**CSC 381Image Processing (CPP)**

**Project: Thinning**

**Student: Jeffrey Pulinat**

**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**

step 0: read the image header

dynamically allocate firstAry and secondAry

step 1: loadImage (firstAry)

step 2: zeroFrame(firstAry)

zeroFrame(secondAry)

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

**B) Source Code:**

#include <iostream>

#include <fstream>

#include <string>

#include <stdlib.h>

**using** **namespace** std**;**

static int numRows**;**

static int numCols**;**

static int minVal**;**

static int maxVal**;**

static bool changeFlag**;**

static int cycleCount**;**

static int**\*\*** firstAry**;**

static int**\*\*** secondAry**;**

static int newMin**;**

static int newMax**;**

static int totalComp**;**

static int numOfConComp **=** 0**;**

static int neighborAry **[**5**];**

static int nAry**[**8**];**

static int**\*** EQAry**;**

static int newLabel **=** 0**;**

static int**\*\*** tempAry**;**

void zeroFramed**(**int **\*\***ary**,** int nRows**,** int nCols**){**

**for(**int col**=**0**;** col**<=**nCols**+**1**;** col**++){**

ary**[**0**][**col**]** **=** 0**;** //top

ary**[**nRows**+**1**][**col**]** **=** 0**;** //bottom

**}**

**for(**int row**=**0**;** row**<=**nRows**+**1**;** row**++){**

ary**[**row**][**0**]** **=** 0**;** //left side

ary**[**row**][**nCols**+**1**]** **=** 0**;** //right side

**}**

**}**

void loadImage**(**ifstream**&** input**){**

int param**;**

input **>>** param**;**

numRows **=** param**;**

input **>>** param**;**

numCols **=** param**;**

input **>>** param**;**

minVal **=** param**;**

input **>>** param**;**

maxVal **=** param**;**

firstAry **=** **new** int **\*[**numRows**+**2**];**

**for(**int i**=**0**;** i**<**numRows**+**2**;** **++**i**){**

firstAry**[**i**]** **=** **new** int**[**numCols**+**2**];**

**}**

secondAry **=** **new** int **\*[**numRows**+**2**];**

**for(**int i**=**0**;** i**<**numRows**+**2**;** **++**i**){**

secondAry**[**i**]** **=** **new** int**[**numCols**+**2**];**

**}**

**for(**int row **=** 1**;** row**<=** numRows**;** row**++){**

**for(**int col **=** 1**;** col**<=** numCols**;** col**++){**

int value**;**

input **>>** value**;**

firstAry**[**row**][**col**]** **=** value**;**

**}**

**}**

**}**

void loadNeighbors**(**int**\*\*** ary**,** int row**,** int col**,** int pass**)** **{** //load 3x3 neighbors

/\* a|b|c

\* d|e|f e = p(i,j)

\* g|h|i

\*/

**if(**pass**==**1**){**

neighborAry**[**0**]=**ary**[**row**-**1**][**col**-**1**];** //a

neighborAry**[**1**]=**ary**[**row**-**1**][**col**];** //b

neighborAry**[**2**]=**ary**[**row**-**1**][**col**+**1**];** //c

neighborAry**[**3**]=**ary**[**row**][**col**-**1**];** //d

//neighborAry[4]=zeroFramedAry[row][col]; //e p(i,j)

neighborAry**[**4**]=**0**;** //setting to 0 to check if disconnect component.

**}**

**if(**pass**==**2**){**

neighborAry**[**4**]=**0**;** //setting to 0 to check if disconnect component.

//neighborAry[4]=zeroFramedAry[row][col]; //e p(i,j)

neighborAry**[**0**]=**ary**[**row**][**col**+**1**];** //f

neighborAry**[**1**]=**ary**[**row**+**1**][**col**-**1**];** //g

neighborAry**[**2**]=**ary**[**row**+**1**][**col**];** //h

neighborAry**[**3**]=**ary**[**row**+**1**][**col**+**1**];** //i

**}**

**}**

int loadNeighbors8**(**int**\*\*** ary**,** int row**,** int col**)** **{** //load 3x3 neighbors

int max **=** 0**;**

int index**=**0**;**

**for(**int i **=-**1**;** i**<=**1**;** i**++){**

**for(**int j**=-**1**;** j**<=**1**;** j**++){**

nAry**[**index**]=**ary**[**row**+**i**][**col**+**j**];**

index**++;**

**if(**nAry**[**index**]>**max**)**max**=**nAry**[**index**];**

**}**

**}**

**return** max**;**

**}**

void updateEQAry**(**int index**,** int value**)** **{**

EQAry**[**index**]** **=** value**;**

**}**

bool zeroNeighbors**(){** //case1

bool allZero **=** **true;**

**for(**int i**=**0**;** i**<=**3**;** i**++){** //0 - 3 excludes p(i,j)

**if** **(**neighborAry**[**i**]** **!=** 0**){**

allZero **=** **false;**

**break;**

**}**

**}**

**return** allZero**;**

**}**

int equalNeighbors**(**int bound**){** //case 2

bool equal **=** **true;**

int value **=** 0**;**

int count **=** 1**;** // keeps track of first non zero element

**for(**int i**=**0**;** i**<=**bound**;** i**++){** //0-3 excludes p(i,j) 0-4 includes p(i,J)

**if** **(**neighborAry**[**i**]** **>** 0 **&&** count **==** 1 **){**

value **=** neighborAry**[**i**];**

count**++;**

**}**

**if(**count**==**2 **&&** neighborAry**[**i**]** **!=** 0 **&&** neighborAry**[**i**]** **!=** value**){**

equal **=** **false;**

value **=** **-**1**;**

**break;**

**}**

**}**

//return equal; Used for my own readability

**return** value**;**

**}**

void ConnectCC\_Pass3**(**int**\*\*** ary**,** int nRows**,** int nCols**)** **{**

**for(**int row **=** 1**;** row**<=**nRows**;** row**++){**

**for(**int col **=** 1**;** col**<=**nCols**;** col**++){**

int EQindex **=** ary**[**row**][**col**];**

int EQvalue **=** EQAry**[**EQindex**];**

ary**[**row**][**col**]=**EQvalue**;**

**}**

**}**

**}**

void ConnectCC\_Pass2**(**int**\*\*** ary**,** int nRows**,** int nCols**)** **{**

int min**;**

bool anyzero **=** **false;**

**for(**int row **=** nRows**;** row**>=**1**;** row**--){**

**for(**int col **=** nCols**;** col**>=**1**;** col**--){**

**if(**ary**[**row**][**col**]>**0**){** //p(i,j) is not 0

loadNeighbors**(**ary**,**row**,**col**,**2**);**

**if(**zeroNeighbors**()==true){** //CASE 1: all neighbors are 0

//do nothing

**}**

**else** **if(**equalNeighbors**(**4**)!=** **-**1**){** //CASE 2: neighbors are equal including (i,j)

//do nothing

**}else** **{** //CASE 3 neighbors are diff values

min **=** 9999**;**

**for(**int i**=**0**;** i**<**5**;** i**++)** **{**

**if(**neighborAry**[**i**]** **>** 0 **&&** neighborAry**[**i**]<**min**)** **{**

min **=** neighborAry**[**i**];**

**}**

**}**

int index **=** ary**[**row**][**col**];**

ary**[**row**][**col**]=**min**;**

updateEQAry**(**index**,**min**);** //(index,value)

**}**

**}else** anyzero **=** **true;**

**}**

**}**

**if(**anyzero**==false)**newMin**=**1**;**

**else** newMin**=**0**;**

newMax**=**newLabel**;**

**}**

void ConnectCC\_Pass1**(**int**\*\*** ary**,** int nRows**,** int nCols**)** **{**

int min**;**

bool anyzero **=** **false;**

**for(**int row **=** 1**;** row**<=**nRows**;** row**++){**

**for(**int col **=** 1**;** col**<=**nCols**;** col**++){**

**if(**ary**[**row**][**col**]>**0**){** //p(i,j) is not 0

loadNeighbors**(**ary**,**row**,**col**,**1**);**

**if(**zeroNeighbors**()==true){** //CASE 1: all neighbors are 0

newLabel**++;**

ary**[**row**][**col**]=**newLabel**;**

**}**

**else** **if(**equalNeighbors**(**3**)!=** **-**1**){** //CASE 2: neighbors are equal

ary**[**row**][**col**]=**equalNeighbors**(**3**);**

**}else** **{** //CASE 3 neighbors are diff values

min **=** 9999**;**

int max **=** 0**;**

**for(**int i**=**0**;** i**<=**3**;** i**++)** **{** //doesnt compare p(i,j)

**if(**neighborAry**[**i**]** **>** 0 **&&** neighborAry**[**i**]<**min**)** **{**

min **=** neighborAry**[**i**];**

**}**

**if(**neighborAry**[**i**]** **>** 0 **&&** neighborAry**[**i**]>**max**)** **{**

max **=** neighborAry**[**i**];**

**}**

**}**

ary**[**row**][**col**]=**min**;**

updateEQAry**(**max**,**min**);** //(index,value)

**}**

**}else** anyzero **=** **true;**

**}**

**}**

**if(**anyzero**==false)**newMin**=**1**;**

**else** newMin**=**0**;**

newMax**=**newLabel**;**

**}**

void manageEQAry**()** **{**

int unique **[**newLabel**];**

totalComp**=**0**;**

**for(**int i**=**0**;** i**<=**newLabel**;** i**++){**

int j**;**

**for** **(**j **=** 0**;** j **<** i**;** j**++){**

**if** **(**EQAry**[**i**]** **==** EQAry**[**j**])break;**

**}**

//fill array with unique values

**if** **(**i **==** j**){**

unique**[**totalComp**]=**EQAry**[**i**];**

totalComp**++;**

**}**

**}**

**for(**int i **=**0**;** i**<**totalComp**;** i**++){**

**for(**int j**=**0**;** j**<=**newLabel**;**j**++){**

**if(**EQAry**[**j**]==**unique**[**i**]){**

EQAry**[**j**]=**i**;**

**}**

**}**

**}**

newMin**=**0**;**

newMax**=**totalComp**-**1**;**

**for(**int i**=**1**;**i**<=**totalComp**;**i**++){**

numOfConComp**++;**

**}**

**}**

void Thinning**(**string side**,** int **\*\***ary**){**

**for(**int row **=** 1**;** row**<=**numRows**;** row**++){**

**for(**int col **=** 1**;** col**<=**numCols**;** col**++){**

int borderVal**;**

**if(**side**==**"north"**)**borderVal**=**ary**[**row**-**1**][**col**];**

**if(**side**==**"south"**)**borderVal**=**ary**[**row**+**1**][**col**];**

**if(**side**==**"east"**)**borderVal**=**ary**[**row**][**col**+**1**];**

**if(**side**==**"west"**)**borderVal**=**ary**[**row**][**col**-**1**];**

**if(** **(**ary**[**row**][**col**]>**0**)** **&&** **(**borderVal**==**0**)** **){**

int count **=**0**;**

**for(**int i**=-**1**;** i**<=**1**;** i**++){**

**for(**int j**=-**1**;** j**<=**1**;** j**++){**

**if(**firstAry**[**row**+**i**][**col**+**j**]>=**0**)**count**++;** //will count itself p(i,j)

**}**

**}**

count**--;** //take away the count of itself p(i,j)

**if(**count**>=**4**){**

tempAry **=** **new** int **\*[**5**];** //3+2 =5 3x3 zeroFrame

**for(**int i**=**0**;** i**<**5**;** **++**i**){**

tempAry**[**i**]** **=** **new** int**[**5**];**

**}**

zeroFramed**(**tempAry**,**3**,** 3**);**

//use 8 connected components algorithms

EQAry **=** **new** int **[(**3**\***3**)/**2**];**

ConnectCC\_Pass1**(**tempAry**,** 3**,** 3**);**

ConnectCC\_Pass2**(**tempAry**,** 3**,** 3**);**

manageEQAry**();**

/\*ConnectCC\_Pass3(tempAry, 3, 3);

for(int row = 1; row<=3; row++){

for(int col = 1; col<=3; col++){

if(loadNeighbors8(ary,row,col)>1){

secondAry[row][col]=0;

changeFlag==true;

}else{

secondAry[row][col]=1;

}

}

}\*/

**if(**numOfConComp**<=**1**){**

secondAry**[**row**][**col**]=**0**;**

changeFlag**==true;**

**}**

**else{**

secondAry**[**row**][**col**]=**1**;**

**}**

**}**

**}**

**}**

**}**

**}**

// always copy from secondAry to firstAry

void copyAry**(){**

**for(**int row **=** 0**;** row**<=**numRows**+**1**;** row**++){**

**for(**int col **=** 0**;** col**<=**numCols**+**1**;** col**++){**

firstAry**[**row**][**col**]=**secondAry**[**row**][**col**];**

**if(**minVal**>**secondAry**[**row**][**col**])**minVal**=**secondAry**[**row**][**col**];**

**if(**maxVal**<**secondAry**[**row**][**col**])**maxVal**=**secondAry**[**row**][**col**];**

**}**

**}**

**}**

void prettyPrint**(**int **\*\***ary**,** ofstream**&** output**,** bool zborder**,** bool iheader**){**

**if(**iheader**==true)**output**<<**endl**<<**numRows**<<**" "**<<**numCols**<<**" "**<<**minVal**<<**" "**<<**maxVal**;** //prints image params

output**<<**endl**;**

**if(**zborder**==false){**

**for(**int row **=** 1**;** row**<=**numRows**;** row**++){** //print out from [1][1] w/o border

**for(**int col **=** 1**;** col**<=**numCols**;** col**++){**

int pixel\_val **=** firstAry**[**row**][**col**];**

**if(**pixel\_val **>** 0**)** output **<<** pixel\_val **<<**" "**;** //if not 0. output 1 space,

**else** output **<<** " "**;** // if 0. output 2 space

**}**

output**<<**endl**;**

**}**

**}**

**if(**zborder**==true){**

**for(**int row **=** 0**;** row**<=**numRows**+**1**;** row**++){** //print out from [0][0] with border

**for(**int col **=** 0**;** col**<=**numCols**+**1**;** col**++){**

int pixel\_val**=** firstAry**[**row**][**col**];**

**if(**pixel\_val **>** 0**)** output **<<** pixel\_val **<<**" "**;** //if not 0. output 1 space,

**else** output **<<** " "**;** // if 0. output 2 space

**}**

output**<<**endl**;**

**}**

**}**

**}**

int main**(**int argc**,** char **\***argv**[])** **{**

ifstream input**(**argv**[**1**]);**

string arg1 **=**argv**[**1**];**

ofstream output2**(**argv**[**3**]);** //outFile2

loadImage**(**input**);**

zeroFramed**(**firstAry**,** numRows**,** numCols**);**

zeroFramed**(**secondAry**,** numRows**,** numCols**);**

cycleCount**=**0**;**

changeFlag**==true;**

**while(true){**

**if(**cycleCount**%**2 **==** 0**){**

output2**<<**cycleCount**<<**" cycle of Thinning"**<<**endl**;**

prettyPrint**(**firstAry**,**output2**,false,false);**

**}**

changeFlag**==false;**

cycleCount**++;**

Thinning**(**"north"**,** firstAry**);**

copyAry**();**

Thinning**(**"south"**,** firstAry**);**

copyAry**();**

Thinning**(**"west"**,** firstAry**);**

copyAry**();**

Thinning**(**"east"**,** firstAry**);**

copyAry**();**

**if(**changeFlag**==false)break;**

**}**

output2**<<**cycleCount**<<**" cycle of Thinning"**<<**endl**;**

prettyPrint**(**firstAry**,**output2**,false,false);**

ofstream output1**(**argv**[**2**]);** //outFile2

prettyPrint**(**firstAry**,**output1**,true,true);**

input**.**close**();**

output2**.**close**();**

output1**.**close**();**

**for** **(**int i **=** 0**;** i **<** numRows**;** **++**i **)delete** **[]** firstAry**[**i**];**

**delete** **[]** firstAry**;**

**}**

**OUTPUT FILES**

**Data1outFile1.txt**

15 19 0 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**Data1outFile2.txt**

0 cycle of Thinning

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

2 cycle of Thinning

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

**Data2Outfile1.txt**

20 19 0 1

1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

**Data2Outfile2.txt**

0 cycle of Thinning

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

2 cycle of Thinning

1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

**Data3Outfile1.txt**

17 17 0 1

1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

**Data3Outfile2.txt**

0 cycle of Thinning

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1

1 1 1

1

1 cycle of Thinning

1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

**Data4Outfile1.txt**

17 17 0 1

1

1 1

1 1

1 1

1 1

1 1

**Data4Outfile2.txt**

0 cycle of Thinning

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1

1 1 1

1

1 cycle of Thinning

1

1 1

1 1

1 1

1 1

1 1