CSCI 381 (Java)

Section 31

Project 8.2 ArcCord Distance

Essam Yousry

Due Date of soft copy: 05/02/2018

Due Date of hard copy: 05/03/2018

Algorithm Steps in main

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

// initiallied to 0.0

- 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)

// find the max of distances of all points in chordAry

// and returns that index

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)

// so the computation will continue wrapped around the boundray

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()

// put each point (x, y)’s corner value (1 or 9) at Img(x, y)

Step 16: prettyPrint (img) to output2

Step 17: close all files

Source Code

import java.util.Scanner;

import java.io.FileReader;

import java.io.IOException;

import java.io.FileWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import java.util.\*;

import java.lang.\*;

public class ArcCordDistance{

    public static void storePt(int x, int y, int index, point [] ptAry){

        point p = new point(x, y);

        ptAry[index] = p;

    }

    public static void plotPt2Img(int [][] imgAry, point [] ptAry, int numPts){

        int index = 0;

        while (index < numPts){

            imgAry[ptAry[index].getRow()][ptAry[index].getCol()] = ptAry[index].getCorner();

            index++;

        }

    }

    public static double computeDistance(int P1, int P2, int Pt, point [] ptAry){

        double distance;

        int x = ptAry[Pt].getRow();

        int y = ptAry[Pt].getCol();

        int x1 = ptAry[P1].getRow();

        int y1 = ptAry[P1].getCol();

        int x2 = ptAry[P2].getRow();

        int y2 = ptAry[P2].getCol();

        int A = (y2 - y1);

        int B = (x1 - x2);

        int C = Math.abs(x2 \* y1) - Math.abs(x1 \* y2);

        double Den = Math.sqrt((A \* A) + (B \* B));

        int Denominator = (int) Den;

        distance = Math.abs((A \* x) + (B \* y) + C) / ((double)(Denominator));

        return Math.abs(distance);

    }

    public static int findMaxDistance(double [] chordAry, int chordLength){

        int max = 0;

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

            if (chordAry[i] > chordAry[max]) max = i;

        }

        return max;

    }

    public static boolean computeLocalMaxima(point [] ptAry, int i, int numPts){

        int mod = ((i - 2) % numPts + numPts) % numPts;

        boolean check = true;

        int votes = ptAry[i].getMaxVotes();

        for (int a = i - 2; a <= i + 2; a++){

            if (votes < ptAry[mod].getMaxVotes()){

                check = false;

                break;

            }

        }

        if (check == true){

            ptAry[i].setLocalMax(1);

            return true;

        }

        else{

            ptAry[i].setLocalMax(0);

            return false;

        }

    }

    public static int SetCorner(point [] ptAry, int i, int numPts){

        int mod = ((i - 2) % numPts + numPts) % numPts;

        boolean check1 = false;

        if (ptAry[i].getLocalMax() == 1){

            check1 = true;

        }

        boolean check2 = false;

        if ((ptAry[(i + 2) % numPts].getLocalMax() == 0) && (ptAry[mod].getLocalMax() == 0)){

            check2 = true;

        }

        if ((check1 == true) && (check2 == true)){

            ptAry[i].setCorner(9);

            return 9;

        }

        else {

            ptAry[i].setCorner(1);

            return 1;

        }

    }

    public static void prettyPrint(int [][] array, int row, int col, PrintWriter pw){

        for(int x = 0; x < row; x++){

            for (int y = 0; y < col; y++){

                if (array[x][y] > 0){

                    pw.print(array[x][y]);

                    pw.print(' ');

                }

                else {

                    pw.print(' ');

                    pw.print(' ');

                }

            }

            pw.println();

        }

    }

    public static void main(String [] args){

        int [] vars = new int [5];

        int row, col, min, max, label;

        int chordLength, P1, P2, maxIndex;

        int numPts = 0;

        File inFile = new File (args[0]);

        File outFile = new File (args[1]);

        File outFile2 = new File (args[2]);

        File outFile3 = new File (args[3]);

        try {

            PrintWriter out1 = new PrintWriter(new FileWriter(outFile));

            PrintWriter out2 = new PrintWriter(new FileWriter(outFile2));

            PrintWriter out3 = new PrintWriter(new FileWriter(outFile3));

            Scanner sc = new Scanner(inFile);

            Scanner user = new Scanner(System.in);

            int c = 0;

            while (sc.hasNextInt() && c != 5){

                vars[c++] = sc.nextInt();

            }

            row = vars[0];

            col = vars[1];

            min = vars[2];

            max = vars[3];

            label = vars[4];

            int count = 0;

            while (sc.hasNextInt()){

                sc.nextInt();

                count++;

            }

            numPts = count / 2;

            File inFile2 = new File (args[0]);

            Scanner sc2 = new Scanner(inFile2);

            int i = 0;

            while (i < 5){

                sc2.nextInt();

                i++;

            }

            point [] ptAry = new point [numPts];

            int index = 0;

            int R;

            int C;

            while (index < numPts){

                R = sc2.nextInt();

                C = sc2.nextInt();

                storePt(R, C, index, ptAry);

                index++;

            }

            int K;

            System.out.println("Please Enter a Value For K: ");

            K = user.nextInt();

            chordLength = 2 \* K;

            double [] chordAry = new double [chordLength];

            P1 = 0;

            P2 = chordLength - 1;

            out3.println(numPts);

            out3.print("P1");

            out3.print(' ');

            out3.print("x1");

            out3.print(' ');

            out3.print("y1");

            out3.print(' ');

            out3.print("P2");

            out3.print(' ');

            out3.print("x2");

            out3.print(' ');

            out3.print("y2");

            out3.print(' ');

            out3.print("CP");

            out3.print(' ');

            out3.print("x");

            out3.print(' ');

            out3.print("y");

            out3.print(' ');

            out3.print("dist");

            out3.print(' ');

            out3.print("ind");

            out3.print(' ');

            out3.print("chordAry[ind]");

            out3.println();

            out3.println();

            double dist;

            do{

                int ind = 0;

                int currPt = (P1) % numPts;

                do{

                    dist = computeDistance(P1, P2, currPt, ptAry);

                    chordAry[ind] = dist;

                    int x = ptAry[currPt].getRow();

                    int y = ptAry[currPt].getCol();

                    int x1 = ptAry[P1].getRow();

                    int y1 = ptAry[P1].getCol();

                    int x2 = ptAry[P2].getRow();

                    int y2 = ptAry[P2].getCol();

                    out3.print(P1);

                    out3.print(' ');

                    out3.print(x1);

                    out3.print(' ');

                    out3.print(y1);

                    out3.print(' ');

                    out3.print(P2);

                    out3.print(' ');

                    out3.print(x2);

                    out3.print(' ');

                    out3.print(y2);

                    out3.print(' ');

                    out3.print(currPt);

                    out3.print(' ');

                    out3.print(x);

                    out3.print(' ');

                    out3.print(y);

                    out3.print(' ');

                    out3.print(dist);

                    out3.print(' ');

                    out3.print(ind);

                    out3.print(' ');

                    out3.print(chordAry[ind]);

                    out3.println();

                    ind++;

                    currPt = (currPt + 1) % numPts;

                } while (ind < chordLength);

                maxIndex = findMaxDistance(chordAry, chordLength);

                int whichIndex = (P1 + maxIndex) % numPts;

                ptAry[whichIndex].setMaxVotes(ptAry[whichIndex].getMaxVotes() + 1);

                if (ptAry[whichIndex].getMaxDist() < chordAry[maxIndex]){

                    ptAry[whichIndex].setMaxDist(chordAry[maxIndex]);

                }

                P1++;

                if (P2 == numPts - 1){

                    P2 = 0;

                }

                else{

                    P2++;

                }

            }while (P2 != (chordLength) - 1);

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

                computeLocalMaxima(ptAry, z, numPts);

            }

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

                SetCorner(ptAry, j, numPts);

            }

            out3.print("-----------------------------------------");

            out3.println();

            out3.print("ptAry");

            out3.println();

            out3.println();

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

                if(ptAry[x].getMaxVotes() > 1){

                    ptAry[x].setCorner(9);

                }

                else ptAry[x].setCorner(1);

            }

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

                out3.print(ptAry[a].getRow());

                out3.print(' ');

                out3.print(ptAry[a].getCol());

                out3.print(' ');

                out3.print(ptAry[a].getMaxDist());

                out3.print(' ');

                out3.print(ptAry[a].getMaxVotes());

                out3.print(' ');

                out3.print(ptAry[a].getCorner());

                out3.println();

            }

            out1.print(row);

            out1.print(' ');

            out1.print(col);

            out1.print(' ');

            out1.print(min);

            out1.print(' ');

            out1.print(max);

            out1.println();

            out1.print(label);

            out1.println();

            out1.print(numPts);

            out1.println();

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

                out1.print(ptAry[k].getRow());

                out1.print(' ');

                out1.print(ptAry[k].getCol());

                out1.print(' ');

                out1.print(ptAry[k].getCorner());

                out1.println();

            }

            int [][] imgAry = new int [row][col];

            plotPt2Img(imgAry, ptAry, numPts);

            prettyPrint(imgAry, row, col, out2);

            sc.close();

            sc2.close();

            user.close();

            out1.close();

            out2.close();

            out3.close();

        }

        catch (Exception e) {

            System.out.println("Error " + e);

        }

    }

}

class point{

    int row, col, corner, maxVotes, localMax;

    double maxDistance;

    public point(int row, int col){

        this.row = row;

        this.col = col;

    }

    void setMaxVotes(int maxVotes){

        this.maxVotes = maxVotes;

    }

    int getMaxVotes(){ return maxVotes; }

    void setMaxDist(double maxDistance){

        this.maxDistance = maxDistance;

    }

    double getMaxDist() { return maxDistance; }

    void setLocalMax(int localMax){

        this.localMax = localMax;

    }

    int getLocalMax(){ return localMax; }

    void setCorner(int corner){

        this.corner = corner;

    }

    int getCorner(){ return corner; }

    int getRow(){ return row; }

    int getCol(){ return col; }

}

Output







