Project 8 (Java): Chain code for image compression (lossless compression) and object recognition (via) boundary Pattern Analysis. You may assume the objects in the image do not have holes in them.

You will be given two sets of files: a) Set1: img1CC and img1Property and b) Set2: img2CC and img2Property.

1) Implement your program based on the specs given below until it passes compilation.
2) Print Img1CC on a piece of paper. Hand-trace the chain-code of the boundary of Img1CC. Then write the result of chain code: image header (one textline) startR, startC, label (one textline) follows by chain code (as illustrated in class.)
3) Run your program using Set1: img1CC and img1Property to get chain code for each object. (the format for chain code output is given below.)
4) Debug your program until your program produces the same chain-code as your hand drawn in 2)
5) Run your program using Set2: img2CC and img2Property.

*** Include in your hard copy:
- Cover page
- The hand tracing of Img1CC and write the traced chain-code. (-1 if not included.)
  (Check to see if your hand traced chain code is the same as your program produces.
- Source Code
- Print img1CC // with caption: "Below is input img1CC"
- Print img1Property // with caption: "Below is img1Property"
- Print outFile1 for Set1 // with caption.
- Print ChainCodeFile for Set1 // with caption.
- Print Boundary file for Set1 // with caption.
- Print LogFile for Set1 // limited to 4 pages, if more.

- Print img2CC // with caption: "Below is the input img2CC"
- Print img2Property // with caption: "Below is img2Property"
- Print outFile1 for Set2 // with caption.
- Print ChainCodeFile Set2 // with caption.
- Print Boundary file for Set2 // with caption.
- Print LogFile for Set1 // limited to 4 pages, if more.

**********

Language: Java

**********

Project Name: Image Compression via Chain-code

Points: 12 pts

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

13/12 (early submission): 11/25/2024, Monday before midnight.

12/12 (on time): 11/30/2024 Saturday before midnight.

-12/12 non-submission): 11/30/2024 Saturday after midnight.

*** Name your soft copy and hard copy files the naming convention given in Project Submission Requirements. (-2 if not.)

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

Email subject: (CV) first name last name <Project 8: Image compression via Chain-code (Java)>

*** Inside the email body include your answer to the 4 questions. Optional screen recording if you wish.

**********

I. Inputs: There are two input files:

**********

a) labelFile (args [0]): An image file with header

b) propFile (args [1]): the connected component properties with the format as below (see the given file):

- 1st text-line, the header of the input image,
- 2nd text-line is the total number of connected components.
- 1 // first CC label
- number of pixels
- upperR upperC //the r c coordinated of the upper left corner
- lowerR lowerC //the r c coordinated of lower right corner
- 2 // second CC label
- number of pixels
- upperR upperC //the r c coordinated of the upper left corner
- lowerR lowerC //the r c coordinated of lower right corner

For an example:

```
45  40  0   9  // image header
9          // there are a total of 9 CCs in the image
1          // CC label 1
187          // 187 pixels in CC label 1
4     9    // upper left corner of the bounding box at row 4 column 9
35  39  // lower right corner of the bounding box at row 35 column 3                    :
2          // CC label 2
:
```

**********

II. Outputs:

a) outFile1 (use args [2]): as specs dictates.

b) chainCodeFile (use args [3]): To store the chain code of the object in the following format:

```
numRows numCols minVal maxVal // image header use one text line
Label startRow startCol // use one text line
code1 code2 code3 .... // All in one text line, with one blank space between codes (for easy reading.)
// In real life, each chain code (0 to 7) only use 3 bits and without blank spaces between codes!
```

c) boundaryFile (use args [4]): To store the reconstructed boundary from chainCode.
// **For future processing.**

d) logFile (use args [5]): as specs dictates.

**********

III. Data structure:

**********

- A chainCode class
  - a point struct
    - (int) row
    - (int) col
  - a CCproperty struct
    - (int) label
    - (int) numPixels
    - (int) minRow, minCol, maxRow, maxCol // bounding box

  - (int) numCC  // number of connected component in the image.
  - (CCproperty) CC // for storing a connected component property.
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal

- (int [][]) imgAry // a 2D array, size of numRows+2 by numCols+2,
// to store the label image, needs to dynamically allocate at run time.
- (int [][]) boundaryAry // a 2D array, size of numRows+2 by numCols+2,
// to store the reconstructed boundary of objects in the labelled image,
// needs to dynamically allocate at run time.
- (int [][]) CCAry // a 2D array, size if numRows+2 by numCols+2, to process the chain code of each c.c.
//needs to dynamically allocate at run time.

- (point) coordOffset [8] // The index is the chain directions from currentP to its eight neighbors;
// coordOffset[i].row and coordOffset[i].col are the offset of currentP's neighbor at i direction.
// i.e., coordOffset [] are: [(0, +1), (-1, +1), (-1, 0), (-1, -1), (0, -1), (+1, -1), (+1, 0), (+1, +1)]
// So, for currentP (r, c), its neighbors at 0, 1, 2, …, 7 directions would be imgAry [r+0,][c+1], //
imgAry [r-1,][c+1],…, imgAry [r+1][ c+1)] You may *hard code* this offset array.

- (int) zeroTable [8] = [6, 0, 0, 2, 2, 4, 4, 6] // The look-up table to update the lastZero chain direction from
//currentP to nextP. You may *hard code* this table as given in the lecture.

- (point) startP // the row and col position of the first none zero pixel.
- (point) currentP // current none zero border pixel
- (point) nextP // next none-zero border pixel

- (int) lastQ // Range from 0 to 7; it is the chain direction of the last zero scanned by currentP
- (int) nextDir // the next scanning direction of currentP to find nextP, range from 0 to 7, need to mod 8.
- (int) PchainDir // chain code direction from currentP to nextP

- methods:
- constructor (…)
- zeroFramed (...) // Reuse code from previous project.
- loadImage (...) // Read from the label file onto inside frame of imgAry begin at (1,1)

- clearCCAry (...) // zero out CCAry, for next
- loadCCAry (ccLabel, CCAry) // Extract the pixels with ccLabel from imgAry to CCAry. On your own.

- getChainCode (…)// see algorithm below.
- (int) findNextP (…) // see algorithm below.

- constructBoundary (…) // on your own.
// Give the chainCode file, create an image contains only the boundary of objects in the labelled file.

- prettyDotPrint (…) // reuse code from your previous project. (The method replaces 0 with a dot ('.').
// note: the method outputs image header.
- prettyPrint (…) // reuse code from your previous project. (This replace 0 with blanks, pixels are line-up nicely).
// note: the method outputs image header.
- AryToFile (inAry, fileOut) // output the image header (same as input image header)
// output pixels inside of frame of inAry to fileOut, with blanks between pixels.
// Make sure pixels line-up nicely.

**********

IV. Main (...)

**********

Step 0: labelFile, propFile, outFile1, logFile, chainCodeFile, boundaryFile ← open via args []
numRows, numCols, minVal, maxVal ← labelFile
numRows, numCols, minVal, maxVal ← propFile // need this read, so you may proceed.
numCC ← propFile
imgAry ← dynamically allocated
zeroFramed (imgAry)
loadImage (labelFile, imgAry)
outFile1 ← "Below is the loaded imgAry of input labelFile"
prettyDotPrint (imgAry, outFile1)
CCAry ←dynamically allocated

Step 1: chainCodeFile ← numRows, numCols, minVal, maxVal // image header, one text line
chainCodeFile ← numCC // one text line

Step 2: CC.label ← propFile
CC.numPixels ← propFile
CC.minRow ← propFile
CC.minCol ← propFile
CC.maxRow ← propFile
CC.maxCol ← propFile

Step 3: clearCCAry () // zero out the old CCAry for next CC

Step 4: loadCCAry (CC.label, CCAry) // Extract the pixels with CC.label from imgAry to CCAry.
logFile ← "Below is the loaded CCAry of connected component label " // write CC.label.
prettyDotPrint CCAry, logFile)

Step 5: getChainCode (CC, CCAry, chainCodeFile, logFile)

Step 6: repeat step 2 to step 5 until all connected components are processed.

Step 7: close chainCodeFile

Step 8: reopen chainCodeFile

Step 9: constructBoundary (boundaryAry, chainCodeFile)
outFile1 ← "** Below is the objects boundaries of the input label image."
prettyDotPrint (boundaryAry, outFile1)
boundaryFile ← "** Below is the objects boundaries of the input label image."
AryToFile (boundaryAry, boundaryFile)

Step 10: close all files

**********
V. getChainCode (CC, CCAry, chainCodeFile, logFile)
**********

Step 0: logFile ← "entering getChainCode method"
chainCodeFile ← numRows, numCols, minVal, maxVal
label ← CC.label

Step 1: scan the CCAry from L to R & T to B

Step 2: if CCAry[iRow][jCol] == label // the beginning of the chain code pixel
ChainCodeFile ← output iRow, jCol, label // In one text-line
startP.row ← iRow
startP.col ← jCol
currentP.row ← iRow
currentP.col ← jCol
lastQ ← 4

Step 3: nextQ ← mod (lastQ+1, 8)

Step 4: PchainDir ← findNextP (currentP, nextQ, logFile)

Step 5: ChainCodeFile ← output PchainDir follows by a blank

Step 6: nextP.row ← currentP.row + coordOffset[chainDir].row
nextP.col ← currentP.col + coordOffset[chainDir].col
currentP ← nextP

Step 7: if PchianDir == 0
lastQ ← zeroTable[ 7]
else
lastQ ← zeroTable[PchianDir-1]

Step 8: logFile ← "lastQ = ; nextQ = ; currentP.row = ; currentP.col = : nextP.row = : nextP.col ="
// fill in all variable's values.

Step 9: repeat step 3 to step 8 until currentP == startP

Step 10: logFile ← "leaving getChainCode"

**********
VI. (int) findNextP (currentP, lastQ, logFile)
**********
Step 0: logFile ← "entering findNextP method"
Step 1: index ← lastQ
found ← false

Step 2: iRow ← currentP.row + coordOffset [index].row
jCol ← currentP.col + coordOffset [index].col

Step 3: if imgAry[iRow][jCol] == label
chainDir ← index
found ← true
else index ← mod (index+1, 8)
Step 4: logFile ← "In findNextP: index=; iRow = ; jCol= ; chainDir = ; found= ; imgAry[iRow][jCol] =" print all values.
Step 5: repeat step 2 to step 4 until (found == true)

Step 6: logFile ← "leaving findNextP method found = ; returning chainDir =" // print values

Step 7: return chainDir