

Section: CV

JAVA

Student: Alex Baraian

Project Due date: 4/19/2023

Project Number: 6

Algorithm Steps:

S1: open inFile and outFile from args[]
S2: read numRows, numCols,minVal,maxVal from inFile
S3: HoughAngle = 180
S4: HoughDist = 2 * diagonal of input image
S5: dynamically allocate imgAry[numRows][numCols]
S6: dynamically allocate CartesianHoughAry[HoughDist][HoughAngle] and PolarHoughAry[HoughDist][HoughAngle]
S7: offset = $\sqrt{\text{numRows}^2 + \text{numCols}^2}$
S8: loadImage(inFile)
S9: PrettyPrint(imgAry,outFile)
S10: buildHoughSpace()
S11: prettyPrint(CartesianHoughAry,outFile)
S12: prettyPrint(PolarHoughAry,outFile)
S13: close all files

Source Code:

Main Class

```
import java.io.*;
import java.util.Scanner;
public class BaraianA_Main {
    public static void main(String[] args)throws IOException{
        File inputFile = new File(args[0]);
        File outFile = new File(args[1]);
        Scanner inputFileScanner = new Scanner(inputFile);
        BufferedWriter writer = new BufferedWriter(new FileWriter(outFile,true));
        BaraianA_HoughTransform HoughSpace = new BaraianA_HoughTransform();

        HoughSpace.numRows=inputFileScanner.nextInt();
        HoughSpace.numCols=inputFileScanner.nextInt();
        HoughSpace.minVal=inputFileScanner.nextInt();
        HoughSpace.maxVal=inputFileScanner.nextInt();
        HoughSpace.HoughAngle=180;
        HoughSpace.HoughDist= (int) (2 *
Math.sqrt(Math.pow(HoughSpace.numRows,2)+Math.pow(HoughSpace.numCols,2)));
        HoughSpace.imgAry= new int[HoughSpace.numRows][HoughSpace.numCols];
        HoughSpace.CartesianHoughAry = new int [HoughSpace.HoughDist][HoughSpace.HoughAngle];
        HoughSpace.PolarHoughAry = new int[HoughSpace.HoughDist][HoughSpace.HoughAngle];
        for(int i=0;i<HoughSpace.numRows;i++) {
            for(int j=0;j<HoughSpace.numCols;j++) {
                HoughSpace.imgAry[i][j]=0;
            }
        }
        for(int i=0;i<HoughSpace.HoughDist;i++) {
```

```

        for(int j=0;j<HoughSpace.HoughAngle;j++) {
            HoughSpace.CartesianHoughAry[i][j]=0;
            HoughSpace.PolarHoughAry[i][j]=0;
        }
    }

    HoughSpace.offset=(int) Math.sqrt(Math.pow(HoughSpace.numRows,2 )+Math.pow(HoughSpace.numCols, 2));

    HoughSpace.loadImage(inputFileScanner);
    writer.write("Img Array\n");
    HoughSpace.reformatPrettyPrint(HoughSpace.imgAry, writer);
    HoughSpace.buildHoughSpace();

    writer.write("Cartesian Hough Array\n");
    HoughSpace.reformatPrettyPrint(HoughSpace.CartesianHoughAry, writer);
    writer.write("Polar Hough Array\n");
    HoughSpace.reformatPrettyPrint(HoughSpace.PolarHoughAry, writer);
    writer.close();
}

}

```

HoughTransform Class

```

import java.io.*;
import java.util.Scanner;
public class BaraianA_HoughTransform {
    int numRows;
    int numCols;
    int minVal;
    int maxVal;
    int HoughDist;
    int HoughAngle=180;
    int imgAry[][];
    int CartesianHoughAry[][];
    int PolarHoughAry[][];
    int angleInDegree;
    double angleInRadians;
    int offset;

    void loadImage(Scanner S) {
        for(int i=0;i<numRows;i++) {
            for(int j=0;j<numCols;j++) {
                imgAry[i][j]=S.nextInt();
            }
        }
    }

    void PrettyPrint(BufferedWriter writer) throws IOException {
        for(int i=0;i<numRows;i++) {
            for(int j=0;j<numCols;j++) {
                writer.write(imgAry[i][j]+" ");
            }
            writer.write("\n");
        }
    }

    void buildHoughSpace() {

```

```

        for(int i=0;i<numRows;i++) {
            for(int j=0;j<numCols;j++) {
                if(imgAry[i][j]>0) {
                    computeSinusoid(i,j);
                }
            }
        }
    }

    void computeSinusoid(int x,int y) {
        angleInDegree=0;
        double dist;
        int distInt;
        while(angleInDegree<=179) {
            angleInRadians = (double)(angleInDegree*Math.PI/180);
            dist=CartesianDist(x,y);
            distInt=(int)dist;
            CartesianHoughAry[distInt][angleInDegree]++;
            dist=PolarDist(x,y);
            distInt=(int)dist;
            PolarHoughAry[distInt][angleInDegree]++;
            angleInDegree++;
        }
    }

    double CartesianDist(int x,int y) {
        double t= angleInRadians- Math.atan(y/x)- Math.PI/2;
        double cartesiandistance=Math.sqrt(Math.pow(x, 2)+Math.pow(y,2)) * Math.cos(t) + offset;
        return cartesiandistance;
    }

    double PolarDist(int x, int y) {
        double polardistance = x* Math.cos(angleInRadians) + y* Math.sin(angleInRadians) + offset;
        return polardistance;
    }

    void reformatPrettyPrint(int Ary[],BufferedWriter writer) throws IOException {
        int max=0;
        for(int row=0;row<Ary.length;row++){
            for(int col=0;col<Ary[0].length;col++){
                if(max<Ary[row][col]){
                    max=Ary[row][col];
                }
            }
        }

        int r=0,c=0;
        while(r<Ary.length) {
            c=0;
            while(c<Ary[0].length) {
                if(max<10) {
                    if(Ary[r][c]==0) {
                        writer.write(" ");
                    }
                    else {
                        writer.write(Ary[r][c] + " ");
                    }
                }
                else {
                    if(Ary[r][c]==0) {
                        writer.write(" ");
                    }
                }
            }
            r++;
        }
    }

```

```

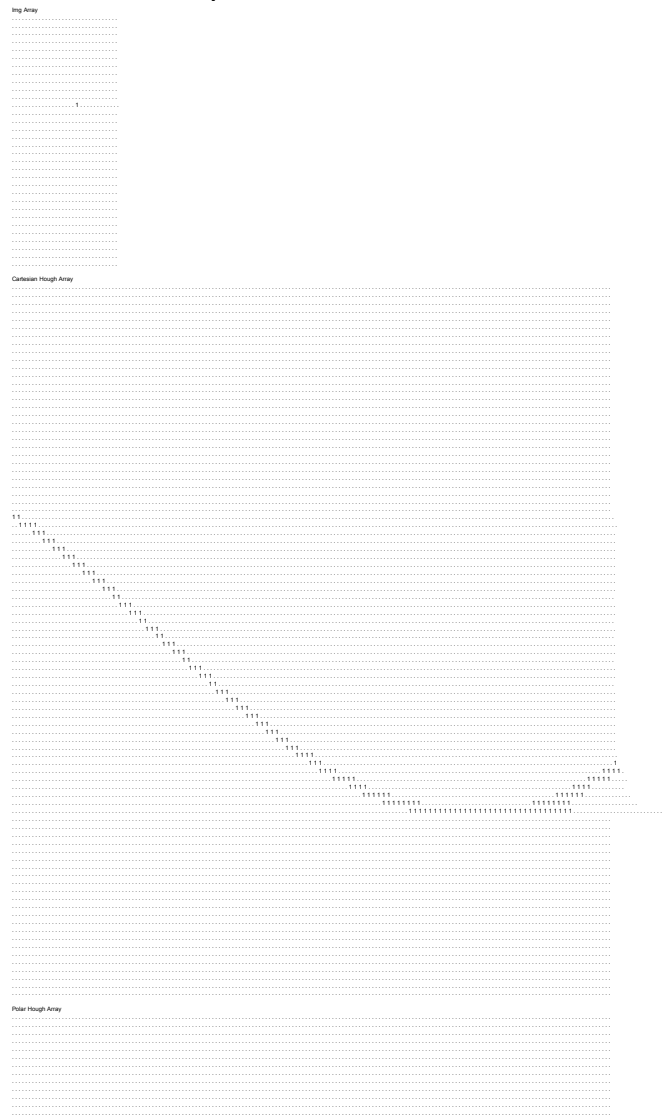
        else if(Ary[r][c]>=10) {
            writer.write(Ary[r][c]+ " ");
        }
        else {
            writer.write(Ary[r][c]+" ");
        }

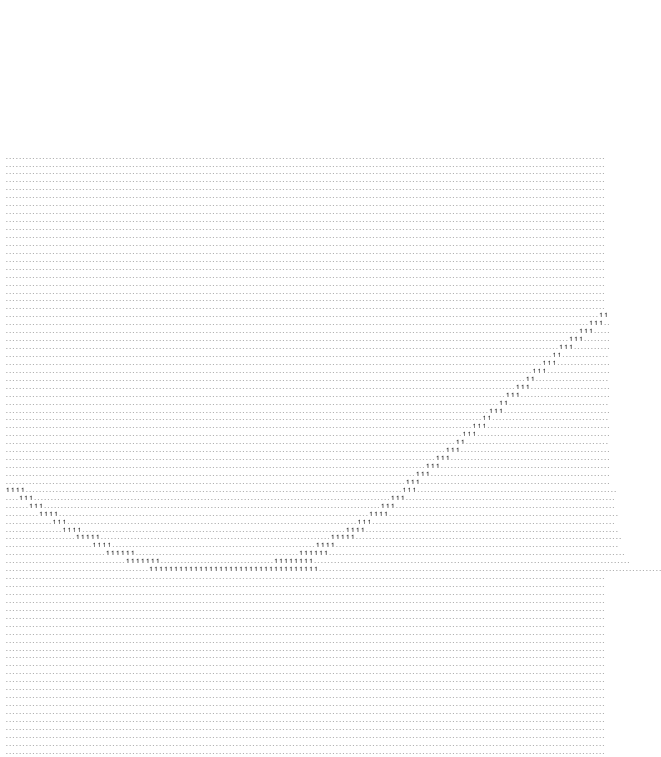
    }
    c++;
}
writer.write("\n");
r++;
}
writer.write("\n");
writer.flush();
}

}
}

```

Outfile From 2)



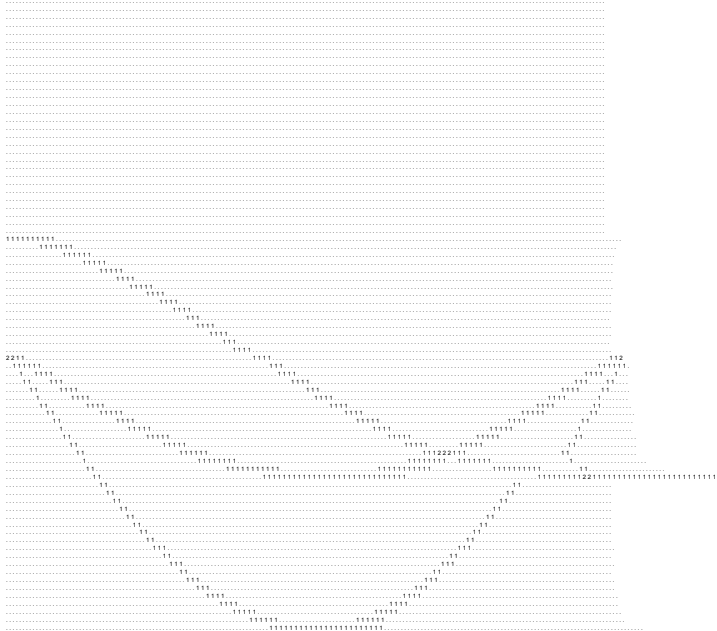


Outfile From 3)

img Array

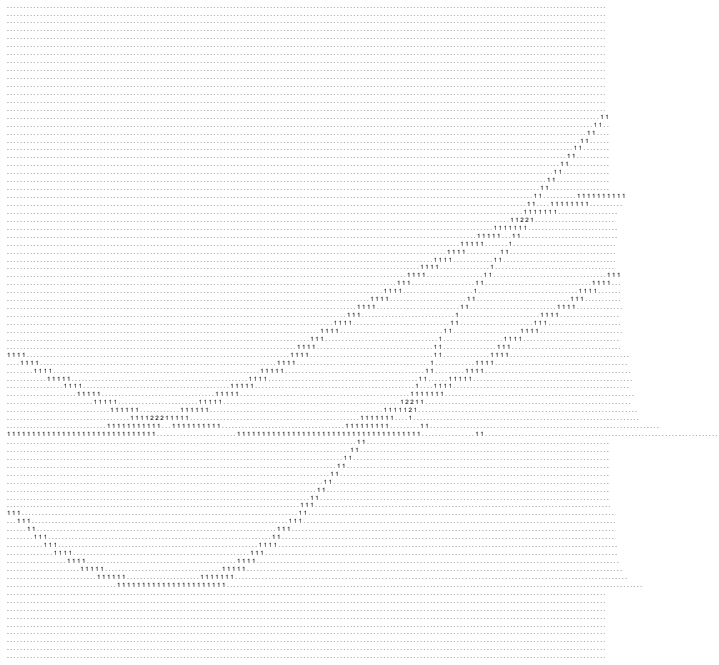


Cartesian Hough Array



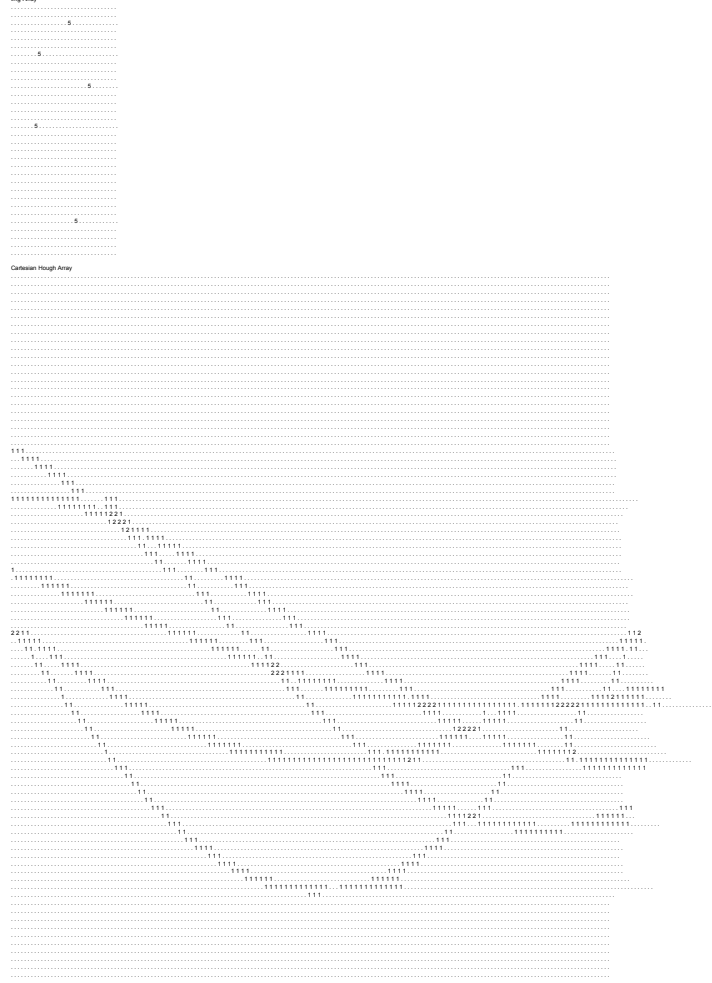
Polar Hough Array



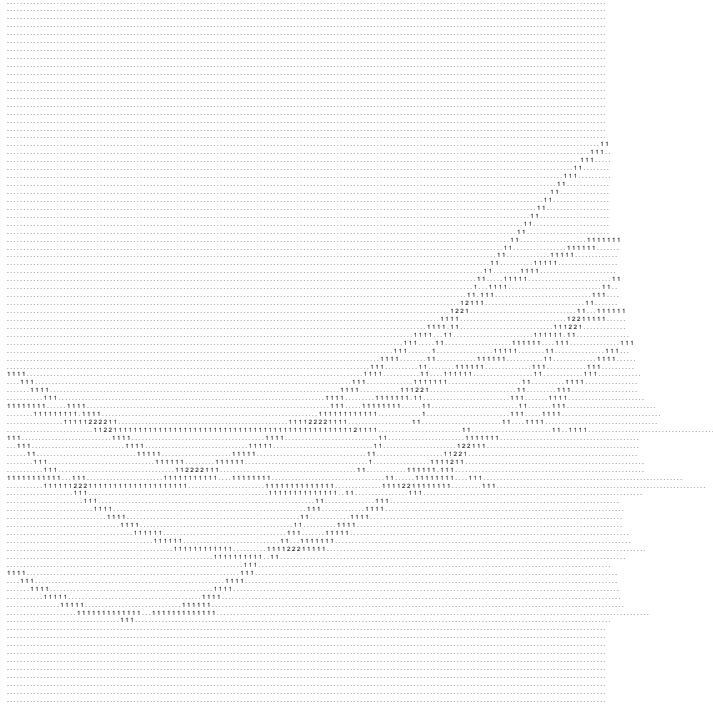


Outfile From 4)

img Array

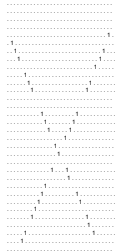


Polar Hough Array

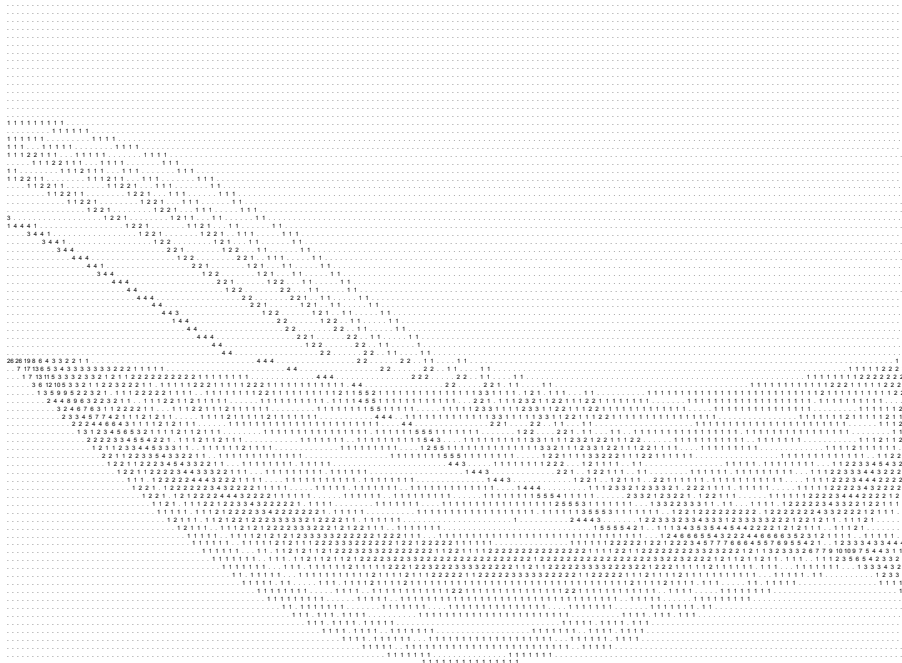


Outfile From 5)

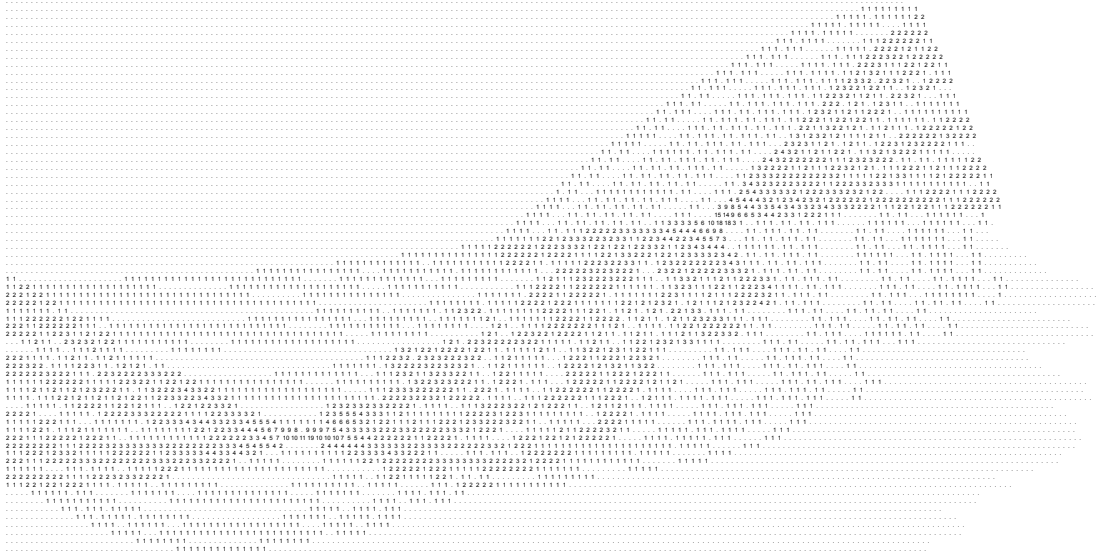
Ing Array



Cartesian Hough Array

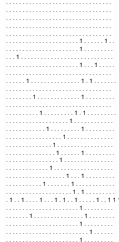


Polar Hough Array



Outfile From 6)

Img Array



Cartesian Hough Array

