Project 6 (C++): You are to implement both 4-connected and 8-connected component algorithms in this project. Your program let the user to choose which algorithm to run the program via argc[2]. Both algorithms consist of the following stages except which neighbors are to be checked:

1) Pass-1: (As taught in class for 8-connectness)

8-connected: check upper 3 neighbors and the left neighbor;

4-connected: check the neighbor right above, and left.

2) Pass-2: (As taught in class for 8-connectness)

8-connected: check itself, the lower 3 and the right neighbors;

4-connected: check itself, the neighbors below and the right.

3) Manage EQ table: (Algorithm was given and taught in class)

4) Pass-3: processing the entire Ary L to R & T to B, begins at (1,1)

Pass-3 accomplishes the followings:

i) re-labelling: It Uses the EQAry to relabel the connected components (CC)on the result of pass-2;

i.e., p(i,j) 🡨 EQAry[p(i,j)]

ii) During the re-labelling process, it also computes all properties for each connected component (see the list of properties below), and keep tracks newMin and newMax for the image header of the re-labelled image.

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

I. Language: C++

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

Points: 12 pts

Due Date: Soft copy and pdf hard copies: 3/22/2020 Sunday before midnight

1 day late: -1 pt 3/23/2020 Monday before midnight

2 days late: -3 pts 3/24/2020 Tuesday before midnight

-12 pts: after 3/24/2020 Tuesday after midnight

\*\*\* Name your pdf file using the same format as your soft copy except

instead of \_CPP use \_HardCopy; for example if your name is Joe Golden

then pdf name would be GoldenJ\_Project6\_HardCopy

\*\*\* All on-line submission MUST include Soft copy and pdf hard copy in the same email with proper file names.

\*\*\* Run your program twice; first with argc[2] = 4, and then with argc[2] = 8

Your hard copies includes:

-Cover page

- source code

- prettyPrintFile for 4-connectness

- labelFile for 4-connectness

- propertyFile for 4-connectness

- prettyPrintFile for 8-connectness

- labelFile for 8-connectness

- propertyFile for 8-connectness

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

I. Input (argv[1]): A binary image.

whichConnectness (argv[2])

II. Outputs: \*\* (All outfiles need to be included in your hard copy -- pdf file)

\*\* a proper caption means the caption should say what the printing is.

prettyPrintFile (argv[3]): \*\* For all prettyPrints

- Pretty print the result of the Pass-1 \*and\*

print the EQAry, with proper captions

- Pretty print the result of the Pass-2 \*and\*

print the EQAry, with proper captions

- Print the EQAry after manage the EQAry, with proper caption

- Pretty print the result of the Pass-3 \*and\*

print the EQAry with proper captions

- Pretty print the result bounding boxes drawing.

- labelFile (argv[4]): \*\* (include in your hard copy)

for the result of Pass-3 -- the labelled image file

with image header, numRows numCols newMin NewMax.

\*\* This file will be used in future processing.

- propertyFile (argv[5]): \*\* (include in your hard copy)

To store the connected component properties.

The format is to be as below:

- 1st text-line, the header of the input image,

- 2nd text-line is the total number of connected components.

- from 3rd text, use four text-lines per each connected component:

- label

- number of pixels

- upperLftR upperLftC //the r c coordinated of the upper left corner

- lowerRgtR lowerRgtC //the r c coordinated of lower right corner

For an example:

45 40 0 9 // image header

9 // there are a total of 9 CCs in the image

1 // CC label 1

187 // 187 pixels in CC label 1

4 9 // upper left corner of the bounding box at row 4 column 9

35 39 // lower right corner of the bounding box at row 35 column 39

:

:

\*\* This file will be used in future processing.

\*\*\* Run your program twice; first with argc[2] = 4, and then with argc[2] = 8

Your hard copies includes:

-Cover page

- source code

- prettyPrintFile for 4-connectness

- labelFile for 4-connectness

- propertyFile for 4-connectness

- prettyPrintFile for 8-connectness

- labelFile for 8-connectness

- propertyFile for 8-connectness

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

III. Data structure:

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

A CCLabel class

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- newMin (int)

- newMax (int)

- newLabel (int) // initialize to 0

- trueNumCC (int) // the true number of connected components in the image

// It will be determined in manageEQAry method.

- zeroFramedAry (int \*\*) // a 2D array, need to dynamically allocate

//at run time of size numRows + 2 by numCols + 2.

- numNb (int) // The number of neighbors to be looked at.

// set to 5 to be sufficient for 4- or 8-connectness.

- NonZeroNeighborAry [numNb]

// For easy programming, a 1-D array is used

// to store pixel(i, j)’s non-zero neighbors

- EQAry (int \*)

// an 1-D array, of size (numRows \* numCols) / 2

// dynamically allocate at run time

// and initialize to its index, i.e., EQAry[i] = i.

- Property (1D struct or class)

- label (int) // The component label

- numpixels (int) // total number of pixels in the cc.

- upperLftR (int)

- upperLftC (int)

- lowerRgtR (int)

- lowerRgtC (int)

// In the Cartesian coordinate system, any rectangular box can be //represented by two points: upper-left corner and the lower-right //corner of the box. Here, the two points:(upperLftR upperLftC) and //(lowerRgtR lowerRgtC) represents the smallest rectangular box that //the cc can fit inside the box.

- CCproperty (\*Property)

// A struct 1D array for storing all components’ properties.

// The size of array is the actual number of cc after manageEQAry

- methods:

- constructor(...) // need to dynamically allocate all arrays;

and assign values to numRows,..., etc.

- zero2D (...) // \*\* Initialized a 2-D array to zero.

- minus1D (...) // \*\* Initialized a 1-D array to -1.

- loadImage (...)

// read from input file and write to zeroFramedAry begin at(1,1)

- connectPass1 (...) // See algorithm below.

- connectPass2 (...) // See algorithm below.

- (int) loadNonZeroPass1 (...)

// see algorithm below.

- (int) loadNonZeroPass2 (...)

// see algorithm below

- connectPass3 (...) // There is no differences between 4-connectness and

// 8-connectness. See the description in the above.

// On your own

- drawBoxes (...) // Draw the bounding boxes on all connected components

// in zeroFramedAry. See algorithm below

- updateEQ (...) // Update EQAry for all non-zero neighbors to minLabel

- (int) manageEQAry (...) // on your own

// The algorithm was taught and given in class.

// The method returns the true number of CCs in

// the labelled image.

- printCCproperty (...) // print the component properties to propertyFile

// using the format given in the above.

// you should know how to do this.

- prettyPrint (...) // prettyPrint zeroFramedAry to prettyPrintFile

// Do NOT need to print image header

// and print the entire zeroFramedAry entirely

// if p(i, j) > 0

output p(i, j) follows by 1 blank space

else

output needed blank spaces to line-up with

labelled p(i, j) so the image will

look “pretty” on the hard copies.

- printEQAry (...) // Print EQAry with index up to newLable or trueLable, not beyond.

- printImg (...) // Output image header and zeroFramedAry to labelFile

//begins at (1,1) and end at ??

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

IV. main(...)

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

step 0: inFile 🡨 open the input file

prettyPrintFile, labelFile, propertyFile 🡨 open from argc[]

numRows, numCols, minVal, maxVal 🡨 read from inFile

dynamically allocate zeroFramedAry.

step 1: zero2D (zeroFramedAry) // \*\* Initialized zeroFramedAry to zero.

step 2: loadImage(inFile, zeroFramedAry)

// read from input file and write to zeroFramedAry begin at(1,1)

step 3: Connectness 🡨 from argv[2]

step 4: newLabel 🡨 connectPass1 (Connectness, zeroFramedAry, NonZeroNeighborAry)

// see algorithm below

step 5: prettyPrint (prettyPrintFile) // Print zeroFramedAry to prettyPrintFile

printEQAry (newLable, prettyPrintFile)

// print the EQAry up to newLable with proper caption

step 6: connectPass2 (Connectness, zeroFramedAry, NonZeroNeighborAry)

step 7: prettyPrint (prettyPrintFile) // Print zeroFramedAry to prettyPrintFile

printEQAry (newLable, prettyPrintFile)

// print the EQAry up to newLable with proper caption

step 8: manageEQAry (EQAry, newLabel)

printEQAry (numCCLable, prettyPrintFile)

// print the EQAry up to newLabel with proper caption

step 9: connectPass3 (...) // See algorithm below

prettyPrint (prettyPrintFile) // Print zeroFramedAry to prettyPrintFile

printEQAry (numCCLable, prettyPrintFile)

// print the EQAry up to numCCLabel with proper caption

step 10: output numRows, numCols, newMin, newMax to labelFile

step 11: printImg (labelFile) // Output the result of pass3 from zeroFramedAry to //labelFile, begins at zeroFramedAry[1, 1] and ending at ??

step 12: printCCproperty (propertyFile) // print cc properties to propertyFile

step 13: drawBoxes(zeroFramedAry, CCproperty)

step 14: prettyPrint (prettyPrintFile) // Print zeroFramedAry to prettyPrintFile

step 15: close all files

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

V. (int) connectPass1(Connectness, Ary, nonZeroAry)

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

step 0: newLabel 🡨 0

step 1: scan the Ary L to R & T to B, begins at (1,1), and ends at (?, ?)

// use index i for row and index j for column

step 2: Ary(i,j) 🡨 next pixel

Step 3: if Ary(i,j) > 0

minLabel 🡨 loadNonZeroPass1 (Ary, Connectness, i, j, nonZeroAry, diffFlag, nonZeroCount)

if minLabel == 0 // case 1

newLabel++

Ary[i,j] 🡨 newLabel

else if minLabel > 0 // case 2 and 3

Ary[i,j] 🡨 minLabel

if (diffFlag == true) // 2 or more difference nonzero labels

updateEQ(EQAry, NonZeroAry, minLabel, nonZeroCount)

step 4: repeat step 2 – step 3 until all pixels inside the framed Ary are processed.

step 5: return newLabel

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

VI. connectPass2(Connectness, Ary, nonZeroAry)

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

step 1: scan the Ary R to L & B to T, begins at (numRows, numCols), and ends at (1, 1)

// using index i for row and index j for column

step 2: Ary(i,j) 🡨 next pixel

Step 3: if Ary(i,j) > 0

minLabel 🡨 loadNonZeroPass2 (Ary, Connectness, i, j, nonZeroAry, diffFlag, nonZeroCount)

if minLabel != Ary[i, j] // not equal to itself

Ary[i,j] 🡨 minLabel

if (diffFlag == true)

updateEQ(EQAry, NonZeroAry, minLabel, nonZeroCount)

step 4: repeat step 2 – step 3 until all pixels inside the framed image are processed.

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

(int) loadNoneZeroPass1 (Ary, Connectness, i, j, nonZeroAry, diffFlag, nonZeroCount)

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

step 0: minus1D (nonZeroAry)// reset nonZeroAry array to -1, very important!!

nonZeroCount 🡨 0

step 1: if Ary[i-1, j] > 0 // check the pixel above

nonZeroAry [nonZeroCount] 🡨 Ary[i-1, j]

nonZeroCount ++

step 2: if Ary[i, j-1] > 0 // check the pixel on the left

nonZeroAry [nonZeroCount] 🡨 Ary[i, j-1]

nonZeroCount ++

step 3: if Connectness == 8 // 8-connectness needs to check two more neighbors

if Ary[i-1, j-1] > 0 // check the pixel on the upperLeft

nonZeroAry [nonZeroCount] 🡨 Ary[i-1, j-1]

nonZeroCount ++

if Ary[i-1, j+1] > 0 // check the pixel on the upperRight

nonZeroAry [nonZeroCount] 🡨 Ary[i-1, j+1]

nonZeroCount ++

step 4: if nonZeroCount <= 0

return 0

step 5: minLabel 🡨 nonZeroAry [0]

diffFlag 🡨 false

step 6: index 🡨 1

// step 7 and step 8 below should be inside the while loop

// otherwise, you may found index out of bound

step 7: if minLabel > nonZeroAry [index]

minLabel 🡨 nonZeroAry [index]

diffFlag 🡨 true

step 8: index ++

step 9: repeat step 7 – step 8 while index < nonZeroCount

step 10: return minLabel

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

(int) loadNonZeroPass2 (Ary, Connectness, i, j, nonZeroAry, diffFlag) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: minus1D (nonZeroAry)// reset nonZeroAry array to -1, very important!!

nonZeroCount 🡨 0

step 1: nonZeroAry [nonZeroCount] 🡨 Ary[i, j] // needs to include pixel itself

NonZeroCount ++ // count p(i, j) itself

step 2: if Ary[i+1, j] > 0 // check the pixel below

nonZeroAry [nonZeroCount] 🡨 Ary[i+1, j]

NonZeroCount ++

step 3: if zeroFramedAry[i, j+1] > 0 // check the pixel on the right

nonZeroAry [nonZeroCount] 🡨 Ary[i, j+1]

NonZeroCount ++

step 4: if Connectness == 8 // 8-connectness needs to check two more neighbors

if Ary[i+1, j-1] > 0 // check the pixel on the lowerLeft

nonZeroAry [nonZeroCount] 🡨 Ary[i+1, j-1]

NonZeroCount ++

if zeroFramedAry[i+1, j+1] > 0 // check the pixel on the lowerRight

nonZeroAry [nonZeroCount] 🡨 Ary[i+1, j+1]

NonZeroCount ++

step 5: minLabel 🡨 nonZeroAry [0]

diffFlag 🡨 false

step 6: index 🡨 1

// step 7 and step 8 below should be inside the while loop

// otherwise, you may found index out of bound

step 7: if minLabel > nonZeroAry [index]

minLabel 🡨 nonZeroAry [index]

diffFlag 🡨 true

step 8: index ++

step 9: repeat step 7 – step 8 while index < nonZeroCount

step 10: return minLabel

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

VII. updateEQ(EQAry, nonZeroAry, minLabel, nonZeroCount)

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

step 0: index 🡨 0

step 1: EQAry[nonZeroAry [index]] 🡨 minLabel

step 2: index ++

step 3: repeat step 1 to step 2 while index < nonZeroCount and nonZeroAry [index]!= -1

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

connectPass3 (...)

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

// processing the entire Ary L to R & T to B, begins at (1,1)

Pass-3 accomplishes the followings:

i) re-labelling: It Uses the EQAry to relabel the connected components (CC)on the result of pass-2;

i.e., p(i,j) 🡨 EQAry[p(i,j)]

ii) During the re-labelling process, it also computes all properties for each connected component (see the list of properties below), and keep tracks newMin and newMax for the image header of the re-labelled image.

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

V. drawBoxes(zeroFramedAry, CCproperty)

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

step 1: index 🡨 1

step 2: minRow 🡨 CCproperty[i].minRow // need to add 1

minCol 🡨 CCproperty[i].minCol // need to add 1

maxRow 🡨 CCproperty[i].maxRow // need to add 1

maxCol 🡨 CCproperty[i].maxCol // need to add 1

label 🡨 CCproperty[i].label

step 3: Assign all pixels on minRow from minCol to maxCol 🡨 label

Assign all pixels on maxRow from minCol to maxCol 🡨 label

Assign all pixels on minCol from minRow to maxRow 🡨 label

Assign all pixels on maxCol from minRow to maxRow 🡨 label

step 4: index++

step 5: repeat step 2 – step 4 while index is within the number of cc.