CS 323\_33 Programming Language: Java

Project #5 QuadTree

Andres Quintero

Due Date:

Soft copy: 3/10/2020

Hard copy: 3/12/2020

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

Step 0: inFile, outFile1, outFile2 🡨open

Step 1: numRows, numCols, minVal, maxVal 🡨read from inFile

Step 2: squareSize 🡨computeSquare(numRows, numCols)

Step 3: imgAry 🡨dynamically allocate the array size of squaresze by squareSide

Step 4: zero2DAry (imgAry)

Step 5: loadImage (inFile, imgAry)

Step 4: QtRoot 🡨BuildQuadTree (imgAry, 0, 0, squareSize)

Step 5: preOrder (QtRoot, outFile2)

Step 6: postOrder (QtRoot, outFile2)

Step 7: close all files

**Source code:**

import java.util.\*;

import java.io.\*;

class QtTreeNode{

int color;

int upperR;

int upperC;

QtTreeNode NWkid = null;

QtTreeNode NEkid = null;

QtTreeNode SWkid = null;

QtTreeNode SEkid = null;

QtTreeNode(int color, int upperR, int upperC, QtTreeNode NWkid, QtTreeNode NEkid, QtTreeNode SWkid, QtTreeNode SEkid){

this.color = color;

this.upperR = upperR;

this.upperC = upperC;

this.NWkid = NWkid;

this.NEkid = NEkid;

this.SWkid = SWkid;

this.SEkid = SEkid;

}

}

public class Main{

public static void main(String[] args) {

// Opening Files

Scanner inFile = null;

PrintWriter outFile1 = null;

PrintWriter outFile2 = null;

try {

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

} catch (FileNotFoundException err) {

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

}

try {

outFile1 = new PrintWriter(args[1]);

outFile2 = new PrintWriter(args[2]);

} catch (FileNotFoundException err) {

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

}

int numRows, numCols, minVal, maxVal, squareSize;

numRows = inFile.nextInt();

numCols = inFile.nextInt();

minVal = inFile.nextInt();

maxVal = inFile.nextInt();

squareSize = computeSquareSize(numRows, numCols);

int[][] imgAry = new int[squareSize][squareSize];

zero2DAry(imgAry); //Is this needed again?

loadImage(inFile, imgAry, numRows, numCols);

//deBugOut

outFile2.println("squareSize: " + squareSize);

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

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

outFile2.print(imgAry[i][j] + " ");

}

outFile2.println();

}

QtTreeNode QtRoot = buildQuadTree(imgAry, 0, 0, squareSize);

outFile1.println("PreOrder Traversal:");

preOrder(QtRoot, outFile1);

outFile1.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

outFile1.println("PostOrder Traversal:");

postOrder(QtRoot, outFile1);

inFile.close();

outFile1.close();

outFile2.close();

}

// Functions

public static QtTreeNode buildQuadTree(int[][] imgAry, int upR, int upC, int size){

QtTreeNode newNode = new QtTreeNode(-1, upR, upC, null, null, null, null);

if(size == 1){

newNode.color = imgAry[upR][upC];

} else {

int newSize = size/2;

newNode.NWkid = buildQuadTree(imgAry,upR,upC,newSize);

newNode.NEkid = buildQuadTree(imgAry,upR,upC+newSize,newSize);

newNode.SWkid = buildQuadTree(imgAry,upR+newSize,upC,newSize);

newNode.SEkid = buildQuadTree(imgAry,upR+newSize,upC+newSize,newSize);

int sumColor = newNode.NWkid.color + newNode.NEkid.color + newNode.SWkid.color + newNode.SEkid.color;

if(sumColor == 0){

newNode.color = 0;

newNode.NWkid = null;

newNode.NEkid = null;

newNode.SWkid = null;

newNode.SEkid = null;

} else if (sumColor == 4){

newNode.color = 1;

newNode.NWkid = null;

newNode.NEkid = null;

newNode.SWkid = null;

newNode.SEkid = null;

} else {

newNode.color = 5;

}

}

return newNode;

}

public static void printQNode(QtTreeNode node, PrintWriter outFile){

String nodeColor = node == null ? "NULL" : Integer.toString(node.color);

String nodeR = node == null ? "NULL" : Integer.toString(node.upperR);

String nodeC = node == null ? "NULL" : Integer.toString(node.upperC);

String nodeNW = node.NWkid == null ? "NULL" : Integer.toString(node.NWkid.color);

String nodeNE = node.NEkid == null ? "NULL" : Integer.toString(node.NEkid.color);

String nodeSW = node.SWkid == null ? "NULL" : Integer.toString(node.SWkid.color);

String nodeSE = node.SEkid == null ? "NULL" : Integer.toString(node.SEkid.color);

outFile.println("(" +nodeColor + " " + nodeR + " " + nodeC + " " +nodeNW + " " +nodeNE + " " +nodeSW + " " +nodeSE + ")");

}

public static void postOrder(QtTreeNode node, PrintWriter outFile){

if(node.NWkid == null && node.NEkid == null && node.SWkid == null && node.SEkid == null){

printQNode(node, outFile);

} else {

postOrder(node.NWkid, outFile);

postOrder(node.NEkid, outFile);

postOrder(node.SWkid, outFile);

postOrder(node.SEkid, outFile);

printQNode(node, outFile);

}

}

public static void preOrder(QtTreeNode node, PrintWriter outFile){

if(node.NWkid == null && node.NEkid == null && node.SWkid == null && node.SEkid == null){

printQNode(node, outFile);

} else {

printQNode(node, outFile);

preOrder(node.NWkid, outFile);

preOrder(node.NEkid, outFile);

preOrder(node.SWkid, outFile);

preOrder(node.SEkid, outFile);

}

}

public static void loadImage(Scanner inFile, int[][] Ary, int rows, int cols){

int value;

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

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

value = inFile.nextInt();

Ary[i][j] = value;

}

}

}

public static void zero2DAry(int[][] Ary){

for(int i = 0; i< Ary.length; i++){

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

Ary[i][j] = 0;

}

}

}

public static int computeSquareSize(int numRows, int numCols){

int square = numRows > numCols ? numRows : numCols;

int power2 = 2;

while(square > power2){

power2 \*= 2;

}

return power2;

}

}

**SquareData: OutFile and QuadTree Representation**



A close up of a piece of paper

Description automatically generated

A close up of text on a whiteboard

Description automatically generated

A close up of a logo

Description automatically generated

**Not\_SquareData: OutFile and QuadTree Representation**

**A close up of a logo

Description automatically generated**

A close up of a logo

Description automatically generated

A picture containing text

Description automatically generated