Computer Vision Programming Language: Java

Project #7 Medial Axis/Distance Transform

Andres Quintero

Due Date:

Soft copy: 3/29/2020

Hard copy: 3/29/2020

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

step 0: inFile 🡨 open input file

numRows, numCols, minVal, maxVal 🡨 read from inFile

dynamically allocate zeroFramedAry with extra 2 rows and 2 cols

dynamically allocate skeletonAry with extra 2 rows and 2 cols

open outFile\_1, outFile\_2

Step 1: skeletonFileName 🡨 argv[1] + “\_skeleton”

Step 2: skeletonFile 🡨 open ( skeletonFileName )

Step 3: decompressedFileName 🡨 argv[1] + “\_decompressed”

Step 4: decompressFile 🡨 open (decompressedFileName)

step 5: setZero (zeroFramedAry)

setZero (skeletonAry)

Step 6: loadImage (inFile, zeroFramedAry) // begins at zeroFramedAry (1,1)

Step 7: compute8Distance (zeroFramedAry, outFile1) // Perform distance transform

Step 8: skeletonExtraction (zeroFramedAry, skeletonAry, skeletonFile, outFile1)

// perform lossless compression

Step 9: skeletonExpansion (zeroFramedAry, skeletonFile, outFile2)

// perform decompression

step 10: Output numRows, numCols, newMinVal, newMaxVal to decompressFile

Step 11: ary2File (zeroFramedAry, decompressFile)

Step 12: close all files

**Source code:**

import java.util.\*;

import java.io.\*;

import java.lang.Math;

public class Main {

public static void main(String[] args) {

int numRows, numCols, minVal, maxVal;

Scanner inFile = null;

PrintWriter outFile1 = null;

PrintWriter outFile2 = null;

// Opened after reading input image

PrintWriter skeletonFile = null;

PrintWriter decompressedFile = null;

// 0

try {

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

} catch (FileNotFoundException err) {

System.out.println("Error in opening inputFile: " + err);

}

numRows = inFile.nextInt();

numCols = inFile.nextInt();

minVal = inFile.nextInt();

maxVal = inFile.nextInt();

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

int[][] skeletonAry = new int[numRows+2][numCols+2];

try{

outFile1 = new PrintWriter(args[1]);

outFile2 = new PrintWriter(args[2]);

} catch (FileNotFoundException err) {

System.out.println("Error in opening files from CLI: " + err);

}

// 1 - 4

// String name correction

String fileName = args[0];

int pos = fileName.indexOf(".txt");

fileName = fileName.substring(0,pos);

// Creating and Opening Skeleton File

String skeletonFileName = new String(fileName+"\_skeleton.txt");

try {

skeletonFile = new PrintWriter(skeletonFileName);

} catch (FileNotFoundException err) {

System.out.println("Error in opening skeleton file: " + err);

}

// Creating and Opening decompressedFile

String decompressedFileName = new String(fileName + "\_decompressed.txt");

try {

decompressedFile = new PrintWriter(decompressedFileName);

} catch (FileNotFoundException err) {

System.out.println("Error in opening decompressed file: " + err);

}

// 5

setZero(zeroFramedAry, numRows+2, numCols+2);

setZero(skeletonAry, numRows+2, numCols+2);

// 6

loadImage(zeroFramedAry, inFile);

//DEBUGSTUFF

// print2DArray(zeroFramedAry, numRows+2, numCols+2);

// outFile1.println("zeroFramedAry prettyPrint test: ");

// prettyPrint(zeroFramedAry, outFile1);

// 7

compute8Distance(zeroFramedAry, outFile1, fileName);

//7.5 finding newMin and newMax values

int newMinVal = Integer.MAX\_VALUE;

int newMaxVal = 0;

for (int i = 1; i < zeroFramedAry.length-1; i++ ){

for (int j = 1; j < zeroFramedAry[0].length-1 ; j++ ){

if(zeroFramedAry[i][j] < newMinVal){newMinVal = zeroFramedAry[i][j];}

if(zeroFramedAry[i][j] > newMaxVal){newMaxVal = zeroFramedAry[i][j];}

}

}

//8

skeletonFile.println(numRows + " " + numCols + " " + (newMinVal+1) + " " + newMaxVal); // skeleton header

skeletonExtraction(zeroFramedAry, skeletonAry, skeletonFile, outFile1, skeletonFileName);

// 8.5 reOpening skeletonFile as skeletonFileRead;

Scanner skeletonFileRead = null;

try { skeletonFileRead = new Scanner(new File(skeletonFileName));}

catch (FileNotFoundException err) {System.out.println("Error in re-opening skeleton file: " + err);}

// 9

skeletonExpansion(zeroFramedAry, skeletonFileRead, outFile2);

// 10

decompressedFile.println(numRows + " " + numCols + " " + minVal + " " + maxVal);

// 11

ary2File(zeroFramedAry, decompressedFile);

// 12

inFile.close();

outFile1.close();

outFile2.close();

skeletonFile.close();

decompressedFile.close();

skeletonFileRead.close();

}

// Functions

static void ary2File(int[][] zeroFramedAry, PrintWriter decompressedFile){

int thresVal = 1;

for(int i = 1; i < zeroFramedAry.length-1; i++){

for(int j = 1; j < zeroFramedAry[0].length-1; j++){

if(zeroFramedAry[i][j] >= thresVal){

decompressedFile.print(1 + " ");

} else {

decompressedFile.print(0 + " ");

}

}

decompressedFile.println();

}

}

static void load(Scanner skeletonFileRead, int[][] zeroFramedAry){

//read header

int r = skeletonFileRead.nextInt();

int c = skeletonFileRead.nextInt();

int min = skeletonFileRead.nextInt();

int max = skeletonFileRead.nextInt();

//

int i , j, value;

while(skeletonFileRead.hasNext()){

i = skeletonFileRead.nextInt();

j = skeletonFileRead.nextInt();

value = skeletonFileRead.nextInt();

zeroFramedAry[i][j] = value;

}

}

static void skeletonExpansion(int[][] zeroFramedAry, Scanner skeletonFileRead, PrintWriter outFile2){

setZero(zeroFramedAry, zeroFramedAry.length, zeroFramedAry[0].length);

load(skeletonFileRead, zeroFramedAry);

// // DEBUGSTUFF

// prettyPrint(zeroFramedAry, outFile2);

firstPassExpansion(zeroFramedAry);

outFile2.println("After firstPassExpansion():");

prettyPrint(zeroFramedAry, outFile2);

secondPassExpansion(zeroFramedAry);

outFile2.println("After secondPassExpansion():");

prettyPrint(zeroFramedAry, outFile2);

}

static void extractLocalMaxima(int[][] skeletonAry, PrintWriter skeletonFile){

for(int i = 1; i < skeletonAry.length-1; i++){

for(int j = 1; j < skeletonAry[0].length-1; j++){

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

skeletonFile.println(i + " " + j + " " + skeletonAry[i][j]);

}

}

}

}

static void computeLocalMaxima(int[][] zeroFramedAry, int[][] skeletonAry){

for (int i = 1; i < zeroFramedAry.length-1; i++ ){

for (int j = 1; j < zeroFramedAry[0].length-1 ; j++ ) {

int max = 0;

// finding max of NEIGHBORS ONLY

for(int a = i-1; a < i+2; a++){

for(int b = j-1; b < j+2; b++){

if(a != 0 && b!= 0){ // skips self

if(zeroFramedAry[a][b] > max){max = zeroFramedAry[a][b];}

}

}

}

if(zeroFramedAry[i][j] >= max){

skeletonAry[i][j] = zeroFramedAry[i][j];

} else {

skeletonAry[i][j] = 0;

}

}

}

}

static void skeletonExtraction(int[][] zeroFramedAry, int[][] skeletonAry, PrintWriter skeletonFile, PrintWriter outFile1, String skeletonFileName){

computeLocalMaxima(zeroFramedAry, skeletonAry);

outFile1.println(skeletonFileName+ " after computeLocalMaxima():");

prettyPrint(skeletonAry, outFile1);

extractLocalMaxima(skeletonAry, skeletonFile);

skeletonFile.close();

}

static int maxNeighbors(int[][] Ary, int i, int j){

int max = 0;

for(int a = i-1; a < i+2; a++){

for(int b = j-1; b < j+2; b++){

if(a != 0 && b!= 0){ // skips self

if(Ary[a][b] > max){max = Ary[a][b];}

}

}

}

return max;

}

static void secondPassExpansion(int[][] Ary){

for(int i = Ary.length-1-1; i > 0 ; i--){

for(int j = Ary[0].length-1-1; j > 0 ; j--){

int max = 0;

// finding max of NEIGHBORS ONLY

for(int a = i-1; a < i+2; a++){

for(int b = j-1; b < j+2; b++){

if(a != 0 && b!= 0){ // skips self

if(Ary[a][b] > max){max = Ary[a][b];}

}

}

}

if(Ary[i][j] < max){

Ary[i][j] = max - 1;

}

}

}

}

static void firstPassExpansion(int[][] Ary){

for(int i = 1; i < Ary.length-1; i++){

for(int j = 1; j < Ary[0].length-1; j++){

if(Ary[i][j] == 0){

int max = 0;

// finding max of NEIGHBORS ONLY

for(int a = i-1; a < i+2; a++){

for(int b = j-1; b < j+2; b++){

if(a != 0 && b!= 0){ // skips self

if(Ary[a][b] > max){max = Ary[a][b];}

}

}

}

max--;

if(Ary[i][j] < max){

Ary[i][j] = max;

}

}

}

}

}

static void secondPass\_8Distance(int[][] Ary){

int min = Integer.MAX\_VALUE;

for(int i = Ary.length-1-1; i > 0 ; i--){

for(int j = Ary[0].length-1-1; j > 0 ; j--){

min = Math.min(min, Ary[i][j+1]); //e

min = Math.min(min, Ary[i+1][j-1]); //f

min = Math.min(min, Ary[i+1][j]); //g

min = Math.min(min, Ary[i+1][j+1]); //h

Ary[i][j] = Math.min(Ary[i][j], min + 1);

min = Integer.MAX\_VALUE;

}

}

}

static void firstPass\_8Distance(int[][] Ary){

int min = Integer.MAX\_VALUE;

for(int i = 1; i < Ary.length-1; i++){

for(int j = 1; j < Ary[0].length-1; j++){

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

min = Math.min(min, Ary[i-1][j-1]); //a

min = Math.min(min, Ary[i-1][j]); //b

min = Math.min(min, Ary[i-1][j+1]); //c

min = Math.min(min, Ary[i][j-1]); //d

Ary[i][j] = min + 1;

min = Integer.MAX\_VALUE;

}

}

}

}

static void compute8Distance(int[][] zeroFramedAry, PrintWriter outFile1, String fileName){

firstPass\_8Distance(zeroFramedAry);

outFile1.println(fileName + " after firstPass\_8Distance():");

prettyPrint(zeroFramedAry, outFile1);

secondPass\_8Distance(zeroFramedAry);

outFile1.println(fileName + " after secondPass\_8Distance():");

prettyPrint(zeroFramedAry, outFile1);

}

static void prettyPrint(int[][]Ary, PrintWriter outFile){

for(int i = 1; i < Ary.length-1; i++){

for(int j = 1; j < Ary[0].length-1; j++){

if(Ary[i][j] == 0){

outFile.print(" ");//2 spaces

} else {

outFile.print(Ary[i][j] + " ");

}

}

outFile.println();

}

}

static void loadImage(int[][] frameAry, Scanner image){

int value;

for (int i = 1; i < frameAry.length-1; i++ ){

for (int j = 1; j < frameAry[0].length-1 ; j++ ) {

value = image.nextInt();

frameAry[i][j] = value;

}

}

}

static void setZero(int[][] Ary, int rows, int cols){

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

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

Ary[i][j] = 0;

}

}

}

// DEBUGSTUFF

static void print2DArray(int[][] Ary, int rows, int cols){

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

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

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

}

System.out.println();

}

}

}

**image1 files:**

**A picture containing street

Description automatically generated**

**A close up of a piece of paper

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**A picture containing plane, white

Description automatically generated**

**END OF image1 OUTPUTS**

**image2 files:**

**A close up of a logo

Description automatically generated**

**A picture containing clock

Description automatically generated**

**A screenshot of a social media post

Description automatically generated**

**A screenshot of a cell phone

Description automatically generated**

**A picture containing window

Description automatically generated**

**END OF image2 OUTPUTS**