**CS381-37: Project 6 (CPP)**

**Yida Tao**

**Due date: Mar. 20, 2018**

Algorithm Steps:

step 1: inFile <-- open the input file, argv[1]

- open outFile1 and outFile2

- read image header from inFile

- output the image header to outFile1

- output image header to outFile2 // per text line

step 2: imgAry <-- dynamically allocated

step 3: loadImage()

step 4: zeroFramed ()

step 5: getChainCode()

step 5.1: scan imgAry from L to R & T to B

step 5.2: if imgAry(iRow, jCol) > 0

output iRow, jCol, imgAry(iRow, jCol) to outFile1

// see format specs above

output iRow, jCol, imgAry(iRow, jCol) to outFile2

// see format specs above

startP <-- (iRow, jCol)

currentP <-- (iRow, jCol)

lastQ <-- 4

step 5.3: nextQ <-- mod(lastQ+1, 8)

step 5.4: PchainDir <-- findNextP(currentP, nextQ, nextP)

// nextP will be determined inside the findNextP method.

step 5.5: output PchainDir to outFile1 // see format given in the above

output PchainDir to outFile2 // see format given in the above

step 5.6: lastQ <-- nextDirTable[(PchainDir+7)%8]

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

step 5.7: repeat step 3 to step 6 until currentP == startP

step 6: close all files

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

class image{

public:

int numRows;

int numCols;

int minVal;

int maxVal;

int \*\*imgAry;

image(string input){

numRows = 0;

numCols = 0;

minVal = 0;

maxVal = 0;

ifstream inFile;

inFile.open(input);

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

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

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

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

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

imgAry[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;

imgAry[r][c] = num;

counter++;

}

inFile.close();

}

~image(){

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

delete imgAry[i];

}

delete[] imgAry;

}

void printHeader(){

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

}

};

class chainCode{

public:

class Point{

public: int row;

int col;

Point(int r,int c){

row = r;

col = c;

}

};

public:

int nextDirTable[8]={6,0,0,2,2,4,4,6};

image \*img;

Point \*\*neighborCoord;

Point \*startP;

Point \*currentP;

Point \*nextP;

int lastQ;

int nextQ;

int nextDir;

int PchainDir;

chainCode(image \*i){

img = i;

neighborCoord = new Point\*[8];

for(int j = 0;j < 8;j++){

neighborCoord[j] = NULL;

}

startP = NULL;

currentP = NULL;

nextP = NULL;

lastQ = 0;

nextQ = 0;

nextDir = 0;

PchainDir = 0;

}

~chainCode(){

delete startP;

delete currentP;

delete nextP;

for(int i = 0; i < 8; i++){

delete neighborCoord[i];

}

delete[] neighborCoord;

delete img;

}

void getChainCode(ofstream& o1,ofstream& o2){

//step 1

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

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

//step 2

int counter = 0;

if(img->imgAry[i][j] > 0){

cout.rdbuf(o1.rdbuf());

cout<<img->imgAry[i][j]<<" "<<i-1<<" "<<j-1<<" ";

cout.rdbuf(o2.rdbuf());

cout<<i-1<<" "<<j-1<<" "<<img->imgAry[i][j]<<endl;;

startP = new Point(i,j);

currentP = new Point(i,j);

lastQ = 4;

//step 3-7

do{

nextQ = (lastQ+1)%8;

findNextP(currentP,nextQ,nextP);

cout.rdbuf(o1.rdbuf());

cout<<PchainDir<<" ";

cout.rdbuf(o2.rdbuf());

cout<<PchainDir<<" ";

if(++counter % 15 == 0)cout<<endl;

lastQ = nextDirTable[(PchainDir+7)%8];

delete currentP;

currentP = new Point(nextP->row,nextP->col);

}while(currentP->row != startP->row || currentP->col != startP->col);

return;

}

}

}

}

void loadNeighborsCoord(int r,int c){

for(int i = 0; i < 8; i++){

delete neighborCoord[i];

}

neighborCoord[0]= new Point(r,c+1);

neighborCoord[1]= new Point(r-1,c+1);

neighborCoord[2]= new Point(r-1,c);

neighborCoord[3]= new Point(r-1,c-1);

neighborCoord[4]= new Point(r,c-1);

neighborCoord[5]= new Point(r+1,c-1);

neighborCoord[6]= new Point(r+1,c);

neighborCoord[7]= new Point(r+1,c+1);

}

void findNextP(Point \*curr,int nextq,Point \*nextp){

loadNeighborsCoord(curr->row,curr->col);

PchainDir = getChainDir(curr,nextq);

delete nextP;

nextP = new Point(neighborCoord[PchainDir]->row,neighborCoord[PchainDir]->col);

}

int getChainDir(Point \*p,int q){

while(img->imgAry[neighborCoord[q]->row][neighborCoord[q]->col]==0){

q = (q+1)%8;

}

return q;

}

};

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();

//step 1,2,3,4

image \*img = new image(argv[1]);

cout.rdbuf(out1.rdbuf());

img->printHeader();

cout.rdbuf(out2.rdbuf());

img->printHeader();

//step 5

chainCode \*chain = new chainCode(img);

chain->getChainCode(out1,out2);

//step 6

cout.rdbuf(console);

cout<<"done"<<endl;

delete chain;

out1.close();

out2.close();

return 0;

}

**Input1**

15 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 1

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

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

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

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

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

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

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

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

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

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

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

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

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

**Output1.1**

15 19 0 1

1 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4

**Output1.2**

15 19 0 1

0 0 1

6 6 6 6 6 6 6 6 6 6 6 6 6 6 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 2 2 2 2 2 2 2 2 2 2 2 2 2

2 4 4 4 4 4 4 4 4 4 4 4 4 4 4

4 4 4 4

**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

1 0 9 5 5 5 5 5 5 5 5 5 0 0 0 0 0 0 0 0 7 5 5 5 5 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 1 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3

**Output2.2**

20 19 0 1

0 9 1

5 5 5 5 5 5 5 5 5 0 0 0 0 0 0

0 0 7 5 5 5 5 5 5 5 5 5 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3 3 3 3 3 3 3 3 3 1 0 0 0 0 0

0 0 0 3 3 3 3 3 3 3 3 3

**Input3**

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 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 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 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 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

**Output3.1**

17 17 0 1

1 0 8 5 5 5 5 5 5 5 5 7 7 7 7 7 7 7 7 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3

**Output3.2**

17 17 0 1

0 8 1

5 5 5 5 5 5 5 5 7 7 7 7 7 7 7

7 1 1 1 1 1 1 1 1 3 3 3 3 3 3

3 3

**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

1 0 8 5 5 5 5 5 6 6 6 6 6 6 7 7 7 7 7 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3

**Output4.2**

17 17 0 1

0 8 1

5 5 5 5 5 6 6 6 6 6 6 7 7 7 7

7 1 1 1 1 1 2 2 2 2 2 2 3 3 3

3 3