Project 6 (Java): You are to implement the Hough Transform algorithm. You will create two Hough arrays, one uses Cartesian distance formula and the other uses Polar distance formula.

****

Language: Java

Project points: 10pts

Due Date: Soft copy (*.zip) and hard copies (*.pdf):

10/11 (early submission): 4/15/2023 Saturday before midnight.

10/10 (on time): 4/19/2023 Wednesday before midnight.

-10/10 (non-submission): 4/19/2023 Wednesday after midnight. **NO LATE Submission**.

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement.

*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.

==========

You will be given 5 test data img1pt, img3pt, img5pt, img2lines, img5lines: contains 1 point, 3 points, 5 points, two colinear lines and five colinear lines

What to do as follows:

1) Implement your program based on the specs given below until pass compilation.

2) Run and debug your program on img1pt until you see 1 sinusoid in both Hough Space.

3) Run and debug your program on img3pt until you see 3 sinusoids in both Hough Space.

4) Run and debug your program on img5 until you see 5 sinusoids in both Hough Space.

5) Run your program on img2lines until you should have multiple sinusoids what intersect at a point (or near-by) in both Hough Space.

6) Run your program on img5lines, you should have multiple sinusoids what intersect at a point (or near-by) in both Hough Space.

*** Include in your hard copies:

- cover page
- source code
- outFile1 from the results of 2) in the above.
- outFile1 from the results of 3) in the above.
- outFile1 from the results of 4) in the above.
- outFile1 from the results of 5) in the above.
- outFile1 from the results of 6) in the above.

****

I. inFile (args [0]): a binary image with header

II. outFile1 (args[1]): prettyPrint for both Hough arrays.

****

III. Data structure:

****

- A HoughTransform class
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal
  - (int) HoughDist // 2 times of the diagonal of the image
  - (int) HoughAngle // 180
  - (int) imgAry [][]// a 2D int array size of numRows by numCols; needs to dynamically allocate.
  - (int) CartesianHoughAry[][] //size of HoughDist by HoughAngle; needs to dynamically allocate.

- (int) PolarHoughAry[][] //size of HoughDist by HoughAngle; needs to dynamically allocate.
- (int) angleInDegree
- (double) angleInRadians
- (int) offSet // Given in class. See your lecture note.

- methods:
  - constructor(...)
  - loadImage (…) // load imgAry from inFile
  - buildHoughSpace (...) // See algorithm steps below
  - (double) CartesianDist (…) // use the Cartesian distance formula given in class
  - (double) PolarDist (…) // use the Polar distance formula given in class
  - prettyPrint (…) // Reuse codes in your previous projects

****
IV. main (…)
****
Step 0: inFile, outFile1 ← open from args []
numRows, numCols, minVal, maxVal ← read from inFile
HoughAngle ← 180
HoughDist ←2 * (the diagonal of the input image)
imgAry ← dynamically allocate
CartesianHoughAry ← dynamically allocate and initialize to zero
PolarHoughAry ← dynamically allocate and initialize to zero
offSet ← // See your lecture note.
Step 1: loadImage (inFile, imgAry)
prettyPrint (imgAry, outFile1)
Step 2: buildHoughSpace (...)
Step 3: prettyPrint (CartesianHoughAry, outFile1) // with caption indicate it is Cartesian Hough space
prettyPrint (PolarHoughAry, outFile1) // with caption indicate it is Polar Hough space
Step 4: close all files

****
V. buildHoughSpace (...)
****
Step 1: scan imgAry left to right and top to bottom
Using x for rows and y for column
Step 2: if imgAry [x, y] > 0
computeSinusoid (x, y)
Step 3: repeat step 1 to step 2 until all pixels are processed

****
VI. computeSinusoid (x, y)
****
Step 1: angleInDegree ← 0
Step 2: angleInRadians ← (double) (angleInDegree / (180.00 * pi))
Step 3: dist ← CartesianDist (x, y, angleInRadians)
Step 4: distInt ← (int) dist // cast dist from double to int
Step 5: CartesianHoughAry[distInt][angleInDegree]++
Step 6: dist ← PolarDist (x, y, angleInRadians)
Step 7: distInt ← (int) dist // cast dist from double to int
Step 8: PolarHoughAry[distInt][angleInDegree]++
Step 9: angleInDegree ++
Step 10: repeat step 2 to Step 9 while angleInDegree <= 179

****
VII. CartesianDist (x, y, angleInRadians)
****
// Use the Cartesian distance formula given in class, see your lecture note.
// x & y need to convert to double in computation.
// add offSet to the computation.

****
VIII. PolarDist (x, y, angleInRadians)
****
// Use the polar distance formula given in class, see your lecture note.
// x & y need to convert to double in computation
// add offSet to the computation.