**CS381-37: Project 8.1 (CPP)**

**Yida Tao**

**Due date: May. 01, 2018**

Algorithm Steps:

step 0: - inFile 🡨 open input files

(numRows, numCols, minVal, maxVal, label) <- get from inFile

- numPts 🡨 countPts (inFile)

- close inFile

- inFile 🡨 open the input file the second time.

- K <-- get from the user from console

- index <-- 0

step 1: (x, y) <-- read from inFile

storePt (x, y, index)// store x, y to PtAry[index]

printPtAry()

step 2: index ++;

step 3: repeat step 1 and step 2 until the end of inFile

step 4: Q 🡨 0

P 🡨 K

R 🡨 2\* K

step 5: index <-- P

curvature <-- computeCurvature (Q, P, R)

store curvature to PtAry[index]

print Q, P, R, index, x, y, curvature of PtAry[index]to argv[4]

step 6: Increment Q, P, R by 1 // each need to mod with numPts

step 7: repeat step 5 to step 6 until P == K-1

step 8: print the info (x, y, curvature) of the entire PtAry to argv[4]

step 9: computeLocalMaxima (PtAry) for all point in PtAry[index], index from 0 to numPts-1

step 10: setCorner (PtAry) do for all point in boundPtAry[index], index from 0 to numPts-1

step 11: output only (x, y, corner) of the entire PtAry to argv[2]

step 12: Img <-- create an image of size numRows by numCols

step 13: plotPt2Img()

step 14: prettyPrint (img) to argv[3]

step 15: close all files

**Source Code**

#include <iostream>

#include <fstream>

#include <string>

#include <cmath>

using namespace std;

class boundaryPt{

public:

int x;

int y;

double curvature;

int localMax;

int corner;

boundaryPt(int xi, int yi){

x = xi;

y = yi;

curvature = 0;

localMax = 0;

corner = 0;

}

void print(){

cout<<"x:"<<x<<", y:"<<y<<", curvature:"<<curvature<<", corner:"<<corner<<endl;

}

void printCurv(){

cout<<"x:"<<x<<", y:"<<y<<", curvature:"<<curvature<<endl;

}

void printCorner(){

cout<<x<<" "<<y<<" "<<corner<<endl;

}

};

class kCurvature{

public:

int numRows;

int numCols;

int minVal;

int maxVal;

int label;

int K;

int countPts;

int numPts;

int beginIndex;

boundaryPt \*\*PtAry;

kCurvature(int k, int num,string input){

numRows = 0;

numCols = 0;

minVal = 0;

maxVal = 0;

label = 0;

K = k;

numPts = num;

beginIndex = 0;

PtAry = new boundaryPt\*[numPts];

ifstream inFile;

inFile.open(input);

inFile >> numRows;

inFile >> numCols;

inFile >> minVal;

inFile >> maxVal;

inFile >> label;

int r = 0;

int c = 0;

while(inFile>>r){

inFile>>c;

storePt(r, c, beginIndex++);

}

inFile.close();

}

~kCurvature(){

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

delete PtAry[i];

}

delete[] PtAry;

}

void storePt(int x, int y, int index){

PtAry[index] = new boundaryPt(x, y);

}

double computeCurvature(int q, int p, int r){

double curv = 0;

double slope1 = (PtAry[q]->y - PtAry[p]->y - 0.0) / (PtAry[q]->x - PtAry[p]->x + 0.01);

double slope2 = (PtAry[p]->y - PtAry[r]->y - 0.0) / (PtAry[p]->x - PtAry[r]->x + 0.01);

//cout<<"P:"<<p<<" slope 1:"<<slope1<<" slope 2:"<<slope2<<endl;

curv = slope1 - slope2;

if(curv < 0) curv = 0 - curv;

return curv;

}

void computeLocalMaxima(){

int firstIndex = 0;

int secondIndex = 0;

int thirdIndex = 0;

int forthIndex = 0;

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

firstIndex = (i+numPts-2)%numPts;

secondIndex = (i+numPts-1)%numPts;

thirdIndex = (i+1)%numPts;

forthIndex = (i+2)%numPts;

if(PtAry[i]->curvature >= PtAry[firstIndex]->curvature

&& PtAry[i]->curvature >= PtAry[secondIndex]->curvature

&& PtAry[i]->curvature >= PtAry[thirdIndex]->curvature

&& PtAry[i]->curvature >= PtAry[forthIndex]->curvature){

PtAry[i]->localMax++;

}

}

}

void setCornor(){

int firstIndex = 0;

int forthIndex = 0;

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

firstIndex = (i + numPts - 2) % numPts;

forthIndex = (i + 2) % numPts;

if(PtAry[i]->localMax == 1

&& PtAry[firstIndex]->localMax == 0

&& PtAry[forthIndex]->localMax == 0){

PtAry[i]->corner = 8;

}

else{

PtAry[i]->corner = 1;

}

}

}

void printPtAry(){

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

PtAry[i]->print();

}

}

void printInfo(){

cout<<"print the info (x, y, curvature) of the entire PtAry:"<<endl;

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

PtAry[i]->printCurv();

}

}

void printCor(){

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

cout<<label<<endl;

cout<<numPts<<endl;

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

PtAry[i]->printCorner();

}

}

};

class image{

public:

int numRows;

int numCols;

int minVal;

int maxVal;

int \*\*imgAry;

image(kCurvature \*kc){

numRows = kc->numRows;

numCols = kc->numCols;

minVal = kc->minVal;

maxVal = kc->maxVal;

imgAry = new int\*[numRows];

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

imgAry[i] = new int[numCols];

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

imgAry[i][j] = 0;

}

}

plotPt2Img(kc);

}

~image(){

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

delete imgAry[i];

}

delete[] imgAry;

}

void plotPt2Img(kCurvature \*kc){

for(int i = 0; i < kc->numPts; i++){

imgAry[kc->PtAry[i]->x][kc->PtAry[i]->y] = kc->PtAry[i]->corner;

}

}

void prettyPrint(){

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

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

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

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

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

}

else{

cout << " ";

}

}

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]);

ofstream out3;

out3.open(argv[4]);

streambuf \*console = cout.rdbuf();

ifstream inFile;

//step 0

inFile.open(argv[1]);

int index = 0;

int temp = 0;

int k = 0;

//skip first 5 numbers

inFile >> temp;

inFile >> temp;

inFile >> temp;

inFile >> temp;

inFile >> temp;

while(inFile >> temp){

inFile >> temp;

index ++;

}

inFile.close();

cout<<"Give me K value:"<<endl;

cin>>k;

//step 1,2,3

kCurvature \*kc = new kCurvature(k,index,argv[1]);

cout.rdbuf(out3.rdbuf());

kc->printPtAry();

//step 4

int Q = 0;

int P = k;

int R = 2\*k;

//step 5,6,7

double curv = 0;

do{

index = P;

curv = kc->computeCurvature(Q,P,R);

kc->PtAry[index]->curvature = curv;

cout<<"Q:"<<Q<<", P:"<<P<<", R:"<<R<<", index:"<<index;

cout<<", x:"<<kc->PtAry[index]->x<<", y:"<<kc->PtAry[index]->y;

cout<<", curvature:"<<kc->PtAry[index]->curvature<<endl;

Q = (Q + 1) % (kc->numPts);

P = (P + 1) % (kc->numPts);

R = (R + 1) % (kc->numPts);

}while(P != k);

//step 8

kc->printInfo();

//step 9

kc->computeLocalMaxima();

//step 10

kc->setCornor();

//step 11

cout.rdbuf(out1.rdbuf());

kc->printCor();

//step 12,13

image \*i = new image(kc);

//step 14

cout.rdbuf(out2.rdbuf());

i->prettyPrint();

//step 15

cout.rdbuf(console);

cout<<"done"<<endl;

delete kc;

out1.close();

out2.close();

out3.close();

return 0;

}

**Input3**

40 40 0 1

1

5 5

6 5

7 5

8 5

9 5

10 5

11 5

12 5

13 5

14 5

15 5

16 6

17 7

18 8

19 9

20 10

21 9

22 8

23 7

24 6

25 5

26 5

27 5

28 5

29 5

30 5

31 5

32 5

33 5

34 5

35 5

35 6

35 7

35 8

35 9

35 10

35 11

35 12

35 13

35 14

35 15

35 16

35 17

35 18

35 19

35 20

35 21

35 22

35 23

35 24

35 25

35 26

35 27

35 28

35 29

34 29

33 29

32 29

31 29

30 29

29 29

28 29

27 29

26 29

25 29

24 30

23 31

22 32

21 33

20 34

19 33

18 32

17 31

16 30

15 29

14 29

13 29

12 29

11 29

10 29

9 29

8 29

7 29

6 29

5 29

5 28

5 27

5 26

5 25

5 24

5 23

5 22

5 21

5 20

5 19

5 18

5 17

5 16

5 15

5 14

5 13

5 12

5 11

5 10

5 9

5 8

5 7

5 6

**Output3\_1**

40 40 0 1

1

108

5 5 8

6 5 1

7 5 1

8 5 1

9 5 1

10 5 1

11 5 1

12 5 1

13 5 1

14 5 1

15 5 8

16 6 1

17 7 1

18 8 1

19 9 1

20 10 8

21 9 1

22 8 1

23 7 1

24 6 1

25 5 8

26 5 1

27 5 1

28 5 1

29 5 1

30 5 1

31 5 1

32 5 1

33 5 1

34 5 1

35 5 8

35 6 1

35 7 1

35 8 1

35 9 1

35 10 1

35 11 1

35 12 1

35 13 1

35 14 1

35 15 1

35 16 1

35 17 1

35 18 1

35 19 1

35 20 1

35 21 1

35 22 1

35 23 1

35 24 1

35 25 1

35 26 1

35 27 1

35 28 1

35 29 8

34 29 1

33 29 1

32 29 1

31 29 1

30 29 1

29 29 1

28 29 1

27 29 1

26 29 1

25 29 8

24 30 1

23 31 1

22 32 1

21 33 1

20 34 8

19 33 1

18 32 1

17 31 1

16 30 1

15 29 8

14 29 1

13 29 1

12 29 1

11 29 1

10 29 1

9 29 1

8 29 1

7 29 1

6 29 1

5 29 8

5 28 1

5 27 1

5 26 1

5 25 1

5 24 1

5 23 1

5 22 1

5 21 1

5 20 1

5 19 1

5 18 1

5 17 1

5 16 1

5 15 1

5 14 1

5 13 1

5 12 1

5 11 1

5 10 1

5 9 1

5 8 1

5 7 1

5 6 1

**Output3\_2**

**8 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 8**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**8 8**

**1 1**

**1 1**

**1 1**

**1 1**

**8 8**

**1 1**

**1 1**

**1 1**

**1 1**

**8 8**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**1 1**

**8 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 8**