Student: Simon Kong

Project Due date: 11/27/2021

Algorithm Steps

****
IV. main (...)
****

Step 0: inFile ← open input file argv [1]
numRows, numCols, minVal, maxVal ← read from inFile
imgAry, reconstructAry ← dynamically allocated and **initialized to 0** in effect the boarder are zero framed.

Step 1: chainCodeFileName ← argv [1] + "_chainCode.txt"
boundaryFileName ← argv [1] + "_boundary.txt"
deCompressedFileName ← argv [1] + "_deCompressed.txt"

Step 2: chainCodeFile ← open (chainCodeFileName)
boundaryFile ← open (boundaryFileName)
deCompressedFile ← open (deCompressedFileName)

Step 3: loadImage (inFile, imgAry, outFile)
reformatPrettyPrint (imgAry, outFile) // prettyPrint imgAry to outFile.

Step 4: getChainCode (imgAry, chainCodeFile) // see algorithm below.
Step 5: close chainCodeFile
Step 6: reopen chainCodeFile

Step 7: constructBoundary (reconstructAry, chainCodeFile)
ary2file (reconstructAry, boundaryFile)

Step 8: whichMethod ← from argv[2]
Step 9: case of whichMethod
case 1: Method1 (reconstructAry)
case 2: Method2 (reconstructAry)
default: console ← print error message: " argv[2] only accept 1 or 2"
exit program
Step 10: ary2file (reconstructAry, deCompressedFile)
Step 11: close all files

Source Code

Main Class

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

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

    string imgFile = argv[1] ;
    string method = argv[2];

    // if( method != "1" && method != "2"){
    //     cout << "Invalid argument. Please enter 1 or 2.";
    //     exit(2);
    // }

    ifstream imgStream, methodStream;
    imgStream.open( imgFile ) ;


    if( imgStream.is_open() ){
        ChainCode* chainObj = new ChainCode( imgStream );
        string chainCodeFile = imgFile + "_chainCode.txt";
        string boundaryFile = imgFile + "_boundaryFile.txt";
        string decompressedFile = imgFile + "_decompressedFile.txt";
        string outFile = "outFile.txt";
        ofstream chainCodeStream, boundaryStream, decompressedStream, outStream;
        chainCodeStream.open(chainCodeFile);
        boundaryStream.open(boundaryFile);
        decompressedStream.open(decompressedFile);
        outStream.open(outFile);
        chainObj -> loadImage(imgStream, chainObj -> imgArr);
        outStream << "Zero Framed Original Image" << endl;
        chainObj -> reformatPrettyPrint(chainObj -> imgArr, outStream);
        chainObj -> getChainCode(chainObj -> imgArr, chainCodeStream);
        chainCodeStream.close();
        ifstream inChainCodeStream;
        inChainCodeStream.open(chainCodeFile);
        chainObj -> constructBoundary(chainObj -> reconstructArr, inChainCodeStream);
        chainObj -> arrToFile(chainObj -> reconstructArr, boundaryStream);
        if(method == "1") chainObj -> method1(chainObj -> reconstructArr);
        else if (method == "2") chainObj -> method2(chainObj -> reconstructArr);
        chainObj -> arrToFile(chainObj -> reconstructArr, decompressedStream);
```

```
        imgStream.close() ;
        boundaryStream.close();
        decompressedStream.close();
        outStream.close();
        inChainCodeStream.close();
        delete chainObj ;
    }
    else{
        cout << "Error: Input" << endl ;
    }

    return 0;
```

ChainCode

```
class ChainCode{
    public:
        int numRows,
            numCols,
            minVal,
            maxVal,
            rowSize,
            colSize,
            label,
            lastQ,
            chainDir;

        int **imgArr, **reconstructArr;

    public:
        ChainCode( ifstream &imgFile ){
            read_header(imgFile);
            this -> rowSize = this -> numRows + 2;
            this -> colSize = this -> numCols + 2;
            this -> imgArr = new int* [this -> rowSize];
            this -> reconstructArr = new int* [this -> rowSize];
            // create 2D array
            for(int i = 0; i < this -> rowSize; i++){
                this -> imgArr[i] = new int [this -> colSize];
                this -> reconstructArr[i] = new int [this -> colSize];
            }
            setZero(this -> imgArr);
            setZero(this -> reconstructArr);

        }

        void read_header( ifstream &inFile ){
            inFile >> 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;
                }
            }
        }
```

```
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 &inFile, int** arr){
    for(int i = 1; i < this -> numRows + 1; i++){
        for(int j = 1; j < this -> numCols + 1; j++){{
            inFile >> arr[i][j];
        }}
    }
}

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;
    }
    output << endl;
}
```

```
void getChainCode(int** arr, ofstream &output){
    Pointer* pointerObj;
    bool found = false;
    int zeroTable[8] = {6,0,0,2,2,4,4,6};
    output << this -> numRows << " " << this -> numCols << " " << this -> minVal << " " << this -> maxVal << endl;
    // search for starting pixel
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(arr[i][j] > 0){
                pointerObj = new Pointer(i, j);
                output << imgArr[i][j] << " " << i << " " << j << " ";
                this -> label = imgArr[i][j];
                this -> lastQ = 4;
                found = true;
                break;
            }
        }
        if(found) break;
    }
    int count = 0;
    while( true ){
        this -> lastQ = (this -> lastQ + 1) % 8;
        findNextP(pointerObj, this -> lastQ);
        output << this -> chainDir << " ";
        pointerObj -> currentRow = pointerObj -> neighborCoords[this -> chainDir][0];
        pointerObj -> currentCol = pointerObj -> neighborCoords[this -> chainDir][1];
        this -> lastQ = zeroTable[ (this -> chainDir - 1) % 8];
        if( pointerObj -> currentRow == pointerObj -> startRow && pointerObj -> currentCol == pointerObj -> startCol) break;
    }
}

void findNextP(Pointer* obj, int q){
    loadNeighborsCoord(obj);
    cout << "Current Position: " << obj -> currentRow << " " << obj -> currentCol << " " << endl;
    int index = q;
    bool found = false;
    while(!found){
        int i = obj -> neighborCoords[index][0];
        int j = obj -> neighborCoords[index][1];
        if(this -> imgArr[i][j] == this -> label){
            this -> chainDir = index;
            found = true;
        }
        index = (index + 1) % 8;
    }
}
```

```
void loadNeighborsCoord(Pointer* obj){
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 2; j++){
            if (j == 0) obj -> neighborCoords[i][j] = obj -> currentRow + obj -> coordOffset[i][0];
            else obj -> neighborCoords[i][j] = obj -> currentCol + obj -> coordOffset[i][1];
        }
    }
}

void constructBoundary(int** arr, ifstream &input){
    read_header(input);
    int  r, c, val, direction;
    input >> val >> r >> c;
    Pointer* obj = new Pointer(r, c);

    while ( true ){
        input >> direction;
        loadNeighborsCoord(obj);
        obj -> currentRow = obj -> neighborCoords[direction][0];
        obj -> currentCol = obj -> neighborCoords[direction][1];
        arr[obj -> currentRow][obj -> currentCol] = val;
        if(obj -> currentRow == obj -> startRow && obj -> currentCol == obj -> startCol) break;

    }
    delete obj;
}

void arrToFile(int** arr, ofstream &output){
     output << this -> numRows << " " << this -> numCols << " " << this -> minVal << " " << this -> maxVal << endl;
     for(int i = 1; i < this -> rowSize - 1; i++){
        for(int j = 1; j < this -> colSize - 1; j++){
            output << arr[i][j] << " ";
        }
        output << endl;
    }
}

void method1(int** arr){
    method1Pass1(arr);
    method1Pass2(arr);
    method1Pass3(arr);

}
```

```
void method1Pass1(int** arr){
    int k = 0;
    int l = 0;
     for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(arr[i][j] == this -> label){
                if(k == 0) k = j;
                else if(l == 0 ) l = j;
            }
            if(k != 0 && l != 0){
                for(int m = k ; m <= l; m++){
                    arr[i][m]++;
                }
                k = l;
                l = 0;
            }
        }
        k = 0;
        l = 0;
    }
}

void method1Pass2(int** arr){
    int k = 0;
    int l = 0;
    for(int i = 0; i < this -> colSize; i++){
        for(int j = 0; j < this -> rowSize; j++){
           if(arr[j][i] >= this -> label + 1){
                if(k == 0) k = j;
                else if(l == 0 ) l = j;
            }
            if(k != 0 && l != 0){
                for(int m = k; m <= l; m++){
                   arr[m][i]++;
                }
                k = l;
                l = 0;
            }
        }
        k = 0;
        l = 0;
    }
}
```

```
void method1Pass3(int** arr){
    for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(arr[i][j] >= 2) arr[i][j] = this -> label;
            else arr[i][j] = 0;



        }
    }
}

void method2(int** arr){
    // just check top to see if there is something
    // i have no idea on how to fill out the bottom half of the image
      for(int i = 0; i < this -> rowSize; i++){
        for(int j = 0; j < this -> colSize; j++){
            if(arr[i][j] == this -> label){
                while(arr[i][j] != 0 && j < this -> colSize) j++;
                if(arr[i-1][j] == this -> label){
                    while(arr[i][j] != this -> label && j < this -> colSize){
                        arr[i][j] = this -> label;
                        j++;
                    }
                }
            }
        }
    }
}
```

Pointer

```
class Pointer{
    public:
        int startRow, startCol, currentRow, currentCol, **coordOffset, **neighborCoords;
    public:
        Pointer(int row, int col){
            this -> startRow = row;
            this -> startCol = col;
            this -> currentRow = row;
            this -> currentCol = col;
            this -> coordOffset = new int*[8];
            this -> neighborCoords = new int*[8];
            for(int i = 0; i < 8; i++){
                this -> neighborCoords[i] = new int[2];
                this -> coordOffset[i] = new int[2];
            }
            initializeArr();
        }
```

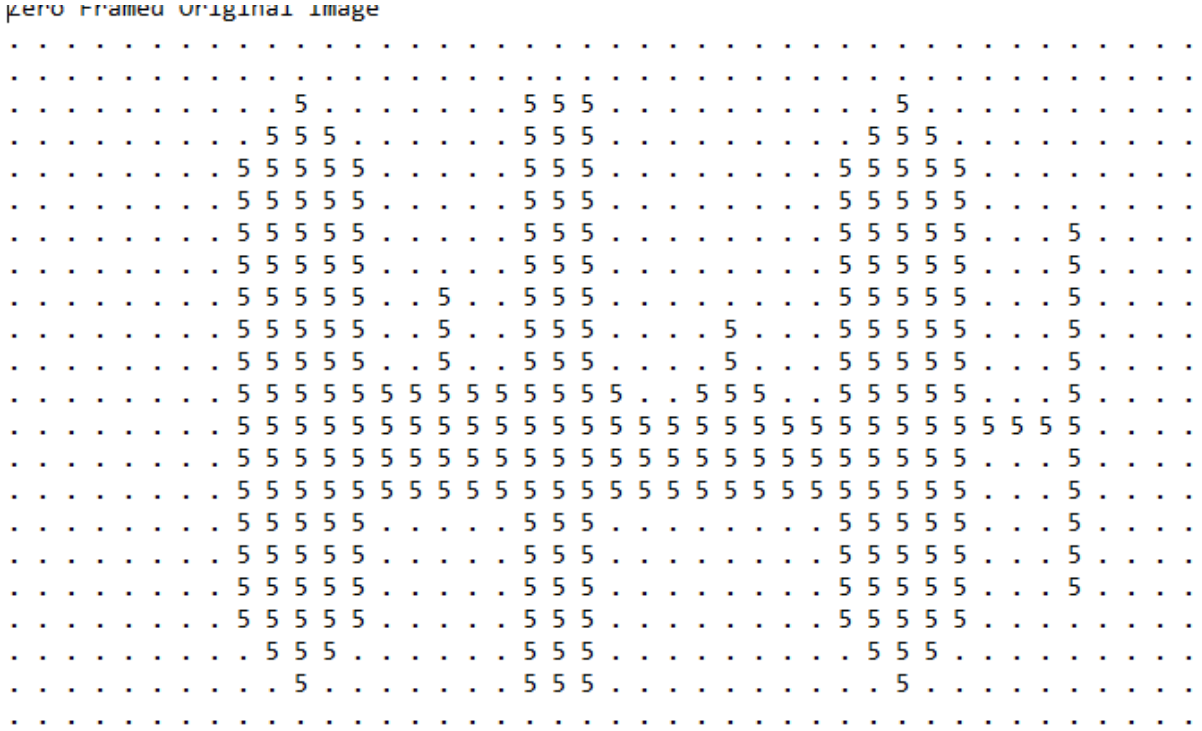
```
void initializeArr(){
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 2; j++){
            this -> neighborCoords[i][j] = 0;
        }
        switch(i){
                case 0: {
                    this -> coordOffset[i][0] = 0;
                    this -> coordOffset[i][1] = 1;
                    break;
                }
                case 1:{
                    this -> coordOffset[i][0] = -1;
                    this -> coordOffset[i][1] = 1;
                    break;
                }
                case 2:{
                    this -> coordOffset[i][0] = -1;
                    this -> coordOffset[i][1] = 0;
                    break;
                }
                case 3:{
                    this -> coordOffset[i][0] = -1;
                    this -> coordOffset[i][1] = -1;
                    break;
                }
                case 4:{
                    this -> coordOffset[i][0] = 0;
                    this -> coordOffset[i][1] = -1;
                    break;
                }
                case 5:{
                    this -> coordOffset[i][0] = 1;
                    this -> coordOffset[i][1] = -1;
                    break;
                }
                case 6:{
                    this -> coordOffset[i][0] = 1;
                    this -> coordOffset[i][1] = 0;
                    break;
                }
                case 7:{
                    this -> coordOffset[i][0] = 1;
                    this -> coordOffset[i][1] = 1;
                    break;
                }
            }
    }
```

Output Files

Data 1

Output File (Pretty Print)

Zero Framed Original Image



Chain Code

20 40 0 5

5 2 10 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 1 1 2 2 2 1 0 0 0 0 7 6 6 6 6 6 0 0 2 2 2 2 2 1 0 0 0 0 0 0 0 7 6 6 6 7 7 1 1 2 2 2 2 2 1 0 0 7 6 6 6 6 2 2 2 2 2 2 2 2 2 2 2 6 6 6 6 6 5 4 4 3 2 2 2 2 2 2 2 3 3 5 5 6 6 6 6 6 6 6 5 4 3 3 2 6 5 5 4 3 3 2 2 2 2 2 2 2 2 4 4 6 6 6 6 6 6 6 6 5 4 3 2 2 6 6 5 4 3 2 2 2 2 2 2 3 3

Boundary File

```
20 40 0 5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 5 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 5 0 0 5 0 5 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 5 0 0 5 0 5 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 5 5 0 5 5 0 0 0 5 0 0 5 0 5 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 0 0 0 5 5 0 0 0 0 0 5 5 5 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 0 5 5 5 5 5 0 0 0 5 5 5 5 5 5 5 5 0 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 5 0 0 0 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0 0 5 0 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
```

Decompressed File Method 1

```
20 40 0 5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 5 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 5 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 5 5 5 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
```

Decompressed File Method 2

```
20 40 0 5
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 5 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 5 0 0 5 5 5 0 0 0 0 5 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 5 5 5 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 5 0 0 0
0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 5 5 5 5 5 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0 0 5 5 5 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5
```

Data 2

Pretty Print

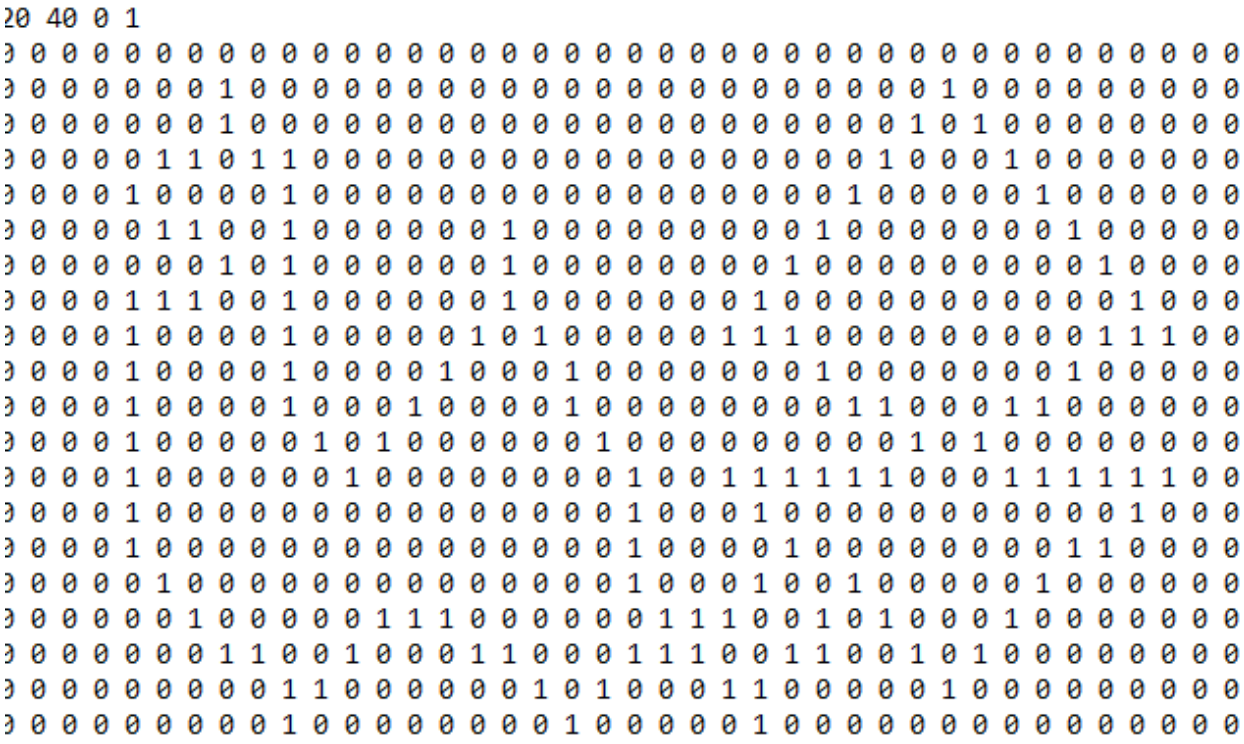
```
Zero Framed Original Image
. . . . . .
. . . . . .
. . . . . 1 . . . . . 1 . . . . . .
. . . . . 1 . . . . . 1 1 1 . . . . . .
. . . . . 1 1 1 1 1 . . . . . 1 1 1 1 1 . . . . . .
. . . . . 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 . . . . . .
. . . . . 1 1 1 1 1 . . . . . 1 . . . . . 1 1 1 1 1 1 1 1 1 . . . . . .
. . . . . 1 1 1 1 1 . . . . . 1 . . . . . 1 1 1 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 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 . . . . . .
. . . . . 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 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 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 1 . . . . . .
. . . . . 1 1 . . . . . 1 1 1 . . . 1 1 . . . . . 1 . . . . . .
. . . . . 1 . . . . . 1 . . . . . 1 . . . . . .
. . . . . .
```

Chain Code

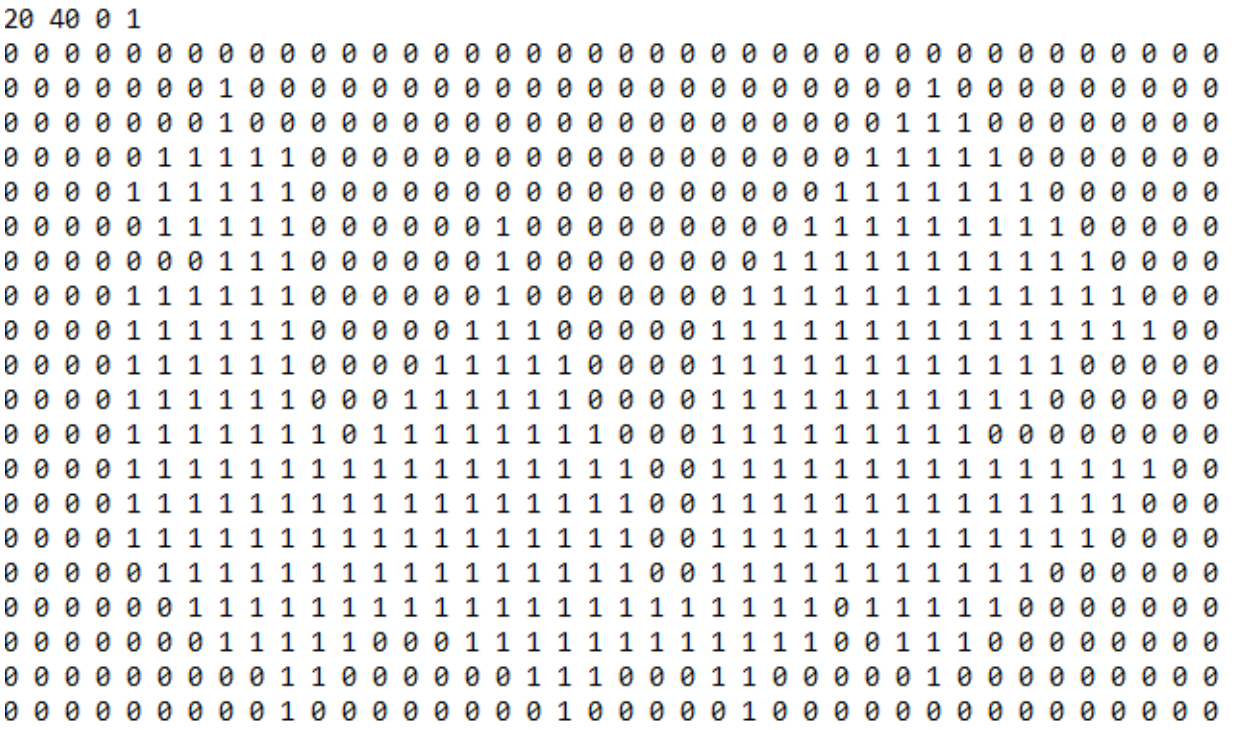
20 40 0 1

1 2 8 6 5 4 5 7 0 7 5 4 4 6 6 6 6 6 6 6 7 7 7 0 7 6 1 1 1 0 0 7 0 7 7 1 1 0 0 7 7 2 1 0
2 1 7 7 7 1 1 1 1 0 1 1 4 4 4 4 4 3 1 0 1 1 0 0 3 3 3 3 3 3 5 5 5 5 5 5 5 0 0 7 7 0 7
5 4 4 4 4 4 7 7 5 5 4 4 3 2 2 2 3 3 2 3 3 2 2 6 6 5 5 5 5 5 3 3 2 2 2 2 2 2 2 4 3 2

Boundary File



Decompressed File Method 1



Decompressed File Method 2

```
20 40 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 1 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 1 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 0 0 0 0 1 1 1 1 1 0 0 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
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 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 0 1 1 1 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 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 0 0 0 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 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 0 0 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 0 1 0 0 0 1 1 1 1 1
0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 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 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```