Project 5: Apply thinning algorithm to a binary objects to obtain

the skeletons of objects. Run this project with each of

the given test image files.

\*\*\* YOU MUST FOLLOW THE SPEC in your implementation, include all methods given in

\*\*\* the specs and algorithm steps!!

\*\*\* NO POINT will be given if otherwise done!

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

Language: C++

Due date: soft copy: 3/7/2018 Wednesday before Midnight

+1 Early submission deadline 3/4/2018 Sunday before midnight

Due date: hard copy: 3/8/2016 Thursday in class

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

I. Input: a binary image to be given.

// Use argv[0] for the input file

II. Outputs:

You should have two output files: // Use argv[2] and argv[3].

- Use argv[1] to do:

1) Create the skeleton image from the final result of the thinning algorithm with image header for future processing.

- Use argv[2] for :

2) Pretty print the input image with proper caption.

3) Pretty print the \*second cycle\* after each thinning direction

(N, S, W, E) with proper caption (i.e., “result 2nd cycle of N thinning, …)

4) Pretty print the \*fouth cycle\* (if not finish)

after each thinning direction (N, S, W, E) with proper caption.

5) Pretty print the final result of the thinning with proper caption.

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

III. Data structure:

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

- A ThinningSkeleton class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- changeflag (bool)

- cycleCount (int)

- firstAry (int \*\*) a 2D array, need to dynamically allocate at run time

of size numRows + 2 by numCols + 2.

- secondAry (int \*\*) a 2D array, need to dynamically allocate at run time

of size numRows + 2 by numCols + 2.

- methods:

- constructor(...)

- need to dynamically allocate firstAry and secondAry

- assign values to numRows,..., etc.

- zeroFramed (Ary) // framing the extra rows and extra columns with zeros.

- loadImage

// Read from the input file onto firstAry

// the first pixel of input image is loaded

// at firstAry[1][1]

- copyAry // always copy from secondAry to firstAry

- DoThinning (i, j)

// check firstAry(i,j)'s 8 neighbors if the following

// conditions are satisfied:

// condition 1: there are 4 or more of firstAry(i,j)'s 8 neighbors >0.

**// condition 2: if flipping firstAry(i,j) from 1 to 0,**

**// it would not create two or more connected components in**

**// firstAry(i,j)'s 3x3 neighborhood.**

// then { secondAry(i,j) <-- 0

changeFlog ++

}// This method keeps the status of changeFlag

// else seconAry(i,j) <-- 1

- NorthThinning

// scan the entire image,

if p(i,j) > 1 and its north neighbor is zero

then call DoThinning within the method

else

secondary[i][j] = firstAry[i][j]

- SouthThinning

// scan the entire image,

if p(i,j) > 1 and its south neighbor is zero

then call DoThinning within the method

- WestThinning

// scan the entire image,

if p(i,j) > 1 and its west neighbor is zero

then call DoThinning within the method

- EastThinning

// scan the entire image,

if p(i,j) > 1 and its east neighbor is zero

then call DoThinning within the method

- prettyPrint(..)

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

III. Thinning Algorithms

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

step 0: read the image header

dynamically allocate firstAry and secondAry

step 1: zeroFrame(firstAry)

zeroFrame(secondAry)

step 2: loadImage (firstAry)

step 3: cycleCount <-- 0

step 4: if cycleCount is 0, or 2, or 4

prettyPrint firstAry to argv[2]

step 5: changeFlag <- false

cycleCount++

step 6: NorthThinning // look at pixels in firstAry and write the result to the secondAry

// need to update flag when there is a change.

copyAry()

step 7: SouthThinning // look at pixels in firstAry and write the result to the econdAry

// need to update flag when there is a change.

copyAry()

step 8: WestThinning // look at firstAry and write the result to secondAry

// need to update flag when there is a change.

copyAry()

step 9: EastThinning // look at firstAry and write the result to secondAry

// need to update flag when there is a change.

copyAry() // always copy from secondAry to firstAry

step 10: repeat step 4 to step 9 while changeFlag is true.

step 11: prettyPrint firstAry to argv[2]

step 12: write image header to argv[1] and copy firstAry from [1][1]

to argv[1]

step 13: close all files