Project 10 (C++): Chain code for image compression (lossless compression) and object recognition (via) boundary Pattern Analysis. Please study the two lecture notes ChainCodeA and ChainCodeB of this topic on the blackboard before reading the specs below.

Abstract: Given a connected component (CC) labeled image (contains one or more connected components) and the corresponding CC property file (i.e., these two files are the output of your connected component project.)

In the property file, it contains the number of CC in the labeled image, follows by the properties of each CC. Properties of each CC are: the cc label, the pixel count, the two points (minRow, minCol) and (maxRow, maxCol) representing the bounding box of the object location within the labelled image.

The chain-code algorithm goes as follows:

For each cc, we scan the region within the labelled image only those pixels within the bounding box (including the boarder pixels) until we find the first non-zero pixel, p(i,j), having the same label as cc’s label. We mark p(i,j) as the starting pixel. Then, starting from p(i,j), we trace the cc’s boarder pixels (must have the same label as cc), counter-clock-wise; we store the direction (with respect the chain-code direction from 0 to 7) from one boarder pixel to the next until we come to p(i,j). The process continue until all cc are traced and chain codes are produced.

\*\*\* What you need to do before running your ChainCode program:

1. You will be given two (2) binary images: chainCodeImg1 and chainCodeImg2.

chainCodeImg1 contains only one object; chainCodeImg2 contains multiple labelled objects.)

2.1. Apply your 8-connected component algorithm to chainCodeImg1 that produces a labelled image, name it chainCodeImg1\_CC and a property file, name it chainCodeImg1\_Property.

2.2 Do the same as 2.1 for chainCodeImg2

3. The labeled image file and the component property file are the two inputs of

this project.

\*\*\* If your connected component algorithm does not work correctly, you may ask for the two input files for this project.

\*\*\* What you need to do in your ChainCode program:

4. In this project, your program uses the information in the property file to extracts each connected component in the labeled image, and trace the boundary of each cc to produce chain code of each cc.

5. Run your chainCode program using chainCodeImg1\_CC and chainCodeImg1\_Property

to get chain code of each object boarder (the format for chain code output is given below.)

6. Run your chainCode program using chainCodeImg2\_CC and chainCodeImg2\_Property the same as 5 in the above.

7. Print chainCodeImg1 (no need to do chainCodeImg2) on a piece of paper. Hand-trace the boundary of chainCodeImg1, similar the illustration in Lecture notes B. Hand-trace of an image boarder could be a question on the final exam.

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

Language: Java

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

Points: 10 pts

Due Date: Soft copy and pdf hard copies:

4/23/2020 Thursday before midnight

Early submission: +1 4/20/2020 Monday before midnight

1 day late: -1 pt 4/24/2020 Friday before midnight

2 days late: -3 pts 4/25/2020 Saturday before midnight

-10 pts: after 4/25/2020 Saturday \*after midnight

\*\*\* Name your pdf file using the same naming convention as given prior

\*\*\* All on-line submission MUST include Soft copy and pdf hard copy

\*\*\* in the same email with correct file names; otherwise, it would not count as submission.

Include in your hard copy:

- Cover page

- The hand tracing of chainCodeImg1 and write the traced chain-code.

(Check to see if your hand traced chain code is the same as your program produces.

- Source Code

- ChainCodeFile for chainCodeImg1

- deBugFile for chainCodeImg1

- ChainCodeFile for chainCodeImg2

- deBugFile for chainCodeImg2

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

I. Inputs: labelFile (argv[1])

propFile argv[2]

II. Outputs:

1) ChainCodeFile (Use argv[3]): for storing the chain-code output.

// We use integers in chain code for easy programming,

// however, in real life, each chain code only use 3 bits!

\*\* Output format \*\*

#rows #cols min max // image header one text line

Label startRow startCol code1 code2 code3 ....

// one text line per connected component, no space I between codes

2) deBugFile (Use argv[3]): for debugging and for printing

the chain code in a more readable format as follows:

#rows #cols min max // one text line

Label startRow startCol // one text line

code1 code2 code3 ....

// 20 chain codes per text line with 1 space in between codes.

:

:

### // indicating the next connected component chain code

Label startRow startCol // one text line

code1 code2 code3 .... /

// 20 codes per text line with 1 space in between codes.

:

:

:

### // indicating the next connected component chain code

Label startRow startCol // one text line

code1 code2 code3 ....

// 20 chain codes per text line with 1 space in between codes.

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

III. Data structure:

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

- An image class // friend of all other classes

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- imageAry (int \*\*) a 2D array to store the label image,

needs to dynamically allocate at run time (numRows+2 by numCols+2)

- CCAry (int \*\*) a 2D array to process the chain code of each c.c.

needs to dynamically allocate at run time (numRows+2 by numCols+2)

methods:

- constructor(s)

- zeroFramed (...) // Reuse code from previous project.

- loadImage (...) // Reuse code from previous project.

// Read from the input label file onto imageAry begin at (1,1)

- A connectCC class

- label (int)

- numbpixels

- boundingBox : (minRow, minCol, maxRow, maxCol)

methods:

- constructor(s)

- clearCC (...) // zero out CCAry

- loadCC (ccLable) // Extract next CC from imageAry of the given label

// and load the connected component from imageAry (1, 1) to CCAry

- a chainCode class

- a Point class

row (int)

col (int)

- neighborCoord [8] (Point) // Give a point,p(i,j), this array

// provide the x-y coordinate of p(i,j)'s eight neigbors

// w.r.t the chain code directoions (0 to 7)

// i.e., p(i,j)'s neighbor of chain-code direction of 2,

// the neighbor's x-y coordinate would be (i-1, j)

- startP (Point)

- currentP (Point) // current none zero border pixel

- nextP (Point) // next none-zero border pixel

- lastQ (int) // Range from 0 to 7,

// it is the direction of the last zero scanned from currentP

- zeroTable[8] = [6, 0, 0, 2, 2, 4, 4, 6]

// the index is the direction from currentP to the last zero

// zeroTable[index] is the direction from nextP to the last zero.

// You may \*hard code\* this table as given in the lecture notes.

- nextDir (int)

// the next scanning direction of currentP's neighbors

// to find nextP, range from 0 to 7, need to mod 8.

- PchainDir // chain code direction from currentP to nextP

methods:

- constructors

- getChainCode(CC, CCAry)// see algorithm below.

- loadNeighborsCoord(...) // you should know how to do this

- findNextP(currentP, nextQ, nextP) // see algorithm below.

// begin chain-code construction for the given connected component

- prettyPrint // you know how to do this

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

III. Main (...)

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

// The algorithm below may contain bugs; debugging is yours, and report bugs to // Dr. Phillip

Step 1: labelFile <-- open label file from argv[1]

propFile <-- open property file from argv[2]

output image header to ChainCodeFile

output image header to deBugFile // per text line

imageAry <-- dynamically allocated

loadImage (imageAry )

CCAry <-- dynamically allocated

Step 2: CC <-- get the next connected component from the property file

Step 3: CClabel <-- get the label of CC

Step 4: clearCC () // zero out the old CClabel for next cc

Step 5: loadCC (CClabel, CCAry)

// Extract the pixels with CClabel from imageAry to CCAry.

Step 6: getChainCode (CC, CCAry) // see algorithm below

Step 7: repeat step 2 to step 5 until all connected components are processed.

Step 8: close all files

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

IV. getChainCode (CC, CCAry)

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

// The algorithm below may contain bugs; debugging is yours, and report bugs to // Dr. Phillips

Step 0: label 🡨 CC’s Label

Step 1: minRow, minCol, maxRow, maxCol <-- get from CC's property

Step 2: scan the CCAry from L to R & T to B within the bounding box

Step 3: if CCAry (iRow, jCol) == label

output iRow, jCol, CCAry (iRow, jCol) to ChainCodeFile

output iRow, jCol, CCAry (iRow, jCol) to deBugFile

// see format specs above

startP <-- (iRow, jCol)

currentP <-- (iRow, jCol)

lastQ <-- 4

step 4: nextQ <-- mod(lastQ+1, 8)

step 5: PchainDir <-- findNextP(currentP, nextQ, nextP)

currentP <-- flip the label of currentP from positive to negative

step 6: output PchainDir to ChainCodeFile // no spaces.

output PchainDir to deBugFile // see format as given above

step 7: lastQ <-- zeroTable[PchianDir-1]

currentP <-- nextP

step 8: repeat step 4 to step 7 until currentP == startP

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

III. findNextP(currentP, nextQ, nextP)

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

step 1: loadNeighborCoord(currentP)

step 2: chainDir <-- scan currentP's 8 neighbors within nighborCoord[] array from lastQ direction (mod 8) until a none zero neighbor with the same label as currentCC is found, chainDir is the index of neighborCoord[] which with the same label as currentP

// You may use the case statement given in Lecture Note B.

step 3: nextP <-- nighborCoord[chainDir]

step 4: returns chainDir