Project 6 : Thinning

Student: Trisha Espejo
Project Due Date: 4/09/2021

Thin North Algorithm:

Step 1: you will need to for loop the outer from loop let r = 1, while r < numRows + 1, and r++. Then the inner for loop would be let c = 1, while c < numCols + 1, and c++.
Step 2 if pixel is greater than zero, you check for the following cases aryOne[r - 1][j] == 0 and objectNeighbor > 4 and it is not connector. If the statement is true aryTwo[r][c] = 0, and changeflag++;

```
  if (aryOne[r][c] > 0){
    if aryOne[r - 1][c] == 0 and neighbor(4, r, c) == true and connector(r, c) == false)
            {
                    aryTwo[r][c] = 0;
                    changeflag++;
            }
}
```

Note for: South Algorithm it is same template but you just need to change aryOne[r - 1][c] == 0 to aryOne[r + 1][c] == 0. And for West Thinning you use same template but change aryOne[r - 1][c] == 0 to aryOne[r][c - 1] == 0 and change neighbor(4,r,c) to neighbor(3,r,c). While for east same template as North thinning but but change aryOne[r - 1][c] == 0 to aryOne[r][c + 1] == 0 and change neighbor(4,r,c) to neighbor(3,r,c).

Neighbor Algorithm:

Step 0: the method neighbor should be called neighbor (int objectNeighbor, int r, int c)
Where as objectNeighbor is how many neighbor does the pixel need to have greater, r is for current row of pixel and c = current column of pixel. That the method must return Boolean value.
Step 1: Initialize count to 0 and then have 2 for loops the outer for loop would be int i = r - 1; i <= r+ 1; i++ and the inner for loop would be int i = c - 1; i <= c + 1; j++. In the inner for loop you have If i = r and c = j continue and if aryOne is not 0 you increment count by 1;
Step 2: After the two for loop is done if count > objectNeighbor then return true. If not the case on next line have return false.

Connected Algorithm :

Step 0: this method should have return Boolean value.
Step 1: create the variables and initialize RHS, LHS, TS and BS to false.

```
Step 2: if (aryOne [i - 1][j] == 0&& aryOne[i][j - 1] == 0){        if top and left = 0
            If (aryOne [i - 1][j - 1] == 1)                              if top-left =1
              Return true;
}
Step 3: if (aryOne [i + 1][j] == 0&& aryOne[i][j - 1] == 0){        if bottom and left = 0
            If (aryOne [i + 1][ j - 1] == 1)                             if bottom-left = 1
              Return true;
}
Step 4: if (aryOne [i - 1][j] == 0&& aryOne[i][j + 1] == 0){        if top and right = 0
            If (aryOne [i - 1][ j + 1] == 1)                             if top-right = 1
              Return true;
}
Step 5: if (aryOne [i + 1][j]== 0&& aryOne[i][j + 1] == 0){         if bottom and right = 0
            If (aryOne [i + 1][j + 1] == 1)                              if bottom-right = 1
              Return true;
}
```

Step 6: have a for loop as follow for(int c = j – 1; c < j + 1; c++) then inside of it if aryOne [i - 1] [c] == 1 then TS = true; and if ary[i + 1][c] == 1 then BS = true;
Step 7: have a for loop as follow for(int r = r – 1; r < i + 1; r++) then inside of it if aryOne [r][j - 1] == 1 then LHS is true; aryOne[r][j + 1] == 1 then RHS = true;

```
Step 7:  if (aryOne [i][j - 1] == 0 && aryOne[i][j + 1] == 0){.      If left and right  = 0
            if (TS && BS) return true;
}
Step 8:  if (aryOne [i - 1][j] == 0 && aryOne[i +1][j ] == 0){.     If top and bottom = 0
            if (LHS && RHS) return true;
}
Step 9: return false;
```

Source Code:

```
#include <iostream>
#include <fstream>
#include <string>


using namespace std;

class Thinning {

    public:
    int numRows = -1;
    int numCols= -1;
    int minVal = -1;
    int maxVal = -1;
    int changeflag = 1;
    int cycleCount = -1;

    int **aryOne;
    int **aryTwo;


    // dynamically allocate aryOne and ary2 and obtain all values of the
header
    void constructor(ifstream & input){
            input >> this->numRows >> this->numCols >> this->minVal >> this-
>maxVal;

            this -> aryOne = new int*[this -> numRows + 2];
            for (int i = 0; i < this -> numRows + 2; ++i){
                aryOne[i] = new int[ this -> numCols + 2];
            }
            this -> aryTwo = new int*[this -> numRows + 2];
            for (int i = 0; i < this -> numRows + 2; ++i){
                aryTwo[i] = new int[ this -> numCols + 2];
            }

        }

    // zero frame the ary
    void zeroFrame (int **Ary){
            for (int i = 0; i < this -> numRows + 2; ++i){
                for (int j = 0; j < this -> numCols + 2; ++j)
                    Ary[i][j] = 0;
            }

    }

    //adds the inFile values to aryOne
    void loadImage (ifstream & input){

        for (int i = 1; i < this -> numRows + 1 ; ++i){
            for (int j = 1; j < this -> numCols + 1 ; ++j)
                input >> aryOne[i][j];
```

```
        }

    }

    // copy all the values of aryTwo to aryOne
    void copyAry(int **ary1, int **ary2){
        for (int i = 0; i < this -> numRows + 2; ++i){
            for (int j = 0; j < this -> numCols + 2; ++j)
            ary1[i][j] = ary2[i][j];
        }

    }

    void NorthThinning(){
        //copyAry(aryTwo, aryOne);

        for (int i = 1; i < numRows +  1; i++){
            for (int j = 1; j < numCols + 1; j++){
                if ( aryOne[i][j] > 0)
                    North(i, j);
            }
        }


    }
    void North(int i, int j){

        bool Objneighbor = false;
        bool connector = false;
        // check if there is at least 4 object neighbor
        Objneighbor = neighbor(4, i, j);
        // check if it is a connector
        connector = connectedness(i, j);

        if (aryOne[i - 1][j] == 0 && Objneighbor == true && connector ==
false){
            aryTwo[i][j] = 0;
            changeflag++;
        }


    }

    void SouthThinning(){
        for (int i = 1; i < numRows +  1; i++){
            for (int j = 1; j < numCols + 1; j++){
                if ( aryOne[i][j] > 0)
                    South(i, j);

            }
        }
    }
    void South(int i, int j){
        bool Objneighbor = false;
        bool connect = false;
```

```
        // if there are at least 4 object neighbor set pixel to zero
        Objneighbor = neighbor(4, i, j);
        //if it is not a connector then flip
        connect = connectedness(i, j);
        //North neighbor == 0 then set pixel to 0
        if (aryOne[i + 1][j] == 0 && Objneighbor == true && connect ==
false){
            aryTwo[i][j] = 0;
            changeflag++;
        }

    }
    void WestThinning(){

        for (int i = 1; i < numRows +  1; i++){
            for (int j = 1; j < numCols + 1; j++){
                if ( aryOne[i][j] > 0)
                    West(i, j);
            }
        }
    }

    void West( int i, int j){
        bool Objneighbor = false;
        bool connect = false;
        // if there are at least 4 object neighbor set pixel to zero
        Objneighbor = neighbor(3, i, j);
        //if it is not a connector then flip
        connect = connectedness(i, j);
        //West neighbor == 0 then set pixel to 0
        if (aryOne[i][j - 1] == 0 && Objneighbor == true && connect ==
false){
            aryTwo[i][j] = 0;
            changeflag++;
        }
    }

    void EastThinning(){
        for (int i = 1; i < numRows +  1; i++){
            for (int j = 1; j < numCols + 1; j++){
                if ( aryOne[i][j] > 0)
                    East(i, j);
            }
        }
    }

    void East( int i, int j){
        bool Objneighbor = false;
        bool connect = false;
        // if there are at least 4 object neighbor set pixel to zero
        Objneighbor = neighbor(3, i, j);
        //if it is not a connector then flip
        connect = connectedness(i, j);
        //West neighbor == 0 then set pixel to 0
```

```
        if (aryOne[i][j + 1] == 0 && Objneighbor == true && connect ==
false){
            aryTwo[i][j] = 0;
            changeflag++;
        }

    }

    // North: objNeighbor = 4
    bool neighbor(int objNeighbor, int i, int j){
        bool result = false;
        int count = 0;
            for ( int row = i - 1; row <= i + 1; row++) {
                for ( int col = j - 1; col <= j + 1; col++) {
                    if ( row == i && col == j)
                        continue;
                    if (aryOne[row][col] > 0)
                        count++;
                }
            }


        if (count > objNeighbor)
            result = true;
        //cout << result;
        return result;
    }



    bool connectedness(int i, int j){

        bool TS = false;
        bool BS = false;
        bool LHS = false;
        bool RHS = false;
        for(int c = j - 1; c < j + 1; c++){
            if(aryOne[i - 1][c] == 1)
                TS = true;
            if(aryOne[i + 1][c] == 1)
                BS = true;
        }
        for(int r = i - 1; r < i + 1; r++){
            if(aryOne[r][j - 1] == 1)
                LHS = true;
            if(aryOne[r][j + 1] == 1)
                RHS = true;
        }


        // if left and right = 0 and TS is 1
        if (aryOne[i][j - 1] == 0 && aryOne[i][j + 1] == 0){
            if (TS && BS)
                return true;
        }
```

```
    //top and bottom == 0 and LHS and RHS == 1
    if (aryOne[i - 1][j] == 0 && aryOne[i + 1][j] == 0){
        if (LHS && RHS)
            return true;
    }
    // top and left = 0 and top-left = 1
    if (aryOne[i - 1][j] == 0 && aryOne[i][j - 1] == 0){
        if (aryOne[i - 1][j - 1] == 1)
            return true;
    }

    // bottom and left = 0 and bottom-left = 1
    if (aryOne[i + 1][j] == 0 && aryOne[i][j - 1] == 0){
        if (aryOne[i + 1][j - 1] == 1)
            return true;
    }

    // top and right = 0 and top-right = 1
    if (aryOne[i - 1][j] == 0 && aryOne[i][j + 1] == 0){
        if (aryOne[i - 1][j + 1] == 1)
            return true;
    }
    // bottom and right = 0 and bottom-right = 1
    if (aryOne[i + 1][j] == 0 && aryOne[i][j + 1] == 0){
        if (aryOne[i + 1][j + 1] == 1)
            return true;
    }
    return false;

}



void imgReformat(int **inAry, ofstream & OutImg){
    OutImg << " " << this->numRows << " "
           << this->numCols << " "
           << this->minVal << " "
           << this->maxVal << endl;
    OutImg << endl;


    for (int r = 1; r < numRows +  1; r++){
        for (int c = 1; c < numCols + 1; c++){
            if  (0 != inAry[r][c])
                OutImg << inAry[r][c] << " ";
            else
                OutImg << ". ";
        }

        OutImg << endl;
    }
    OutImg << endl;
}

void free_Heap (){
```

```
        for (int i = 0; i < this->numRows + 2; ++i)
            delete[] this->aryOne[i];
        delete[] this->aryOne;


        for (int i = 0; i < this->numRows + 2; ++i)
            delete[] this->aryTwo[i];
        delete[] this->aryTwo;
    }




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

    string inputName = argv[1];
    ifstream input;
    input.open(inputName);


    string output1 = argv[2];
    ofstream outFile1;
    outFile1.open(output1);


    string output2 = argv[3];
    ofstream outFile2;
    outFile2.open(output2);


    Thinning* read_img = new Thinning();
    read_img -> constructor(input);

    read_img -> zeroFrame(read_img -> aryOne);
    read_img -> zeroFrame(read_img -> aryTwo);
    read_img -> loadImage(input);

    read_img -> cycleCount = 0;
    outFile2 << "Original Image: " << "Cycle " << read_img -> cycleCount  <<
endl;
    read_img -> imgReformat(read_img -> aryOne, outFile2);

    read_img -> copyAry(read_img -> aryTwo, read_img -> aryOne);
    read_img -> changeflag = 0;
    read_img -> copyAry(read_img -> aryTwo, read_img -> aryOne);

    read_img -> NorthThinning();
    read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);


    read_img -> SouthThinning();
    read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);
```

```
    read_img -> WestThinning();
    read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);


    read_img -> EastThinning();
    read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);
    read_img -> cycleCount++;

    outFile2 << "Thinning Image : " << "Cycle " << read_img -> cycleCount <<
endl;
    read_img -> imgReformat(read_img -> aryOne, outFile2);

    while (read_img -> changeflag > 0){
        read_img -> changeflag = 0;
        read_img -> NorthThinning();
        read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);


        read_img -> SouthThinning();
        read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);


        read_img -> WestThinning();
        read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);

        read_img -> EastThinning();
        read_img -> copyAry(read_img -> aryOne, read_img -> aryTwo);
        read_img -> cycleCount++;
        outFile1 << "Thinning Image : " << "Cycle " << read_img -> cycleCount
<< endl;
        read_img -> imgReformat(read_img -> aryOne, outFile1);

    }

    read_img -> free_Heap();
    outFile1.close();
    outFile2.close();

    return 0;

}
```

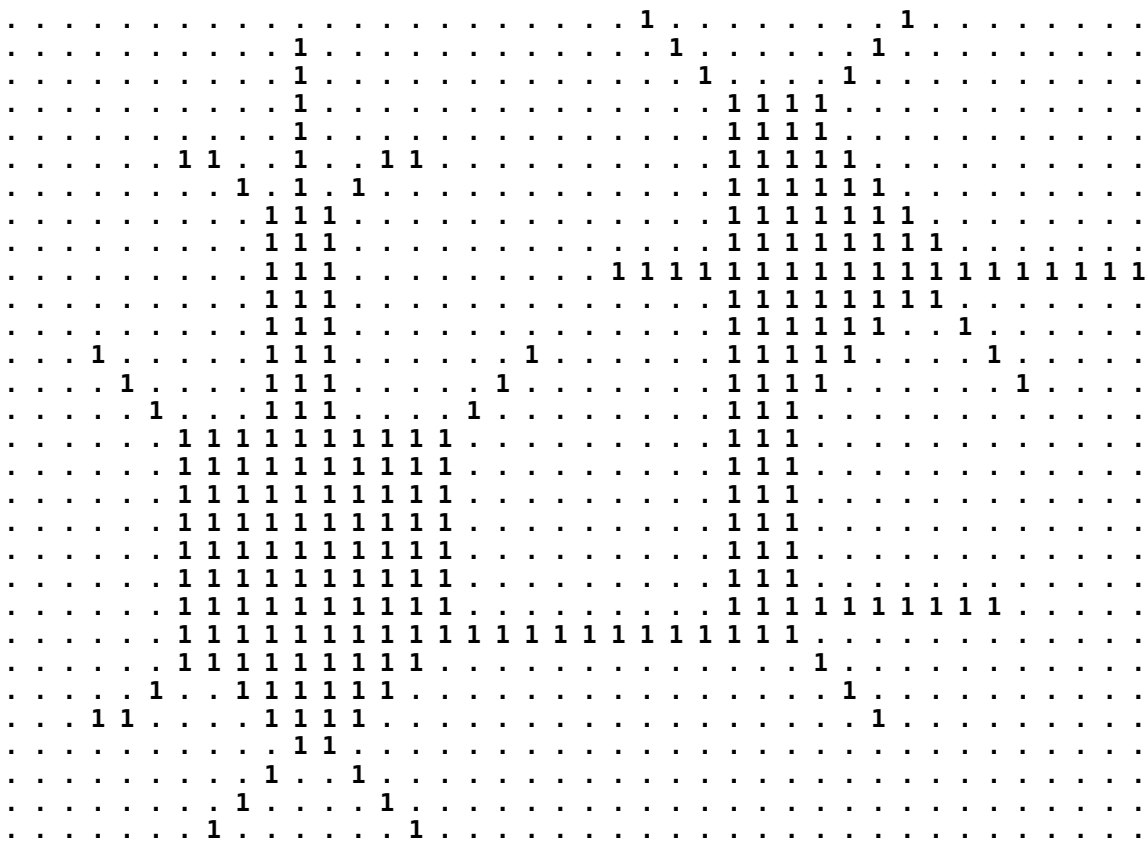
Image 1: (image1.txt)

OutFile1.txt:

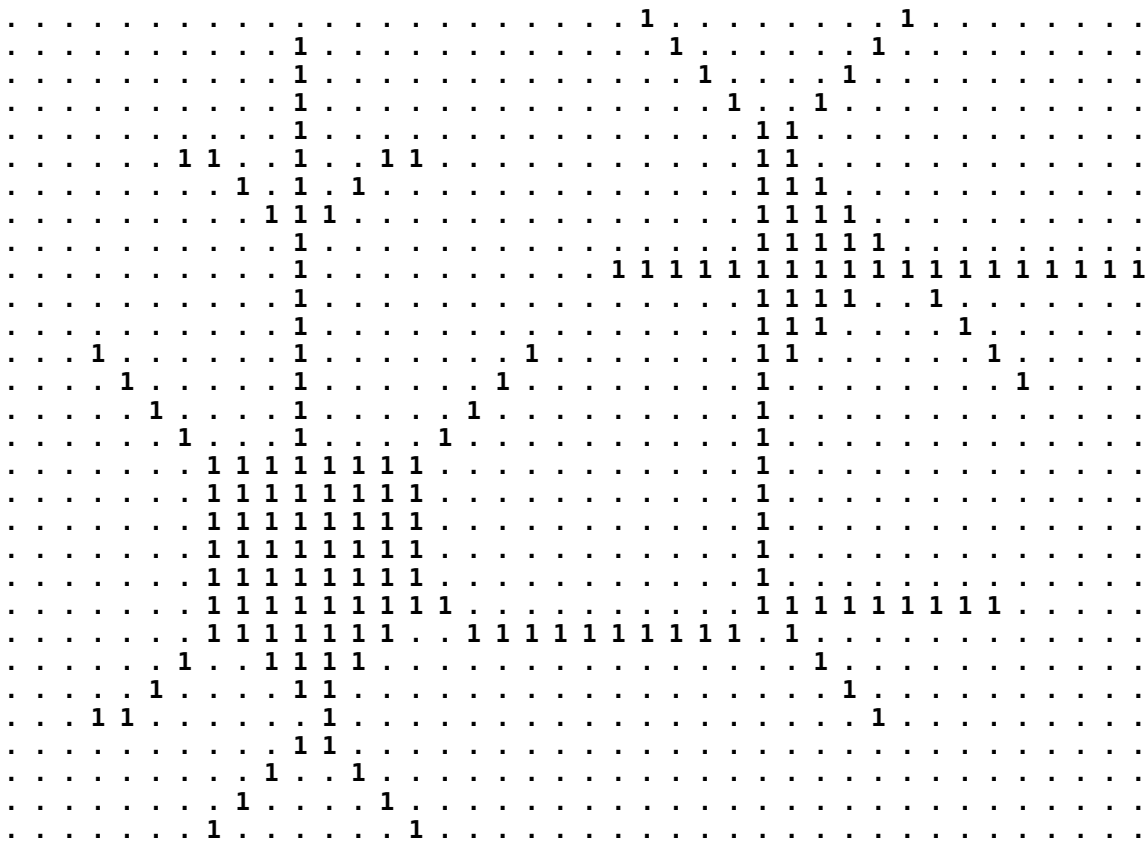
```
Thinning Image : Cycle 2
 30 40 0 1

. . . . . . . . . . . . . . . . . . . . . . 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 1 1 . . . . . . .
. . . . . . . . 1 1 1 1 1 . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . . . . . . 1 1 1 1 1 . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . . . . . 1 1 1 1 1 . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . . . . 1 1 1 1 1 . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . . . . . 1 1 1 1 1 . . . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . 1 . . . . 1 1 1 1 1 . . . . . 1 . . . . . 1 1 1 1 1 1 1 1 . . 1 . . . . .
. . . . 1 . . . 1 1 1 1 1 . . . . 1 . . . . . . 1 1 1 1 1 1 1 . . . . 1 . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . 1 . . . . . . . . . .
. . . 1 1 . . 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . 1 . . . . . . . . .
. . . . . . . . 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . 1 . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
```

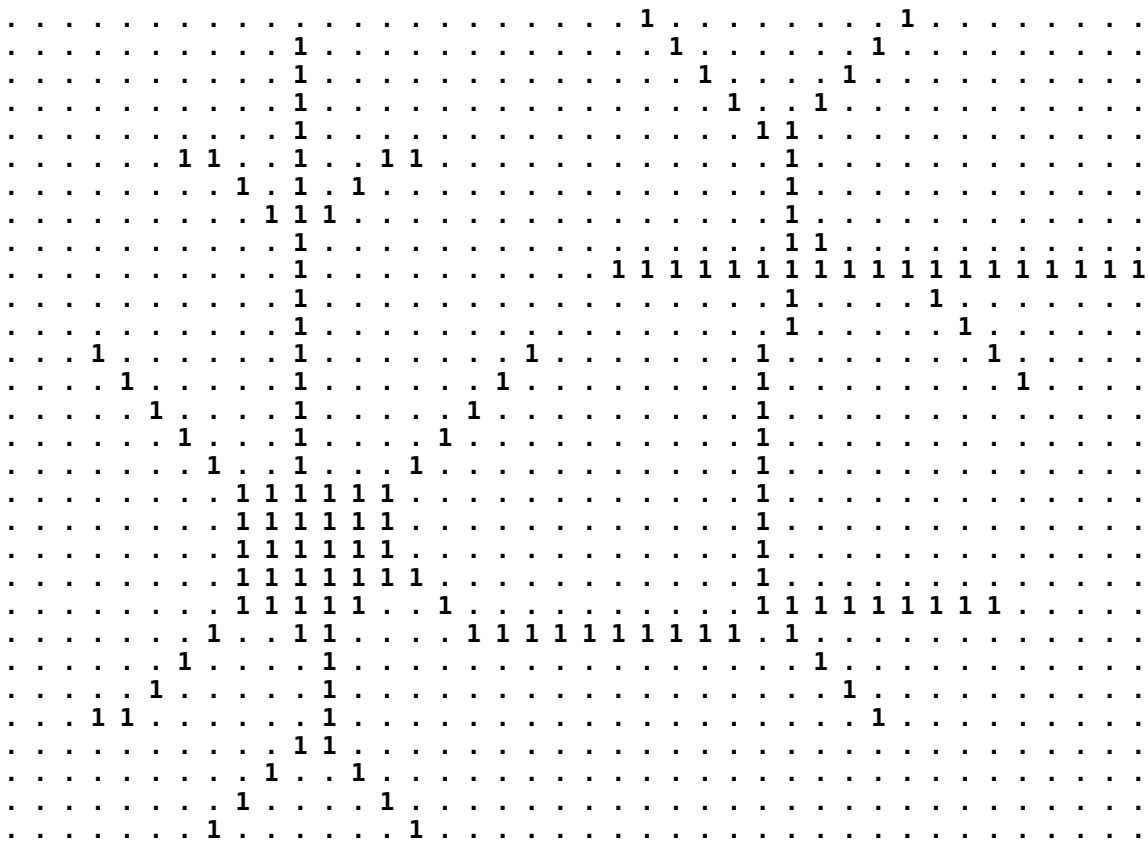
Thinning Image : Cycle 3
30 40 0 1



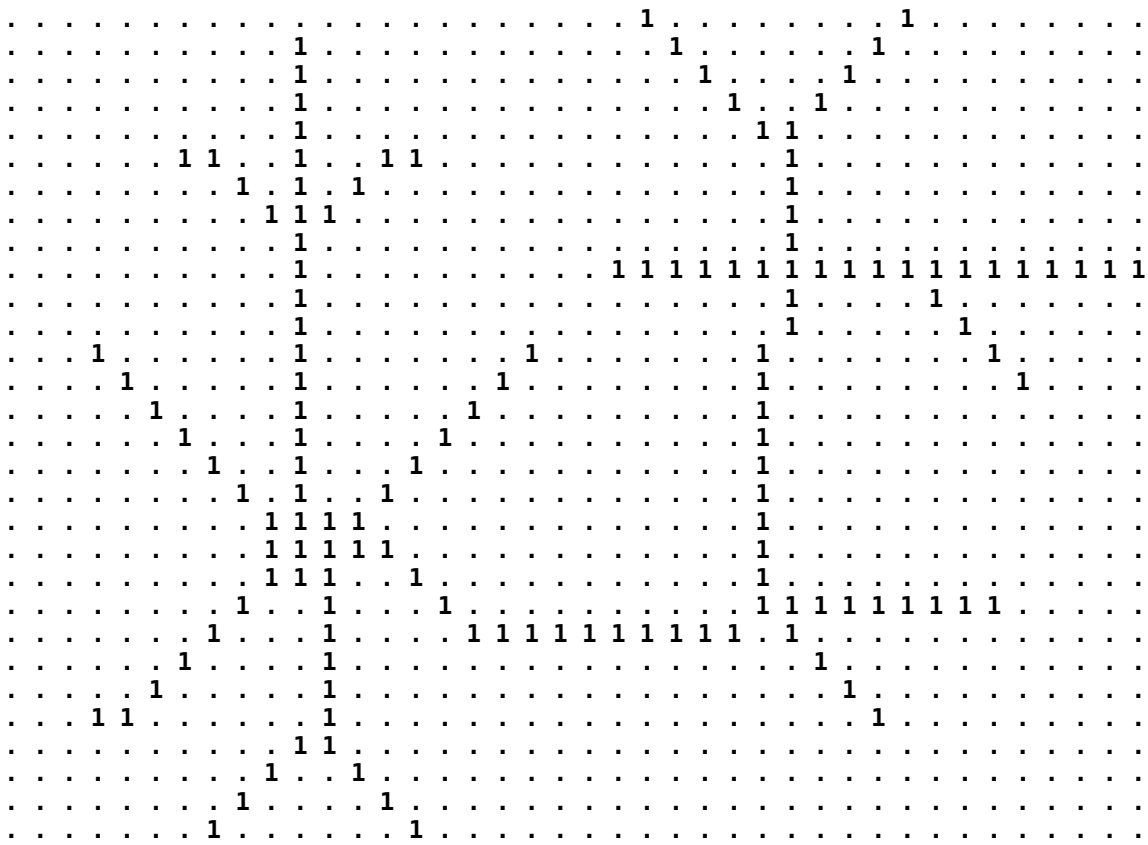
Thinning Image : Cycle 4
30 40 0 1



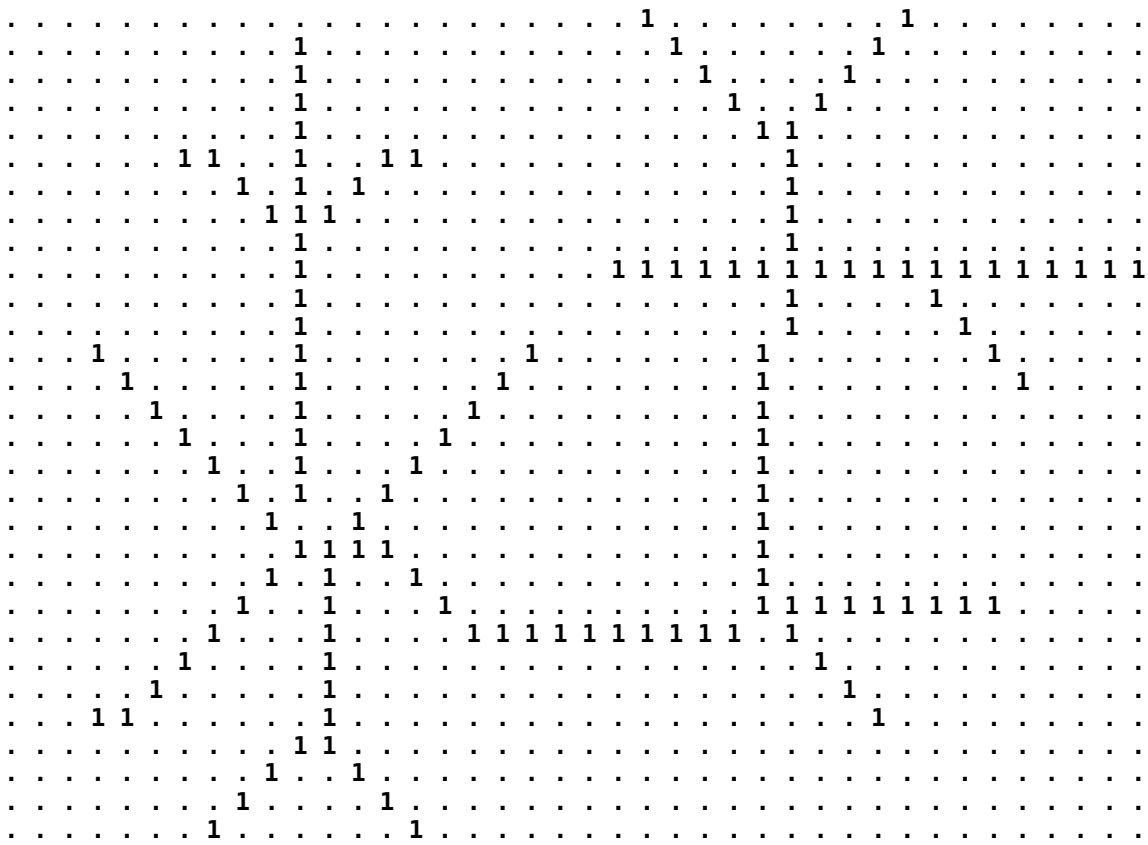
Thinning Image : Cycle 5
30 40 0 1



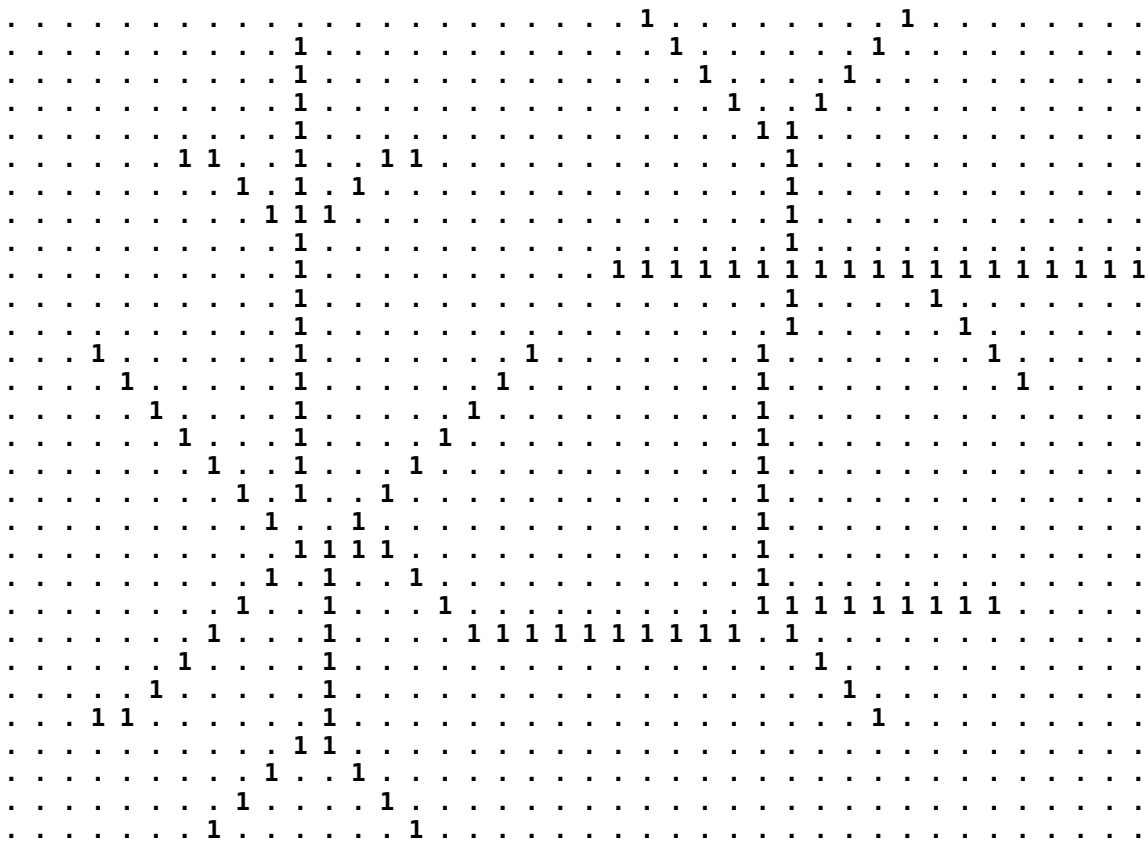
Thinning Image : Cycle 6
30 40 0 1



Thinning Image : Cycle 7
30 40 0 1



Thinning Image : Cycle 8
30 40 0 1



OutFile2.txt:

```
Original Image: Cycle 0
 30 40 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 . . . . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 .
. . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 1 . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . .
. . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . .
```

Thinning Image : Cycle 1

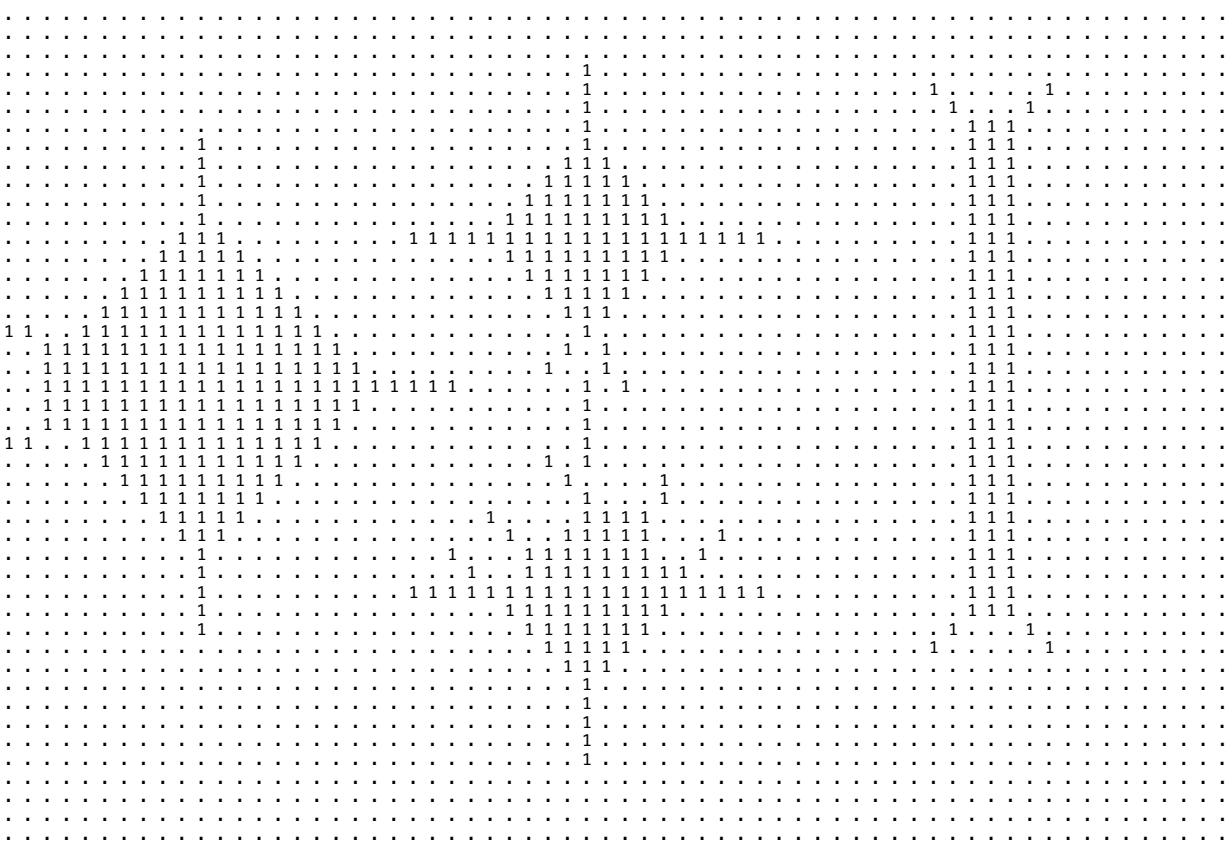
```
 30 40 0 1

. . . . . . . . . . . . . . . . . . . . . . 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 . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 . . . . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . .
. . . . . . . 1 1 1 1 1 1 1 . . . . . . . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . .
. . . 1 . . . 1 1 1 1 1 1 1 . . . . 1 . . . . 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 1 1 1 . . 1 . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 . . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 . . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 . . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . .
. . . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . .
. . . 1 1 1 1 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . 1 . . . . . . . . .
. . . . . . 1 1 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 1 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . 1 1 1 1 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . 1 . . . . . . 1 . . . . . . . . . . . . . . . . . . . . . . . . .
```

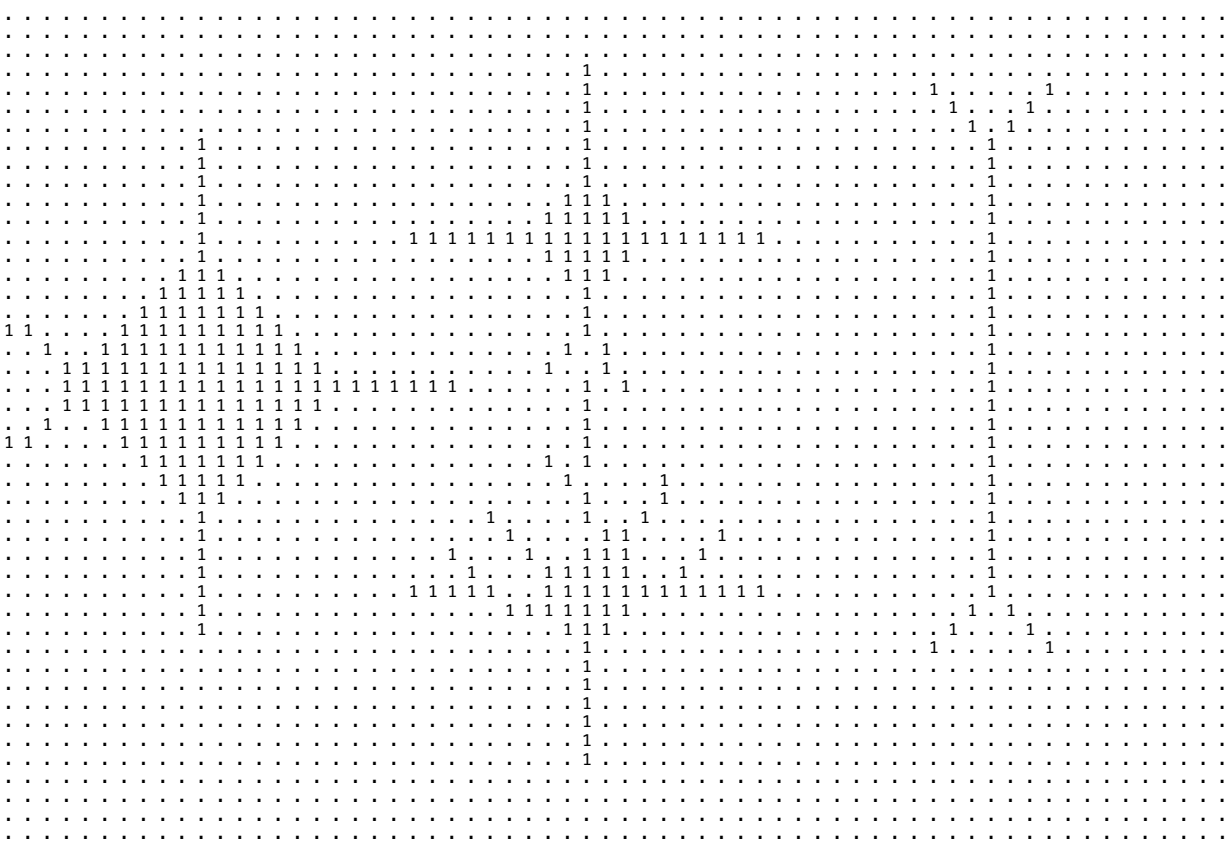
Image 2: (image2.txt)

OutFile1.txt:

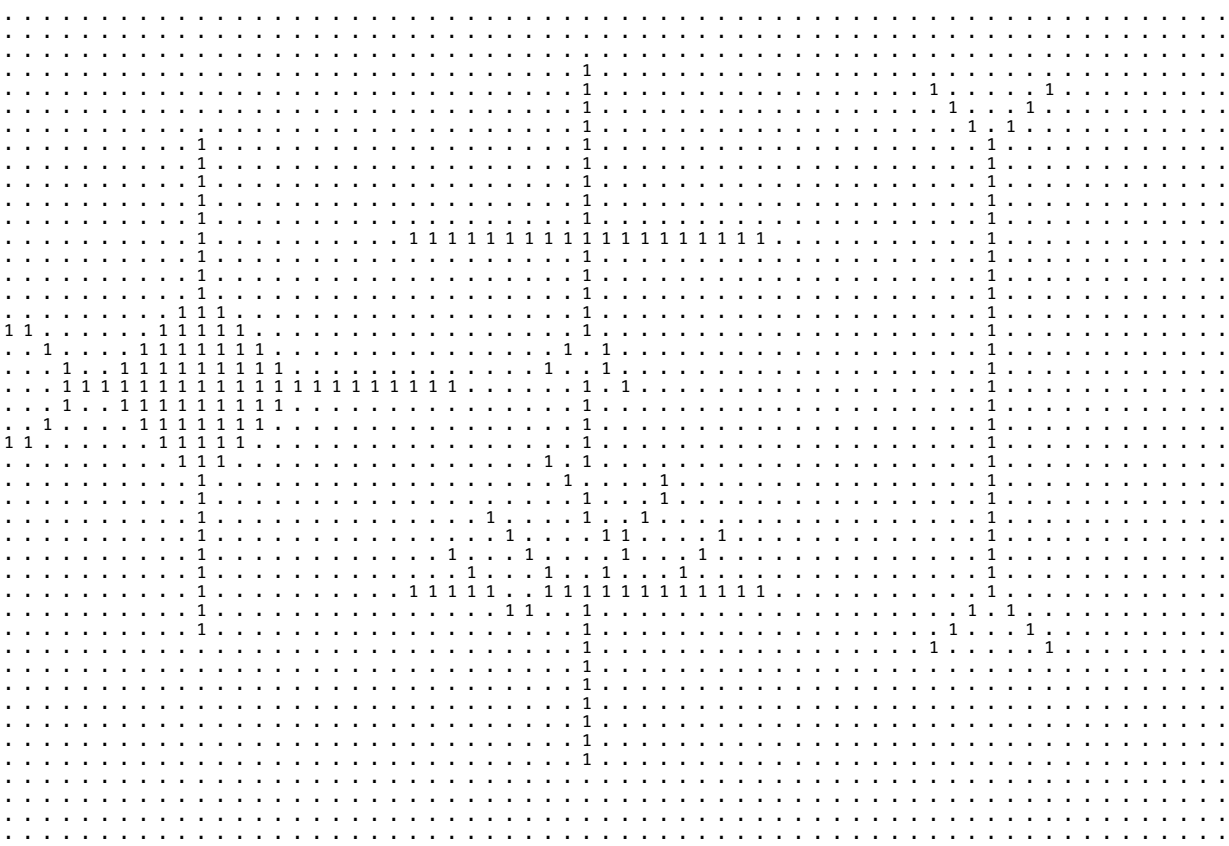
```
Thinning Image : Cycle 2
 45 64 0 1
```



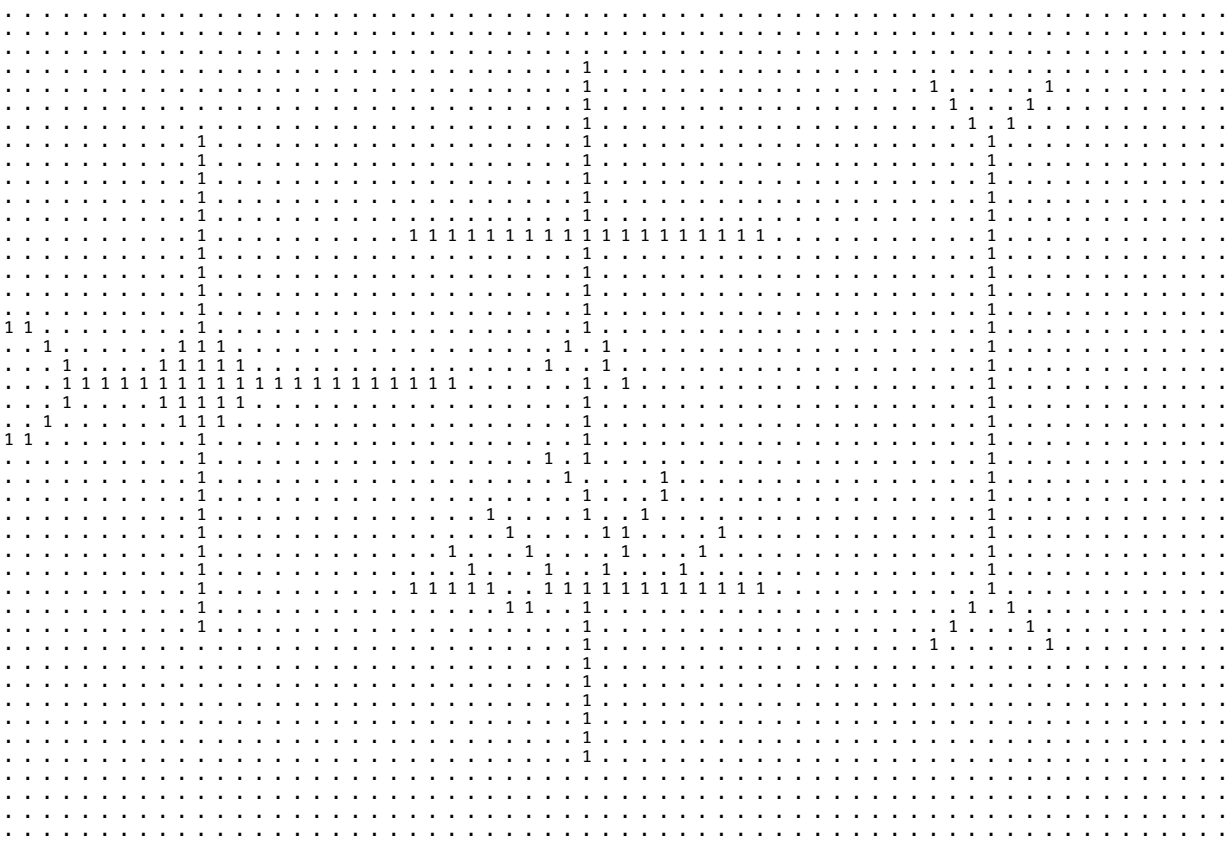
Thinning Image : Cycle 3
 45 64 0 1



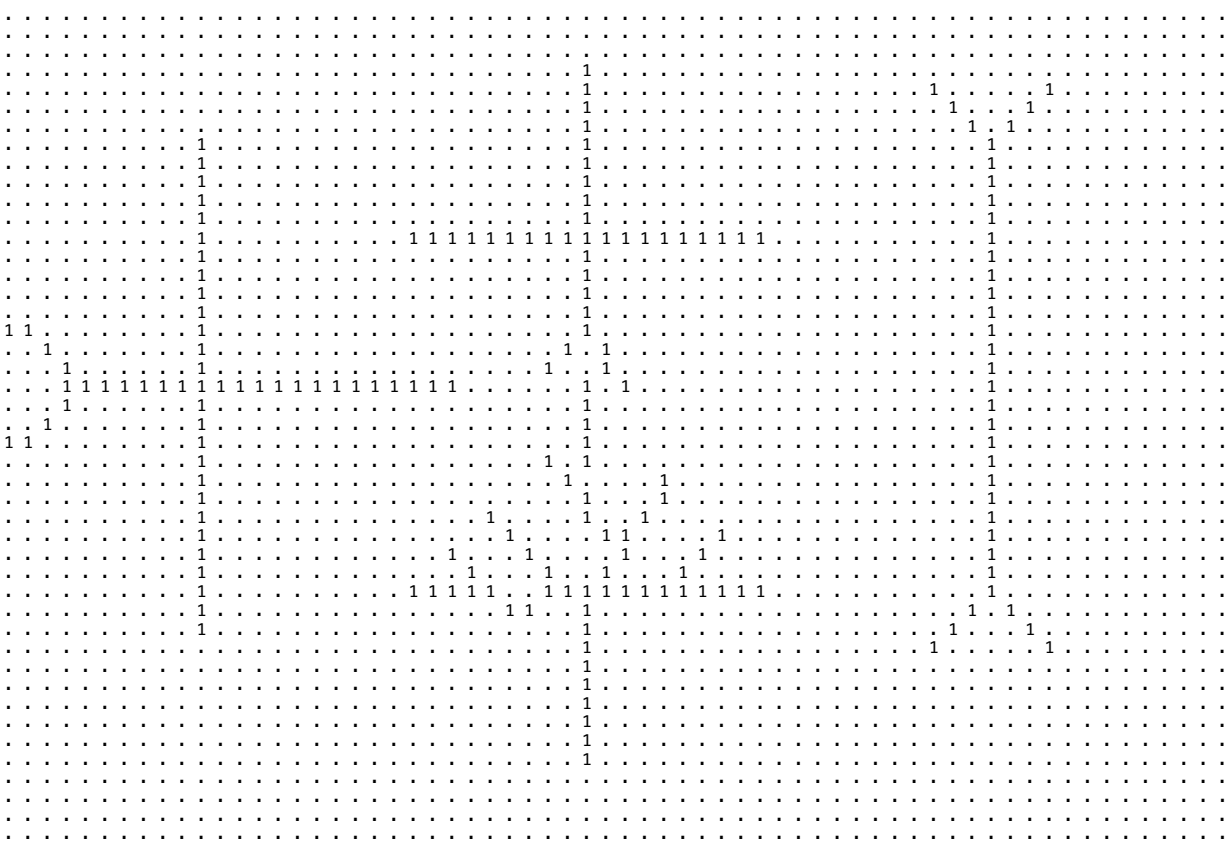
Thinning Image : Cycle 4
 45 64 0 1



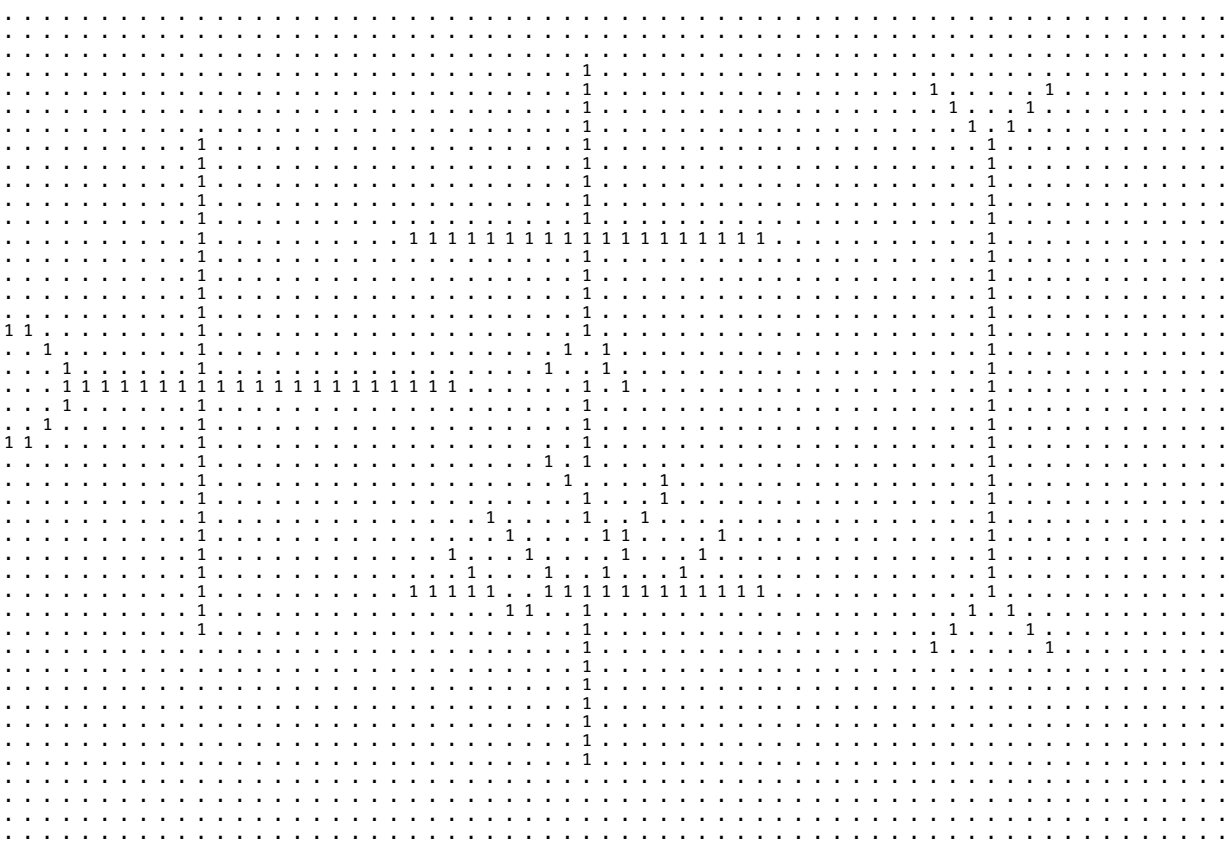
Thinning Image : Cycle 5
 45 64 0 1



Thinning Image : Cycle 6
 45 64 0 1

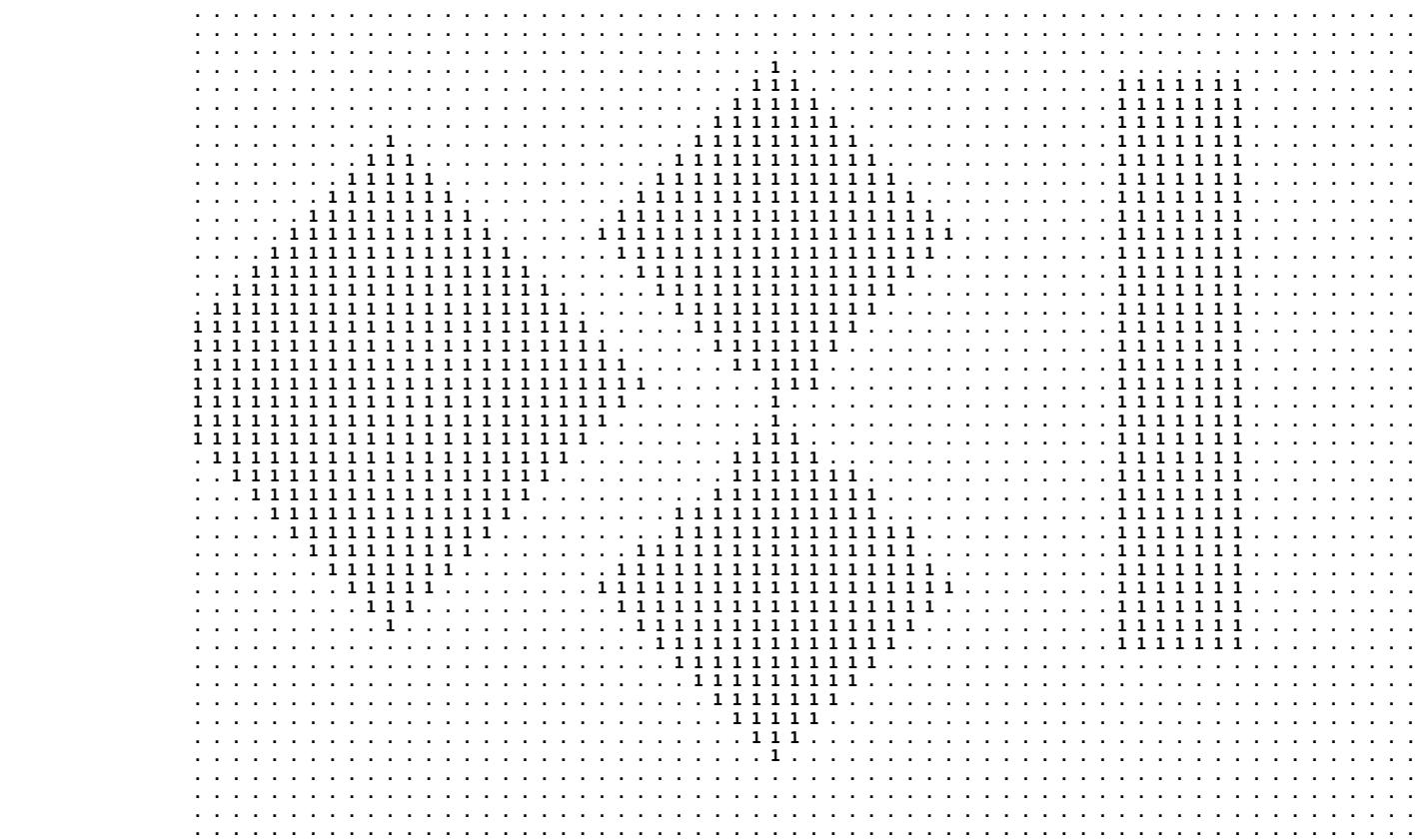


Thinning Image : Cycle 7
 45 64 0 1



OutFile2.txt:

```
Original Image: Cycle 0
 45 64 0 1
```



Thinning Image : Cycle 1
45 64 0 1