Student: Michael Grossman

Due Date: 4/8/2022

Algorithm Steps for Draw Boxes given a property file and an image with labeled components:

1. Index ←1
2. minRow ← CCProperty[Index].minR + 1
3. minCol ← CCProperty[Index].minC + 1
4. maxRow ← CCProperty[Index].maxR + 1
5. maxCol ← CCProperty[Index].maxC + 1
6. label ← CCProperty[Index].label
7. Assign all pixels on minRow from minCol to maxCol ←label
8. Assign all pixel on maxRow from minCol to maxCol ← label
9. Assign all pixels on minCol from minRow to maxRow ← label
10. Assign all pixels on maxCol from minRow to maxRow ← label
11. Index++
12. Repeat steps 2 to 11 while index <= trueNumCC

**Main.cpp**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Property{

    int label, numpixels, minR, minC, maxR, maxC;

};

class CClabel{

    public:

    //variables

    int numRows, numCols, minVal, maxVal;

    int newMin, newMax, newLabel, trueNumCC;

    int \*\*zeroFramedAry;

    int\* nonZeroNeighborAry;

    int\* EQAry;

    Property\* CCProperty;

    //constructor + destructor

    CClabel(int \* params, ifstream &inp);

    ~CClabel();

    //functions

    void zero2D();

    void minus1D();

    void loadImage(ifstream &inp);

    void imgReformat(ofstream &outp);

    void connect8Pass1();

    void connect8Pass2();

    void connect4Pass1();

    void connect4Pass2();

    void connectPass3();

    void drawBoxes();

    void updateEQ(int\* inp, int\* eq, int count, int min);

    int manageEQAry();

    void printCCProperty(ofstream &outp);

    void printEQAry(ofstream &outp);

    void printImg(ofstream &outp);

};

int main(int argc, char\*\* argv){

    //pull command line args

    ifstream image(argv[1]);

    int connectedness = atoi(argv[2]);

    ofstream RFPrettyPrintFile(argv[3]), labelFile(argv[4]), propertyFile(argv[5]);

    //read in the header info from the image

    int params[4];

    for(int i = 0; i < 4; ++i){

        image >> params[i];

    }

    RFPrettyPrintFile << argv[1] << " - First Pass of Connected Components ";

    CClabel cclabel(params, image);

    if(connectedness == 4){

        RFPrettyPrintFile << "with 4-connected:\n";

        cclabel.connect4Pass1();

        cclabel.imgReformat(RFPrettyPrintFile);

        cclabel.printEQAry(RFPrettyPrintFile);

        RFPrettyPrintFile << argv[1] << " - Second Pass of Connected Components with 4-connected:\n";

        cclabel.connect4Pass2();

    }

    else{

        RFPrettyPrintFile << argv[1] << " - with 8-connected:\n";

        cclabel.connect8Pass1();

        cclabel.imgReformat(RFPrettyPrintFile);

        cclabel.printEQAry(RFPrettyPrintFile);

        RFPrettyPrintFile << argv[1] << " - Second Pass of Connected Components with 8-connected:\n";

        cclabel.connect8Pass2();

    }

    cclabel.imgReformat(RFPrettyPrintFile);

    cclabel.printEQAry(RFPrettyPrintFile);

    cclabel.trueNumCC = cclabel.manageEQAry();

    RFPrettyPrintFile << argv[1] << " - updating equality table labels:\n";

    cclabel.printEQAry(RFPrettyPrintFile);

    cclabel.connectPass3();

    RFPrettyPrintFile << argv[1] << " - Third Pass of Connect Components:\n";

    cclabel.imgReformat(RFPrettyPrintFile);

    cclabel.printEQAry(RFPrettyPrintFile);

    labelFile << to\_string(cclabel.numRows) + " " + to\_string(cclabel.numCols) + " " + to\_string(cclabel.newMin) + " " + to\_string(cclabel.newMax) + "\n";

    cclabel.printImg(labelFile);

    cclabel.printCCProperty(propertyFile);

    cclabel.drawBoxes();

    cclabel.imgReformat(RFPrettyPrintFile);

    RFPrettyPrintFile << "True Number of Connected Components: " + to\_string(cclabel.trueNumCC) + "\n";

    //close all files

    image.close();

    RFPrettyPrintFile.close();

    labelFile.close();

    propertyFile.close();

    return 0;

}

CClabel::CClabel(int\* params, ifstream &inp){

    numRows = params[0];

    numCols = params[1];

    minVal = params[2];

    maxVal = params[3];

    int rows = numRows + 2, cols = numCols + 2;

    zeroFramedAry = new int\*[rows];

    for(int i = 0; i < rows; ++i){

        zeroFramedAry[i] = new int[cols];

    }

    EQAry = new int[(numRows\*numCols)/4 + 1];

    nonZeroNeighborAry = new int[5];

    //set EQarray to store all -1's

    minus1D();

    EQAry[0] = 0;

    //make all indices of zeroFramedArray 0

    zero2D();

    //load image into zeroFramedArray

    loadImage(inp);

    newLabel = 0;

}

CClabel::~CClabel(){

    int rows = numRows + 2;

    for(int i = 0; i < rows; ++i){

        delete[] zeroFramedAry[i];

    }

    delete[] zeroFramedAry;

    delete[] EQAry;

    delete[] nonZeroNeighborAry;

}

void CClabel::zero2D(){

    int rows = numRows + 2, cols = numCols + 2;

    for(int i = 0; i < rows; ++i){

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

            zeroFramedAry[i][j] = 0;

        }

    }

}

void CClabel::minus1D(){

    int len = numRows \* numCols;

    len /= 4;

    for(int i = 0; i < len; ++i){

        EQAry[i] = -1;

    }

}

void CClabel::loadImage(ifstream &inp){

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

        for(int j = 1; j<= numCols; ++j){

            inp >> zeroFramedAry[i][j];

        }

    }

}

void CClabel::imgReformat(ofstream &outp){

    int width = to\_string(newLabel).length();

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

        for(int j = 1; j<= numCols; ++j){

            if(zeroFramedAry[i][j] < 1){

                outp << ". ";

            }

            else{

                outp << to\_string(zeroFramedAry[i][j]) << " ";

            }

            for(int ww = to\_string(zeroFramedAry[i][j]).length(); ww < width; ++ww){

                outp << " ";

            }

        }

        outp << " \n";

    }

    outp << "\n\n";

}

void CClabel::connect8Pass1(){

    int \*\*p = zeroFramedAry, min = 99999, max = 0;

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

        for(int j = 1; j<= numCols; ++j){

            if(p[i][j] > 0){

                max = 0;

                min = 99999;

                nonZeroNeighborAry[0] = p[i-1][j-1];

                nonZeroNeighborAry[1] = p[i-1][j];

                nonZeroNeighborAry[2] = p[i-1][j+1];

                nonZeroNeighborAry[3] = p[i][j-1];

                for(int k = 0; k < 4; ++k) max |= nonZeroNeighborAry[k];

                //case 1

                if(max == 0){

                    newLabel++;

                    EQAry[newLabel] = newLabel;

                    p[i][j] = newLabel;

                }

                else{

                    max = 0;

                    for(int k = 0; k < 4 ; ++k){

                        if(nonZeroNeighborAry[k] != 0){

                            max = EQAry[nonZeroNeighborAry[k]] > max ? EQAry[nonZeroNeighborAry[k]] : max;

                            min = EQAry[nonZeroNeighborAry[k]] < min ? EQAry[nonZeroNeighborAry[k]] : min;

                        }

                    }

                    //case 2

                    if(min == max){

                        p[i][j] = max;

                    }

                    //case 3

                    else{

                        p[i][j] = min;

                        updateEQ(nonZeroNeighborAry, EQAry, 4, min);

                    }

                }

            }

        }

    }

}

void CClabel::connect8Pass2(){

    int \*\*p = zeroFramedAry, min = 99999, max = 0;

    for(int i = numRows; i > 0; --i){

        for(int j = numCols; j > 0; --j){

            if(p[i][j] > 0){

                max = 0;

                min = 99999;

                nonZeroNeighborAry[0] = p[i+1][j-1];

                nonZeroNeighborAry[1] = p[i+1][j];

                nonZeroNeighborAry[2] = p[i+1][j+1];

                nonZeroNeighborAry[3] = p[i][j+1];

                nonZeroNeighborAry[4] = p[i][j];

                for(int k = 0; k < 4; ++k) max |= nonZeroNeighborAry[k];

                //case 1 if max == 0 do nothing

                if(max != 0){

                    max = 0;

                    for(int k = 0; k < 5 ; ++k){

                        if(nonZeroNeighborAry[k] != 0){

                            max = EQAry[nonZeroNeighborAry[k]] > max ? EQAry[nonZeroNeighborAry[k]] : max;

                            min = EQAry[nonZeroNeighborAry[k]] < min ? EQAry[nonZeroNeighborAry[k]] : min;

                        }

                    }

                    //case 2 - if they are all the same do nothing

                    if(max != min){

                        //case 3

                        EQAry[p[i][j]] = min;

                        p[i][j] = min;

                        updateEQ(nonZeroNeighborAry, EQAry, 5, min);

                    }

                }

            }

            p[i][j] = EQAry[p[i][j]];

        }

    }

}

void CClabel::connect4Pass1(){

    int\*\* p = zeroFramedAry, max = 0;

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

        for(int j = 1; j<= numCols; ++j){

            if(p[i][j] > 0){

                nonZeroNeighborAry[0] = p[i-1][j];

                nonZeroNeighborAry[1] = p[i][j-1];

                max =  nonZeroNeighborAry[0] | nonZeroNeighborAry[1];

                //case 1

                if(max == 0){

                    newLabel++;

                    EQAry[newLabel] = newLabel;

                    p[i][j] = newLabel;

                }

                else{

                    //case 2

                    if(nonZeroNeighborAry[0] == nonZeroNeighborAry[1]){

                        p[i][j] = nonZeroNeighborAry[0];

                    }

                    else if(nonZeroNeighborAry[0] == 0 || nonZeroNeighborAry[1] == 0){

                        p[i][j] = nonZeroNeighborAry[0] + nonZeroNeighborAry[1];

                    }

                    //case 3

                    else{

                        p[i][j] = EQAry[nonZeroNeighborAry[0]] < EQAry[nonZeroNeighborAry[1]] ? EQAry[nonZeroNeighborAry[0]] : EQAry[nonZeroNeighborAry[1]];

                        updateEQ(nonZeroNeighborAry, EQAry, 2, p[i][j]);

                    }

                }

            }

        }

    }

}

void CClabel::connect4Pass2(){

    int \*\*p = zeroFramedAry, min = 99999, max = 0;

    for(int i = numRows; i > 0; --i){

        for(int j = numCols; j > 0; --j){

            if(p[i][j] > 0){

                min = 99999;

                nonZeroNeighborAry[0] = p[i+1][j];

                nonZeroNeighborAry[1] = p[i][j+1];

                nonZeroNeighborAry[2] = p[i][j];

                max =  nonZeroNeighborAry[0] | nonZeroNeighborAry[1];

                //case 1 if max == 0 do nothing

                if(max != 0){

                    max = 0;

                    for(int k = 0; k < 3 ; ++k){

                        if(nonZeroNeighborAry[k] != 0){

                            max = EQAry[nonZeroNeighborAry[k]] > max ? EQAry[nonZeroNeighborAry[k]] : max;

                            min = EQAry[nonZeroNeighborAry[k]] < min ? EQAry[nonZeroNeighborAry[k]] : min;

                        }

                    }

                    //case 2 - if they are all the same do nothing

                    if(max != min){

                        //case 3

                        EQAry[p[i][j]] = min;

                        p[i][j] = min;

                        updateEQ(nonZeroNeighborAry, EQAry, 3, min);

                    }

                }

            }

            p[i][j] = EQAry[p[i][j]];

        }

    }

}

void CClabel::connectPass3(){

    CCProperty = new Property[trueNumCC+1];

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

        CCProperty[i].label = i;

        CCProperty[i].numpixels = 0;

        CCProperty[i].minR = 99999;

        CCProperty[i].minC = 99999;

        CCProperty[i].maxR = 0;

        CCProperty[i].maxC= 0;

    }

    int\*\* p = zeroFramedAry;

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

        for(int j = 1; j <= numCols; ++j){

            if(p[i][j] > 0){

                p[i][j] = EQAry[p[i][j]];

                CCProperty[p[i][j]].numpixels++;

                CCProperty[p[i][j]].minR = i < CCProperty[p[i][j]].minR ? i : CCProperty[p[i][j]].minR;

                CCProperty[p[i][j]].minC = j < CCProperty[p[i][j]].minC ? j : CCProperty[p[i][j]].minC;

                CCProperty[p[i][j]].maxR = i > CCProperty[p[i][j]].maxR ? i : CCProperty[p[i][j]].maxR;

                CCProperty[p[i][j]].maxC = j > CCProperty[p[i][j]].maxC ? j : CCProperty[p[i][j]].maxC;

            }

        }

    }

}

void CClabel::drawBoxes(){

    int sr, sc, er, ec, label;

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

        //only draw if you are not overwriting an existing pixel

        for(int j = CCProperty[i].minC; j <= CCProperty[i].maxC; ++j){

            if(zeroFramedAry[CCProperty[i].minR][j] < 1){

                zeroFramedAry[CCProperty[i].minR][j] = CCProperty[i].label;

            }

            if(zeroFramedAry[CCProperty[i].maxR][j] < 1){

                zeroFramedAry[CCProperty[i].maxR][j] = CCProperty[i].label;

            }

        }

        for(int j = CCProperty[i].minR; j <= CCProperty[i].maxR; ++j){

            if(zeroFramedAry[j][CCProperty[i].minC] < 1){

                zeroFramedAry[j][CCProperty[i].minC] = CCProperty[i].label;

            }

            if(zeroFramedAry[j][CCProperty[i].maxC] < 1){

                zeroFramedAry[j][CCProperty[i].maxC] = CCProperty[i].label;

            }

        }

    }

}

void CClabel::updateEQ(int\* inp, int\* eq, int count, int min){

    for(int m = 0; m < count; ++m){

        if(inp[m] > 0) eq[inp[m]] = min;

    }

}

int CClabel::manageEQAry(){

    int label = 0, index = 1;

    while(index <= newLabel){

        if(EQAry[index] != index) EQAry[index] = EQAry[EQAry[index]];

        else{

            label++;

            EQAry[index] = label;

        }

        index++;

    }

    return label;

}

void CClabel::printCCProperty(ofstream &outp){

    outp << numRows << " " << numCols << " " << minVal << " " << maxVal << "\n";

    outp << trueNumCC << "\n";

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

        outp << CCProperty[i].label << "\n";

        outp << CCProperty[i].numpixels << "\n";

        //subtract frame from locatiion

        outp << CCProperty[i].minR - 1 << " " << CCProperty[i].minC - 1 << "\n";

        outp << CCProperty[i].maxR - 1 << " " << CCProperty[i].maxC - 1 << "\n";

    }

    outp << "\n\n";

}

void CClabel::printEQAry(ofstream &outp){

    outp << "Equality Array:\n";

    int width = to\_string(newLabel).length();

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

        outp << "| " << i << " ";

        for(int j = to\_string(i).length(); j < width; ++j){

            outp << " ";

        }

    }

    outp << "|\n";

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

        outp << "| " << EQAry[i] << " ";

        for(int j = to\_string(EQAry[i]).length(); j < width; ++j){

            outp << " ";

        }

    }

    outp << "|\n\n";

}

void CClabel::printImg(ofstream &outp){

    outp << numRows << " " << numCols << " " << minVal << " " << maxVal << "\n";

    int\*\* p = zeroFramedAry;

    int width = to\_string(newLabel).length();

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

        for(int j = 1; j <= numCols; ++j){

            outp << p[i][j] << " ";

            for(int ww = to\_string(p[i][j]).length(); ww < width; ++ww){

                outp << " ";

            }

        }

        outp << "\n";

    }

    outp << "\n\n";

}

**OUTPUT**

