Project 8.2: Maximum arc-Chord distance edge detector (Java)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: \*\*\*\* Java \*\*\*\*

Due date: soft copy: 5/2/2018 Wednesday before Midnight

Early submission +1 deadline: 4/29/2018 Sunday before Midnight

Due date: hard copy: 5/3/2016 Thursday in class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. Inputs:

- Input1 (argv[]) : A text file contains the boundary points of an object in an image.

The format of the input is as follows:

#rows #cols minVal maxVal // image header

label // the label of the object

r1 c1

r2 c2

r3 c3

:

:

- Input2 (console): ask the user from condole for K. 2\*K will be used as

the length of the arc-Chord in the maximum arc-Chord distance computation

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

II. Outputs:

- Output1 (argv[]): The result of the maximum arc-Chord distance of the object boundary points plus corner indicating label. The format of this output file is as follows:

#rows #cols minVal maxVal // image header

label // the label of the object.

#pts // the number of boundary points

r1 c1 1 // not a corner

r2 c2 9 // a corner (use 9 for corner indicator for this project)

r3 c3 1 // not a corner

:

:

- Output2 (argv[]): Pretty print (displaying) as an image of the result of the Maximum arc-Chord distance corner detection, where corner points are printed as 9 and non-corner points are printed as 1.

- Output3 (argv[]): for all debugging output

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Data structure:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- An image class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- img (int\*\*) // a 2D array for display, initially set to 0

- constructor

- plotPt2Img()

// put each point (x, y)’s corner indicating value (1 or 9) at Img(x, y)

- prettyPrint (img) // print img, if pixel(i,j) == 0

// print blank, otherwise, print its value.

- A boundaryPt class

- x (int)

- y (int)

- maxVotes (int) // initallized to 0

- maxDist (double) // to keep track of the maximum distance,

// initialized to 0.0

- corner (int) // initallized to 1, not corner

- constructor

- A arcChord class

- chordLength (int)

// Ask user to input K from console,

// chordLength is set to (2\*K)

- countPts (inFile) // reads and returns the count of the boundary points

// in the input file.

- numPts (int) // get from input-1

- PtAry (boundaryPt \*) // an 1D array of boundaryPt class,

// need to dynamically allocate.

// use mod function to get index during compute

// the 2K points without extending the tail of the array,

// the array size is numPts.

- printPtAry() // print the content of the entire PtAry

- chordAry (double \*) // used during computation, size of chordLength,

// need to be dynamically allocated

- P1 (int) // the array index of the first-end of the arc Chord;

// initially set to 0

- P2 (int) // the array index of the second-end of the arc Chord;

// initially set to chordLength

- loadData (inFile) // read and store data to PtAry

- computeDistance (P1, P2, Pt) // It computes the orthogonal distance

// from Pt to the line formed by P1 and P2 (with respect

// to their xy coordinates.

// it returns the computed distance.

// Use the distance formula given in class

- findMax (chordAry)

// find which index in chordAry having the maximum distance

// and returns the index that has

// the maximum distances, for voting.

- computeLocalMaxima (PtAry)

// Go thru the entire PtAry,

// p(i) is a local maxima

// iff p(i)’s maxVote >= all the maxVotes of

// its 1x5 neighborhood: two points from its left

// and two points from its right.

- setCorner (PtAry) // a boundary point, p(i) is a corner (returns 9), if

// (a) p(i) is a local maxima, and

// (b) within its 5 neighborhood, p(i-2), p(i-1), p(i+1), p(i+2)

// only p(i-1) or p(i+1) can be a local maxima,

// otherwise, p(i) is not a corner (return 1).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Algorithms

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: - inFile 🡨 open input files

(numRows, numCols, minVal, maxVal, label) <- get from inFile

- dynamically allocate image array of size numRows by numCols

- numPts 🡨 countPts (inFile)

- close inFile

- inFile 🡨 open the input file the second time.

- dynamically allocate PtAry with size of numPts

- K 🡨 get from the user from console

- chordLength 🡨 (2\*K)

- dynamically allocate chordAry with size of chordLength

// initiallied to 0.0

- loadData (inFile)

Step 1: P1 <-- 0

P2 <-- chordLength-1

step 2: index <-- 0

currPt <-- P1 + 1

step 3: dist <-- computeDistance (P1, P2, currPt )

chordAry[index]🡨 dist

index ++

currPt ++

step 4: repeat step 3 while index < chordLength

step 5: print chordAry to debugging file (Output3)

step 6: maxIndex <-- findMaxDist(chordAry)

// find the max of distances of all points in chordAry

// and returns that index

whichIndex <-- P1 + maxIndex

PtAry[whichIndex]'s maxVotes ++

update PtAry[whichIndex]'s maxDist if it is less then chordAry[maxIndex]

step 7: print PtAry from P1 to P2 to output3, debugging file

step 8: Increment P1, and P2, and then

mod (P1, numPts) and mod (P2, numPts)

// so the computation will continue wrapped around the boundray

step 9: repeat step 2 to step 8 until P2 == (chordLength / 2)

step 10: printPtAry() // five pts per text line

step 11: computeLocalMaxima (PtAry)

step 12: setCorner (PtAry) do for all point in boundPtAry[index], index from 0 to numPts-1

step 13: output only (x, y, corner) of the entire PtAry to output1

step 14: Img <-- create an image of size numRows by numCols

step 15: plotPt2Img()

// put each point (x, y)’s corner value (1 or 9) at Img(x, y)

step 16: prettyPrint (img) to output2

step 17: close all files