadImg() {
                for(int i=1; i<=numImgRows; i++) {
                        for(int j=1; j<=numImgCols; j++) {
                                if(inFile.hasNextInt()) {
                                zeroFramedAry[i][j] = inFile.nextInt();
                                }
                        }
                }
        }

        public void distance8Pass1() {
                int mNeighbor;
      for(int i=1; i<numImgCols+1; i++){
        for(int j=1; j<numImgRows+1; j++){
          if(i<zeroFramedAry.length && j<zeroFramedAry[i].length && zeroFramedAry[i][j] >
0){
                        mNeighbor = Math.min( Math.min(zeroFramedAry[i][j-1],
zeroFramedAry[i-1][j-1]) , Math.min(zeroFramedAry[i-1][j], zeroFramedAry[i-1][j+1]) );
                        zeroFramedAry[i][j] =  mNeighbor + 1;
          }
        } // inner
      }
        }

        public void distance8Pass2() {
                 int minNeighbor;
               for(int i=numImgRows; i>=1; i--){
                 for(int j=numImgCols; j>=1; j--){

                   if(i<zeroFramedAry.length && j<zeroFramedAry[i].length &&
zeroFramedAry[i][j] > 0){
```

```
                        minNeighbor = Math.min( Math.min(zeroFramedAry[i][j+1],
zeroFramedAry[i+1][j+1]) , Math.min(zeroFramedAry[i+1][j], zeroFramedAry[i+1][j-1]) );
                        if (zeroFramedAry[i][j] > minNeighbor+1){
                            zeroFramedAry[i][j] = minNeighbor+1;
                        }
                        if (zeroFramedAry[i][j] >  newMaxVal)  newMaxVal = zeroFramedAry[i][j];
                        if(zeroFramedAry[i][j] < newMinVal) newMinVal = zeroFramedAry[i][j];
                    }
                }
            }
        }

        public void imageDecompression(){
        //*outFile << "\n********DE-COMPRESSION********\n\n";
        firstPassExpansion();

                try {
                        prettyPrintFile1.write("Entering basicOperations method \n");
                        reformatPrettyPrint(prettyPrintFile1, zeroFramedAry);
                } catch (IOException e2) {
                        e2.printStackTrace();
                }


        secondPassExpansion();
                try {
                        prettyPrintFile1.write("Entering basicOperations method \n");
                        reformatPrettyPrint(prettyPrintFile1, zeroFramedAry);
                } catch (IOException e2) {
                        e2.printStackTrace();
                }
    }

        public void firstPassExpansion() {
            int maxNeighbor;
             for(int i =1; i<=numImgRows; i++){
                for(int j =1; j<=numImgCols; j++){
                    if(zeroFramedAry[i][j] == 0){
                            maxNeighbor = Math.max(
                                        Math.max(Math.max(zeroFramedAry[i][j-1],
zeroFramedAry[i-1][j-1]), Math.max(zeroFramedAry[i-1][j],zeroFramedAry[i-1][j+1])),
                                        Math.max(Math.max(zeroFramedAry[i][j+1],
zeroFramedAry[i+1][j+1]), Math.max(zeroFramedAry[i+1][j],zeroFramedAry[i+1][j-1]))
                             );
                                if(maxNeighbor > 2){
                                    zeroFramedAry[i][j] = maxNeighbor-1;
```

```
                    }
                }
            }
        }
    }

    public void reformatPrettyPrint(BufferedWriter file, int[][] ary) {
  for(int i =1; i<numImgCols+1; i++){
    for(int j=1; j<numImgRows+1; j++){
            try {
                    if(i<zeroFramedAry.length && j<zeroFramedAry[i].length) {
                      if (ary[i][j] < 10){    // 2 padded spaces
                              file.write(ary[i][j] + "   ");
            }
            else if(ary[i][j] < 100){   // 1 padded space
                file.write(ary[i][j] + "  ");
            }
            else{   // no spaces
                file.write(ary[i][j]);
            }
                    }

                    } catch (IOException e2) {
                            e2.printStackTrace();
                    }
    }

    try {
                            file.write("\n");
                    } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                    }
  }
    }

public Boolean isLocalMaxima(int r, int c){
  for(int i=r-1; i<=r+1; i++){
    for(int j=c-1; j<=c+1; j++){
      if( zeroFramedAry[r][c] < zeroFramedAry[i][j]) return false;
    }
  }
  return true;
}

    public void secondPassExpansion() {
```

```
            int maxNeighbor;

            for(int i=numImgRows; i>=1; i--){
               for(int j =numImgCols; j>=1; j--){
                     maxNeighbor = Math.max(
                                 Math.max(Math.max(zeroFramedAry[i][j-1], zeroFramedAry[i-1][j-1]),
Math.max(zeroFramedAry[i-1][j],zeroFramedAry[i-1][j+1])),
                                 Math.max(Math.max(zeroFramedAry[i][j+1],
zeroFramedAry[i+1][j+1]), Math.max(zeroFramedAry[i+1][j],zeroFramedAry[i+1][j-1]))
                                 );


                     if( zeroFramedAry[i][j] < maxNeighbor-1) zeroFramedAry[i][j] = maxNeighbor-1;
               }
            }
               }
         public void computeLocalMaxima(){
            for(int i =1; i<=numImgRows; i++){
               for(int j=1; j<=numImgCols; j++){
                  if (isLocalMaxima(i,j) ) skeletonAry[i][j] = zeroFramedAry[i][j];
                  else skeletonAry[i][j] = 0;
               }
            }
         }

         public void skeletonExtraction(BufferedWriter file){
               try {
                           debugFile.write("Entering skeletonExtraction");
                     } catch (IOException e1) {
                           e1.printStackTrace();
                     }

               computeLocalMaxima();
               try {
                           file.write("LocalMaxima \n");
                     } catch (IOException e1) {
                           e1.printStackTrace();
                     }
            reformatPrettyPrint(file, skeletonAry);
            compression(file);

               try {
                           debugFile.write("Leaving skeletonExtraction");
                     } catch (IOException e1) {
                           e1.printStackTrace();
                     }
```

```
        }


            public void compression(BufferedWriter file) {
            try {
                        file.write(numImgRows + " " + numImgCols +" 0 " + newMaxVal
+"\n");
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                for(int i =1; i<=numImgRows; i++){
                   for(int j=1; j<=numImgCols; j++){
                      if (skeletonAry[i][j] > 0){
                        int id1 = i-1;
                        int id2 = j-1;
                       try {
                                                    file.write(id1+ " " + id2 + " " + skeletonAry[i][j] +
"\n");
                                                } catch (IOException e) {
                                                    e.printStackTrace();
                                                }
                      }
                   }
                }
                   }
                   public void decompression(BufferedWriter prettyPrintFile2, String str) {
                int skip4;
                Scanner skely;

                    try {
                        skely = new Scanner(new File(str));
                   for(int i=0; i<4; i++){
                      if(skely.hasNextInt()) {
                      skip4 = skely.nextInt();
                      }
                   }
                   int i=0, j=0, val = 0;
                   for(int k=0; k<3; k++){
                      if(skely.hasNextInt()) {
                        i = skely.nextInt();
                      }
                      if(skely.hasNextInt()) {
                         j = skely.nextInt();
                      }
                      if(skely.hasNextInt()) {
```

```
                    val = skely.nextInt();
                }

                    zeroFramedAry[i+1][j+1] = val;
            }



                } catch (FileNotFoundException e) {
                    e.printStackTrace();
                }


                try {
                    decompressedFile.write("\ndecompression\n\n");
                    firstPassExpansion();
                prettyPrintFile2.write("\nZero Framed Array after First Expansion\n");
                reformatPrettyPrint(prettyPrintFile2, zeroFramedAry);
                secondPassExpansion();
                prettyPrintFile2.write("\nZero Framed Array after Second Expansion\n");
                reformatPrettyPrint(prettyPrintFile2, zeroFramedAry);
                } catch (IOException e) {
                    e.printStackTrace();
                }




    }

void binaryThreshHold(){
  for(int i =1; i<=numImgCols; i++){
    try {
                        decompressedFile.write("\n");
                         for(int j=1; j<=numImgRows; j++){
                    if (zeroFramedAry[i][j] >= 1){
                        decompressedFile.write("1 ");
                    }
                    else {
                        decompressedFile.write("0 ");
                    }
                }

                } catch (IOException e) {
```

```
                                        e.printStackTrace();
                                }

                        }
                }

        }


        public static void main(String[] args) {
                File imgFile = new File(args[0]);
                File outFile1 = new File(args[1]);
                File outFile2 = new File(args[2]);
                String skeletonFileName = args[0] + "_skeleton.txt";
                String decompressedFileName = args[0] + "_decompressed.txt";


                try {
                        skeletonFile = new BufferedWriter(new FileWriter(new
File(skeletonFileName)));
                        prettyPrintFile1 = new BufferedWriter(new FileWriter(outFile1));
                        prettyPrintFile2 = new BufferedWriter(new FileWriter(outFile2));
                        decompressedFile = new BufferedWriter(new FileWriter(new
File(decompressedFileName)));
                        debugFile = new BufferedWriter(new FileWriter(args[3]));
                } catch (IOException e1) {
                        e1.printStackTrace();
                }

                Imageprocessing imageprocessing = null;
                int numImgRows = 0, numImgCols = 0,imgMin = 0,imgMax =0;

                try {
                        inFile = new Scanner(imgFile);
                        numImgRows = inFile.nextInt();
                        numImgCols = inFile.nextInt();
                        imgMin = inFile.nextInt();
                        imgMax = inFile.nextInt();
                        imageprocessing = new
Imageprocessing(numImgRows,numImgCols,imgMin,imgMax);

                } catch (FileNotFoundException e) {
                        e.printStackTrace();
                }
```

```
                imageprocessing.setZero(imageprocessing.zeroFramedAry, numImgRows+2,
numImgCols+2);
                imageprocessing.setZero(imageprocessing.skeletonAry, numImgRows+2,
numImgCols+2);
                imageprocessing.loadImg();
        try {
                debugFile.write("Entering Distance8 \n");
                } catch (IOException e1) {
                        e1.printStackTrace();
                }

                try {
                        prettyPrintFile1.write("1st pass distance transform \n");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                imageprocessing.distance8Pass1();
                imageprocessing.reformatPrettyPrint(prettyPrintFile1,
imageprocessing.zeroFramedAry);

                try {
                        prettyPrintFile1.write("2nd pass distance transform \n");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                imageprocessing.distance8Pass2();
                imageprocessing.reformatPrettyPrint(prettyPrintFile1,
imageprocessing.zeroFramedAry);

        try {
                debugFile.write("Leaving Distance8 \n");
                } catch (IOException e1) {
                        e1.printStackTrace();
                }

        imageprocessing.setZero(imageprocessing.zeroFramedAry, numImgRows+2, numImgCols);



                imageprocessing.skeletonExtraction(skeletonFile);
                imageprocessing.decompression(prettyPrintFile2, skeletonFileName);

                try {
                        decompressedFile.write(numImgRows + " " + numImgCols + " " + imgMin + "
" + imgMax);
                } catch (IOException e) {
```

```
                e.printStackTrace();
            }

            try {
                skeletonFile.close();
                prettyPrintFile1.close();
                prettyPrintFile2.close();
                debugFile.close();

            } catch (IOException e) {
                e.printStackTrace();
            }


        }

}
```

Output1

1st pass distance transform

```
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 1 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 2 3 3 3 3 3 2
0 0 0 0 0 0 0 0 1 1 2 1 1 0 0 0 0 0 0 0 0 0 1 2 3 4 4 4 3 2
0 0 0 0 0 0 0 1 1 2 2 2 1 1 0 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 1 2 2 3 2 2 1 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 2 3 3 3 2 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 3 4 3 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 4 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 1 1 1 2 3 4 5 4 3 2 1 1 1 1 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 2 2 2 3 4 5 4 3 2 2 2 2 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 3 3 3 4 5 4 3 3 3 3 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 4 4 4 5 4 4 4 4 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 5 5 5 5 5 5 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 6 6 6 6 6 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 7 7 7 6 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 1 1 1 1 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 5 4 3 2 2 2 2 2 2 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 5 4 3 3 3 3 3 3 3 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 5 4 4 4 4 4 4 4 4 4 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 5 5 5 5 5 5 5 5 5 5 5 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 6 6 6 6 6 6 6 6 6 6 6 6 5 4 3 2
0 0 0 1 2 3 4 5 6 7 8 8 7 7 7 7 7 7 7 7 7 7 7 7 6 5 4 3 2
0 0 0 0 1 2 3 4 5 6 7 8 8 8 8 8 8 8 0 0 0 0 0 0 0 0 0 0 0 0
```

0 0 0 0 0 1 2 3 4 5 6 7 8 9 9 9 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 2 3 4 5 6 7 8 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2nd pass distance transform
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 1 0 0 0 0 0 0 0 0 0 0 0 1 2 2 2 2 2 2 2
0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 2 3 3 3 3 3 2
0 0 0 0 0 0 0 0 1 1 2 1 1 0 0 0 0 0 0 0 0 0 1 2 3 4 4 4 3 2
0 0 0 0 0 0 0 1 1 2 2 2 1 1 0 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 1 2 2 3 2 2 1 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 2 3 3 3 2 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 3 4 3 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 4 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 1 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 1 1 1 2 3 4 5 4 3 2 1 1 1 1 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 2 2 2 3 4 5 4 3 2 2 2 2 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 3 3 3 4 5 4 3 3 3 3 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 4 4 4 5 4 4 4 4 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 5 5 5 5 5 5 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 6 6 6 6 6 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 1 2 3 4 5 6 7 7 7 7 6 5 4 3 2 1 0 0 0 1 2 3 4 5 4 3 2
0 0 0 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
0 0 0 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 2
0 0 0 1 2 3 4 5 5 6 6 6 6 5 5 4 3 3 3 3 3 3 3 3 4 5 4 3 2
0 0 0 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 2
0 0 0 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
0 0 0 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
0 0 0 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
0 0 0 0 1 1 2 2 3 3 4 4 3 3 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 1 2 2 3 3 3 3 2 2 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 2 2 2 2 2 2 1 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 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

The other outputs did not produce correct values. **If you gave me 15 more minutes** and i didnt have to submit it by 12 sharp I promise you i coulve got there correct outputs