Student: Simon Kong

Project Due date: 12/8/2021

Algorithm Steps

1) Create DocImage and initialize all arrays to zero
2) Copy input data onto imgArr
3) Compute HPP and VPP
4) Output HPP and VPP to outputFile2
5) Threshold HPP and VPP using thrVal. Threshold version should be saved into HPPbin and VPPbin
6) Output HPPbin and VPPbin to outputFile2
7) Find zone box using HPPbin and VPPbin
8) Output zone box to outputFile2
9) Apply 1D morphological closing with mask of 1**1**1 to HPPbin and VPPbin. Save results onto HPPMorph and VPPMorph
10) Output HPPMorph and VPPMorph to outputFile2
11) Find document reading direction using HPPMorph and VPPMorph
12) Output reading direction to outputFile1
13) If Horizontal direction: find bounding box of text using HPPMorph

    If Vertical direction: find bounding box text using VPPMorph

    Insert bounding boxes onto queue.
14) Create Linked List of bounding boxes using queue
15) Output input data with bounding boxes applied to outputFile1
16) Output box type and box coordinates to outputFile2
17) Close all files

Source Code

Box, BoxNode, DocImage Class

```cpp
#include <iostream>
#include <fstream>
#include <string>
#include <queue>
using namespace std;

// I thought a text-line was a single row or column so, my code is very long
// You can get text-line boxes from zone box columns and HPPMorph but I didn't do it that way
// because of the misinterpretation.
// Use threshold 4 since 3 will combine columns
class Box{
    public:
        int minR, minC, maxR, maxC;
    public:
        Box(int minR, int minC, int maxR, int maxC){
            this -> minR = minR;
            this -> minC = minC;
            this -> maxR = maxR;
            this -> maxC = maxC;
        }
};

class BoxNode{
    public:
        int boxType;
        Box* bBox;
        BoxNode* next;
    public:
        BoxNode(int type, int minR, int minC, int maxR, int maxC){
            this -> boxType = type;
            this -> bBox = new Box(minR, minC, maxR, maxC);
        }
};

class DocImage{
    public:
        int numRows, numCols, minVal, maxVal, rowSize, colSize, thrVal, hppRuns, vppRuns;
        int *HPP, *VPP, *HPPbin, *VPPbin, *HPPMorph, *VPPMorph;
        int **imgArr, **boundArr;
        BoxNode* listHead;
        Box* zoneBox;
        queue<BoxNode*> boxQ;
```

```
public:
    DocImage( ifstream &input, string threshold ){
        read_header(input);
        this -> thrVal = stoi(threshold);
        this -> rowSize = this -> numRows + 2;
        this -> colSize = this -> numCols + 2;
        this -> imgArr = new int* [this -> rowSize];
        this -> boundArr = new int* [this-> rowSize];
        this -> HPP = new int [this -> rowSize];
        this -> VPP = new int [this -> colSize];
        this -> HPPbin = new int [this -> rowSize];
        this -> VPPbin = new int [this -> colSize];
        this -> HPPMorph = new int [this -> rowSize];
        this -> VPPMorph = new int [this -> colSize];

        for(int i = 0; i < this -> rowSize; i++){
            this -> HPP[i] = 0;
            this -> HPPbin[i] = 0;
            this -> HPPMorph[i] = 0;
            this -> boundArr[i] = new int[this -> colSize];
            this -> imgArr[i] = new int [this -> colSize];
        }

        for(int i = 0; i < colSize; i++){
            this -> VPP[i] = 0;
            this -> VPPbin[i] = 0;
            this -> VPPMorph[i] = 0;
        }

        setZero(this -> imgArr);
        setZero(this -> boundArr);
    }



    void read_header( ifstream &input ){
        input >> this -> numRows >> this -> numCols >> this -> minVal >> this -> maxVal ;
    }

    void setZero(int** arr){
        for(int i = 0; i < this -> rowSize; i++){
            for(int j = 0; j < this -> colSize; j++)
                arr[i][j] = 0;
        }
    }
```

```cpp
void testArr(int** arr){
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++)
            cout << arr[i][j] << " ";
        cout << endl;
    }
}

void loadImage(ifstream &input, int** arr){
    for(int i = 1; i < this -> rowSize - 1; i++){
        for(int j = 1; j < this -> colSize - 1; j++)
            input >> arr[i][j];
    }
}

void computeHPP(int** imgArr, int* hpp){
    int count = 0;
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++)
            if(imgArr[i][j] > 0) count++;
        hpp[i] = count;
        count = 0;
    }
}

void computeVPP(int**imgArr, int* vpp){
    int count = 0;
    for(int i = 0; i < this -> colSize; i++){
        for(int j = 0; j < this -> rowSize; j++){
            if(imgArr[j][i] > 0) count++;
        }
        vpp[i] = count;
        count = 0;
    }
}

void printPP(int* hpp, int* vpp, ofstream &output){
    // Print HPP
    for(int i = 0; i < this -> rowSize; i++)
        output << hpp[i] << " ";
    output << endl;

    // Print VPP
    for(int i = 0; i < this -> colSize; i++)
        output << vpp[i] << " ";
    output << endl;
}
```

```cpp
// i dont usually use ternary, just felt like it
void threshold(int thrVal){
    // Threshold HPP
    for(int i = 0; i < this -> rowSize; i++)
        this -> HPP[i] >= thrVal ? this -> HPPbin[i] = 1 : this -> HPPbin[i] = 0;

    // Threshold VPP
    for(int i = 0; i < this -> colSize; i++)
        this -> VPP[i] >= thrVal ? this -> VPPbin[i] = 1 : this ->VPPbin[i] = 0;
}

void setZone(int* hpp, int*vpp){
    int top = 9999;
    int bot = 0;
    int left = 9999;
    int right = 0;

    for(int i = 0; i < this -> rowSize; i++){
        if(hpp[i] > 0 && top > i) top = i;
        if(hpp[i] > 0 && bot < i) bot = i;
    }

    for(int i = 0; i < this -> colSize; i++){
        if(vpp[i] > 0 && left > i) left = i;
        if(vpp[i] > 0 && right < i) right = i;
    }

    this -> zoneBox = new Box(top,left,bot,right);
}

void printBoundingBox(int** arr, Box* bounds, ofstream &output){
    applyBound(arr, bounds);
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            output << this -> boundArr[i][j] << " ";
        }
        output << endl;
    }
}
```

```
void applyBound(int** arr, Box* bounds){
    // copy array
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            this -> boundArr[i][j] = arr[i][j];
        }
    }

    int minR = bounds -> minR;
    int minC = bounds -> minC;
    int maxR = bounds -> maxR;
    int maxC = bounds -> maxC;

    // apply horizontal bound
    for(int i = 0; i < this -> rowSize; i++){
        if(i == minR || i == maxR){
            for(int j = minC; j < maxC; j++)
                this -> boundArr[i][j] = 3;
        }
    }

    // apply vertical bound
    for(int i = 0; i < this -> colSize; i++){
        if(i == minC || i == maxC){
            for(int j = minR; j < maxR; j++){
                this -> boundArr[j][i] = 3;
            }
        }
    }

    this -> boundArr[maxR][maxC] = 9; //fix bottom right corner
}

void morphClosing(int* hpp, int* vpp, int* hppOut, int* vppOut){
    int tempHPP[this -> rowSize] = {0};
    int tempVPP[this -> colSize] = {0};
    oneDDilation(hpp, vpp, tempHPP, tempVPP);
    oneDErosion(tempHPP, tempVPP, hppOut, vppOut);
}
```

```
void oneDDilation(int* hpp, int* vpp, int* hppOut, int* vppOut){
    for(int i = 1; i < this -> rowSize - 1; i++){
        int a = i - 1;
        int b = i + 1;
        if(hpp[i] == 1){
            hppOut[a] = 1;
            hppOut[b] = 1;
            hppOut[i] = 1;
        }
    }

    for(int i = 1; i < this -> colSize - 1; i++){
        int a = i - 1;
        int b = i + 1;
        if(vpp[i] == 1){
            vppOut[a] = 1;
            vppOut[b] = 1;
            vppOut[i] = 1;

        }
    }
}

void oneDErosion(int* hpp, int*vpp, int* hppOut, int* vppOut){
    for(int i = 1; i < this -> rowSize - 1; i++){
        int a = i - 1;
        int b = i + 1;
        if(hpp[i] == 1)
            hpp[a] == 0 || hpp[b] == 0 ? hppOut[i] = 0 : hppOut[i] = 1;
    }

     for(int i = 1; i < this -> colSize - 1; i++){
        int a = i - 1;
        int b = i + 1;
        if(vpp[i] == 1)
            vpp[a] == 0 || vpp[b] == 0 ? vppOut[i] = 0 : vppOut[i] = 1;
    }

}
```

```
int findReadDirection(int* hpp, int* vpp, ofstream &output){
    this -> hppRuns = computeHPPRuns(hpp);
    this -> vppRuns = computeVPPRuns(vpp);
    int hppRuns = this -> hppRuns;
    int vppRuns = this -> vppRuns;

    int factor = 2;
    if(hppRuns <= 2 && vppRuns <= 2){
        output << "The zone may be a non-text zone";
        output.close();
        exit(2);
    }

    if(  hppRuns >= (factor * vppRuns) ){
        output << "Horizontal Reading Direction" << endl;
        return 1;
    }
    else if( vppRuns >= (factor * hppRuns) ){
        output << "Vertical Reading Direction" << endl;
        return 2;
    }
    else{
        output << "The zone may be a non-text zone";
        output.close();
        exit(2);
    }
}
```

```
int computeHPPRuns(int* hpp){
    int count = 0;
    for(int i = 0; i < this -> rowSize; i++){
        if(hpp[i] > 0){
            count++;
            for(int j = i; j < this -> rowSize - 1; j++){
                int next = hpp[j+1];
                if(next == 0){
                    i = j + 1;
                    break;
                }
            }
        }

    }
    return count;
}

int computeVPPRuns(int* vpp){
    int count = 0;
    for(int i = 0; i < this -> colSize; i++){
        if(vpp[i] > 0){
            count++;
            for(int j = i; j < this -> colSize - 1; j++){
                int next = vpp[j+1];
                if(next == 0){
                    i = j + 1;
                    break;
                }
            }
        }

    }
    return count;
}
```

```
void computeHorizontalBBox(int* ppArr, int** imgArr, int runs){
    int index = 0;
    int rowCoords[runs][2] = {0};
    for(int i = 1; i < this -> rowSize - 1; i++){
        if(ppArr[i] > 0 && ppArr[i-1] == 0)
            rowCoords[index][0] = i;
        else if(ppArr[i] > 0 && ppArr[i+1] == 0){
            // there is edge case where if last index is 1 it wont count row
            // but the array is padded so, last index should always be 0
            rowCoords[index][1] = i;
            index++;
        }
    }

    for(int i = 0; i < runs; i++)
        findHorizontalTextlineBoxes(imgArr, rowCoords[i], this -> boxQ);
}
```

```
void findHorizontalTextlineBoxes(int** imgArr, int coords[], queue<BoxNode*> &boxQ){
    // find column start and end for every line
    // find column start and end for zone
    int size = coords[1] - coords[0];
    int lineColCoords[size][2] = {0};
    int index = 0;
    int min = 9999; //right column per line
    int max = 0; //left column per line
    for(int i = coords[0]; i <= coords[1]; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(min > j && imgArr[i][j] > 0) min = j;
            if(max < j && imgArr[i][j] > 0) max = j;
        }
        if(min == 9999 && max == 0){
            lineColCoords[index][0] = -1;
            lineColCoords[index][1] = -1;
            index++;
            continue;
        }
        lineColCoords[index][0] = min;
        lineColCoords[index][1] = max;
        index++, min = 9999, max = 0;
    }

    for(int i = 0; i <= size; i++){
        // reuse min max to be left and right bounds for zone box
        if(lineColCoords[i][0] < min && lineColCoords[i][0] > 0) min = lineColCoords[i][0];
        if(lineColCoords[i][1] > max && lineColCoords[i][1] > 0) max = lineColCoords[i][1];
        // I can put textlines in queue here but i want to keep order of zone -> textlines
    }

    // changed from 3 to 4
    BoxNode* zone = new BoxNode(4, coords[0], min, coords[1], max);
    boxQ.push(zone);

    // originally for finding box per row or column
    // index = 0;
    // for(int i = coords[0]; i <= coords[0] + size; i++){
    //     BoxNode* line = new BoxNode(4, i, lineColCoords[index][0], i, lineColCoords[index][1]);
    //     index++;
    //     boxQ.push(line);
    // }
}
```

```
void computeVerticalBBox(int* ppArr, int** imgArr, int runs){
    int index = 0;
    int colCoords[runs][2] = {0};
    for(int i = 1; i < this -> colSize - 1; i++){
        if(ppArr[i] > 0 && ppArr[i-1] == 0)
            colCoords[index][0] = i;
        else if(ppArr[i] > 0 && ppArr[i+1] == 0){
            colCoords[index][1] = i;
            index++;
        }
    }

    for(int i = 0; i < runs; i++)
        findVerticalTextlineBoxes(imgArr, colCoords[i], this -> boxQ);
}
```

```
void findVerticalTextlineBoxes(int** imgArr, int coords[], queue<BoxNode*> &boxQ){
    int size = coords[1] - coords[0];
    int lineRowCoords[size][2] = {0};
    int index = 0;
    int min = 9999; //top row per line
    int max = 0; //bottom row per line
    for(int i = coords[0]; i <= coords[1]; i++){
        for(int j = 0; j < this -> rowSize; j++){
            if(min > j && imgArr[j][i] > 0) min = j;
            if(max < j && imgArr[j][i] > 0) max = j;
        }
        if(min == 9999 && max == 0){
            lineRowCoords[index][0] = -1;
            lineRowCoords[index][1] = -1;
            index++;
            continue;
        }
        lineRowCoords[index][0] = min;
        lineRowCoords[index][1] = max;
        index++, min = 9999, max = 0;
    }

    for(int i = 0; i <= size; i++){
        if(lineRowCoords[i][0] < min && lineRowCoords[i][0] > 0) min = lineRowCoords[i][0];
        if(lineRowCoords[i][1] > max && lineRowCoords[i][1] > 0) max = lineRowCoords[i][1];
    }

    BoxNode* zone = new BoxNode(4, min, coords[0], max, coords[1]);
    boxQ.push(zone);

    // originally for finding box per row or column
    // index = 0;
    // for(int i = coords[0]; i <= coords[0] + size; i++){
    //     BoxNode* line = new BoxNode(4, lineRowCoords[index][0], i, lineRowCoords[index][1], i);
    //     index++;
    //     boxQ.push(line);
    // }
}
```

```
void applyZoneBoxes(int** boundArr, BoxNode* &head){
    BoxNode* node = head;
    while(node -> boxType != -1){
        int type = node -> boxType;
        int minR = node -> bBox -> minR;
        int minC = node -> bBox -> minC;
        int maxR = node -> bBox -> maxR;
        int maxC = node -> bBox -> maxC;

        // apply vertical bound
        if(minR == -1 || minC == -1 || maxR == -1 || maxC == -1){
            node = node -> next;
            continue;
        }

        for(int i = minR; i <= maxR; i++){
            boundArr[i][minC] = type;
            boundArr[i][maxC] = type;
        }

        for(int i = minC; i <= maxC; i++){
            boundArr[minR][i] = type;
            boundArr[maxR][i] = type;
        }

        node = node -> next;
    }
}

void assembleBoxLL(queue<BoxNode*> q, BoxNode* &head, Box* &zone){
    head = new BoxNode(3, zone -> minR, zone -> minC, zone -> maxR, zone -> maxC);
    BoxNode* currentNode = head;
    int size = q.size();
    for(int i = 0; i < size; i++){
        BoxNode* nextNode = q.front();
        currentNode -> next = nextNode;
        currentNode = nextNode;
        q.pop();
    }

    BoxNode* tail = new BoxNode(-1, -1, -1, -1, -1);
    currentNode -> next = tail;
}
```

```
void printBoxQueue(BoxNode* &head, ofstream &output){
    BoxNode* node = head;

    while(node -> boxType != -1){
        int type = node -> boxType;
        int minR = node -> bBox -> minR;
        int minC = node -> bBox -> minC;
        int maxR = node -> bBox -> maxR;
        int maxC = node -> bBox -> maxC;
        output << "Box Type: " << type << endl;
        output << minR << " " << minC << " " << maxR << " " << maxC << endl;
        node = node -> next;
    }
}

void reformatPrettyPrint(int** arr, ofstream &output){
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(arr[i][j] > 0) output << arr[i][j] << " ";
            else output << ". ";
        }
        output << endl;
    }
}
```

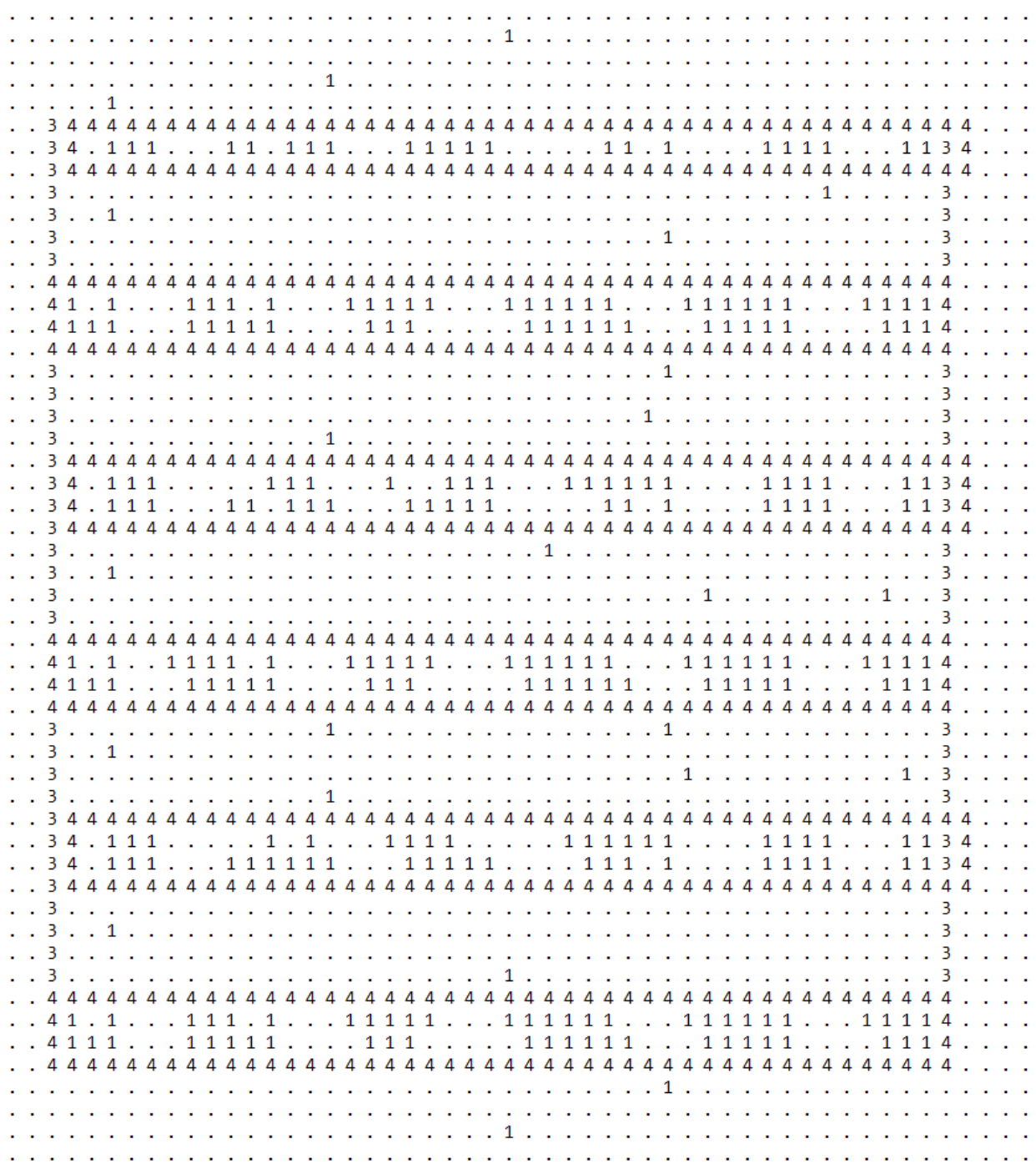
Main Class

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

    if(argc != 3){
        cout << "Invalid amount of arguments. Please enter input file name and threshold value.";
        exit(1);
    }

    string input = argv[1] ;
    string threshold = argv[2];
    string output1 = "outFile1.txt";
    string output2 = "outFile2.txt";
    ifstream inputStream;
    ofstream outputStream1, outputStream2;
    inputStream.open( input ) ;
    outputStream1.open( output1 );
    outputStream2.open( output2 );
```

```
if( inputStream.is_open() ){

    DocImage* img = new DocImage( inputStream, threshold );
    img -> loadImage(inputStream, img -> imgArr);
    img -> computeHPP(img -> imgArr, img -> HPP);
    img -> computeVPP(img -> imgArr, img -> VPP);
    outputStream2 << "HPP and VPP: how many pixels in every row/col" << endl;
    img -> printPP(img -> HPP, img -> VPP, outputStream2);
    img -> threshold(img -> thrVal);
    outputStream2 << "HPP and VPP: Binary" << endl;
    img -> printPP(img -> HPPbin, img -> VPPbin, outputStream2);
    img -> setZone(img -> HPPbin, img -> VPPbin);
    outputStream2 << "Zone Box Bounding Box" << endl;
    img -> printBoundingBox(img -> imgArr, img -> zoneBox, outputStream2);
    img -> morphClosing(img -> HPPbin, img -> VPPbin, img -> HPPMorph, img -> VPPMorph);
    outputStream2 << "HPP and VPP: Morph" << endl;
    img -> printPP(img -> HPPMorph, img -> VPPMorph, outputStream2);
    int direction = img -> findReadDirection(img -> HPPMorph, img -> VPPMorph, outputStream1);
    direction == 1 ? img -> computeHorizontalBBox(img -> HPPMorph, img -> imgArr, img -> hppRuns) : img -> computeVerticalBBox(img -> VPPMorph, img -> imgArr, img -> vppRuns);
    img -> assembleBoxLL(img -> boxQ, img -> listHead, img -> zoneBox);
    img -> applyZoneBoxes(img -> boundArr, img -> listHead);
    outputStream1 << "Pretty Print of text-line bounding boxes applied. \n4 = text-line box \n9 = document box \nYou will notice that the text-line bounding box has overwritten
    img -> reformatPrettyPrint(img -> boundArr, outputStream1);
    outputStream2 << "Bounding Box Coordinates and Box Type: \n3 = Zone Box \n4 = Text-line Box\n";
    img -> printBoxQueue(img -> listHead, outputStream2);


    inputStream.close() ;
    outputStream1.close();
    outputStream2.close();
    delete img;
}
else{
    cout << "Error: Input" << endl ;
}

return 0;
```

Zone 1 Output 1 Threshold 4

```
Horizontal Reading Direction
Pretty Print of text-line bounding boxes applied.
4 = text-line box
3 = zone box
You will notice that the text-line bounding box has overwritten parts of the zone box.
```



Zone 1 Output 2 Threshold 4

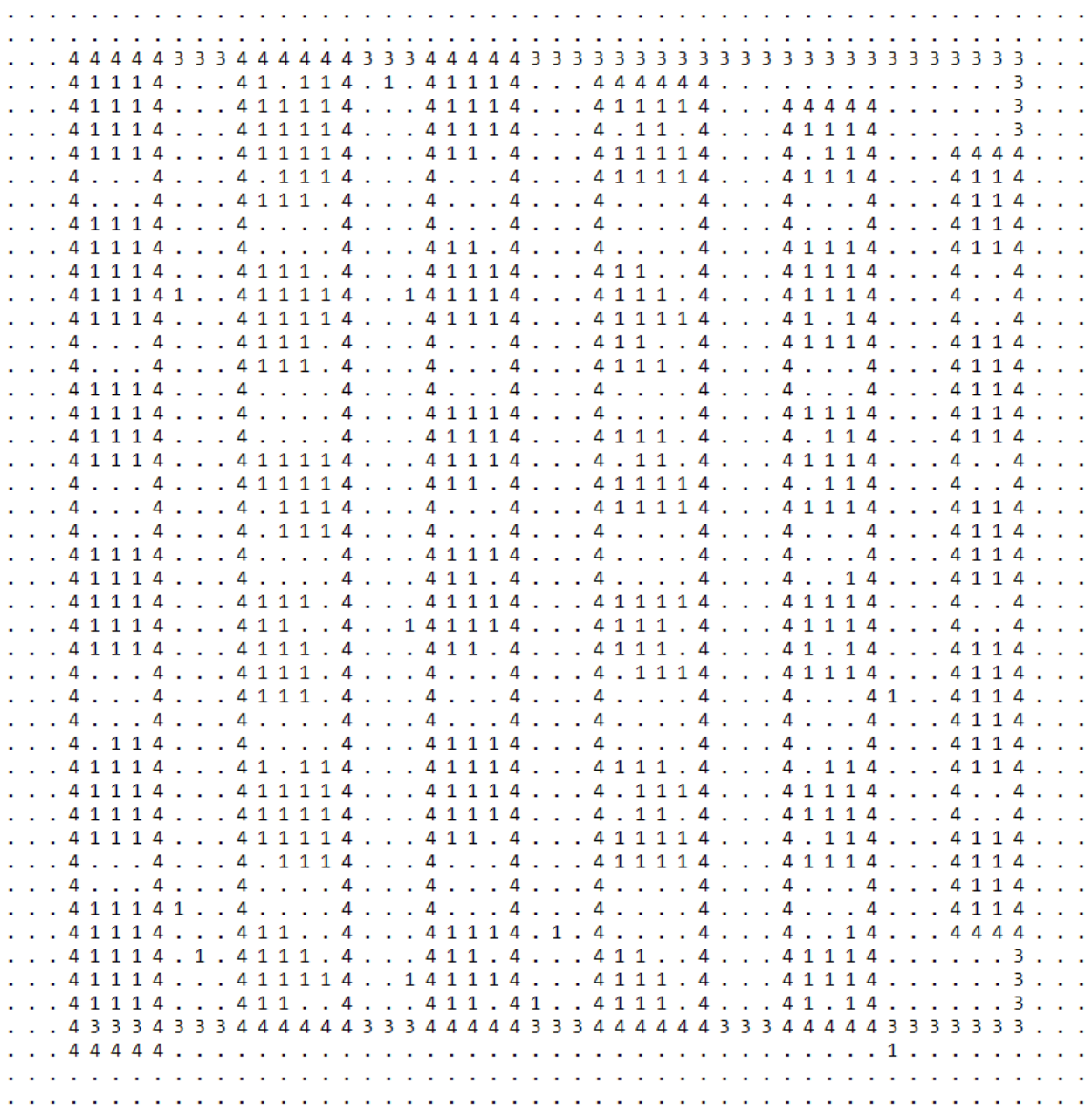
```
HPP and VPP: how many pixels in every row/col
0 1 0 1 1 19 24 25 1 1 1 0 24 27 26 4 1 0 1 1 8 23 24 27 1 1 2 0 25 28 26 8 2 1 2 1 10 22 26 26 0 1 0 1 25 27 26 18 1 0 1 0
0 0 7 14 11 24 10 7 5 9 10 10 13 14 8 10 10 6 9 15 16 14 9 6 5 9 9 11 16 18 20 14 8 13 8 12 10 13 16 16 10 10 3 4 8 18 18 12 3 0 0 0
HPP and VPP: Binary
0 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 0
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 1 1 1 1 1 0 0 0 0
```

```
Zone Box Bounding Box
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0
0 0 3 1 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 3 0 0 0 0
0 0 3 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 3 1 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 3 0 0 0 0
0 0 3 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 3 0 0 0 0
0 0 3 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 3 0 0 0 0
0 0 3 1 0 1 1 1 0 0 0 1 1 0 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 3 0 0 0 0
0 0 3 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 3 1 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 1 1 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 3 0 0 0 0
0 0 3 1 0 1 0 0 1 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 3 0 0 0 0
0 0 3 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 3 0 0 0 0
0 0 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 1 1 0 0 0 0 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 1 1 3 0 0 0 0
0 0 3 1 0 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 1 1 3 0 0 0 0
0 0 3 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 1 3 1 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0
0 0 3 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 3 0 0 0 0
0 0 3 1 0 1 0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 3 0 0 0 0
0 0 3 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 3 0 0 0 0
0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 9 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
HPP and VPP: Morph
0 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 0
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 0 0 0 0
Bounding Box Coordinates and Box Type:
3 = Zone Box
4 = Text-line Box
Box Type: 3
5 2 47 47
Box Type: 4
5 3 7 48
Box Type: 4
12 2 15 47
Box Type: 4
20 3 23 48
Box Type: 4
28 2 31 47
Box Type: 4
36 3 39 48
Box Type: 4
44 2 47 47
```

Zone 2 Output 1 Threshold 4

```
Vertical Reading Direction
Pretty Print of text-line bounding boxes applied.
4 = text-line box
3 = zone box
You will notice that the text-line bounding box has overwritten parts of the zone box.
```



Zone 2 Output 2 Threshold 4

```
HPP and VPP: how many pixels in every row/col
0 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 0
0 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 0
HPP and VPP: Binary
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 1 1 1 1 1 1 1 1 0 0 0
0 0 0 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 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0
```

```
Zone Box Bounding Box
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 0
0 0 0 3 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 1 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 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 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 1 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 0 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 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 0 0 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 3 0 0 0
0 0 0 3 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 1 0 0 0 1 1 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 3 0 0 0
0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1 3 0 0 0
0 0 0 3 1 1 1 1 0 1 0 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 1 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 0 0 3 0 0 0
0 0 0 3 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 3 0 0 0
0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 9 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
HPP and 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 0 0 0
0 0 0 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 1 1 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0
Bounding Box Coordinates and Box Type:
3 = Zone Box
4 = Text-line Box
Box Type: 3
2 3 43 48
Box Type: 4
2 3 44 7
Box Type: 4
2 11 43 16
Box Type: 4
2 20 43 24
Box Type: 4
3 28 43 33
Box Type: 4
4 37 43 41
Box Type: 4
6 45 39 48
```

Zone 3 Output 1 Threshold 4

```
The zone may be a non-text zone
```

Zone 3 Output 2 Threshold 4

```
HPP and VPP: how many pixels in every row/col
0 0 2 4 10 13 15 15 20 24 20 19 18 32 30 28 26 26 21 12 7 7 7 0
0 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 0
HPP and VPP: Binary
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
Zone Box Bounding Box
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 3 0 0 0 0
0 0 0 0 0 3 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 3 1 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 0 0 0
0 0 0 0 0 3 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 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 3 0 0 0 0
0 0 0 0 0 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 3 0 0 0 0
0 0 0 0 0 3 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 0 3 0 0 0 0
0 0 0 0 0 3 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 0 3 0 0 0 0
0 0 0 0 0 3 0 0 0 1 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 0 0 0 0 1 1 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 3 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 3 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 3 0 0 0 0
0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 9 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
HPP and VPP: Morph
0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 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 0 0 0 0
```