\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CSCI 381-26 Project 11: text-line detection Language: C++

Name: Akshar Patel

Due date: Soft copy: 5/8/2020

Hard copy: 5/8/2020

Submission date: Soft copy: 5/8/2020

Hard copy: 5/8/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Part 1: Algorithm**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. main (…)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Step 1: inFile <-- open image file from argv[1]

outFiles <-- open outFiles file from argv[3], argv[4]

read image header from inFile

get threshold value from argv[2]

Step 2: dynamically allocate image array, HPP, VPP, HPPbin, VPPbin, HPPMorph, VPPMorph

and initialize to 0.

Step 3: loadImage(inFile, imageAry)

Step 4: computeVPP & computeHPP

outFIle VPP & HPP with caption.

Step 5: computeVPPbin & computeHPPbin

outFile with caption.

Step 6: perform closing operation on VPPbin & HPPbin

Using 1 1 1 structure element and

Store output in HPPMorph & VPPMorph.

Step 7: find reading direction

outFile direction if horizontal or vertical.

Step 8: close all files

**Part 2: Source code**

#include <iostream>

#include<fstream>

using namespace std;

class box{

public:

int minR, minC, maxR, maxC;

};

class boxNode{

public:

int boxType;

box BBox;

boxNode\* next;

};

class boxQ{

public:

boxNode\* Qfront;

boxNode\* QBack;

};

class imagePP{

public:

int numRows, numCols, minVal, maxVal, thrVal, HPPruns, VPPruns;

int\*\* imgAry;

int\* HPP;

int\* VPP;

int\* HPPbin;

int\* VPPbin;

int\* HPPMorph;

int\* VPPMorph;

string direction;

void set2DZero(int\*\* Ary){

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

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

Ary[i][j] = 0;

}

}

}

void loadImage(ifstream& file, int\*\* Ary){

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

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

file>>Ary[i][j];

}

}

}

void prettyPrint(ofstream& file){

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

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

file<<imgAry[i][j]<<" ";

}

file<<endl;

}

}

void computeHPP(int\*\* imAry, int\* hAry){

int sum = 0;

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

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

if(imAry[i][j] == 1){

sum++;

}

}

if(sum >= thrVal){

HPPbin[i] = 1;

}

hAry[i] = sum;

sum = 0;

}

}

void computeVPP(int\*\* imAry, int\* vAry){

int sum = 0;

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

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

if(imAry[j][i] == 1){

sum++;

}

}

if(sum >= thrVal){

VPPbin[i] = 1;

}

vAry[i] = sum;

sum = 0;

}

}

void prettyPrinthP(ofstream& file, int\* Ary){

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

file<<Ary[i]<<" ";

}

}

void prettyPrintvP(ofstream& file, int\* Ary){

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

file<<Ary[i]<<" ";

}

}

void morphClosingHPP(int\* bin, int\* morph){

int temp[numRows + 2];

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

temp[i] = 0;

}

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

morph[i] = bin[i];

temp[i] = bin[i];

}

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

if(bin[i] > 0){

morph[i-1] = 1;

morph[i+1] = 1;

temp[i-1] = 1;

temp[i+1] = 1;

}

}

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

if(temp[i] > 0){

if (temp[i-1] == 0 || temp[i+1] == 0){

morph[i] = 0;

}

}

}

}

void morphClosingVPP(int\* bin, int\* morph){

int temp[numCols + 2];

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

temp[i] = 0;

}

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

morph[i] = bin[i];

temp[i] = bin[i];

}

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

if(bin[i] > 0){

morph[i-1] = 1;

morph[i+1] = 1;

temp[i-1] = 1;

temp[i+1] = 1;

}

}

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

if(temp[i] > 0){

if (temp[i-1] == 0 || temp[i+1] == 0){

morph[i] = 0;

}

}

}

}

string findDirection(int\* HPPMo, int\* VPPMo){

HPPruns = 0;

VPPruns = 0;

countruns(HPPruns,VPPruns, HPPMo, VPPMo);

if(HPPruns >= VPPruns){

return "Horizontal";

}

else if(VPPruns >= HPPruns){

return "Vertical";

}

else return "Can not determine the reading direction." ;

}

void countruns(int& hruns, int& vruns, int\* HPPMo, int\* VPPMo){

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

if(HPPMo[i] == HPPMo[i+1]){

hruns++;

}

}

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

if(VPPMo[i] == VPPMo[i+1]){

vruns++;

}

}

}

};

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

imagePP l;

string inputName = argv[1];

ifstream inFile;

inFile.open( inputName );

string outputName1 = argv[3];

ofstream outFile1;

outFile1.open( outputName1 );

string outputName2 = argv[4];

ofstream outFile2;

outFile2.open( outputName2 );

if(inFile.is\_open()){

if(outFile1.is\_open() && outFile2.is\_open()){

inFile>>l.numRows>>l.numCols>>l.minVal>>l.maxVal;

l.thrVal = stoi(argv[2]);

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

l.HPP = new int [l.numRows + 2];

l.VPP = new int [l.numCols + 2];

l.HPPbin = new int [l.numRows + 2];

l.VPPbin = new int [l.numCols + 2];

l.HPPMorph = new int [l.numRows + 2];

l.VPPMorph = new int [l.numCols + 2];

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

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

l.HPP[i] = 0;

l.HPPbin[i] = 0;

l.HPPMorph[i] = 0;

}

for(int i = 0; i < l.numCols + 2; i++){

l.VPP[i] = 0;

l.VPPbin[i] = 0;

l.VPPMorph[i] = 0;

}

l.set2DZero(l.imgAry);

l.loadImage(inFile, l.imgAry);

l.computeVPP(l.imgAry, l.VPP);

l.computeHPP(l.imgAry, l.HPP);

outFile2<<"----------- HPP -----------"<<endl;

l.prettyPrinthP(outFile2, l.HPP);

outFile2<<endl<<"----------- VPP -----------"<<endl;

l.prettyPrintvP(outFile2, l.VPP);

outFile2<<endl<<"----------- HPPbin -----------"<<endl;

l.prettyPrinthP(outFile2, l.HPPbin);

outFile2<<endl<<"----------- VPPbin -----------"<<endl;

l.prettyPrintvP(outFile2, l.VPPbin);

l.morphClosingHPP(l.HPPbin, l.HPPMorph);

outFile2<<endl<<"----------- HPPMorph -----------"<<endl;

l.prettyPrinthP(outFile2, l.HPPMorph);

l.morphClosingVPP(l.VPPbin, l.VPPMorph);

outFile2<<endl<<"----------- VPPMorph -----------"<<endl;

l.prettyPrintvP(outFile2, l.VPPMorph);

l.direction = l.findDirection(l.HPPMorph,l.VPPMorph);

outFile2<<endl<<"The reading direction is: "<<l.direction<<endl;

inFile.close();

outFile1.close();

outFile2.close();

}else{cout<<"Error!! Could NOT create output file"<<endl ;}

}else{cout<<"Error!! Could NOT open input file"<<endl;}

}

**Part 3: Output**

**outFile2**

----------- HPP -----------

0 0 1 1 19 24 25 1 1 1 24 27 26 4 1 0 1 1 8 23 24 27 1 1 2 25 28 26 8 2 1 2 1 10 22 26 26 0 1 0 25 27 26 18 1 0 0

----------- VPP -----------

0 0 7 14 11 24 10 7 5 9 10 10 13 14 8 10 10 6 9 15 16 14 9 6 5 6 9 11 16 18 20 14 8 13 8 12 10 13 16 16 10 10 3 4 8 18 18 12 3 0 0 0

----------- HPPbin -----------

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

----------- VPPbin -----------

0 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 0 0 0

----------- HPPMorph -----------

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

----------- VPPMorph -----------

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 0 0 0

The reading direction is: Vertical