**CS381-37: Project 8.2 (JAVA)**

**Yida Tao**

**Due date: May. 03, 2018**

Algorithm Steps:

step 0: - inFile 🡨 open input files

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

- dynamically allocate image array of size numRows by numCols

- numPts 🡨 countPts (inFile)

- close inFile

- inFile 🡨 open the input file the second time.

- dynamically allocate PtAry with size of numPts

- K 🡨 get from the user from console

- chordLength 🡨 (2\*K)

- dynamically allocate chordAry with size of chordLength

- loadData (inFile)

Step 1: P1 <-- 0

P2 <-- chordLength-1

step 2: index <-- 0

currPt <-- P1 + 1

step 3: dist <-- computeDistance (P1, P2, currPt )

chordAry[index]🡨 dist

index ++

currPt ++

step 4: repeat step 3 while index < chordLength

step 5: print chordAry to debugging file (Output3)

step 6: maxIndex <-- findMaxDist(chordAry)

whichIndex <-- P1 + maxIndex

PtAry[whichIndex]'s maxVotes ++

update PtAry[whichIndex]'s maxDist if it is less then chordAry[maxIndex]

step 7: print PtAry from P1 to P2 to output3, debugging file

step 8: Increment P1, and P2, and then

mod (P1, numPts) and mod (P2, numPts)

step 9: repeat step 2 to step 8 until P2 == (chordLength / 2)

step 10: printPtAry() // five pts per text line

step 11: computeLocalMaxima (PtAry)

step 12: setCorner (PtAry) do for all point in boundPtAry[index], index from 0 to numPts-1

step 13: output only (x, y, corner) of the entire PtAry to output1

step 14: Img <-- create an image of size numRows by numCols

step 15: plotPt2Img()

step 16: prettyPrint (img) to output2

step 17: close all files

**Source Code**

**import** java.io.\*;

**import** java.io.\*;

**public** **class** Project8\_2 {

**public** **static** **void** main(String[] args) {

//step 0

**int** k = 0;

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));

**try**{

System.***out***.println("argument 1:" + args[0]);

System.***out***.println("argument 2:" + args[1]);

System.***out***.println("argument 3:" + args[2]);

System.***out***.println("argument 4:" + args[3]);

}**catch**(ArrayIndexOutOfBoundsException e){

System.***out***.println("no arguments");

System.*exit*(0);

}

System.***out***.println("Give me a K:");

**try**{

k = Integer.*parseInt*(br.readLine());

} **catch** (IOException e) {

e.printStackTrace();

} **catch**(NumberFormatException e){

System.***out***.println("input is not a number!");

System.*exit*(0);

}

image i = **new** image(args[0]);

arcChord ac = **new** arcChord(i.numPts,k);

ac.loadData(args[0]);

//ac.printPtAry();

//step 1

FileOutputStream fos = **null**;

**try** {

fos = **new** FileOutputStream(args[3]);

System.*setOut*(**new** PrintStream(fos));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

**int** P1 = 0;

**int** P2 = 2\*k - 1;

**int** index = 0;

**int** currPtIndex = 0;

**int** maxIndex;

**double** dist = 0.0;

//step 2 - 9

**do** {

index = 0;

currPtIndex = (P1 + 1) % ac.numPts;

**while**(index < 2\*k){

dist = ac.computeDistance(P1, P2, currPtIndex);

ac.chordAry[index] = dist;

index++;

currPtIndex = (currPtIndex + 1) % ac.numPts;

}

//ac.printChordAry();

maxIndex = ac.findMax();

ac.PtAry[(P1+maxIndex+1) % ac.numPts].maxVotes++;

**if**(ac.PtAry[(P1+maxIndex+1) % ac.numPts].maxDist < dist){

ac.PtAry[(P1+maxIndex+1) % ac.numPts].maxDist = dist;

}

//ac.printPtAry(P1, P2);

P1 = (P1+1) % ac.numPts;

P2 = (P2+1) % ac.numPts;

}**while**(P2 != 2\*k-1);

ac.printVotes();

//step 10

ac.print5();

//step 11 12

ac.computeLocalMax();

ac.setCorner();

//step 13

**try** {

fos = **new** FileOutputStream(args[1]);

System.*setOut*(**new** PrintStream(fos));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

ac.printCorners();

//step 14 15

i.plotPt2Img(ac);

//step 16

**try** {

fos = **new** FileOutputStream(args[2]);

System.*setOut*(**new** PrintStream(fos));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

i.prettyPrint();

//step17

**try** {

fos.close();

br.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** image {

**int** numRows;

**int** numCols;

**int** minVal;

**int** maxVal;

**int** label;

**int** numPts;

**int**[][] img;

**public** image(String input){

numPts = 0;

Scanner sc = **null**;

**try** {

sc = **new** Scanner(**new** File(input));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

System.***out***.println("Cant find the file" + input);

}

**try** {

numRows = sc.nextInt();

numCols = sc.nextInt();

minVal = sc.nextInt();

maxVal = sc.nextInt();

label = sc.nextInt();

img = **new** **int**[numRows][numCols]

**while**(sc.hasNextInt()){

sc.nextInt();

sc.nextInt();

numPts++;

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

sc.close();

}

}

**public** **void** plotPt2Img(arcChord ac){

**int** r = 0;

**int** c = 0;

**int** corner = 0;

**for**(**int** i = 0; i < ac.PtAry.length; i++){

r = ac.PtAry[i].x;

c = ac.PtAry[i].y;

corner = ac.PtAry[i].corner;

img[r][c] = corner;

}

}

**public** **void** prettyPrint(){

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

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

**if**(img[i][j] > 0){

System.***out***.print(img[i][j] + " ");

}

**else**{

System.***out***.print(" ");

}

}

System.***out***.println();

}

}

**public** **class** boundaryPt {

**int** x;

**int** y;

**int** maxVotes;

**double** maxDist;

**int** corner;

**int** localMax;

**public** boundaryPt(**int** r, **int** c){

x = r;

y = c;

maxVotes = 0;

maxDist = 0.0;

corner = 1;

localMax = 0;

}

**public** **void** printPt(){

System.***out***.println("x:"+x + " y:"+y);

}

**public** **void** printPtInOneLine(){

System.***out***.print("x:"+x + " y:"+y + ", ");

}

**public** **void** printCorner(){

System.***out***.println(x +" "+ y +" "+ corner);

}

**public** **void** printVotes(){

System.***out***.println(x +" "+ y +" "+ maxVotes);

}

}

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.util.Scanner;

**public** **class** arcChord {

**int** numRows;

**int** numCols;

**int** minVal;

**int** maxVal;

**int** label;

**int** chordLength;

**int** numPts;

boundaryPt[] PtAry;

**double**[] chordAry;

**public** arcChord(**int** num, **int** k){

chordLength = 2\*k;

numPts = num;

PtAry = **new** boundaryPt[numPts];

chordAry = **new** **double**[chordLength];

}

**public** **void** loadData(String input){

Scanner sc = **null**;

**try** {

sc = **new** Scanner(**new** File(input));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

System.***out***.println("Cant find the file" + input);

}

**try** {

numRows = sc.nextInt();

numCols = sc.nextInt();

minVal = sc.nextInt();

maxVal = sc.nextInt();

label = sc.nextInt();

**int** counter = 0;

**int** r = 0;

**int** c = 0;

**while**(sc.hasNextInt()){

r = sc.nextInt();

c = sc.nextInt();

PtAry[counter] = **new** boundaryPt(r,c);

counter++;

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

sc.close();

}

}

**public** **double** computeDistance(**int** p1, **int** p2, **int** currPtIndex){

**double** dist = 0.0;

**int** x = PtAry[currPtIndex].x;

**int** y = PtAry[currPtIndex].y;

**int** x1 = PtAry[p1].x;

**int** x2 = PtAry[p2].x;

**int** y1 = PtAry[p1].y;

**int** y2 = PtAry[p2].y;

**double** A = y2 - y1 + 0.00001;

**double** B = x1 - x2 + 0.00001;

**double** C = x2 \* y1 - x1 \* y2 + 0.00001;

dist = Math.*abs*(A\*x + B\*y + C) / Math.*sqrt*(A\*A + B\*B);

**return** dist;

}

**public** **int** findMax(){

**int** maxIndex = 0;

**for**(**int** i = 1; i < chordAry.length; i++){

**if**(chordAry[i] > chordAry[maxIndex]){

maxIndex = i;

}

}

**return** maxIndex;

}

**public** **void** computeLocalMax(){

**int** first = 0;

**int** second = 0;

**int** third = 0;

**int** forth = 0;

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

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

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

third = (i + 1) % numPts;

forth = (i + 2) % numPts;

**if**(PtAry[i].maxVotes > PtAry[first].maxVotes

&&PtAry[i].maxVotes > PtAry[second].maxVotes

&&PtAry[i].maxVotes > PtAry[third].maxVotes

&&PtAry[i].maxVotes > PtAry[forth].maxVotes){

PtAry[i].localMax++;

}

}

}

**public** **void** setCorner(){

**int** first = 0;

**int** forth = 0;

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

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

forth = (i + 2) % numPts;

**if**(PtAry[first].localMax == 0

&&PtAry[forth].localMax == 0

&&PtAry[i].localMax == 1){

PtAry[i].corner = 9;

}

**else**

PtAry[i].corner = 1;

}

}

**public** **void** printPtAry(){

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

PtAry[i].printPt();

}

}

**public** **void** printPtAry(**int** p1, **int** p2){

System.***out***.println("From " + p1 + " to " + p2);

**for**(**int** i = p1; i < numPts + p2; i++){

PtAry[i%numPts].printPt();

}

}

**public** **void** print5(){

System.***out***.println("Five pts per text line:");

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

PtAry[i].printPtInOneLine();

**if**((i+1)%5 == 0){

System.***out***.println();

}

}

}

**public** **void** printChordAry(){

System.***out***.println("ChordAry:");

**for**(**int** i = 1; i < chordAry.length; i++){

System.***out***.println(chordAry[i]);

}

}

**public** **void** printCorners(){

System.***out***.println(numRows + " " + numCols + " " + minVal + " " + maxVal);

System.***out***.println(label);

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

PtAry[i].printCorner();

}

}

**public** **void** printVotes(){

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

PtAry[i].printVotes();

}

}

}

**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

5 5 9

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 9

16 6 1

17 7 1

18 8 1

19 9 1

20 10 9

21 9 1

22 8 1

23 7 1

24 6 1

25 5 9

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 9

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 9

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 9

24 30 1

23 31 1

22 32 1

21 33 1

20 34 9

19 33 1

18 32 1

17 31 1

16 30 1

15 29 9

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 9

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**

9 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 9

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

9 9

1 1

1 1

1 1

1 1

9 9

1 1

1 1

1 1

1 1

9 9

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

1 1

9 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 9