Project 7: You are to implement the Chain code algorithm and the reconstruction of the object from the resulting chain code. In this project, you may assume that the input binary image contains only one object without holes.

Summary of what your program will do:

1. Opens the input file and get the header of the input image.

1. Allocate a ZeroFramedAry with extra 2 rows and extra 2 cols; initialized the array to zero.
2. Read and load the input onto ZeroFramedAry begins at (1,1).
3. Performs chain code (as taught in class) on ZeroFramedAry to produce a chainCode file, treated as a compressed file.

5) The name of the chainCode file is to be created during the run time of your program, using the original file name with an extension “\_chainCode.txt”. For example, if the name of the input file is “image”, then the name of the compressed file should be “image\_chainCode.txt”.

6) Close the chainCode file (image\_chainCode.txt)

// To make sure your program works correctly; you are going to do a reconstruction of the object from chainCode file as follows:

7) re-open the chainCode file (image\_chainCode.txt).

8) Read the image header from chainCode file and allocate a 2D imgAry, size of numRows by numCols; initialize imgAry to zero.

9) Read object’s label from the chainCode file.

10) boundaryLabel 🡨 object’s label + 2. // see if you know why.

11) Read startRow and startCol from chainCode file and assign boundaryLabel to imgAry [startRow, startCol].

12) Read, one-by-one, chain code from chainCode file to construct the boundary of the object. Assign all boundary pixels the boundaryLabel.

13) Open boundary file and output imgAry to the boundary File.

14) The name of the boundary file is to be created during the run time of your program, using the original file name with an extension “\_Boundary.txt”. For example, if the name of the input file is “image”, then the name of the compressed file should be “image\_ Boundary.txt”.

15) Fill the interior of the object: (There are a few methods to do this. Below are two suggested methods for you to try out.)

Method A: Scan the imgAry, L to R & T to B

Step 1: process next row L to R

Step 2: Find the boundary pixel whose right neighbor is 0.

Step 3: Assign 1 to that 0 pixel and all subsequent 0 pixels until reach another boundary pixel

Step 4: repeat step 2 – step 3 until the end of row.

Step 5: if the last pixel at the end of row has label 1, then you need

to back track and un-do all the background pixels with the label 1 to 0.

Step 6: repeat step 1 to step 5 until all rows are processed

Step 7: Relabeling: Scan the entire imgAry, relabel all non-zero pixels to object’s label

Method B)

1. rowBYrow: Scan the imgAry, row-by-row from L to R and T to B.

Increment all 0 pixels between two boundary points on the same row by 1.

1. colBYcol: Scan the imgAry, column-by-column from T to B and L to R.

Increment all 1 pixels between two boundary points on the same column by 1

III. Relabeling: Scan the entire imgAry left-right & top-bottom

If p(i, j) >= 2

p(i, j) 🡨 object’s label

else

p(i, j) 🡨 0

16) After fill the interior of the object, output image header and imgAry to the Decompressed file.

17) The name of the Decompressed file is to be created during the run time of your program, using the name of the input file with an extension “\_chainCodeDecompressed.txt”. i.e., if the name of the input file is “image”, then the name of the compressed file should be “image\_ chainCodeDecompres.txt”.

18) Closed all files.

// after this step your directory should have these three files: image, image\_chainCode.txt, image\_ Boundary.txt, and image\_ chainCodeDecompressed.txt.

19) If your program works correctly, image\_ chainCodeDecompressed.txt should be identical to the input file.

20) Include in your hard copies:

(a) Print input file

(b) Print chainCode file

(c) Print Boundary file

(d) Print chainCodeDecompressed file

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

Language: C++

Due date (soft copy): 4/28/2019 Sunday before Midnight

Early submission +2 deadline: 4/25/2019 Thursday before Midnight

-1 pt due: 4/29/2019 Monday before midnight

After 4/29/2019, -12 pts for all students who did not submit soft copy

Due Date (hard copy): 4/30/2019 Tuesday in class,

-1 pt for late hard copy submission after Thursday 3/21/2019 (in class).

All projects without hard copy after 4/30/2019 will receive 0 pt, even you have submitted soft copy on time and even if it works.

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

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

II. Outputs:

- console: for output the total boundary points and for error messages if any.

- chainCode file (not from argv [])

format:

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

startRow startCol Gray code1 code2 code3 ....

// all in 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!

* Decompressed file (not from argv [])

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

III. Data structure:

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

- An image class

friend of chainCode class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- label (int) // the object label

- numBoundaryPts (int) // the total number of boundary pixels.

- ZeroFramedAry (int \*\*)

// a 2D array of size numRows + 2 by numCols + 2;

// it needs to be dynamically allocate and initialized to zero

- imgAry (int \*\*) // a 2D array of size numRows by numCols

// for image reconstruction;

// it needs to be dynamically allocate,

methods:

- constructor(s)

- setZero (ZeroFramedAry) // set ZeroFramedAry to zero.

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

- a chainCode class

- a Point class

row (int)

col (int)

- startP (point)

// the x-y coordinate of the first non-zero pixel of the object.

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

- nextP (Point) // next none zero neighbor of currentP.

- neighborCoord [8] (Point)

// This array store the x-y coordinates of currentP's eight neighbors.

// Index of the array are the chain code directions from currentP.

// i.e., neighborCoord [2] has the x-y coordinate of currentP’s

// neighbor at chain-code direction 2, which is (i-1, j)

// This array is very useful for finding the next non-zero neighbor of

// currentP.

- lastQ (int) // the last zero before getting to currentP.

- DirTable [8] // A look-up table to get the next scan direction.

// You may \*hard code\* this table as given in class

- chainDir // chain code direction from currentP to nextP

methods:

- constructors

- getChainCode (ZeroFramedAry, chainCodeFile)//see algorithm below.

- int findNextP (lastQ, neighborCoord) // returns chain direction

// see algorithm below.

- loadNeighborsCoord (currentP) // on your own.

// store the xy-coordinates of currentP’s 8 neighbors in neighborCoord [].

* reconstructObject (chainCodeFile, deCompressFile, imgAry)

* constructBoundary (chainCodeFile, imgAry, Tmplabel) // on your own.

// Constructs the boundary of the object from chainCode File

- fillInterior (imgAry) // on your own.

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

IV. main (...)

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

step 0: inFile 🡨 open input file

numRows, numCols, minVal, maxVal 🡨 read from inFile

dynamically allocate imgAry size of numRows by numCols

dynamically allocate ZeroFramedAry size of numRows+2 by numCo+2

Step 1: chainCodeFileName 🡨 argv[1] + “\_chainCode.txt”

Step 2: chainCodeFile 🡨 open (chainCodeFileName)

Step 3: loadImage (inFile, ZeroFramedAry) // begins at ZeroFramedAry (1,1)

Step 4: getChainCode (ZeroFramedAry, chainCodeFile)

Step 5: close (chainCodeFile)

Step 6: re-open (chainCodeFile)

Step 7: deCompressFileName 🡨 argv[1] + “\_chainCodeDecompressed.txt”

Step 8: deCompressFile 🡨 open (deCompressFileName)

Step 9: reconstructObject (chainCodeFile, deCompressFile, imgAry)

Step 10: close all files

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

V. getChainCode (ZeroFramedAry, chainCodeFile)

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

Step 0: Write numRows, numCols, minVal, maxVal to chainCodeFile

numBoundaryPts 🡨 0

Step 1: scan ZeroFramedAry from L to R & T to B

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

output iRow, jCol, ZeroFramedAry (iRow, jCol) to chainCodeFile

// see the chain code file format in the above

currentP <-- (iRow, jCol)

lastQ <-- 4

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

step 4: chainDir <-- findNextP (currentP, lastQ)

// nextP will be determined inside the method.

NumBoundaryPts++

Step 5: output chainDir to chainCodeFile // see format given in the above

Step 6: nextP <-- neighborCoord [chainDir]

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

lastQ <-- DirTable[chainDir]

Step 7: output numBoundaryPts to **console** with caption “The total boundary pts is ”

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

VII. int findNextP (currentP, lastQ)

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

Step 0: loadNeighborsCoord (currentP)

Step 1: index 🡨 lastQ

found 🡨 false

Step 2: iRow 🡨 neighborCoord [index].row

jCol 🡨 neighborCoord [index].col

step 3: if ZeroFramedAry[iRow][jCol] == label

chainDir 🡨 index

found 🡨 true

Step 4: index 🡨 mod(index+1, 8)

Step 5: repeat step 2 - step 3 until (found == true)

Step 6: return chainDir

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

VIII. reconstructObject (chainCodeFile, deCompressFile, imgAry)

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

Step 1: numRows, numCols, minVal, maxVal 🡨 read from ChainCodeFile

Step 2: Write numRows, numCols, minVal, maxVal to deCompressFile

Step 3: label 🡨 read from chainCodeFile

Tmplabel 🡨 label + 2

Step 4: constructBoundary (chainCodeFile, imgAry, Tmplabel)

Step 5: fillInterior (imgAry)