

Name: Alif Rahi
Section: CSCI 381 - Computer Vision / Tues-Thurs 1:40-2:55pm
Project 8
Due: May 11, 2023

IV. main (...)

Step 0: open all files from argv[]

thrVal \leftarrow argv[2]

numRows, numCols, minVal, maxVal \leftarrow inFile

numStructRows, numStructCols, StructMin, StructMax, rowOrigin, colOrigin \leftarrow structElemFile or hard coded
use constructor to establish, allocate, and initialize all members of docImage class; unlike Java, C++ does NOT do
any initialization; therefore, make sure to initialize as indicate in the date structure in the above.

Step 1: loadImage (inFile, imgAry)

outFile1 \leftarrow "Below is the input image"

imgReformat (imgAry, outFile1)

Step 2: computePP (imgAry)

outFile2 \leftarrow "Below is HPP"

printPP (HPP, outFile2)

outFile2 \leftarrow "Below is VPP"

printPP (VPP, outFile2)

Step 3: binaryThreshold (HPP, thrVal, binHPP)

binaryThreshold (VPP, thrVal, binVPP)

outFile2 \leftarrow "Below is binHPP"

printPP (binHPP, outFile2)

outFile2 \leftarrow "Below is binVPP"

printPP (binVPP, outFile2)

Step 4: listHead \leftarrow get a boxNode, as the dummy node for listHead to point to.

(boxNode*) zBox \leftarrow computeZoneBox (binHPP, binVPP)

listInsert (listHead, zBox) // insert zBox to the front of linked list, after dummy

outFile2 \leftarrow "Below is the linked list after insert input zone box"

printBox (listHead, outFile2)

Step 5: morphClosing (binHPP, structElem, morphHPP)

morphClosing (binVPP, structElem, morphVPP)

outFile2 \leftarrow "Below is morphHPP after performing morphClosing on HPP"

outFile2 \leftarrow printPP (morphHPP)

outFile2 \leftarrow "Below is morphVPP after performing morphClosing on VPP"

printPP (morphVPP)

Step 6: runsHPP \leftarrow computePPruns (morphHPP, numRows)

runsVPP \leftarrow computePPruns (morphVPP, numCols)

outFile2 \leftarrow The number of runs in morphHPP-runsHPP is " // fill in value.

outFile2 \leftarrow The number of runs in morphVPP – runsVPP is " // fill in value.

Step 7: readingDirection \leftarrow computeDirection (runsHPP, runsVPP)

outFile2 \leftarrow "readingDirection is" // fill in value.

Step 8: if readingDirection == 1

computeHorizontalTextBox (zoneBox, morphHPP, numRows)

else if readingDirection == 2

computeVerticalTextBox (zoneBox, morphVPP, numCols)

Step 9: overlayBox (listHead, imgAry)

Step 10: outFile1 \leftarrow "Below is the input image overlay with bounding boxes"

imgReformat (imgAry)

Step 11: outFile1 \leftarrow "Output the boxNode in the list"

printBox (listHead, outFile1)

Step 12: close all files.

Source code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <cmath>
using namespace std;

class box
{
public:
    int minRow, minCol, maxRow, maxCol;
    box() : minRow(0), minCol(0), maxRow(0), maxCol(0) {}
    box(int a, int b, int c, int d) : minRow(a), minCol(b), maxRow(c), maxCol(d) {}
};

class boxNode
{
public:
    int boxType; // 1 for zone, 2 for texLine

    box *boundBox;
    boxNode *next;

    boxNode()
    {
        boxType = 99;
        next = nullptr;
    }

    boxNode(int t, box *b)
    {
        boxType = t;
        boundBox = b;
        next = nullptr;
    }
};

class boxQueue
```

```

{

public:

    boxNode *front;
    boxNode *back;

    boxQueue()
    {
        front = new boxNode();
        back = new boxNode();
        back->next = front;
    }

    void insert(boxNode *q)
    {
        q->next = back->next;
        back->next = q;
    }

    boxNode *pop()
    {
        boxNode *temp = back;
        boxNode *hold;
        // check if empty
        if (isEmpty())
            return nullptr;

        while (temp->next->next != front)
            temp = temp->next;

        if (temp->next->next == front)
        {
            hold = temp->next;
            temp->next = front;
            return temp;
        }

        return nullptr;
    }

    bool isEmpty()
    {

```

```

        return back->next == front;
    }
};

class docImage
{
public:
    int numRows, numCols, minVal, maxVal;
    int numStructRows, numStructCols, structMin, structMax, rowOrigin, colOrigin;
    int **imgAry;
    boxQueue *queue;
    boxNode *listHead;
    box *zoneBox;

    int *HPP;
    int *VPP;
    int *HPPbin;
    int *VPPbin;
    int *HPPmorph;
    int *VPPmorph;
    int HPPruns, VPPruns, thrVal;

    int readingDir;

    ifstream inFile;
    ofstream outFile1, outFile2;

    docImage(char *argv[], int tv, int r, int c, int mV, int mxV, int **arr)
    {
        numRows = r;
        numCols = c;
        minVal = mV;
        maxVal = mxV;
        thrVal = tv;

        cout << numRows << " " << numCols << endl;
        imgAry = arr;

        for (int i = 0; i < numRows + 2; i++)
        {
            for (int j = 0; j < numCols + 2; j++)

```

```

        {
            imgAry[i][j] = arr[i][j];
        }
    }

    ofstream outFile1(argv[3]);
    ofstream outFile2(argv[4]);

    HPP = new int[numRows + 2];
    HPPbin = new int[numRows + 2];
    HPPmorph = new int[numRows + 2];
    VPP = new int[numCols + 2];
    VPPbin = new int[numCols + 2];
    VPPmorph = new int[numCols + 2];

    for (int i = 0; i < numRows + 2; i++)
    {
        HPP[i] = 0;
        HPPbin[i] = 0;
        HPPmorph[i] = 0;
        VPP[i] = 0;
        VPPbin[i] = 0;
        VPPmorph[i] = 0;
    }

    // step 2
    computeHPP();
    computeVPP();
    outFile2 << "\nBelow is HPP : \n";
    printPP(HPP, numRows, outFile2);
    outFile2 << "\nBelow is VPP : \n";
    printPP(VPP, numCols, outFile2);

    // step 3
    threshold(tv);
    outFile2 << "\n\nBelow is binHPP : \n";
    printPP(HPPbin, numRows, outFile2);
    outFile2 << "\nBelow is binVPP : \n";
    printPP(VPPbin, numCols, outFile2);

    // step 4
    computeZoneBox();

```

```

        outFile2 << "\n\nBelow is the linked list after insert input zone box : \n"
                << zoneBox->minRow << " " << zoneBox->minCol << " " << zoneBox->maxRow
<< " " << zoneBox->maxCol;

// step 5
morphClosing();
outFile2 << "\n\nBelow is morphHPP after performing morphClosing on HPP : \n";
printPP(HPPmorph, numRows, outFile2);
outFile2 << "\n\nBelow is morphVPP after performing morphClosing on VPP : \n";
printPP(VPPmorph, numCols, outFile2);

// step 6
queue = new boxQueue();
queue->insert(new boxNode(1, zoneBox));
printBox(outFile2);

HPPruns = computePPRuns(HPPbin, numRows);
VPPruns = computePPRuns(VPPbin, numCols);
outFile2 << "\n\nThe number of runs in morphHPP-runsHPP is: " << HPPruns <<
"\n";
outFile2 << "\n\nThe number of runs in morphVPP-runsVPP is: " << VPPruns <<
"\n";

readingDir = computeDirection();
outFile1 << "\n\nReading Direction : ";

if (readingDir == 1)
{
    outFile1 << "Horizontal\n";
    computeHorizontalTextBox();
}
else if (readingDir == 2)
{
    outFile1 << "Vertical\n";
    computeVerticalTextBox();
}
else
{
    outFile1 << "The zone may be a non-text zone!\n";
}

printBox(outFile2);

```

```

        overlayImgAry();
        reformatPrettyPrint(imgAry, numRows, numCols, outFile1);

        // close resources
        inFile.close();
        outFile1.close();
        outFile2.close();
    }

void overlayImgAry()
{
    boxNode *thisBox = queue->pop();
    int label = 1;
    int minR, minC, maxR, maxC;

    while (thisBox != 0 && thisBox != queue->back)
    {
        minR = thisBox->boundBox->minRow;
        maxR = thisBox->boundBox->maxRow;
        minC = thisBox->boundBox->minCol;
        maxC = thisBox->boundBox->maxCol;
        for (int i = minR; i <= maxR; i++)
        {
            for (int j = minC; j <= maxC; j++)
            {
                imgAry[i][j] = label;
            }
        }

        thisBox = queue->pop();
    }
}

void computeHorizontalTextBox()
{
    int minR = zoneBox->minRow;
    int maxR = minR;
    int minC = zoneBox->minCol;
    int maxC = zoneBox->maxCol;

    while (maxR <= numRows)

```

```

{
    while (HPPmorph[maxR] == 0 && maxR <= numRows)
        maxR++;

    minR = maxR;
    while (HPPmorph[maxR] > 0 && maxR <= numRows)
        maxR++;

    queue->insert(new boxNode(2, new box(minR, minC, maxR, maxC)));

    minR = maxR;
    while (minR == 0 && minR <= numRows)
        minR++;
}
}

void computeVerticalTextBox()
{
    int minR = zoneBox->minRow;
    int maxR = zoneBox->maxRow;
    int minC = zoneBox->minCol;
    int maxC = minC;
    while (maxC <= numCols)
    {
        while (VPPmorph[maxC] == 0 && maxC <= numCols)
            maxC++;

        minC = maxC;
        while (VPPmorph[maxC] > 0 && maxC <= numCols)
            maxC++;

        queue->insert(new boxNode(2, new box(minR, minC, maxR, maxC)));

        minC = maxC;
        while (minC == 0 && minC <= numCols)
            minC++;
    }
}

int computeDirection()
{
    int factor = 2;

```



```

    if (HPPruns <= 2 && VPPruns <= 2)
        return 0;

    else if (HPPruns >= factor * VPPruns)
        return 1;

    else if (VPPruns >= factor * HPPruns)
        return 2;

    else
        return 0;
}

void computeZoneBox()
{

    int minR = 1;
    int minC = 1;
    int maxR = numRows;
    int maxC = numCols;

    // step 1
    while (HPPbin[minR] == 0 && minR <= numRows)
        minR++;

    // step 3
    while (HPPbin[maxR] == 0 && maxR >= 1)
        maxR--;

    // step 6
    while (VPPbin[minC] == 0 && minC <= numCols)
        minC++;

    // step 8
    while (VPPbin[maxC] == 0 && maxC >= 1)
        maxC--;

    zoneBox = new box(minR, minC, maxR, maxC);
}

int computePPRuns(int *pp, int l)

```

```

{
    int numRuns = 0;
    int i = 0;
    while (i <= 1)
    {

        while (pp[i] == 0 && i <= 1)
            i++;
        if (pp[i] > 0)
        {
            numRuns++;
            while (pp[i] > 0 && i <= 1)
                i++;
        }
    }
    return numRuns;
}

void morphClosing()
{
    // hard coded struct element is 111
    for (int i = 1; i <= numRows; i++)
        if (HPPbin[i - 1] == 1 && HPPbin[i] == 1 && HPPbin[i + 1] == 1)
            HPPmorph[i] = 1;

    for (int i = 1; i <= numCols; i++)
        if (VPPbin[i - 1] == 1 && VPPbin[i] == 1 && VPPbin[i + 1] == 1)
            VPPmorph[i] = 1;
}

void computeHPP()
{
    for (int row = 1; row <= numRows; row++)
    {
        int numThisRow = 0;
        for (int col = 1; col <= numCols; col++)
            if (imgAry[row][col] > 0)
                numThisRow++;
        HPP[row] = numThisRow;
    }
}

```

```

void computeVPP()
{
    for (int col = 1; col <= numCols; col++)
    {
        int numThisRow = 0;
        for (int row = 1; row <= numRows; row++)
            if (imgAry[row][col] > 0)
                numThisRow++;
        VPP[col] = numThisRow;
    }
}

void threshold(int val)
{
    // thresholding HPP
    for (int i = 0; i < numRows + 2; i++)
    {
        if (HPP[i] >= val)
            HPPbin[i] = 1;
        else
            HPPbin[i] = 0;
    }

    // thresholding VPP
    for (int j = 0; j < numCols + 2; j++)
    {
        if (VPP[j] >= val)
            VPPbin[j] = 1;
        else
            VPPbin[j] = 0;
    }
}

void printBox(ofstream &outFile)
{
    outFile << "\n\nPrinting Box Queue:\n\n";

    boxNode *temp = queue->back->next;

```

```

while (temp != queue->front)
{
    outFile << temp->boxType << endl;
    if (temp->boxType == 2)
    {
        outFile << temp->boundBox->minRow << " " << temp->boundBox->minCol << "
" << temp->boundBox->maxRow << " " << temp->boundBox->maxCol << endl;
    }
    temp = temp->next;
}

}

void printPP(int *ary, int l, ofstream &outFile)
{
    for (int i = 1; i <= l; i++)
        outFile << ary[i] << " ";

    outFile << "\n";
}

void reformatPrettyPrint(int **ary, int r, int c, ofstream &outFile)
{
    for (int i = 1; i <= r; i++)
    {
        for (int j = 1; j <= c; j++)
        {
            if (ary[i][j] > 0)
            {
                if (ary[i][j] < 10)
                {
                    outFile << ary[i][j] << "  ";
                }
                else if (ary[i][j] < 100)
                {
                    outFile << ary[i][j] << " ";
                }
                else
                {
                    outFile << ary[i][j];
                }
            }
        }
    }
}

```

```

        }

        }

        else

            outFile << ".  ";

    }

    outFile << "\n";

}

}

void loadImage(ifstream &inFile, int **imgAry)
{
    for (int i = 0; i < numRows + 2; i++)
    {
        for (int j = 0; j < numCols + 2; j++)
        {
            imgAry[i][j] = 0;
        }
    }

    for (int i = 1; i <= numRows; i++)
    {
        for (int j = 1; j <= numCols; j++)
        {
            int val;
            inFile >> val;
            imgAry[i][j] = val;
        }
    }
}

};

int main(int argc, char *argv[])
{

    int threshVal = stoi(argv[2]);

    ifstream inFile(argv[1]);

    int r, c, mV, mxV;
    inFile >> r >> c >> mV >> mxV;
    int **arr = new int *[r + 2];
    for (int i = 0; i < r + 2; i++)

```

```
{  
    arr[i] = new int[c + 2];  
}  
  
for (int i = 1; i <= r; i++)  
{  
    for (int j = 1; j <= c; j++)  
    {  
        int val;  
        inFile >> val;  
        arr[i][j] = val;  
    }  
}  
  
docImage docImage(argv, threshVal, r, c, mV, mxV, arr);  
}
```

[illegible]

output2.txt

[illegible]

output1.txt

output2.txt

Input: zone3.txt

output1.txt

Reading Direction : The zone may be a non-text zone!

output2.txt

Below is HPP :

0 2 4 10 13 15 15 20 24 20 19 18 32 30 28 26 26 21 12 7 7 7

Below is VPP :

0 0 0 0 9 12 13 17 19 19 8 6 6 7 8 10 14 14 13 8 6 2 3 8 11 9 11 11 14 17 21 16 13 11 8 6 4 2 0 0

Below is binHPP :

```
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Below is binVPP :

0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

Below is the linked list after insert input zone box :

3 5 22 37

Below is morphHPP after performing morphClosing on HPP :

0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

VBelow is morphVPP after performing morphClosing on VPP :

0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

Printing Box Queue:

1

The number of runs in morphHPP-runsHPP is: 1

The number of runs in `morphVPP-runsVPP` is: 2

Printing Box Queue:

1