Project 6: You are to implement the Chain code algorithm as taught in class. In this project, you may assumed that the input binary image contain only one object without holes.

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

Language: C++

Due date: soft copy: 3/19/2018 Monday before midnight

Early submission deadline: 3/17/2018 Saturday before midnight

Due date: hard copy: 3/20/2018 Tuesday in class

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

I. Input (use argv[]): a binary image contain only one object without hole.

II. Outputs: You should have two output files:

outFile1 and outFile2 // Use argv[]

- outFile1: it is for storing the chain-code of the object for future image decompression.

Output format:

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

Label startRow startCol code1 code2 code3 ....

// one text line with one blank space between codes.

// In real life, each code (0 to 7) only use 3 bits

// and without blank spaces between codes!

- outFile2: it is for debugging and for printing the chain code

in a more readable format as follows:

#rows #cols min max // one text line

startRow startCol Label // one text line

code1 code2 code3 ....

// 15 chain codes per text line with one blank space in between codes.

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

III. Data structure:

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

- An image class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- imgAry (int \*\*) a 2D array to store the input image,

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

methods:

- constructor(s)

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

- zeroFramed ()

- a chainCode class

- a Point class

row (int)

col (int)

- neighborCoord[8] (Point)

// Given a point, p(i,j), this array

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

// w.r.t the chain code directions.

// 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) // the x-y coordinate of the first pixel of the object.

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

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

- lastQ (int) // Range from 0 to 7, it is the direction of

// the last zero scanned from currentP

- nextQ (int)

- nextDirTable[8] // You may \*hard code\* this table as given in class

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

// nextDirTable[index] is from nextP looking at the last zero.

- 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()

- loadNeighborsCoord(...)

- findNextP (...)

- getChainDir(currentP, nextQ) // on your own

// getChianDir will scan currentP 's 8 neighbors in

//neighborCoord [] array from nextQ direction (mod 8)

//until a none zero neighbor is found,

//chainDir is the index of neighborCoord[] of the none zero neighbor

- prettyPrint

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

III. In Main

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

step 1: inFile <-- open the input file, argv[1]

- open outFile1 and outFile2

- read image header from inFile

- output the image header to outFile1

- output image header to outFile2 // per text line

step 2: imgAry <-- dynamically allocated

step 3: loadImage()

step 4: zeroFramed ()

step 5: getChainCode()

step 6: close all files

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

IV. getChainCode algorithm steps

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

step 1: scan imgAry from L to R & T to B

step 2: if imgAry(iRow, jCol) > 0

output iRow, jCol, imgAry(iRow, jCol) to outFile1

// see format specs above

output iRow, jCol, imgAry(iRow, jCol) to outFile2

// see format specs above

startP <-- (iRow, jCol)

currentP <-- (iRow, jCol)

lastQ <-- 4

step 3: nextQ <-- mod(lastQ+1, 8)

step 4: PchainDir <-- findNextP(currentP, nextQ, nextP)

// nextP will be determined inside the findNextP method.

step 5: output PchainDir to outFile1 // see format given in the above

output PchainDir to outFile2 // see format given in the above

step 6: lastQ <-- nextDirTable[PchainDir]

currentP <-- nextP // nextP was determined inside the findNextP method.

step 7: repeat step 3 to step 6 until currentP == startP

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

III. findNextP(currentP, nextQ, nextP)

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

step 1: loadNeighborsCoord (currentP)

step 2: chainDir <-- getChainDir(currentP, nextQ)

step 3: nextP <-- neighborCoord [chainDir]

step 4: returns chainDir