CSCI 381 - Computer Vision Project 9 (C++) : Text Line Detection C++
Student: Matin Nazamy
Due Date: 12/10/2021

Cover Page

****
IV. main (...)
****
Step 0: open all files from argv[]
thrVal argv[2]
numRows, numCols, minVal, maxVal inFile
numStructRows, numStructCols, StructMin, StructMax, rowOrigin, colOrigin structElemFile or hard coded
use constructor to establish, allocate, and initialize all members of docImage class
Step 1: loadImage (inFile)
outFile1 reformatPrettyPrint (imgAry)
Step 2: computePP (imgAry)
outFile2 printPP (HPP) // to outFile2 with proper captions
outFile2 printPP (VPP) // to outFile2 with proper captions
Step 3: threshold (HPP, thrVal, binHPP)
threshold (HPP, thrVal, binVPP)
outFile2 printPP (binHPP) // to outFile2 with proper captions
outFile2 printPP (binVPP) // to outFile2 with proper captions
Step 4: zoneBox computeZoneBox (binHPP, binVPP)
listInsert (zoneBox) // insert zoneBox to the back of linked list of listHead.
outFile2 printBoxQueue (...) // to outFile2 with proper captions
Step 5: morphClosing (binHPP, structElem, morphHPP)
morphClosing (binVPP, structElem, morphVPP)
outFile2 printPP (morphHPP) // to outFile2 with proper captions

outFile2 printPP (morphVPP) // to outFile2 with proper captions
Step 6: runsHPP computePPruns (morphHPP, numRows)
runsVPP computePPruns (morphVPP, numCols)
outFile2 printPP (morphHPP) // to outFile2 with proper captions // morphHPP
outFile2 printPP (morphVPP) // to outFile2 with proper captions
Step 7: readingDirection computeDirection (runsHPP, runsVPP)
Step 8: if readingDirection == 1

computeTBoxHorizontal (zoneBox, morphHPP, numRows)

else if readingDirection == 2

computeTBoxVertical (zoneBox, morphVPP, numCols)

Step 9: overlayBox (listHead, imgAry)
Step 10: reformatPrettyPrint (imgAry)
Step 11: outFile1 printBoxQueue (...) // to outFile1 with proper captions

Table Of Contents

Source Code:

```cpp
#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 page, 2 for column, 3 for zone, 4 for texLine, 5 textWord, etc
                  // in this project we use 3 and 4
    box* boundBox;   // bounding box
    boxNode* next;  // points to boxNode in the same level

    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;

            // go to the 2nd to the front node
        while(temp->next->next != front) temp=temp->next;

            // set 2nd front to point to front
            // remove any references to the old front
        if(temp->next->next == front){
            hold = temp->next;
            temp->next = front;
            return temp;
        }
        return nullptr;
    }
    bool isEmpty(){
        return back->next == front;
    }

};
```

```
class docImage{

    public:

        // Class Variables

        int numRows, numCols, minVal, maxVal;
        int** imgAry;
        boxQueue* queue;
        boxNode* listHead;
        box* zoneBox;


        int* HPP;   // a 1D array to store the horizontal/vertical projection profile
        int* VPP;
        int* HPPbin;    // a 1D array of binarized HPP/VPP.
        int* VPPbin;
        int* HPPmorph;
        int* VPPmorph;
        int HPPruns, VPPruns, thrVal;

        int readingDir;

        ifstream inFile;
        ofstream outFile1, outFile2;

        // Constructor
        docImage(string f, int tv){

                // Read input data, load image
            inFile.open(f);
            thrVal = tv;
            loadImage();

            outFile1.open("outFile1.txt");
            outFile2.open("outFile2.txt");

                // Dynamically allocate a 1D array for the project profiles
            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++){    // initialize to zero
                HPP[i] = 0; HPPbin[i] = 0; HPPmorph[i] = 0;
                VPP[i] = 0; VPPbin[i]=0; VPPmorph[i] = 0;
            }

                //  Compute the HPP and the VPP from the input image.
            computeHPP(); computeVPP();
            outFile2 << "\nHPP  : \n";  printPP(HPP, numRows, &outFile2);
            outFile2 << "\nVPP  : \n"; printPP(VPP, numCols, &outFile2);

                // Threshold and output binary PP
            threshold(tv);
            outFile2 << "\n\nHPP Binary Threshold : \n";  printPP(HPPbin, numRows,
&outFile2);
            outFile2 << "\nVPP Binary Threshold : \n"; printPP(VPPbin, numCols, &outFile2);

                //  Compute the zone bounding box based on HPPBinary and VPPBinary.
            computeZoneBox();
            outFile2 << "\n\nZone Bounding Box : \n" << zoneBox->minRow << " " <<
zoneBox->minCol << " " << zoneBox->maxRow << " " << zoneBox->maxCol;

                //  Apply 1D morphological closing
            morphClosing();
            outFile2 << "\n\nHPP Morph : \n";  printPP(HPPmorph, numRows, &outFile2);
            outFile2 << "\nVPP Morph : \n"; printPP(VPPmorph, numCols, &outFile2);



                // Start building the box queue
            queue = new boxQueue();
            queue->insert(new boxNode(3,zoneBox));
            printBoxQueue(&outFile2);

                // Compute the number of runs
            HPPruns = computePPRuns(HPPbin, numRows);
            VPPruns = computePPRuns(VPPbin, numCols);
            outFile2 << "\n\nHPP Runs: " << HPPruns << "        VPP Runs: " << VPPruns;

                //  Using HPPMorph and VPPMorph to determine the reading direction of the
text-zone.
            readingDir = determineReadingDirection();
```

```
        outFile1 << "\n\nReading Direction : ";

            // Determine the reading direction
        if(readingDir == 1) {outFile1 << "Horizontal\n"; computeTBoxHorizontal();}
        else if(readingDir == 2){ outFile1 << "Vertical\n"; computeTBoxVertical();}
        else {outFile1 << "The zone may be a non-text zone!\n"; }

            // Overlay the zone box and text-line bounding boxes onto the image array.
        printBoxQueue(&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 computeTBoxHorizontal(){

            int minR = zoneBox->minRow; int maxR = minR;
            int minC = zoneBox->minCol; int maxC = zoneBox->maxCol;

            while(maxR <= numRows){
                    // find the start row of this text box
                while (HPPmorph[maxR] == 0 && maxR<= numRows) maxR++;

                    // find the end row of this text box
                minR = maxR;
                while(HPPmorph[maxR] > 0 && maxR <= numRows) maxR++;

                    // insert this node in the queue
                queue->insert( new boxNode( 4 , new box(minR, minC, maxR, maxC) ) );

                    // skip zeroes in between to next text box
                minR = maxR;
                while(minR == 0 && minR <= numRows) minR++;
            }
        }

        void computeTBoxVertical(){

            int minR = zoneBox->minRow; int maxR = zoneBox->maxRow;
            int minC = zoneBox->minCol; int maxC = minC;
            while(maxC <= numCols){
                    // find the start col of this text box
                while (VPPmorph[maxC] == 0 && maxC<= numCols) maxC++;
                    // find the end col of this text box
                minC = maxC;
                while(VPPmorph[maxC] > 0 && maxC <= numCols) maxC++;

                    // insert this node in the queue
                queue->insert( new boxNode( 4 , new box(minR, minC, maxR, maxC) ) );

                    // skip zeroes in between to next text box
                minC = maxC;
                while(minC == 0 && minC <= numCols) minC++;
            }
        }
```

```
int determineReadingDirection(){

    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(){
    //Computes the zone bounding box based on HPPBinary and VPPBinary.
    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){
            // computes the number of run in morphPP, labelling each run, in sequence: 1, 2,
3, ...
            // overwriting morphPP and returns the number of runs.
            int numRuns = 0;
            int i =0;
            while(i<=l){

                while(pp[i] == 0 && i<=l) i++;  // skip zeroes

                if(pp[i] > 0){  // once we found a run, skip through this run
                    numRuns++;
                    while( pp[i] > 0 && i<=l) i++;
                }
            }
            return numRuns;
        }

        void morphClosing(){

                    // hpp morph
                for(int i =1; i<=numRows; i++) if( HPPbin[i-1] == 1 && HPPbin[i] == 1 &&
HPPbin[i+1] == 1) HPPmorph[i] = 1;

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

        }
```

```
        void computeHPP(){
            // compute the horizontal projection profile of object pixels within imgBox.
            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(){
            // compute the vertical projection profile of object pixels within imgBox.
            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;
            }
        }
```

```
        // I/O Methods

    void printBoxQueue(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 == 4){
                *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 ){
        // reuse code from your previous project

        for(int i =1; i<=l; i++) *outFile << ary[i] << " ";

        *outFile << "\n";
    }
```

```
void reformatPrettyPrint(int** ary, int r, int c, ofstream* outFile ){

    // reuse code from your previous project
    for(int i =1; i<=r; i++){
        for(int j=1; j<=c; j++){

            if(ary[i][j] > 0){
                if (ary[i][j] < 10){    // 2 padded spaces
                    *outFile << ary[i][j] << "   ";
                }
                else if(ary[i][j] < 100){   // 1 padded space
                    *outFile << ary[i][j] << "  ";
                }
                else{   // no spaces
                    *outFile << ary[i][j];
                }
            }
            else *outFile << ".   ";

        }
        *outFile << "\n";
    }
}

void loadImage(){

        // Read in img header
    inFile >> numRows >> numCols >> minVal >> maxVal;

        // build our 2D array
    imgAry = new int*[numRows+2];
    for(int i =0; i<numRows+2; i++){
        imgAry[i] = new int[numCols+2];
    }

    for(int i=0; i<numRows+2; i++){
        for(int j =0; j<numCols+2; j++){
            imgAry[i][j] = 0;
        }
    }
```

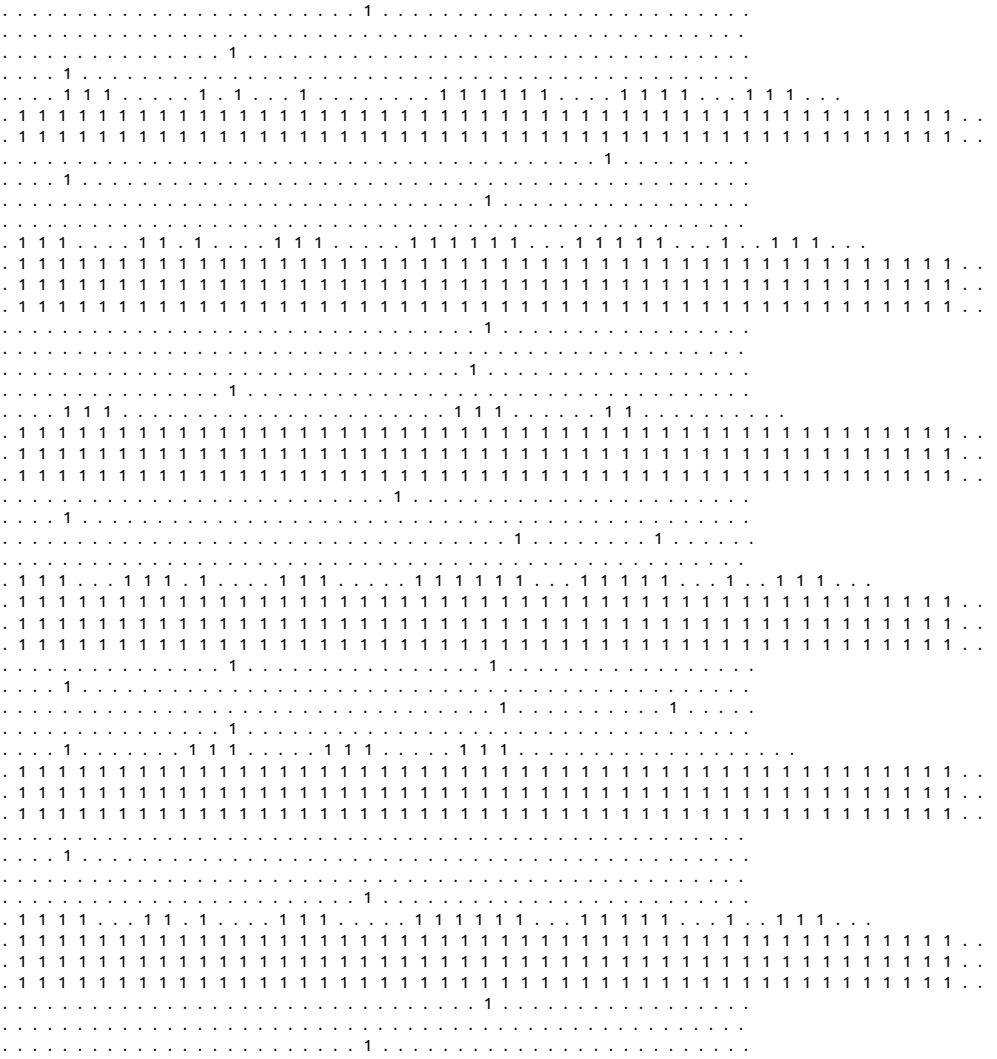
```
                // Fill in our img Ary
            for(int i =1; i<=numRows; i++){
                for(int j =1; j<= numCols; j++){
                    inFile >> imgAry[i][j];
                }
            }

        }

}; // end class




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

    if (argc != 3){ // If not the correct amount of arguments
        cout << "Error: \n Expected 2 arguments" << endl << "Received " << argc-1 << "
arguments" << endl;
        return -1;
    }

    int threshVal = stoi(argsv[2]);

    docImage* d = new docImage(argsv[1], threshVal);

}
```

Output Files

zone1:

outFile1

Reading Direction : Horizontal



outFile2

HPP :
1 0 1 1 19 24 25 1 1 1 0 24 27 26 4 1 0 1 1 8 23 24 27 1 1 2 0 25 28 26 8 2 1 2 1 10 22 26 26 0 1 0 1 25 27 26 18 1 0 1

VPP :
0 7 14 11 24 10 7 5 9 10 10 13 14 8 10 10 6 9 15 16 14 9 6 5 9 9 11 16 18 20 14 8 13 8 12 10 13 16 16 10 10 3 4 8 18 18 12 3 0 0

HPP Binary Threshold :
0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0

VPP Binary Threshold :
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

Zone Bounding Box :
5 2 47 48

HPP Morph :
0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 1 1 0 0 0 0

VPP Morph :
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

Printing Box Queue:

3

HPP Runs: 6 VPP Runs: 1

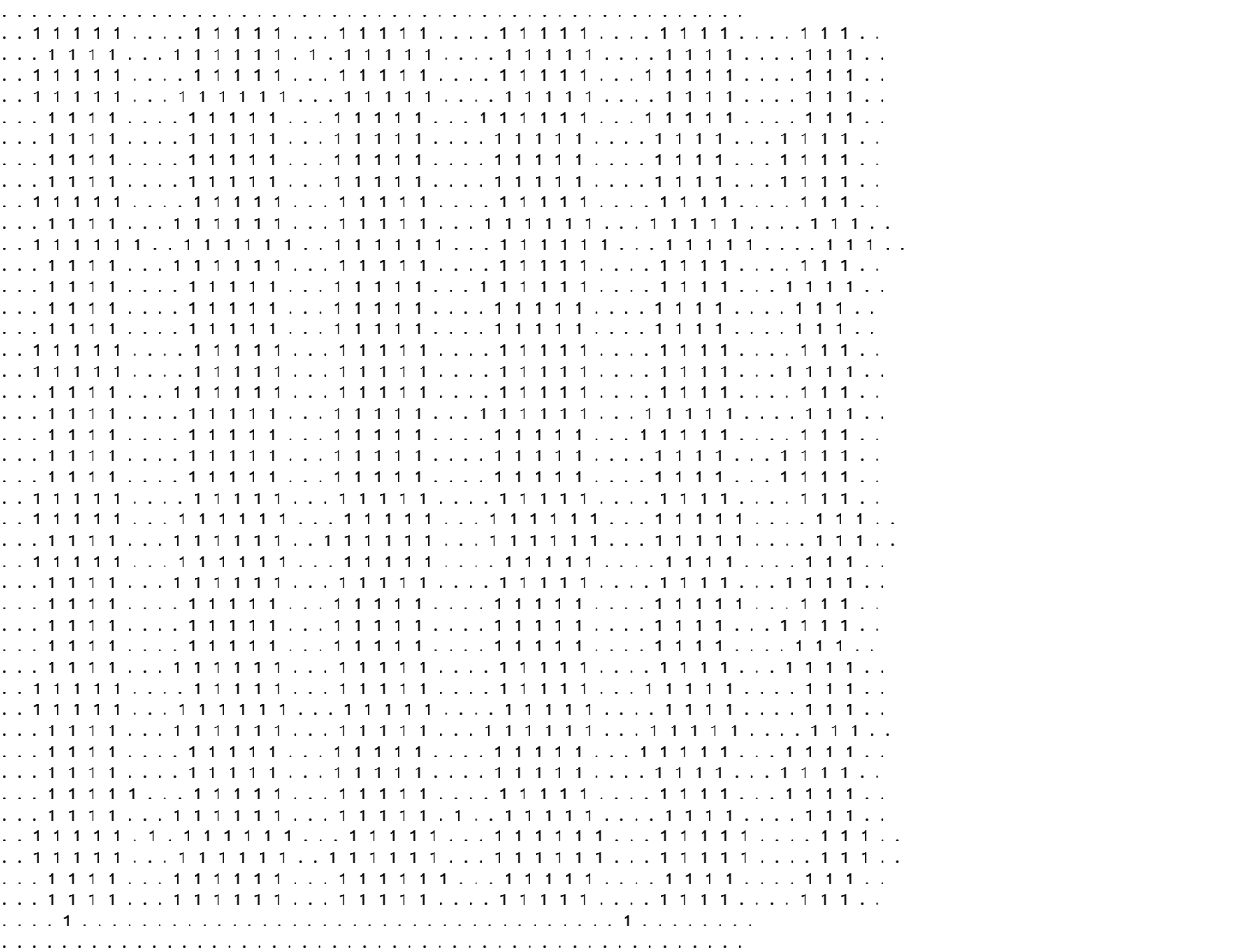
Printing Box Queue:

4
51 2 51 48
4
45 2 47 48
4
37 2 39 48
4
29 2 31 48
4
21 2 23 48
4
13 2 15 48
4
6 2 7 48
3

zone2:

outFile1

Reading Direction : Vertical



outFile2

HPP :
0 9 17 23 22 23 15 6 7 13 19 23 19 16 9 7 14 19 21 17 15 6 12 12 22 19 21 14 7 3 9 21 22 22 25 16 3 9 16 20 22 16 12 2 0

VPP :
0 0 14 28 30 29 18 2 1 0 18 26 28 26 17 4 0 1 3 24 28 28 20 7 1 1 0 10 21 26 23 11 13 0 0 0 13 22 24 29 16 2 0 0 13 25 25 18 0 0

HPP Binary Threshold :
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0

VPP Binary Threshold :
0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0

Zone Bounding Box :
2 3 43 48

HPP Morph :
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0

VPP Morph :
0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 0 0 0

Printing Box Queue:

3

HPP Runs: 1 VPP Runs: 6

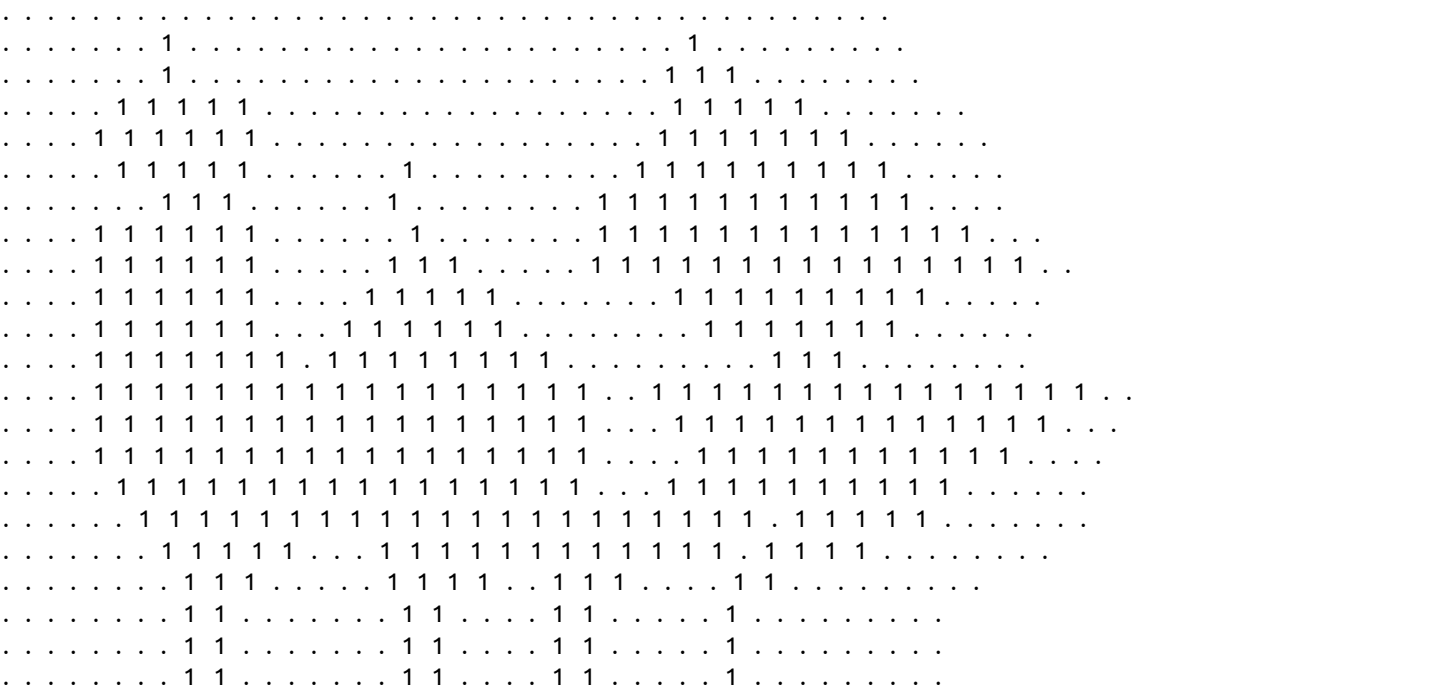
Printing Box Queue:

4
2 51 43 51
4
2 46 43 48
4
2 38 43 41
4
2 29 43 33
4
2 20 43 24
4
2 12 43 16
4
2 4 43 7
3

zone3:

outFile1

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



outFile2

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

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

HPP Binary Threshold :
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

VPP Binary Threshold :
0 0 0 0 1 1 1 1 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 1 1 1 1 0 0 0

Zone Bounding Box :
3 5 22 37

HPP Morph :
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

VPP Morph :
0 0 0 0 0 1 1 1 1 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 1 1 1 1 0 0 0 0

Printing Box Queue:

3

HPP Runs: 1 VPP Runs: 2

Printing Box Queue:

3