**CS381-37: Project 5 (CPP)**

**Yida Tao**

**Due date: Feb. 07, 2018**

Algorithm Steps:

step 1: zeroFrame(firstAry)

zeroFrame(secondAry)

step 2: loadImage (firstAry)

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

copyAry()

step 7: SouthThinning

copyAry()

step 8: WestThinning

copyAry()

step 9: EastThinning

copyAry()

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

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class ThinningSkeleton{

private:

int numRows;

int numCols;

int minVal;

int maxVal;

int \*\*firstAry;

int \*\*secondAry;

public:

int cycleCount;

bool changeflag;

ThinningSkeleton(string input){

numRows = 0;

numCols = 0;

minVal = 0;

maxVal = 0;

cycleCount = 0;

changeflag = true;

ifstream inFile;

inFile.open(input);

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

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

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

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

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

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

for(int j = 0; j < numCols+2; j++){

firstAry[i][j] = 0;

secondAry[i][j] = 0;

}

}

int num = 0;

int counter = 0;

int r = 0;

int c = 0;

while(inFile>>num){

r = counter/numCols + 1;

c = counter%numCols + 1;

firstAry[r][c] = num;

counter++;

}

inFile.close();

}

~ThinningSkeleton(){

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

delete firstAry[i];

delete secondAry[i];

}

delete[] firstAry;

delete[] secondAry;

}

void northThinning(){

for(int i = 1; i < numRows + 1; i++){

for(int j = 1; j < numCols + 1; j++){

if(firstAry[i][j] > 0 && firstAry[i-1][j] == 0){

DoThinning(i,j);

}

else{

secondAry[i][j] = firstAry[i][j];

}

}

}

copyAry();

}

void southThinning(){

for(int i = 1; i < numRows + 1; i++){

for(int j = 1; j < numCols + 1; j++){

if(firstAry[i][j] > 0 && firstAry[i+1][j] == 0){

DoThinning(i,j);

}

else{

secondAry[i][j] = firstAry[i][j];

}

}

}

copyAry();

}

void westThinning(){

for(int i = 1; i < numRows + 1; i++){

for(int j = 1; j < numCols + 1; j++){

if(firstAry[i][j] > 0 && firstAry[i][j-1] == 0){

DoThinning(i,j);

}

else{

secondAry[i][j] = firstAry[i][j];

}

}

}

copyAry();

}

void eastThinning(){

for(int i = 1; i < numRows + 1; i++){

for(int j = 1; j < numCols + 1; j++){

if(firstAry[i][j] > 0 && firstAry[i][j+1] == 0){

DoThinning(i,j);

}

else{

secondAry[i][j] = firstAry[i][j];

}

}

}

copyAry();

}

void prettyPrint(){

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

for(int j = 0; j < numCols + 2; j++){

if(firstAry[i][j] > 0){

cout << firstAry[i][j] <<" ";

}

else{

cout <<" ";

}

}

cout << endl;

}

}

void printResult(){

cout<< numRows << " " << numCols << " " << minVal << " " << maxVal << endl;

for(int i = 1; i < numRows + 1; i++){

for(int j = 1; j < numCols + 1; j++){

if(firstAry[i][j] < 10){

cout << firstAry[i][j] <<" ";

}

else{

cout << firstAry[i][j] <<" ";

}

}

cout << endl;

}

}

private:

void copyAry(){

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

for(int j = 0; j < numCols + 2; j++){

firstAry[i][j] = secondAry[i][j];

}

}

}

void DoThinning(int r, int c){

int nonZeroNeighbors = countNeighbors(r,c);

bool check = true;

//check 4 corners,if corners > 0 and corners' adjacent == 0, check = false

//top left

if(firstAry[r-1][c-1]>0 && firstAry[r][c-1]==0 && firstAry[r-1][c]==0){

check = false;

}

//top right

if(firstAry[r-1][c+1]>0 && firstAry[r][c+1]==0 && firstAry[r-1][c]==0){

check = false;

}

//bottom left

if(firstAry[r+1][c-1]>0 && firstAry[r][c-1]==0 && firstAry[r+1][c]==0){

check = false;

}

//bottom right

if(firstAry[r+1][c+1]>0 && firstAry[r][c+1]==0 && firstAry[r+1][c]==0){

check = false;

}

//top and bottom

if(firstAry[r-1][c]==0 && firstAry[r+1][c]==0){

check = false;

}

//left and right

if(firstAry[r][c-1]==0 && firstAry[r][c+1]==0){

check = false;

}

if(nonZeroNeighbors >= 4 && check){

secondAry[r][c] = 0;

changeflag = true;

}

else{

secondAry[r][c] = 1;

}

}

int countNeighbors(int r, int c){

int counter = 0;

for(int i = -1; i < 2; i++){

for(int j = -1; j < 2; j++){

if(firstAry[r+i][c+j] > 0){

counter++;

}

}

}

return counter-1; //dont count itself

}

};

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

if(argv[1]==NULL) {

cout<<"no parameter"<<endl;

return 0;

}

ofstream out1;

out1.open(argv[2]);

ofstream out2;

out2.open(argv[3]);

streambuf \*console = cout.rdbuf();

cout.rdbuf(out2.rdbuf());

//step 0,1,2

ThinningSkeleton \*ts = new ThinningSkeleton(argv[1]);

//step 3

ts->cycleCount = 0;

//step 10 : 4 - 9

while(ts->changeflag){

//step 4

if(ts->cycleCount == 0 ||ts->cycleCount == 2 ||ts->cycleCount == 4){

cout<<"Cycle Count:"<< ts->cycleCount <<endl;

ts->prettyPrint();

}

//step 5

ts->changeflag = false;

ts->cycleCount ++;

//step 6-9

ts->northThinning();

ts->southThinning();

ts->westThinning();

ts->eastThinning();

}

//step 11

cout<<"Final Cycle Count:"<< ts->cycleCount <<endl;

ts->prettyPrint();

//step 12

cout.rdbuf(out1.rdbuf());

ts->printResult();

//step 13

cout.rdbuf(console);

cout<<"done"<<endl;

out1.close();

out2.close();

delete ts;

return 0;

}

**Input2**

20 19 0 1

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

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

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

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

**Output2\_1**

20 19 0 1

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 1 0 1 0 1 1 1 0 0 0 0 0

0 0 1 1 1 0 0 0 0 1 0 0 0 0 1 1 1 0 0

1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 1

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 0 0

0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0

1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1

**Output2\_2**

Cycle Count: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 1 1 1 1 1 1 1 1 1 1 1

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

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

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

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

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

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

1 1 1 1 1 1 1 1 1 1 1 1 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 Count:2

1

1

1

1

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1

1

1

1

1

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1

Final Cycle Count:4

1

1

1

1

1

1

1 1 1

1 1 1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 1 1

1

1

1

1

1

1

1 1 1

1 1 1 1 1 1

1 1 1 1 1 1

1 1 1 1

**Input4**

17 17 0 1

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0

0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0

0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0

0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

**Output4\_1**

17 17 0 1

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0

**Output4\_2**

Cycle Count: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 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1

1 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 Count:2

1

1

1

1

1

1 1 1

1 1 1 1 1

1 1 1 1 1 1 1

1 1 1 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 Count:4

1

1

1

1

1

1

1

1

1 1 1 1 1 1 1

1

1

1

1

1

1

1

1

Final Cycle Count:5

1

1

1

1

1

1

1

1

1 1 1 1 1 1 1

1

1

1

1

1

1

1

1