Project 5.2: 8 Chain Code (C++)

CSC 381 Rafael Carmilema

Due Date: Oct / 20 /2016

III. ChainCode Algorithms  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  step 0: - open the component file and property files  
          - allocate and initalize all data structures  
          - currentCC = 0  
 step 1: currentCC++  
        - minRowOffset <-- get from currentCC's bounding box in the property  
        - maxRowOffset <-- get from currentCC's bounding box in the property  
        - minColOffset <-- get from currentCC's bounding box in the property  
        - maxColOffset <-- get from currentCC's bounding box in the property  
step 2: scan the zeroFramed image  
        iRow from minRowOffset+1 to maxRowOffset+1 and  
        jCol from minColOffset+1 to maxColffset+1  
step 3: if image(iRow, jCol) == currentCC  
        output iRow, jCol, currentCC to outFile1  
        startP <-- (iRow, jCol)  
        currentP <-- (iRow, jCol)  
        lastQ <-- 4  
step 4: nextQ <-- mod(lastQ+1, 8)  
step 5: Pchain <-- findNextP(currentP, nextQ, currentCC)  
        nextP <-- nighborCoord[Pchain]  
step 6: output Pchain to outFile1 (give one blank space after)  
step 7: lastQ <-- nextDirTable[Pchian]  
        currentP <-- nextP  
step 8: repeat step 4 to step 7 until currentP == startP  
step 9: start next text line for next CC chain code  
step 10: repeat step 1 to step 9 until currentCC > maxCC  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
III. findNextP(currentP, nextQ, currentCC, nextP)  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
step 1: loadNeighborCoord(currentP)  
step 2: chainDir <-- scan currentP's 8 neighbors within nighborCoord[] array from nextQ direction (mod 8)  
        until a none zero neighbor with the same label as currentCC is found, chainDir is the index of  
        neighborCoord[] which with the same label as currentP  
step 3: nextP <-- nighborCoord[chainDir]  
  
step 4: returns chainDir

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <limits>

using namespace std;

class Image{

int numRows;

int numCols;

int minVal;

int maxVal;

int\*\* zeroFramedAry;

public:

Image(ifstream& the\_file){

int data;

//get the header values

the\_file>>data;

numRows=data;

the\_file>>data;

numCols=data;

the\_file>>data;

minVal=data;

the\_file>>data;

maxVal=data;

// allocate zeroframed 2d array

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

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

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

}

//ini

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

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

zeroFramedAry[i][j]=0;

}

}

for(int r= 0; r<numRows;r++){

for(int c= 0; c<numCols;c++){

while(the\_file>>data){

loadImage(data,r ,c);

//cout<<r<<" "<< c<<" "<<string<<endl;

break;

}

}

// cout<<endl;

}

}

void loadImage(int value, int r, int c){

zeroFramedAry[r+1][c+1]=value;

}

void prettyPrint(ofstream& out1){

out1<<"Pretty Print"<<endl;

out1<<numRows<<" "<<numCols<<" "<<minVal<<" "<<maxVal<<endl;

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

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

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

out1<<zeroFramedAry[i][j];

}

else{

out1<<" ";

}

//outfile1<<zeroFramedAry[i][j]<<" ";

}

out1<<endl;

//cout<<endl;

}

}

friend class chainCode;

};

class CCproperty{

struct property{

int label;

int numPixels;

int minRow;

int minCol;

int maxRow;

int maxCol;

};

int maxCC;

property\* table;

public:

CCproperty(ifstream& the\_PP){

int mCC;

// thiw will go back to the beggining of the file rather than open and close it

while(the\_PP>>mCC){

the\_PP.ignore(numeric\_limits<streamsize>::max(),'\n');

}

the\_PP.clear();

the\_PP.seekg(0,the\_PP.beg);

maxCC=mCC;

table = new property[maxCC+1];

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

the\_PP>>table[i].label;

the\_PP>>table[i].minRow;

the\_PP>>table[i].minCol;

the\_PP>>table[i].maxRow;;

the\_PP>>table[i].maxCol;

the\_PP>>table[i].numPixels;

}

}

friend class chainCode;

};

class chainCode{

public:

struct point{

int row;

int col;

};

//public:

int currentCC;

int minRowOffset;

int maxRowOffset;

int minColOffset;

int maxColOffset;

CCproperty\* t;

Image\* imag;

point neighborCoord[8];

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

point currentP, nextP,startP;

int nextQ, Pchain, lastQ;

public:

chainCode(ifstream& the\_file,ifstream& the\_PP){

t=new CCproperty(the\_PP);

imag=new Image(the\_file);

currentCC=0;

}

void calculateChain(ofstream& out1,ofstream& out2){

while(currentCC<t->maxCC){

currentCC++;

cout<<currentCC<<endl;

minRowOffset=t->table[currentCC].minRow;

maxRowOffset=t->table[currentCC].maxRow;

minColOffset=t->table[currentCC].minCol;

maxColOffset=t->table[currentCC].maxCol;

for(int i= minRowOffset+1; i< maxRowOffset+1;i++){

for(int j=minColOffset+1; j<maxColOffset+1;j++){

if(imag->zeroFramedAry[i][j]==currentCC){

out1<<i<<" "<<j<<" "<<currentCC<<endl;

startP.row=i;

startP.col=j;

currentP.row=i;

currentP.col=j;

goto finish;

//break;

}

}

}

finish:

out2<<currentP.row<<" out "<<currentP.col<<endl;

out1<<"PChain:"<<endl;

lastQ=4;

nextQ=(lastQ)%8;

Pchain=findNextP(currentP,nextQ,currentCC);

out1<<Pchain<<" ";

lastQ=nextDirTable[Pchain];

currentP.row=nextP.row;

currentP.col=nextP.col;

while(currentP.row!=startP.row||currentP.col!=startP.col){

nextQ=(lastQ)%8;

Pchain=findNextP(currentP,nextQ,currentCC);

out1<<Pchain<<" ";

lastQ=nextDirTable[Pchain];

currentP.row=nextP.row;

currentP.col=nextP.col;

}//end of while

out1<<endl;

out1<<endl;

out2<<minRowOffset<<" "<<maxRowOffset<<" "<<minColOffset<<" "<<maxColOffset<<endl;

}

}

void loadNeighborsCoord(point currP){

int i=currP.row;

int j=currP.col;

//upper left

neighborCoord[3].row=i-1;

neighborCoord[3].col=j-1;

//upper middle

neighborCoord[2].row=i-1;

neighborCoord[2].col=j;

//upper right

neighborCoord[1].row=i-1;

neighborCoord[1].col=j+1;

//middle left

neighborCoord[4].row=i;

neighborCoord[4].col=j-1;

//middle right

neighborCoord[0].row=i;

neighborCoord[0].col=j+1;

//bottom left

neighborCoord[5].row=i+1;

neighborCoord[5].col=j-1;

//bottom middle

neighborCoord[6].row=i+1;

neighborCoord[6].col=j;

//bottom right

neighborCoord[7].row=i+1;

neighborCoord[7].col=j+1;

}

int findNextP(point currP, int nQ, int currCC){

loadNeighborsCoord(currP);

int chainDir;

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

if(imag->zeroFramedAry[neighborCoord[i%8].row][neighborCoord[i%8].col]==currCC){

//out2<<neighborCoord[i%8].row<<" nextp "<<neighborCoord[i%8].col<<endl;

chainDir=i%8;

break;

}//end of if

}//end of for

nextP.row=neighborCoord[chainDir].row;

nextP.col=neighborCoord[chainDir].col;

return chainDir;

}

void prettyPrint(ofstream& out1){

imag->prettyPrint(out1);

}

};

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

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

if ( argc <4 ) // argc should be 4 for correct execution

// We print argv[0] assuming it is the program name

cout<<"usage: "<< argv[0] <<" <filename>\n";

else {

// We assume argv[1] is a filename to open

ifstream the\_file ( argv[1] );

ifstream the\_PP(argv[2]);

// Always check to see if file opening succeeded

if ( !the\_file.is\_open() )

cout<<"Could not open file\n";

else {

ofstream outfile1;

ofstream outfile2;

outfile1.open(argv[3]);

outfile2.open(argv[4]);

chainCode testing(the\_file,the\_PP);

testing.calculateChain(outfile1,outfile2);

testing.prettyPrint(outfile1);

the\_file.close();

the\_PP.close();

outfile1.close();

outfile2.close();

}//else

}

return 0;

}

Out1:

2 16 1

PChain:

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

6 8 2

PChain:

6 0 7 6 6 6 6 6 6 7 7 7 0 7 6 6 0 0 0 0 0 0 0 4 4 4 4 3 2 2 4 4 3 3 2 0 3 2 0 1 3 2 3 4 4 4 3 2 4

9 22 3

PChain:

6 5 5 4 4 4 5 6 0 0 7 6 6 6 2 2 1 0 1 2 2 1 3 2

22 13 4

PChain:

5 7 6 4 4 4 6 0 0 0 0 0 7 0 0 0 0 1 0 2 3 3 4 4 6 6 5 4 4 3 2 2 2 4

31 13 5

PChain:

6 6 7 7 6 7 1 2 1 1 2 2 4 4 4 4 4 4

Pretty Print

38 31 0 5

1

111

11111

1111111

22 111111111

22 11

2222 11

22222 11 3

22222 11 3

222222 333

22222 333333

2222 3333333

22222 3333333

222 333

222 3

22222 3

222 3

222

22222222

44

444 444

44 4444

44444 44444

4444444444444

44444

5555555

5555555

5555555

55555

555

555

5

Out2:

2 out 16

1 9 11 19

6 out 8

5 19 7 21

9 out 22

8 17 15 22

22 out 13

21 26 9 21

31 out 13

30 36 12 18