Project 7 (Java): Implement the thinning algorithm as taught in class. Thinning is another method to obtain the skeletons of objects in a given binary image.

What you need to do:
1) You will have for (4) date files: data1, data2, data3 and data4 to test your program.
2) Print the image of data1 on a paper, and cycle those pixels to be the skeleton of the object.
3) Run and debug you program using data1 until your program produces the same result as your drawing.
4) Run your program three more times using data2, data3 and data4. Eyeball the results for correctness.
3) Include in your hard copies:
- cover page
- your drawing. (-1 if omitted.)
- source code
- data1 // with caption
- outFile1 for data1 // with caption
- logFile for data1 // with caption, limit to 3 pages if more
- data2 // with caption
- outFile1 for data2 // with caption
- logFile for data2 // with caption, limit to 3 pages if more
- data3 // with caption
- outFile1 for data3 // with caption
- logFile for data3 // with caption, limit to 3 pages if more
- data4 // with caption
- outFile1 for data4 // with caption
- logFile for data4 // with caption, limit to 3 pages if more

****
Programming Language: Java
****
Project name: Object skeleton via thinning
Project points: 10 pts
Due Date: (11/10) +1 early submission: 11/11/2024 Monday before midnight, 11:59pm.
(10/10) on time: 11/15/2024 Friday before midnight
(-10/10) non-submission: 11/15/2024 Friday after midnight

*** Name your soft copy and hard copy files using the naming convention given in Project Submission Requirements; -2 otherwise.
*** 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 7: Object skeleton via thinning (Java)>

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

****
I. Input: inFile (args [0]): a binary image with image header
****
II. Outputs: There are two outfiles:
a) outFile1 (args [1]): as program dictates.
b) logFile (args [2]): as program dictates.

****
III. Data structure:
****
- A Thinning class
- (int) numRows
- (int) numCols
- (int) minVal
- (int) maxVal
- (int) changeCount
- (int) cycleCount
- (int) aryOne [][] // a 2D array, need to dynamically allocate at run time of size numRows + 2 by numCols + 2.
// initialized to zero. aryOne is for checking those three conditions for thinning.

- (int) aryTwo [][] // a 2D array, need to dynamically allocate at run time of size numRows + 2 by numCols + 2
// initialized to zero. aryTwo is for storing the intermediate result after each side of thinning.

- methods:
  - constructor(...) // dynamically allocate aryOne and aryTwo, etc.
  - zeroFramed (…) // **zero framing** the two extra rows and two extra columns of aryOne and aryTwo.
  - loadImage (inFile, aryOne) // Read from the inFile and load to inside frame of aryOne, begins at (1, 1).
  - (int) countNonZeroNeighbors (...) // On your own.
    //counts non-zero neighbors of aryOne[i][j] (**excluding aryOne[i][j] itself**.), and returns count.
  - copyArys (…) // always copy aryTwo to aryOne
  - thinning (…) // call the four thinning methods to thin one layer in each iteration.
  - northThinning (…) // See algorithm below.
  - southThinning (…) // Similar to northThinning, except the different side of zero check.
  - eastThinning (…) // Similar to northThinning, except the different side of zero check.
  - westThinning (…) // Similar to northThinning, except the different side of zero check.
  - (bool) checkConnector (…) //checks the connector's six configurations (see below), if any one of six configurations
    //is true, returns true, else returns false. x means can be either 1 or 0; therefore, no need to check x.

```
x 0 x   x x x   1 0 x   x 0 1   x x x   x x x
x x x   0 x 0   0 x x   x x 0   0 x x   x x 0
x 0 x   x x x   x x x   x x x   1 0 x   x 0 1
```

  - prettyDotPrint (…) // reuse code from your previous project. (This method replaces 0 with a dot ('.'), pixels are line-up nicely.
    // 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.

****
IV. main (...)
****

Step 0: inFile, outFile1, logFile ← open via args []
numRows, numCols, minVal, maxVal ← read from inFile
outFile1 ← "input image header"
outFile1 ← write numRows, numCols, minVal, maxVal
dynamically allocate all arrays and initialize via constructor.
changeCount ← 0
cycleCount ← 0
Step 1: loadImage (inFile, **aryOne**)
Step 2: ouFile1 ← "In main(), before thinning, changeCount = ; cycleCount =" // print values.
prettyDotPrint (**aryOne**, outFile1) // using dots.
Step 3: thinning (aryOne, aryTwo, logFile)
Step 4: cycleCount ++
Step 5: ouFile1 ← "In main (), inside iteration; changeCount = ; cycleCount =" // print values.
prettyDotPrint (**aryOne**, outFile1) // using dots.
Step 6: repeat step 3 to step 5 until changeCount <= 0
Step 7: outFile1 ← "in main (), the final skeleton, changeCount = ; cycleCount =" // print values.
prettyPrint (aryOne, outFile1) // Use blank, no dots.
Step 8: close all files

****
V. thinning (aryOne, aryTwo, logFile)
****

Step 0: logFile ← "Entering thinning () before thinning 4 sides, aryOne is below:"
prettyDotPrint (**aryOne**, logFile) // using dots.
changeCount ← 0
Step 1: NorthThinning (aryOne, aryTwo, logFile)
logFile ← "after northThinning; aryTwo is below:
prettyDotPrint (**aryTwo**, logFile) // using dots.
copyArys (aryTwo, aryOne)

Step 2: SouthThinning (aryOne, aryTwo, logFile)
logFile ← "in thinning, after SouthThinning aryTwo is below:
prettyDotPrint (**aryTwo**, logFile) // using dots.
copyArys (aryTwo, aryOne)

Step 3: WestThinning (aryOne, aryTwo, logFile)
logFile ← "after WestThinning aryTwo is below
prettyDotPrint (**aryTwo**, logFile) // using dots.
copyArys (aryTwo, aryOne)

Step 4: EastThinning (aryOne, aryTwo, logFile)
logFile ← "after EastThinning aryTwo is below
prettyDotPrint (**aryTwo**, logFile) // using dots.
copyArys (aryTwo, aryOne)

Step 5: logFile ← "Leaving thinning (); cycleCount = ; changeCount = " // print values

****

VI. northThinning (aryOne, aryTwo, logFile)

****

Step 0: logFile ← "Entering northThinning (); cycleCount = ; changeCount = " // print values.
Step 1: i ← 1
Step 2: j ← 1
Step 3: if aryOne [i][j] > 0 and aryOne[i-1][j] == 0 // an object pixel and its north neighbor is zero.
nonZeroCount ← countNonZeroNeighbors (neighborAry, aryOne, i, j)
Flag ← checkConnector (neighborAry)
logFile ← "In northThinning, i=; j=; nonZeroCount=; Flag=" // print values
if nonZeroCount >= 4 and Flag is false
aryTwo[i][j] ← 0
changeCount++
else
aryTwo[i][j] ← aryOne[i][j]
Step 4: j++
Step 5: repeat step 3 to step 4 until j > numCols+1
Step 6: i++
Step 7: repeat step 2 to step 6 until i > numRows +1
Step 8: logFile ← "Leaving northThinning (); cycleCount = ; changeCount = " // print values.