CV **Thinning** C++

Student: Shuhua Song

Project Due Date: 04/09/2021

(1) Algorithm Steps for Thin in North(South/West/East) (inArray, outAry):
step 1: Scan image left->right & top->bottom
step2: p(i, j) <- get next pixel
step3: if p(i, j) > 0:
change p(i,j) to 0 under the following condition
(1) North neighbor of p(i,j) is 0
(2) p(i,j) has at least 4 object neighbors
(3) p(i,j) is not a connector, by flipping p(i,j) to 0 will not create more htan one connected component in p's 3X3 neighborhood

(2) Algorithm Steps for thinning (inArray, outAry):
step 1: image ← given binary image
step 2: Thin North (1 layer)
step 3: Thin South (1 layer)
step 4: Thin West (1 layer)
step 5: Thin Each (1 layer)
step 6: repeat steps 1 to 4 until no more pixels change from 1 to 0

Source Code:

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

class Thinning{
public:
    int numRows, numCols;
    int minVal, maxVal;
    int changeFlag;
    int cycleCount;

    int** aryOne; //numRows+2 by numCols+2
    int** aryTwo;
    Thinning(){ }
    Thinning(int numRows, int numCols, int minVal, int maxVal, int changeFlag, int
cycleCount){
        this->numRows = numRows;
        this->numCols = numCols;
        this->minVal = minVal;
        this->maxVal =maxVal;
        this->changeFlag = changeFlag;
        this->cycleCount = cycleCount;


        aryOne = new int*[numRows+2];
```

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

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

    void zeroFrame(int** aryOne){
        for(int i=0; i<numRows+2; i++){
            for(int j=0; j<numCols+2; j++){
                aryOne[i][j] = 0;
            }
        }
    }

    void loadImage(ifstream& inFile, int** aryOne, int** aryTwo){
       int value = -1;
        for(int i=1; i<numRows+1; i++){
            for(int j=1; j<numCols+1; j++){
                inFile >> value;
                aryOne[i][j] = value;
                aryTwo[i][j] = value;
            }
        }
    }
    void WestThinning(int** aryOne, int** aryTwo){
        //cout << endl << "West: " << endl;
        changeFlag = 0;
        bool zeroWestNeighbor = false;
        bool with3Neighbor = false;
        bool isConnector = false;
        for(int i=1; i<numRows+1; i++){
            for(int j=1; j<numCols+1; j++){
                if(aryOne[i][j] > 0){
                    zeroWestNeighbor = ( aryOne[i][j-1]==0 ? true : false );
                    if(zeroWestNeighbor){
                        with3Neighbor = checkNeighbors_3(aryOne, i, j);
                        isConnector = checkConnector(aryOne, i, j);
                       // cout << i << " " << j << " " << aryOne[i][j] <<
zeroWestNeighbor << " " << with3Neighbor << " " << !isConnector << endl;
                        if( with3Neighbor && !isConnector){
                            aryTwo[i][j] = 0;
                            changeFlag++;
                        }
                    }
                }
            }
        }
    }
    void SouthThinning(int** aryOne, int** aryTwo){
      //  cout << endl << "South: " << endl;
        changeFlag = 0;
```

```
        bool zeroSouthNeighbor = false;
        bool with4Neighbor = false;
        bool isConnector = false;
        for(int i=1; i<numRows+1; i++){
            for(int j=1; j<numCols+1; j++){
                if(aryOne[i][j] > 0){
                    zeroSouthNeighbor = ( aryOne[i+1][j]==0 ? true : false );
                    if(zeroSouthNeighbor){
                        with4Neighbor = checkNeighbors_4(aryOne, i, j);
                        isConnector = checkConnector(aryOne, i, j);
                       // cout << i << " " << j << " " << aryOne[i][j] <<
zeroSouthNeighbor << " " << with4Neighbor << " " << !isConnector << endl;
                        if( with4Neighbor && !isConnector){
                            aryTwo[i][j] = 0;
                            changeFlag++;
                        }
                    }
                }
            }
        }
    }

    void EastThinning(int** aryOne, int** aryTwo){
        //cout << "East" << endl;
        changeFlag = 0;
        bool zeroEastNeighbor = false;
        bool with3Neighbor = false;
        bool isConnector = false;
        for(int i=1; i<numRows+1; i++){
            for(int j=1; j<numCols+1; j++){
                if(aryOne[i][j] > 0){
                    zeroEastNeighbor  = ( aryOne[i][j+1]==0 ? true : false );
                    if(zeroEastNeighbor ){
                        with3Neighbor = checkNeighbors_3(aryOne, i, j);
                        isConnector = checkConnector(aryOne, i, j);
                       //cout << i << " " << j << " " << aryOne[i][j] <<
zeroEastNeighbor  << " " << with3Neighbor << " " << !isConnector << endl;
                        if( with3Neighbor && !isConnector){
                            aryTwo[i][j] = 0;
                            changeFlag++;
                        }
                    }
                }
            }
        }
    }

    void NorthThinning(int** aryOne, int** aryTwo){
        // cout << "North" << endl;
         changeFlag = 0;
         bool zeroNorthNeighbor = false;
         bool with4Neighbor = false;
         bool isConnector = false;
         for(int i=1; i<numRows+1; i++){
             for(int j=1; j<numCols+1; j++){
                 if(aryOne[i][j] > 0){
                     zeroNorthNeighbor = ( aryOne[i-1][j]==0 ? true : false );
                     if(zeroNorthNeighbor){
                         with4Neighbor = checkNeighbors_4(aryOne, i, j);
                         isConnector = checkConnector(aryOne, i, j);
                        // cout << i << " " << j << " " << aryOne[i][j] <<
```

```
zeroNorthNeighbor << " " << !with4Neighbor << " " << isConnector << endl;
                        if( with4Neighbor && !isConnector){
                            aryTwo[i][j] = 0;
                            changeFlag++;
                        }
                    }
                }
            }
        }
    }

    bool checkConnector(int** aryOne, int i, int j){
        if(aryOne[i][j-1]==0 && aryOne[i][j+1]==0){
            if((aryOne[i-1][j-1]==1 || aryOne[i-1][j]==1 || aryOne[i-1][j+1]==1) &&
                    (aryOne[i+1][j-1]==1 || aryOne[i+1][j]==1 ||
aryOne[i+1][j+1]==1)){
                return true;
            }
        }
        else if(aryOne[i-1][j]==0 && aryOne[i+1][j]==0){
            if((aryOne[i-1][j-1]==1 || aryOne[i][j-1]==1 || aryOne[i+1][j-1]==1) &&
                    (aryOne[i-1][j+1]==1 || aryOne[i][j+1]==1 ||
aryOne[i+1][j+1]==1)){
                return true;
            }
        }
        else if(aryOne[i-1][j]==0 && aryOne[i][j-1]==0 && aryOne[i-1][j-1]==1){
            return true;
        }
        else if(aryOne[i][j-1]==0 && aryOne[i+1][j]==0 && aryOne[i+1][j-1]==1){
            return true;
        }
        else if(aryOne[i-1][j]==0 && aryOne[i][j+1]==0 && aryOne[i-1][j+1]==1){
            return true;
        }
        else if(aryOne[i][j+1]==0 && aryOne[i+1][j]==0 && aryOne[i+1][j+1]==1){
            return true;
        }
        return false;
    }

    bool checkNeighbors_3(int** aryOne, int i, int j){
        int sum = 0;
        for(int x=i-1; x<=i+1; x++){
            for(int y=j-1; y<=j+1; y++){
                if(x==i && y==j) continue;
                sum += aryOne[x][y];
            }
        }
        return sum >= 3;
    }

    bool checkNeighbors_4(int** aryOne, int i, int j){
        int sum = 0;
        for(int x=i-1; x<=i+1; x++){
            for(int y=j-1; y<=j+1; y++){
                if(x==i && y==j) continue;
                sum += aryOne[x][y];
            }
        }
        return sum >= 4;
```

```
    }

    void copyArys(int** aryOne, int** aryTwo){
        for(int i=1; i<numRows+1; i++){
            for(int j=1; j<numCols+1; j++){
                aryOne[i][j] = aryTwo[i][j];
            }
        }
    }
    void reformatPrettyPrint(int** inArray, ofstream& outFile){
        outFile << endl;
        for(int i=0; i<numRows+2; i++){
            for(int j=0; j<numCols+2; j++){
                if(inArray[i][j]==0){
                    outFile << "." << " ";
                }else{
                    outFile << inArray[i][j] << " ";
                }
            }
            outFile << endl;
        }
    }
};

int main(int argc, const char *argv[]) {
    ifstream inFile(argv[1]);
    ofstream outFile1(argv[2]);
    ofstream outFile2(argv[3]);
    int numRows = 0, numCols = 0, minVal = 0, maxVal = 0;
    inFile >> numRows;
    inFile >> numCols;
    inFile >> minVal;
    inFile >> maxVal;
    cout << numRows << " " << numCols << " " << minVal << " "<< maxVal << endl;
    outFile1 << numRows << " " << numCols << " " << minVal << " "<< maxVal << endl;
    int changeFlag = 0;
    int cycleCount = 0;
    Thinning *thinning = new Thinning(numRows, numCols, minVal, maxVal, changeFlag,
cycleCount);
    changeFlag = thinning->cycleCount;
    int** aryOne =  thinning->aryOne;
    int** aryTwo = thinning->aryTwo;
    thinning->zeroFrame(aryOne);
    thinning->zeroFrame(aryTwo);
    thinning->loadImage(inFile, aryOne, aryTwo);

    outFile2 << endl << "input image: " << endl;
    thinning->reformatPrettyPrint(aryOne, outFile2);

    changeFlag = INT_MAX;
    cycleCount = 1;
    while(changeFlag > 0){
        thinning->NorthThinning(aryOne, aryTwo);
        thinning->copyArys(aryOne, aryTwo);
        changeFlag = thinning->changeFlag;

        thinning->SouthThinning(aryOne, aryTwo);
        thinning->copyArys(aryOne, aryTwo);
        changeFlag = thinning->changeFlag;

        thinning->WestThinning(aryOne, aryTwo);
```

```
        thinning->copyArys(aryOne, aryTwo);
        changeFlag = thinning->changeFlag;

        thinning->EastThinning(aryOne, aryTwo);
        thinning->copyArys(aryOne, aryTwo);
        changeFlag = thinning->changeFlag;
        outFile2 << endl << "result of thinning: cycle - " << cycleCount << endl;
        thinning->reformatPrettyPrint(aryTwo, outFile2);
        cycleCount++;
    }
    thinning->reformatPrettyPrint(aryOne, outFile1);
    inFile.close();
    outFile1.close();
    outFile2.close();
    return 0;
}
```

Output

Image1-outFile1

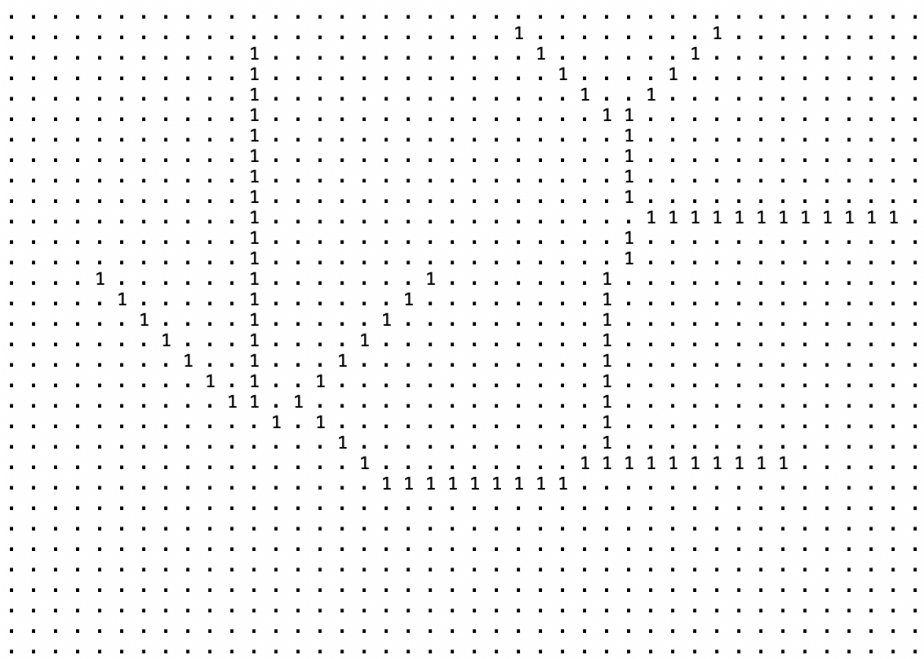
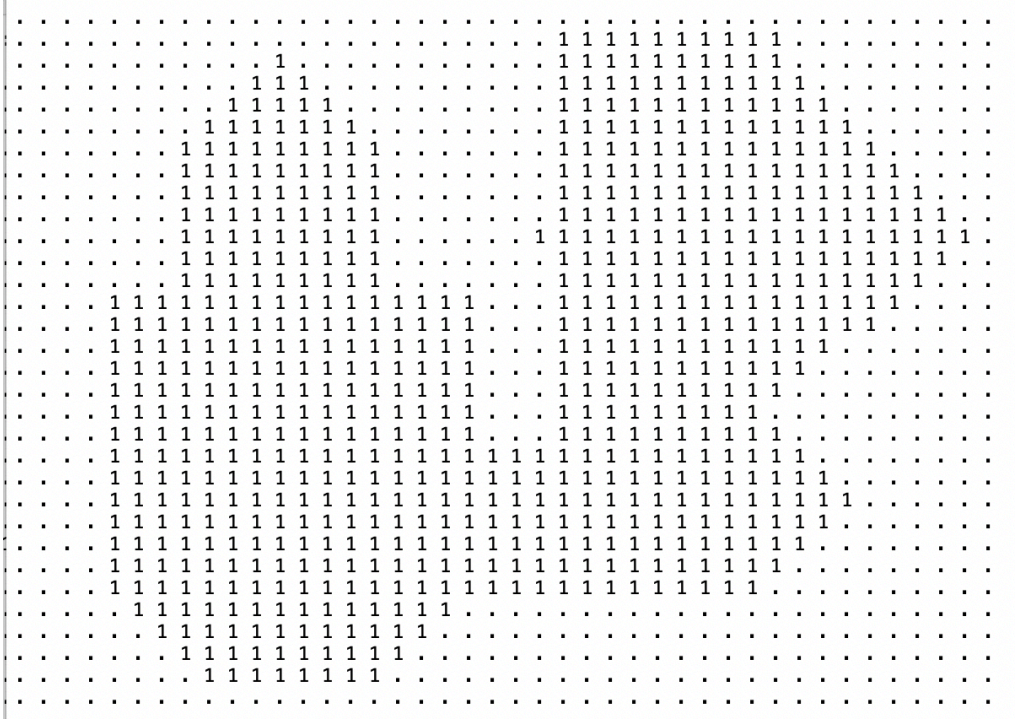
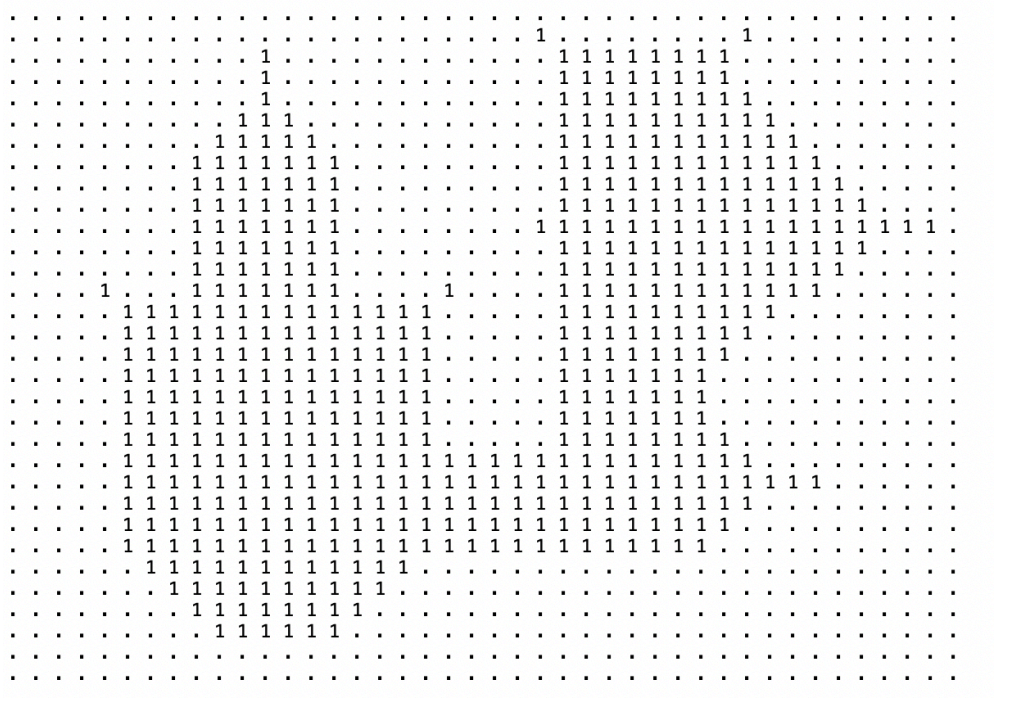


Image1-outFile2

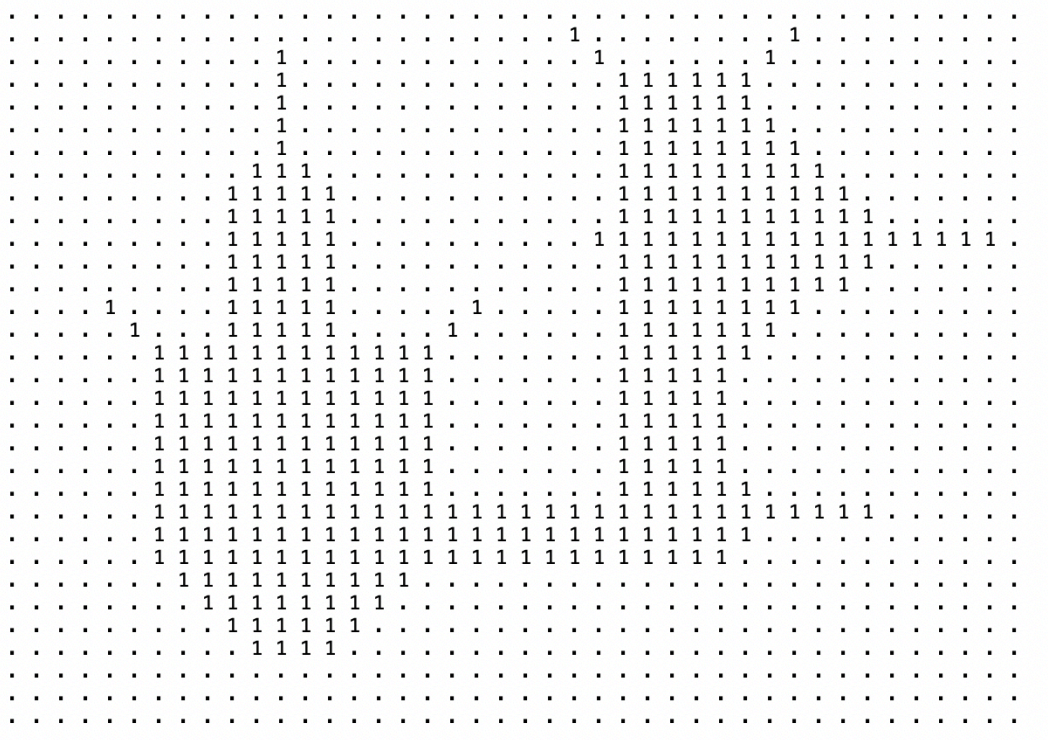
input image:



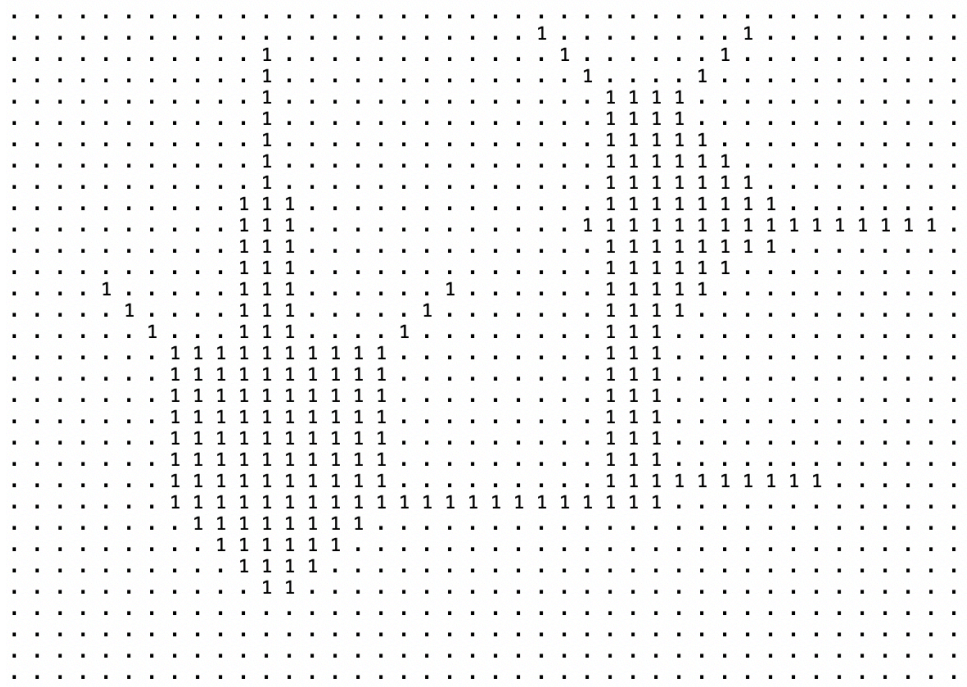
result of thinning: cycle = 1



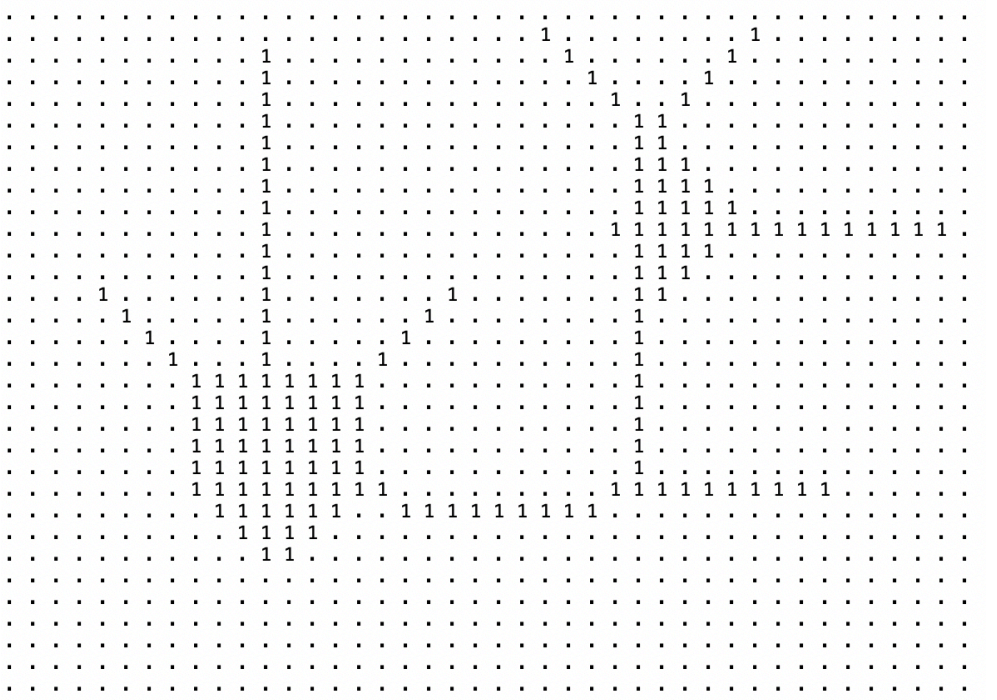
result of thinning: cycle - 2



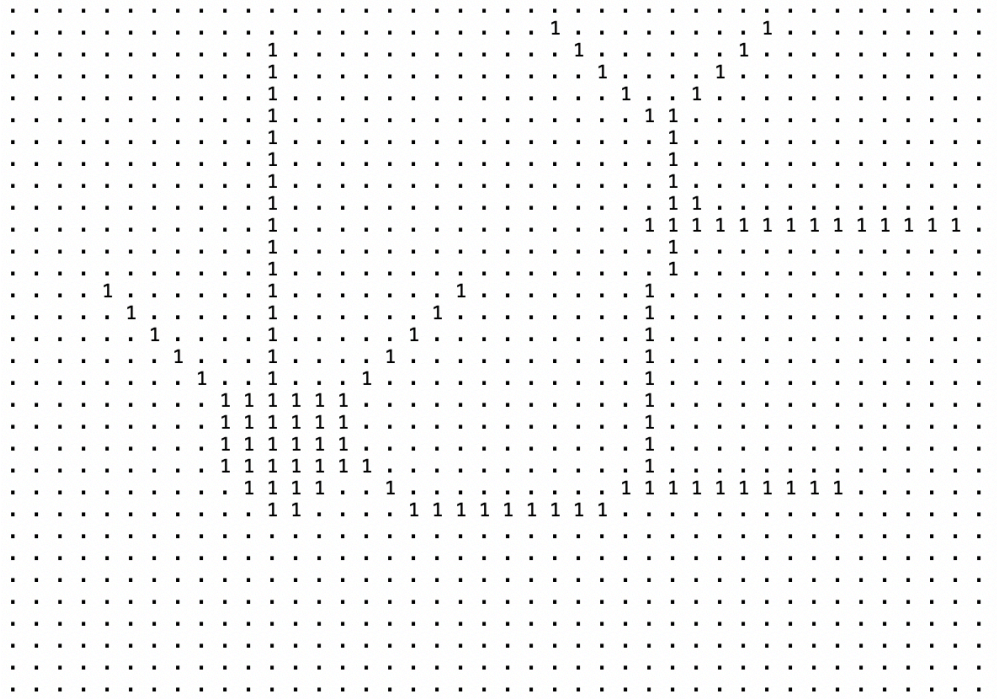
result of thinning: cycle - 3



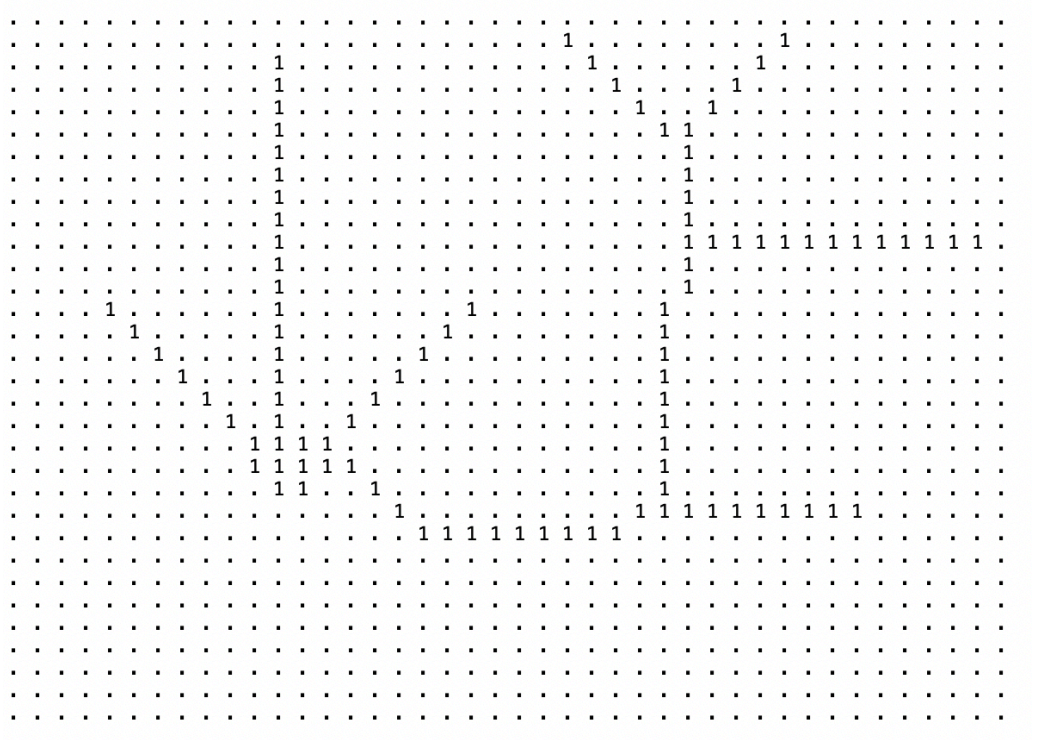
result of thinning: cycle = 4



result of thinning: cycle = 5



result of thinning: cycle - 6



result of thinning: cycle - 7

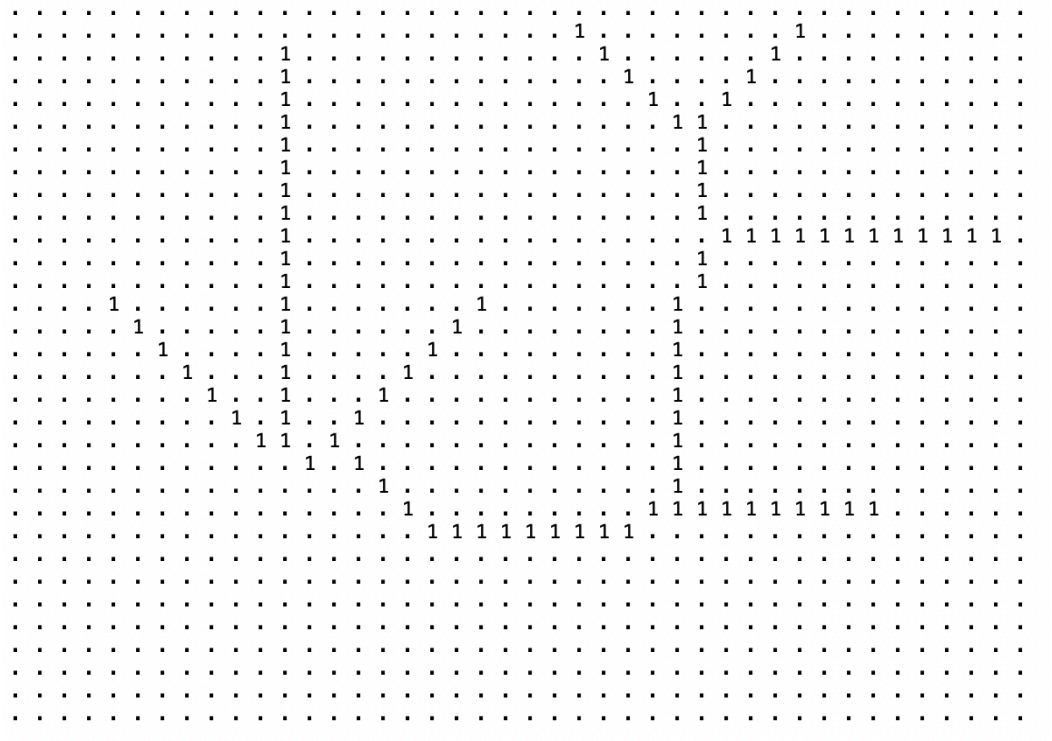


Image2-outFile1

45 64 0 1

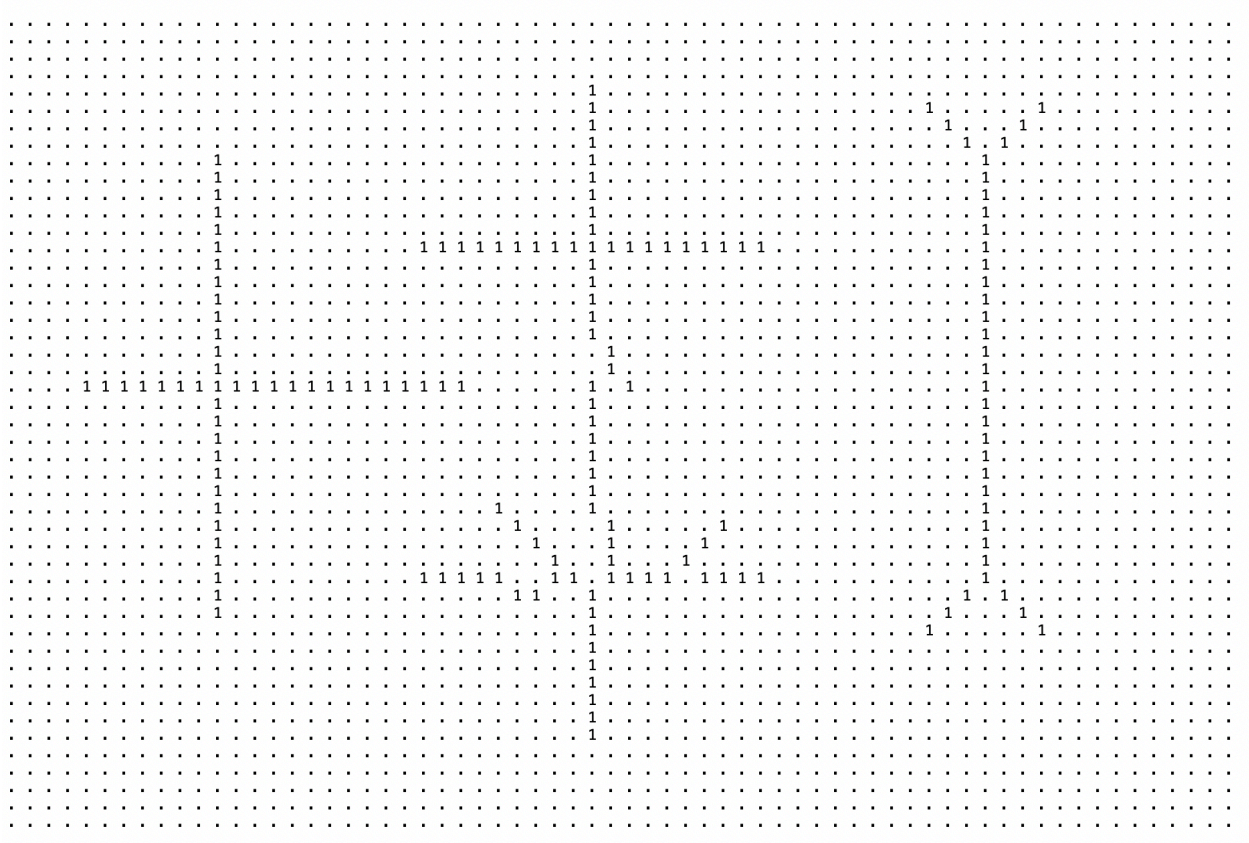
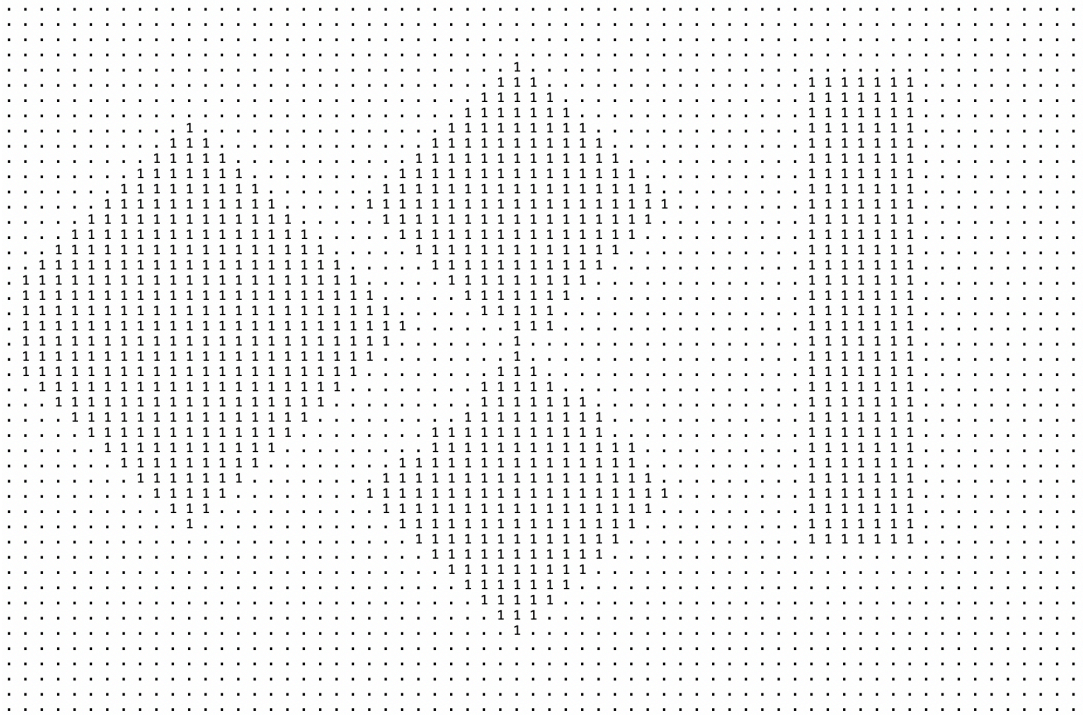
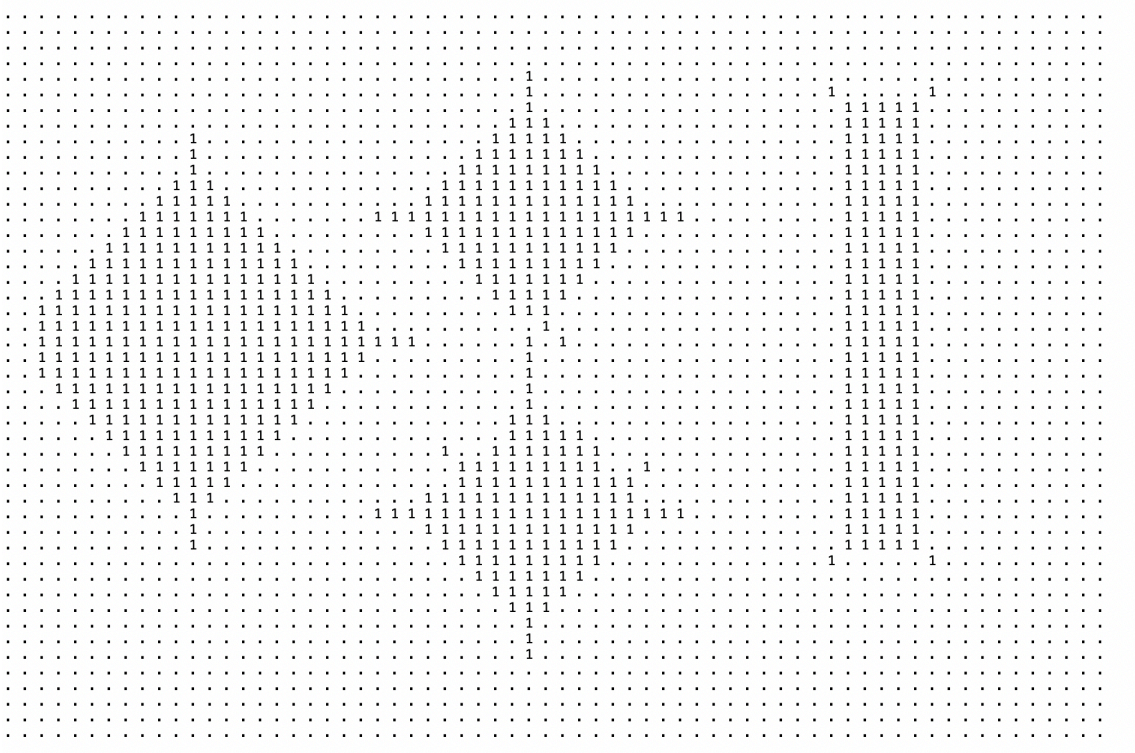


Image2-outFile2

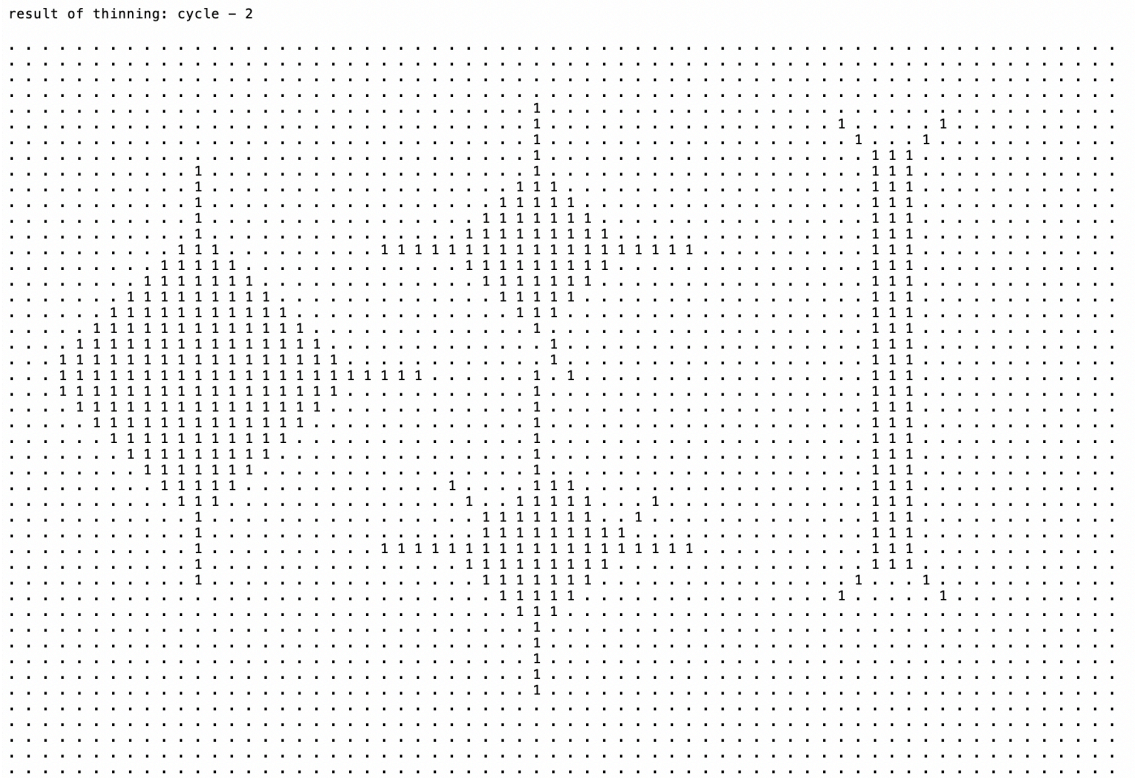
input image:



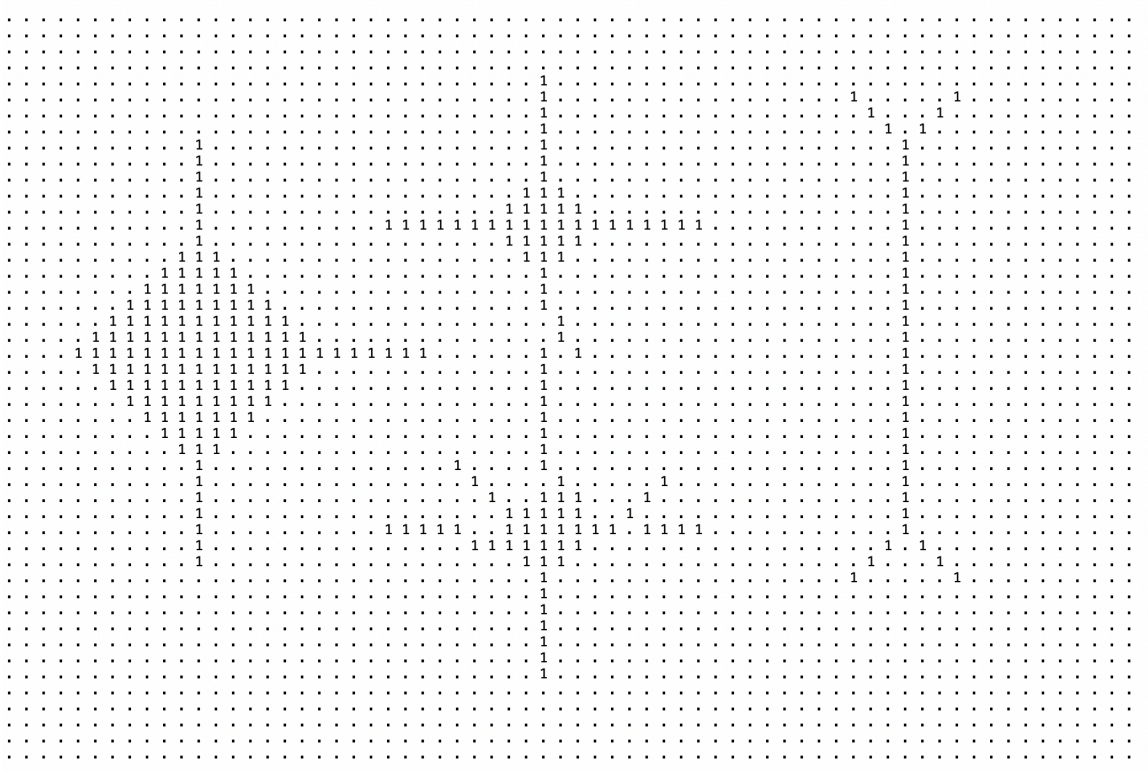
result of thinning: cycle – 1



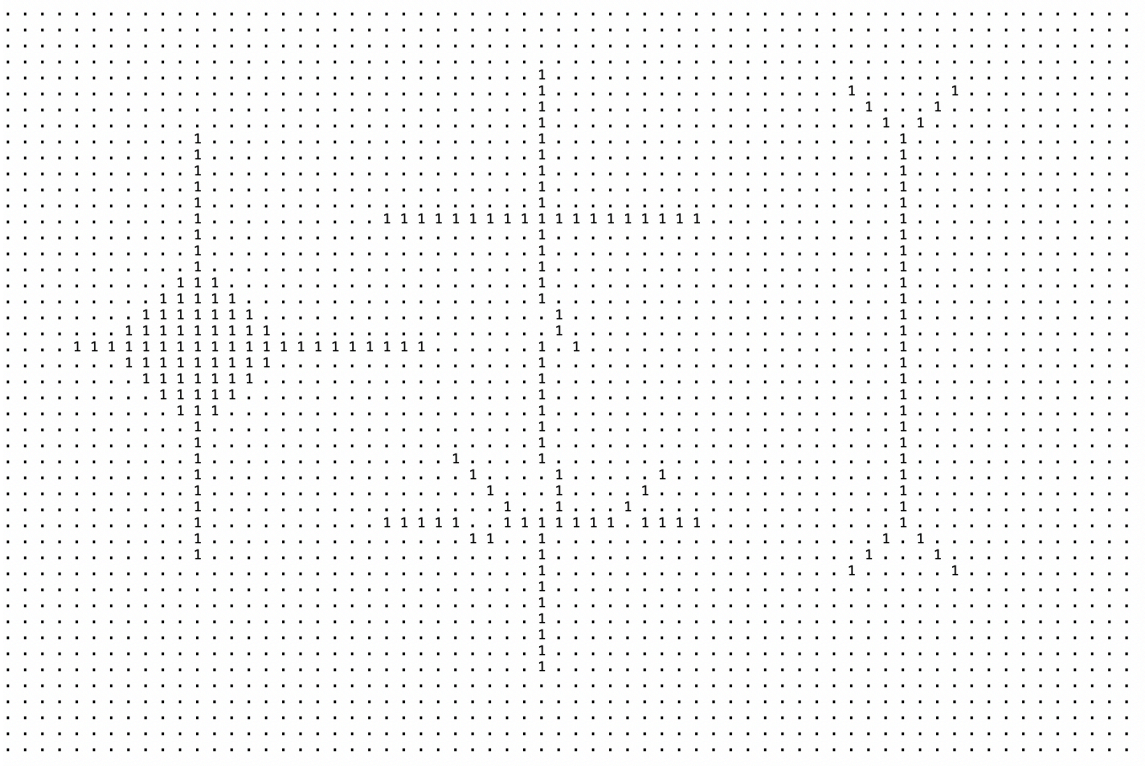
result of thinning: cycle – 2



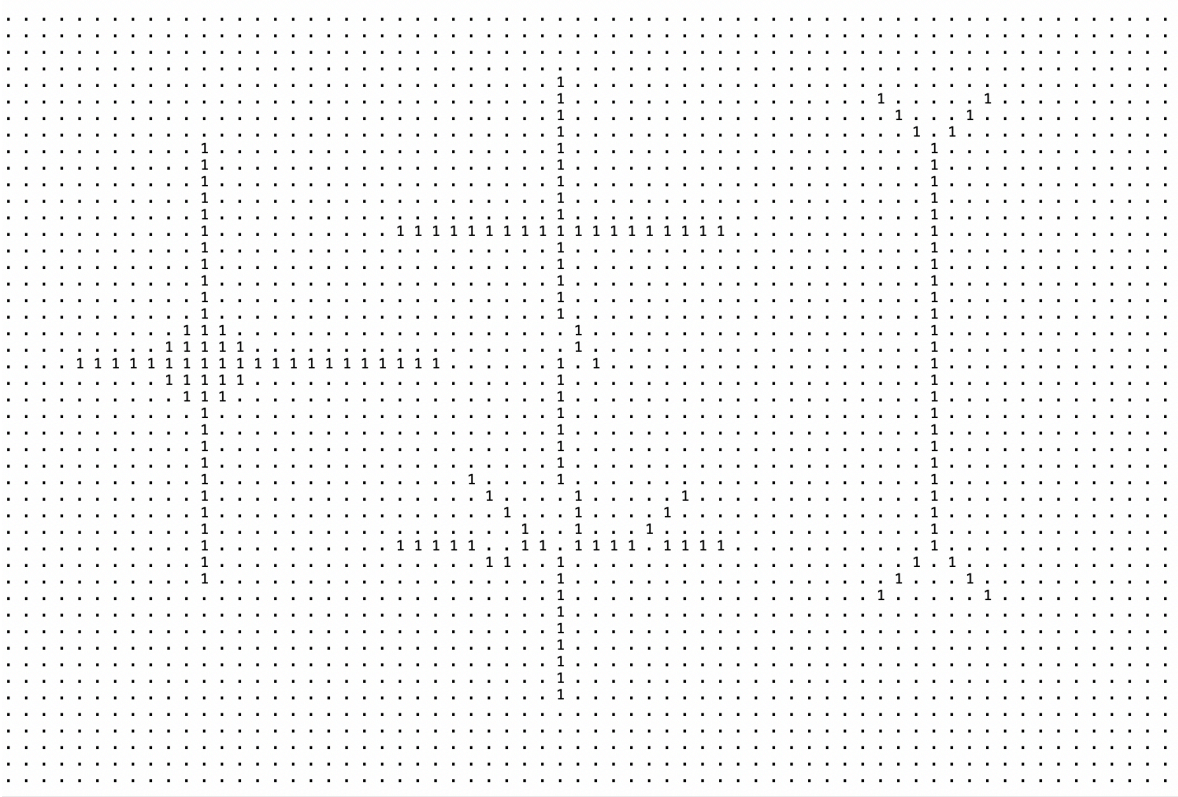
result of thinning: cycle = 3



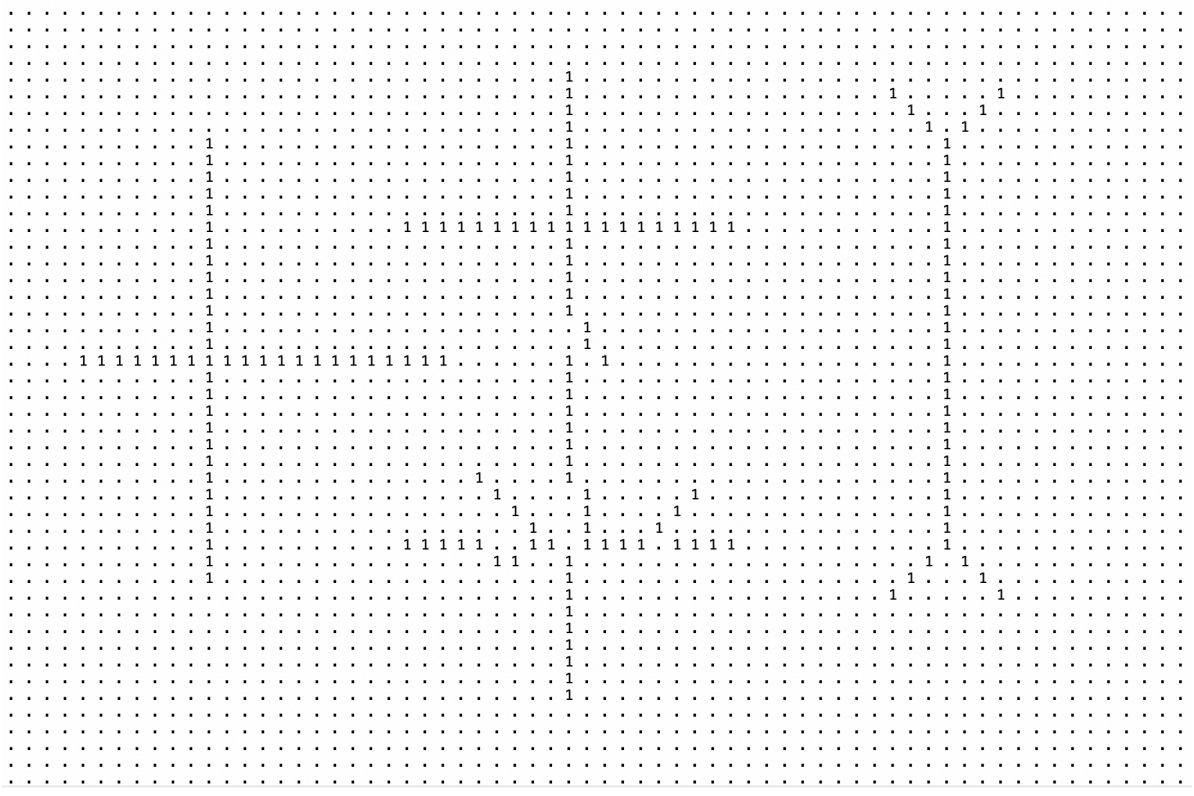
result of thinning: cycle = 4



result of thinning: cycle = 5



result of thinning: cycle = 6



result of thinning: cycle = 7