**CS381-37: Project 4 (CPP)**

**Yida Tao**

**Due date: Feb. 27, 2018**

Algorithm Steps:

step 0: read the image header

dynamically allocate zerpFramedAry with extra 2 rows and 2 cols

step 1: zeroFrame the zerpFramedAry.

Step 2: loadImage

step 3: firstPassDistance (…) // 8-distance algorithm taught in class

step 4: prettyPrintof the result of Pass-1 to outFile2

with caption indicating the result of pass-1,

step 5: secondPassDistance (…) // 8-distance algorithm taught in class

// In second pass, you need to keep track the newMinVal and newMaxVal

Step 6: prettyPrintDistace of the result of Pass-2 to outFile2 w/ caption

Step 7: Write the result of Pass-2 (without the 2 extra rows and columns) to outFile1 with updated image header.

Step 8: close all files.

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class ImageProcessing{

private:

int numRows;

int numCols;

int minVal;

int maxVal;

int newMin;

int newMax;

int \*\*zeroFramedAry;

int neighborAry[5] = {0};

public:

ImageProcessing(string input){

numRows = 0;

numCols = 0;

minVal = 0;

maxVal = 0;

newMin = 99;

newMax = 0;

ifstream inFile;

inFile.open(input);

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

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

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

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

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

zeroFramedAry[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;

zeroFramedAry[r][c] = num;

counter++;

}

inFile.close();

}

~ImageProcessing(){

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

delete zeroFramedAry[i];

}

delete[] zeroFramedAry;

}

private:

void loadNeighbors1(int i, int j){

neighborAry[0] = zeroFramedAry[i - 1][j - 1];

neighborAry[1] = zeroFramedAry[i - 1][j];

neighborAry[2] = zeroFramedAry[i - 1][j + 1];

neighborAry[3] = zeroFramedAry[i][j - 1];

neighborAry[4] = 99;

}

void loadNeighbors2(int i, int j){

neighborAry[0] = zeroFramedAry[i][j];

neighborAry[1] = zeroFramedAry[i][j + 1];

neighborAry[2] = zeroFramedAry[i + 1][j - 1];

neighborAry[3] = zeroFramedAry[i + 1][j];

neighborAry[4] = zeroFramedAry[i + 1][j + 1];

}

int minNeighbor(){

int min = neighborAry[0];

for(int i = 1; i < (sizeof(neighborAry)/sizeof(\*neighborAry)); i++){

if(neighborAry[i] < min) min = neighborAry[i];

}

return min;

}

public:

void firstPassDistance(){

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

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

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

loadNeighbors1(i, j);

zeroFramedAry[i][j] = minNeighbor() + 1;

}

}

}

}

void secondPassDistance(){

int minN = 0;

for(int i = numRows + 1; i > 1; i--){

for(int j = numCols + 1; j > 1; j--){

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

loadNeighbors2(i, j);

minN = minNeighbor();

if(zeroFramedAry[i][j] > minN){

zeroFramedAry[i][j] = minN + 1;

}

if(zeroFramedAry[i][j] > newMax)

newMax = zeroFramedAry[i][j];

if(zeroFramedAry[i][j] < newMin)

newMin = zeroFramedAry[i][j];

}

}

}

}

void prettyPrint(){

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

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

if(zeroFramedAry[i][j] > 0 && zeroFramedAry[i][j] < 10){

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

}

else if(zeroFramedAry[i][j] >9){

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

}

else{

cout <<" ";

}

}

cout << endl;

}

}

void printResult(){

cout<< numRows << " " << numCols << " " << newMin << " " << newMax << endl;

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

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

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

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

}

else{

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

}

}

cout << endl;

}

}

};

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

ImageProcessing \*ip = new ImageProcessing(argv[1]);

//step 3

ip->firstPassDistance();

//step 4

cout<<"------first Pass Distance------"<<endl;

ip->prettyPrint();

//step 5

ip->secondPassDistance();

//step 6

cout<<"------second Pass Distance------"<<endl;

ip->prettyPrint();

//step 7

cout.rdbuf(out1.rdbuf());

ip->printResult();

//step 8

cout.rdbuf(console);

cout<<"done"<<endl;

out1.close();

out2.close();

delete ip;

return 0;

}

**Input: Distance\_Data4.txt**

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

**Output1**

17 17 1 5

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 2 1 1 0 0 0 0 0 0

0 0 0 0 0 1 1 2 2 2 1 1 0 0 0 0 0

0 0 0 0 1 1 2 2 3 2 2 1 1 0 0 0 0

0 0 0 1 1 2 2 3 3 3 2 2 1 1 0 0 0

0 0 0 1 2 2 3 3 4 3 3 2 2 1 0 0 0

0 0 0 1 2 3 3 4 4 4 3 3 2 1 0 0 0

0 0 0 1 2 3 4 4 5 4 4 3 2 1 0 0 0

0 0 0 1 2 3 3 4 4 4 3 3 2 1 0 0 0

0 0 0 1 2 2 3 3 4 3 3 2 2 1 0 0 0

0 0 0 1 1 2 2 3 3 3 2 2 1 1 0 0 0

0 0 0 0 1 1 2 2 3 2 2 1 1 0 0 0 0

0 0 0 0 0 1 1 2 2 2 1 1 0 0 0 0 0

0 0 0 0 0 0 1 1 2 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

**Output2**

------first Pass Distance------

1

1 1 1

1 1 2 1 1

1 1 2 2 2 1 1

1 1 2 2 3 2 2 1 1

1 1 2 2 3 3 3 2 2 1 1

1 2 2 3 3 4 3 3 2 2 1

1 2 3 3 4 4 4 3 3 2 1

1 2 3 4 4 5 4 4 3 2 1

1 2 3 4 5 5 5 4 3 2 1

1 2 3 4 5 6 5 4 3 2 1

1 2 3 4 5 6 5 4 3 2 1

1 2 3 4 5 5 4 3 2

1 2 3 4 5 4 3

1 2 3 4 4

1 2 3

1

------second Pass Distance------

1

1 1 1

1 1 2 1 1

1 1 2 2 2 1 1

1 1 2 2 3 2 2 1 1

1 1 2 2 3 3 3 2 2 1 1

1 2 2 3 3 4 3 3 2 2 1

1 2 3 3 4 4 4 3 3 2 1

1 2 3 4 4 5 4 4 3 2 1

1 2 3 3 4 4 4 3 3 2 1

1 2 2 3 3 4 3 3 2 2 1

1 1 2 2 3 3 3 2 2 1 1

1 1 2 2 3 2 2 1 1

1 1 2 2 2 1 1

1 1 2 1 1

1 1 1

1