Computer Vision

Project 6: Thinning

Student: Andrew Alleyne

Algorithm Steps for Thinning

1. Scan image left to right top to bottom.
2. If the current pixel's value is greater than zero. Change its value to 0 under these conditions.
   a. Direction (North, South, East, and West) pixel is zero.
   b. The current pixel has at least 4 object neighbors
   c. The current pixel is not a connector.
   d. Repeat until all pixels are processed.

Source Code

```
/* Andrew Alleyne

Project 6 (C++): Thinning

31/03/21

 */


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

using namespace std;

class Thinning
{

private:
    int numRows;
    int numCols;
    int minVal;
    int maxVal;

public:
    int changeFlag = 0;

public:
    Thinning(int numRows, int numCols, int minVal, int maxVal)
    {
        this->numRows = numRows;
        this->numCols = numCols;
        this->minVal = minVal;
        this->maxVal = maxVal;
    }

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

    void loadImage(fstream &ifs, int **aryOne)
    {
        int data;
        for (int i = 1; i <= numRows; i++)
        {
            for (int j = 1; j <= numCols; j++)
            {
                if (ifs >> data)
                {
                    aryOne[i][j] = data;
                }
            }
        }
    }
```

```
    void reformatPrettyPrint(int **aryOne, ofstream &ofs2)
    {

        for (int i = 0; i <= numRows; i++)
        {
            for (int j = 0; j <= numCols; j++)
            {
                if (aryOne[i][j] > 0)
                {
                    ofs2 << aryOne[i][j] << "  ";
                }
                else
                {
                    ofs2 << ".  ";
                }
            }
            ofs2 << endl;
        }
    }

    void NorthThinning(int **aryOne, int **aryTwo)
    {

        int N = 1;
        for (int i = 1; i <= numRows; i++)
        {
            for (int j = 1; j <= numCols; j++)
            {

                if (aryOne[i][j] > 0)
                {
                    if (neighborZero(aryOne, i, j, N) && fourObjectNeigh(aryOne, i, j, 4) &&
noConnector(aryOne, i, j))
                    {
                        aryTwo[i][j ] = 0;
                    }
                    else
                    {
                        aryTwo[i][j] = 1;
                        changeFlag = 0;
                    }
                }
            }
        }
        copyArray(aryOne, aryTwo);
    }



    void SouthThinning(int **aryOne, int **aryTwo)
    {
        int S = 2;
        for (int i = 1; i <= numRows; i++)
        {
            for (int j = 1; j <= numCols; j++)
            {
                if (aryOne[i][j] > 0)
                {
                    if (neighborZero(aryOne, i, j, S) && fourObjectNeigh(aryOne, i, j, 4) &&
noConnector(aryOne, i, j))
```

```
                    {

                        aryTwo[i][j] = 0;
                        changeFlag++;
                    }
                    else
                    {
                        aryTwo[i][j] = 1;
                    }
                }
            }
        }
        copyArray(aryOne, aryTwo);
    }

    void EastThinning(int **aryOne, int **aryTwo)
    {
        int E = 4;
        for (int i = 1; i <= numRows; i++)
        {
            for (int j = 1; j <= numCols; j++)
            {
                if (aryOne[i][j] > 0)
                {
                    if (neighborZero(aryOne, i, j, E) && fourObjectNeigh(aryOne, i, j, 3) &&
noConnector(aryOne, i, j))
                    {

                        aryTwo[i][j] = 0;

                    }
                    else
                    {
                        aryTwo[i][j] = 1;




                    }
                }
            }
        }
        copyArray(aryOne, aryTwo);
    }


    void WestThinning(int **aryOne, int **aryTwo)
    {
        int W = 3;
        for (int i = 1; i <= numRows; i++)
        {
            for (int j = 1; j <= numCols; j++)
            {
                if (aryOne[i][j] > 0)
                {
                    if (neighborZero(aryOne, i, j, W) && fourObjectNeigh(aryOne, i, j, 3) &&
noConnector(aryOne, i, j))
                    {

                        aryTwo[i][j] = 0;
                        changeFlag++;
```

```
                }
                else
                {
                    aryTwo[i][j] = 1;

                }
            }
        }
    }
    copyArray(aryOne, aryTwo);
}

void copyArray(int **aryOne, int **aryTwo)
{
    for (int i = 1; i <= numRows; i++)
    {
        for (int j = 1; j <= numCols; j++)
        {
            aryOne[i][j] = aryTwo[i][j];
        }
    }
}

bool neighborZero(int **aryOne, int i, int j, int direction)
{
    bool status = false;

    if (direction == 1)
    {
        if (aryOne[i - 1][j] == 0)
        {
            status = true;
        }
    }

    if (direction == 2)
    {
        if (aryOne[i + 1][j] == 0)
        {
            status = true;
        }
    }

    if (direction == 3)
    {
        if (aryOne[i][j - 1] == 0)
        {
            status = true;
        }else{ status = false;}
    }

    if (direction == 4)
    {
        if (aryOne[i][j + 1] == 0)
        {
            status = true;
        }
    }

    return status;
}
```

```
//Has at least 4 objects neighbors.
bool fourObjectNeigh(int **aryOne, int i, int j, int direction)
{
    int neigh[8];
    bool status = false;


    neigh[0] = aryOne[i - 1][j - 1];
    neigh[1] = aryOne[i - 1][j];
    neigh[2] = aryOne[i - 1][j + 1];

    neigh[3] = aryOne[i][j - 1];
    neigh[4] = aryOne[i][j + 1];

    neigh[5] = aryOne[i + 1][j - 1];
    neigh[6] = aryOne[i + 1][j];
    neigh[7] = aryOne[i + 1][j + 1];

    if (direction == 4)
    {
        int count;
        count = 0;
        for (int m = 0; m < 8; m++)
        {
            if (neigh[m] == 1)
            {
                count++;
            }
        }
        if (count >= 4)
        {
            status = true;
        }
    }

    if (direction == 3)
    {
        int count;
        count = 0;
        for (int m = 0; m < 8; m++)
        {
            if (neigh[m] == 1)
            {
                count++;
            }
        }
        if (count >= 3)
        {
            status = true;
        }
    }

    return status;
}
//Check for connectedness
bool noConnector(int **aryOne, int i, int j)
{
    bool status = true;

    int topLeft = aryOne[i - 1][j - 1];
    int top = aryOne[i - 1][j];
```

```
        int topRight = aryOne[i - 1][j + 1];

        int left = aryOne[i][j - 1];
        int right = aryOne[i][j + 1];

        int bottomLeft = aryOne[i + 1][j - 1];
        int bottom = aryOne[i + 1][j];
        int bottomRight = aryOne[i + 1][j + 1];

        if (left == 0 && right == 0)
        {
            if (topLeft == 1 || top == 1 || topRight == 1 && bottom == 1 || bottomLeft == 1 ||
bottomRight == 1)
            {
                status = false;
            }
        }
        if (top == 0 && bottom == 0)
        {
            if (topLeft == 1 || left == 1 || bottomLeft == 1 && topRight == 1 || right == 1 ||
bottomRight == 1)
            {
                status = false;
            }
        }

        if (top == 0 && left == 0 && topLeft == 1)
        {
            status = false;
        }

        if (bottom == 0 && left == 0 && bottomLeft == 1)
        {
            status = false;
        }

        if (top == 0 && right == 0 && topRight == 1)
        {
            status = false;
        }

        if (bottom == 0 && right == 0 && bottomRight == 1)
        {
            status = false;
        }

        return status;
    }

    bool Cflag()
    {
        bool status = false;

        if( changeFlag > 0){
            status = true;
        }
    return status;
    }
};

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

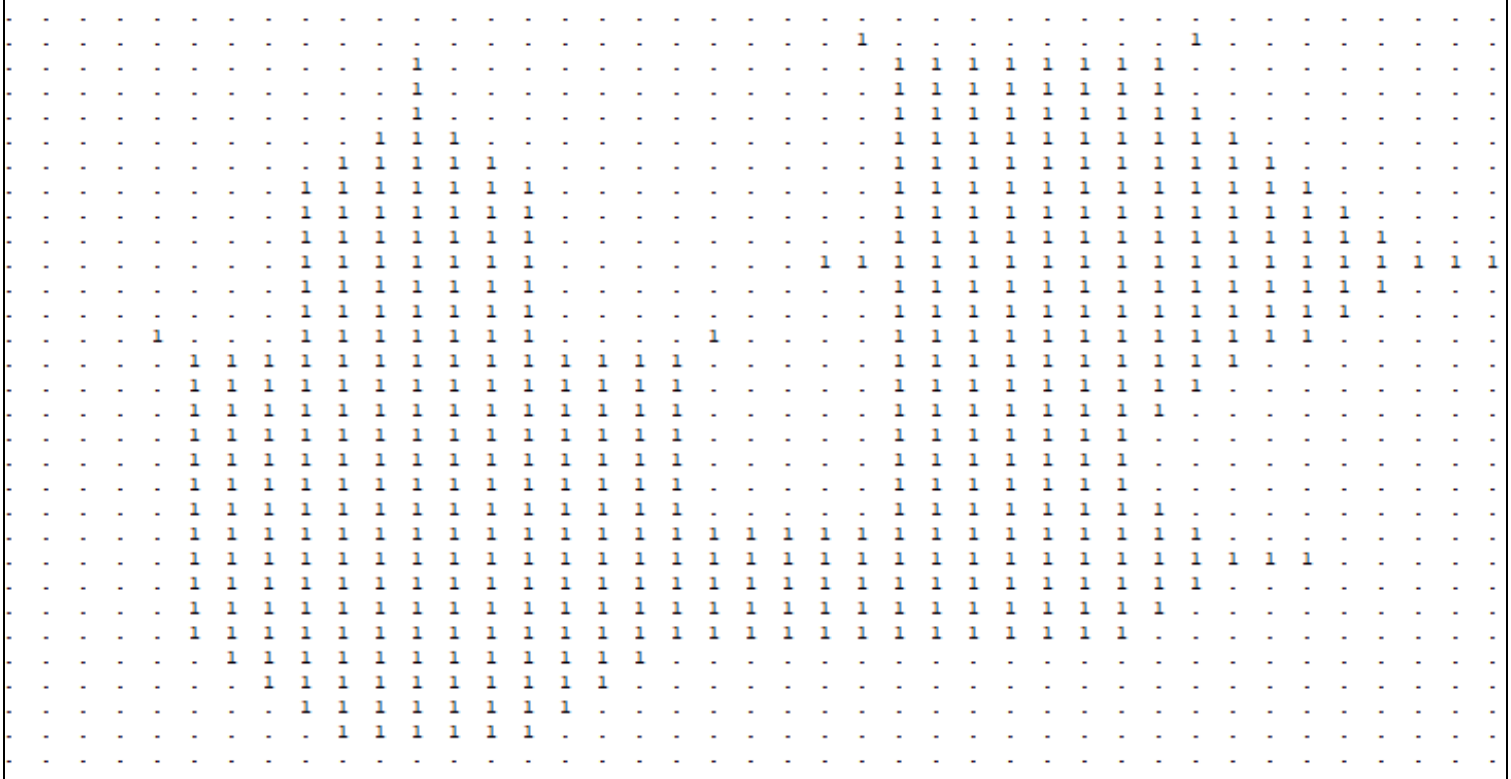
```
{

    if (argc < 3)
    {
        cout << "More arguments are needed!" << endl;
    }

    string filename = argv[1];
    string oFname = argv[2];
    string oFname2 = argv[3];

    int numRows;
    int numCols;
    int minVal;
    int maxVal;
    int cycleCount = 0;
    int **aryOne;
    int **aryTwo;

    fstream inputFile;
    inputFile.open(filename);

    if (inputFile)
    {
        inputFile >> numRows >> numCols >> minVal >> maxVal;
    }

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

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

    ofstream ofs;
    ofs.open(argv[2]);

    ofstream ofs2;
    ofs2.open(argv[3]);

    Thinning thin(numRows, numCols, minVal, maxVal);

    thin.zeroFrame(aryOne);
    thin.zeroFrame(aryTwo);
    thin.loadImage(inputFile, aryOne);

    ofs << " Original Image PrettyPrinted: " << endl;
    thin.reformatPrettyPrint(aryOne, ofs);
    ofs << endl;

    bool changes = 0;

    do    {



        thin.NorthThinning(aryOne, aryTwo);
        ofs << endl;
```

```
        thin.SouthThinning(aryOne, aryTwo);
        ofs << endl;



        thin.WestThinning(aryOne, aryTwo);
        ofs << endl;

        thin.EastThinning(aryOne, aryTwo);
        thin.reformatPrettyPrint(aryOne, ofs);
        ofs << endl;



        changes = thin.Cflag();


        cycleCount++;
        cout << " Changes : " << changes << endl;
        ofs << " Results of thinning cycle - " << cycleCount << endl;
        ofs << endl;
    }while(changes > 0);

    ofs2 << numRows << " " << numCols << " " << minVal  << " " << maxVal << endl;
    thin.reformatPrettyPrint(aryOne, ofs2);

inputFile.close();
osf.close();
osf2.close();

}
```

Code Output

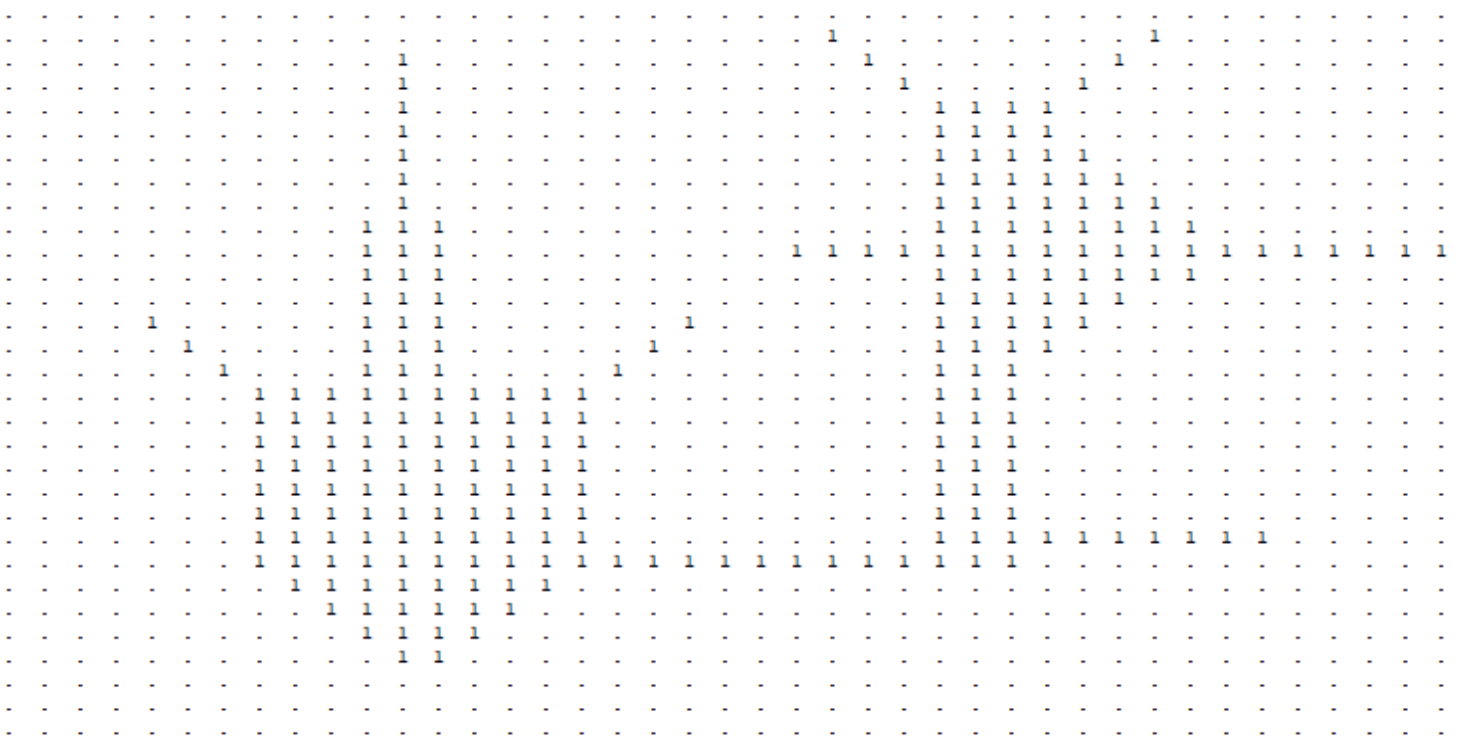
For image 1

Original Image PrettyPrinted:



Results of thinning cycle = 1

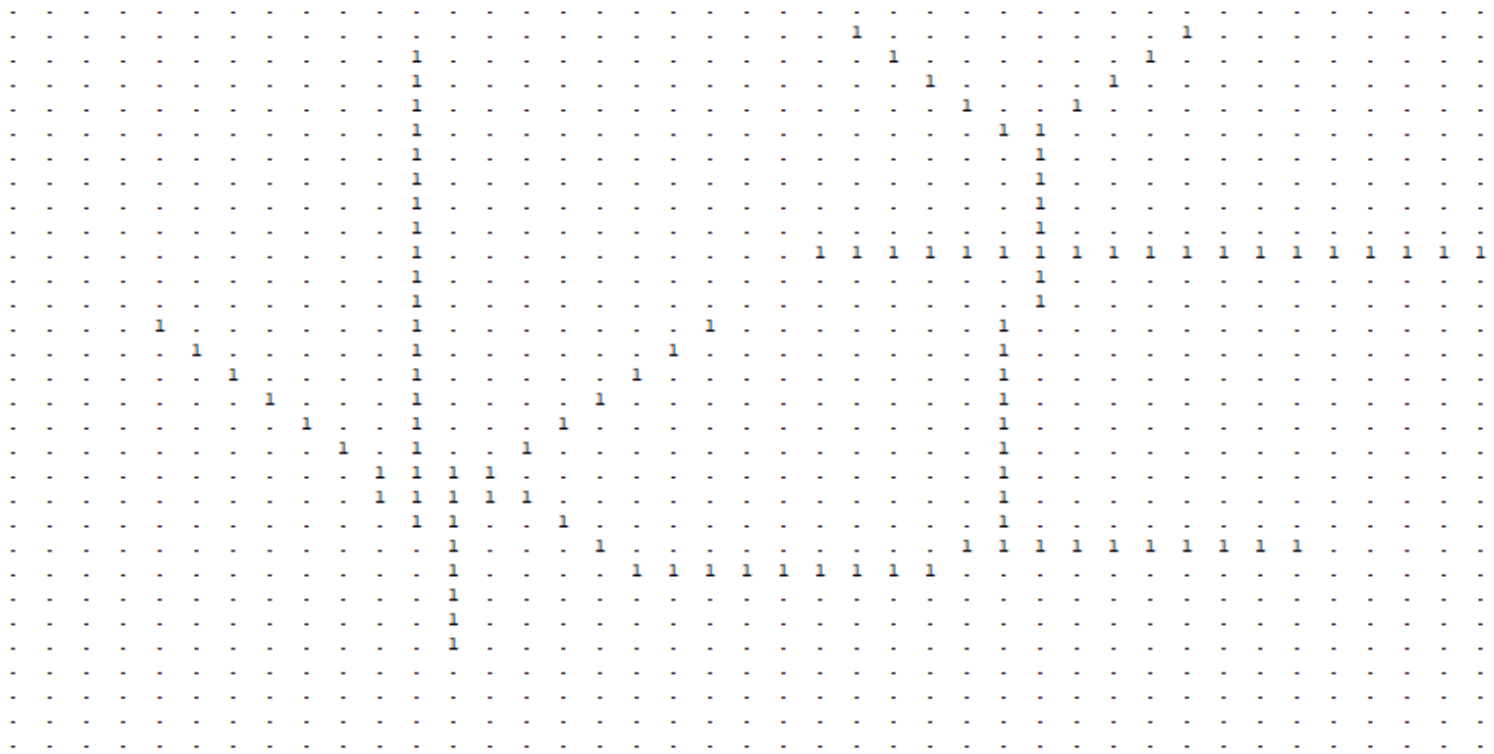
Results of thinning cycle - 2



Results of thinning cycle - 3

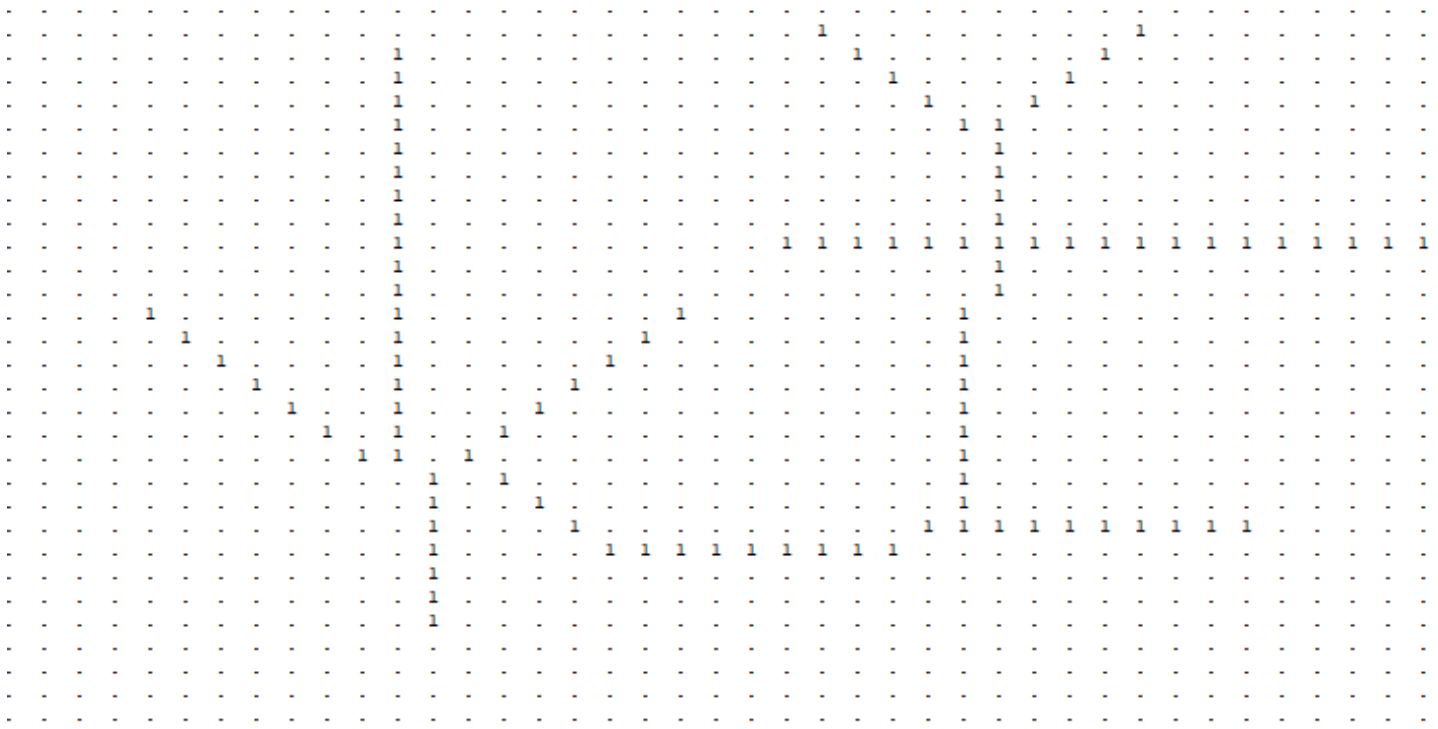
Results of thinning cycle - 4

Results of thinning cycle - 5



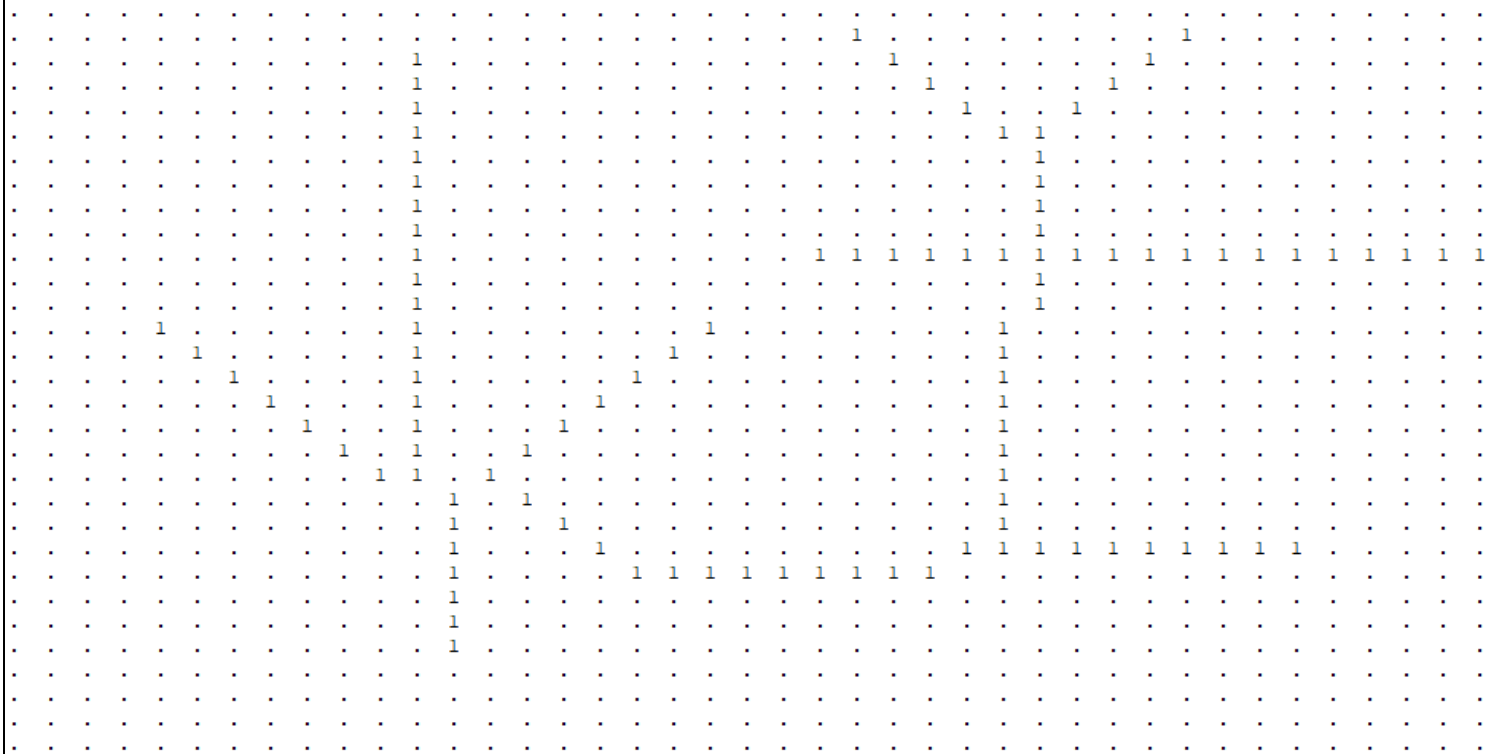
Results of thinning cycle - 6

Results of thinning cycle - 7



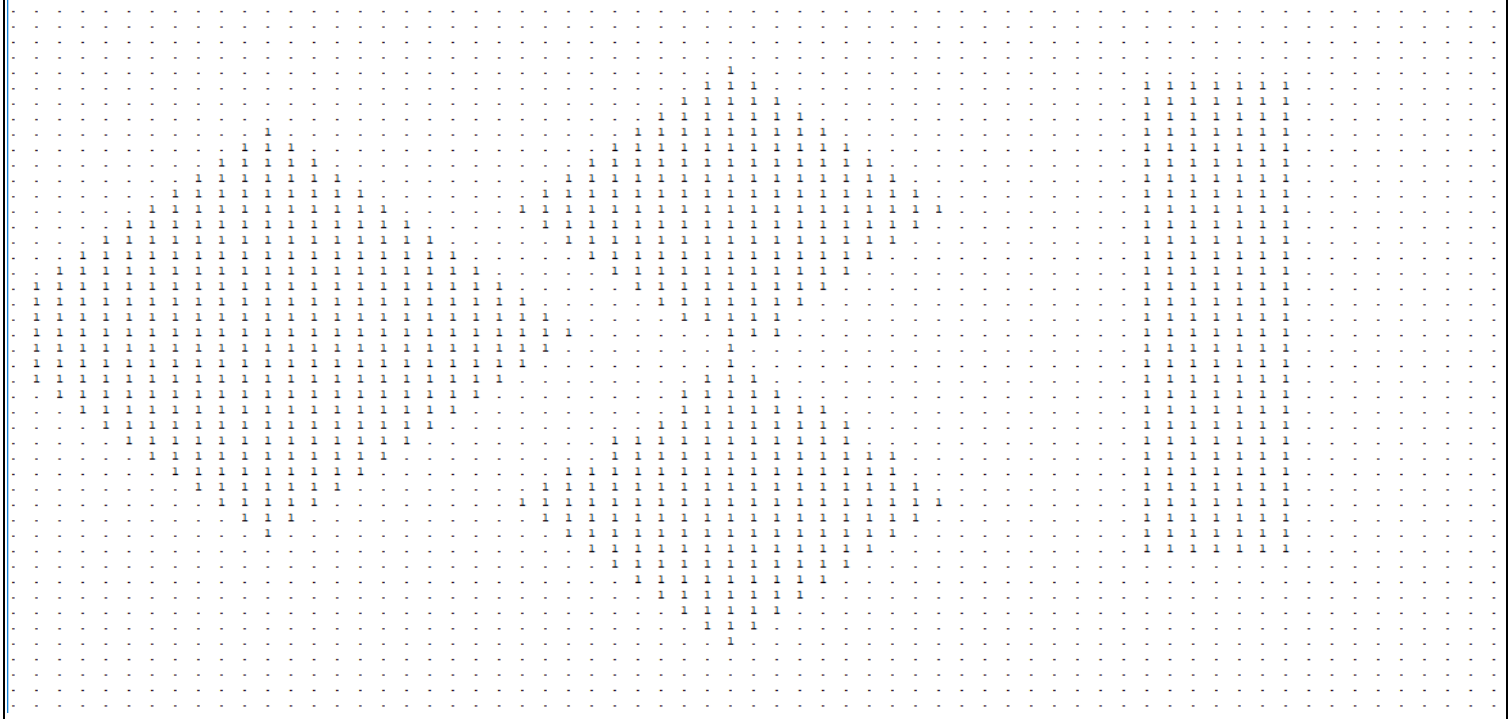
Results of thinning cycle = 8

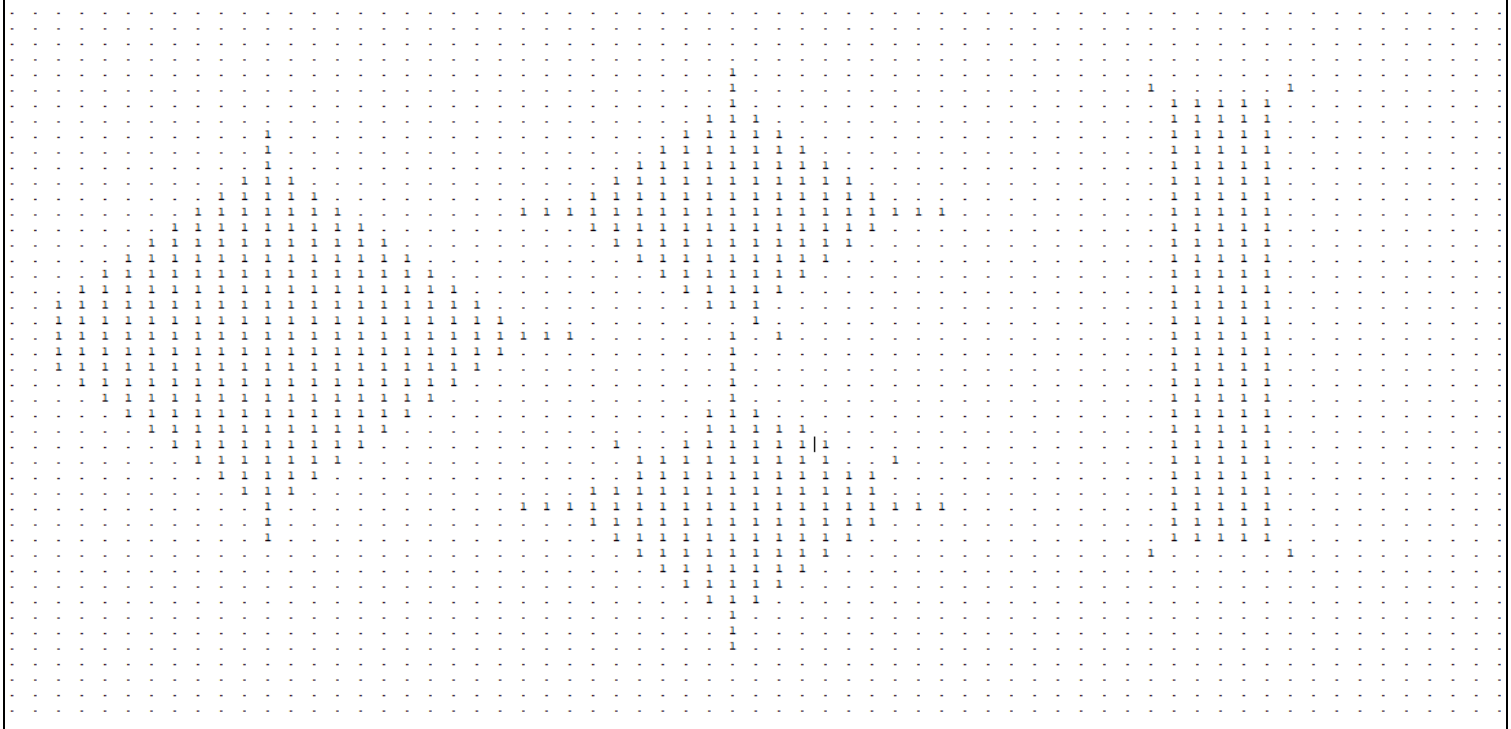
30 40 0 1



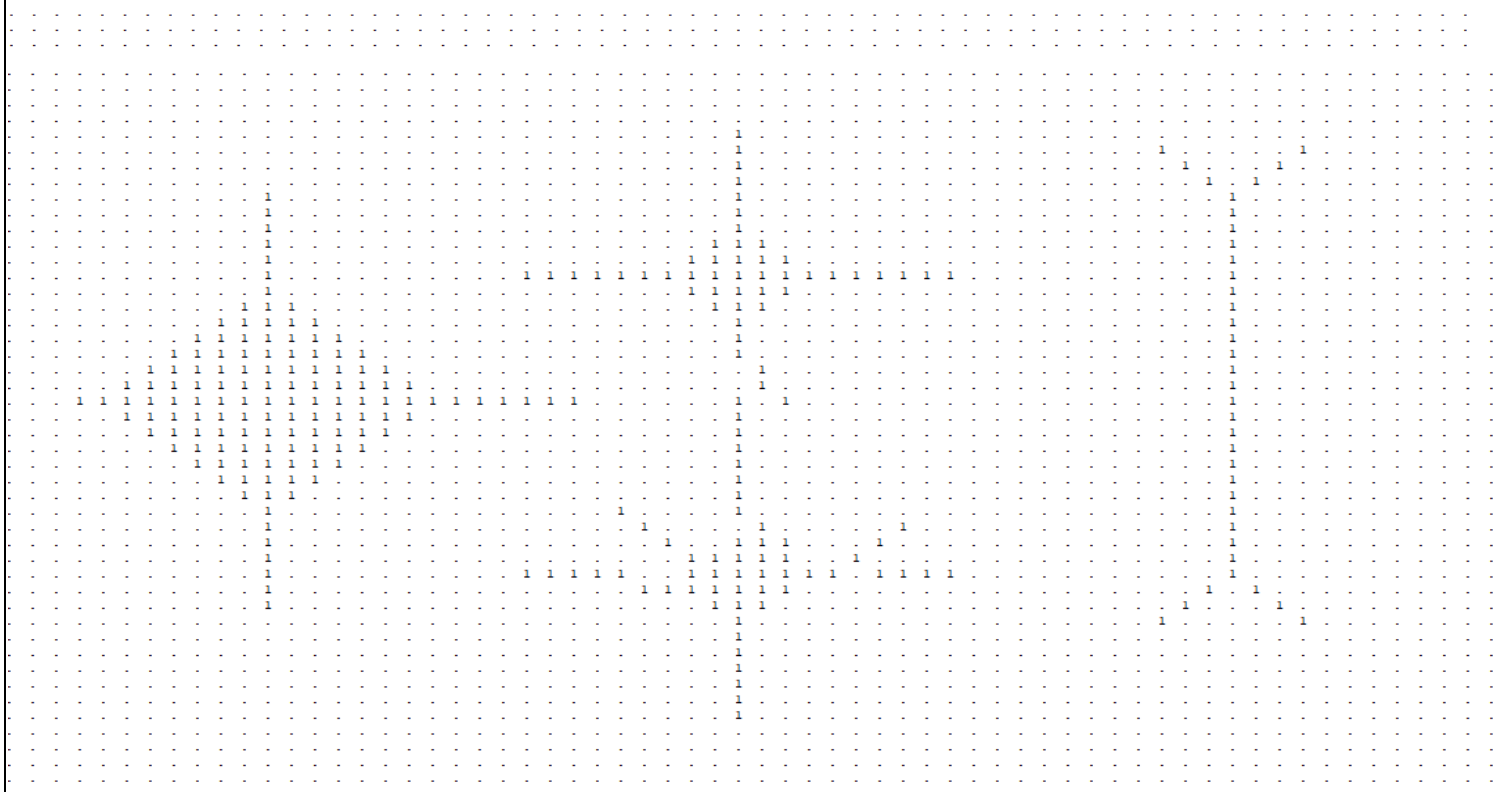
Code Output for image 2

Original Image PrettyPrinted:

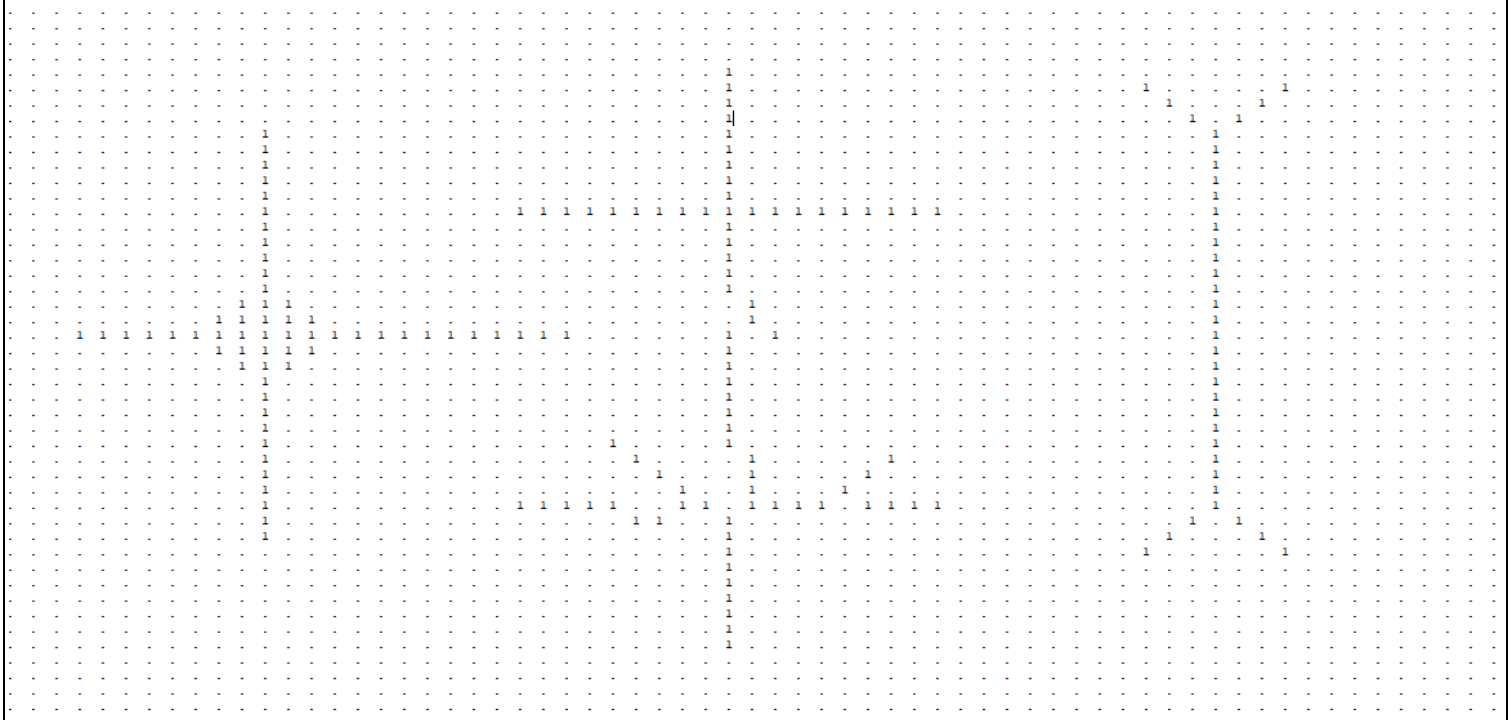




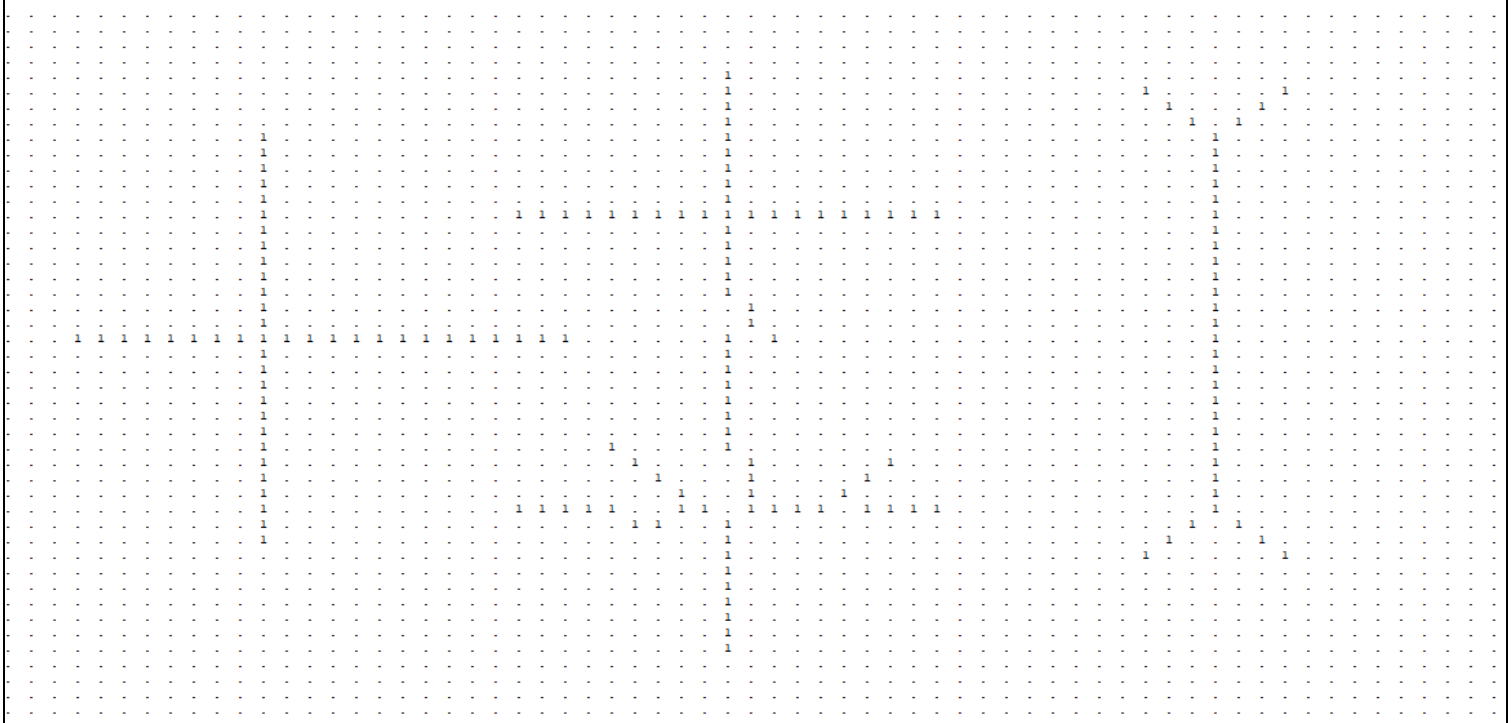
Results of thinning cycle = 1



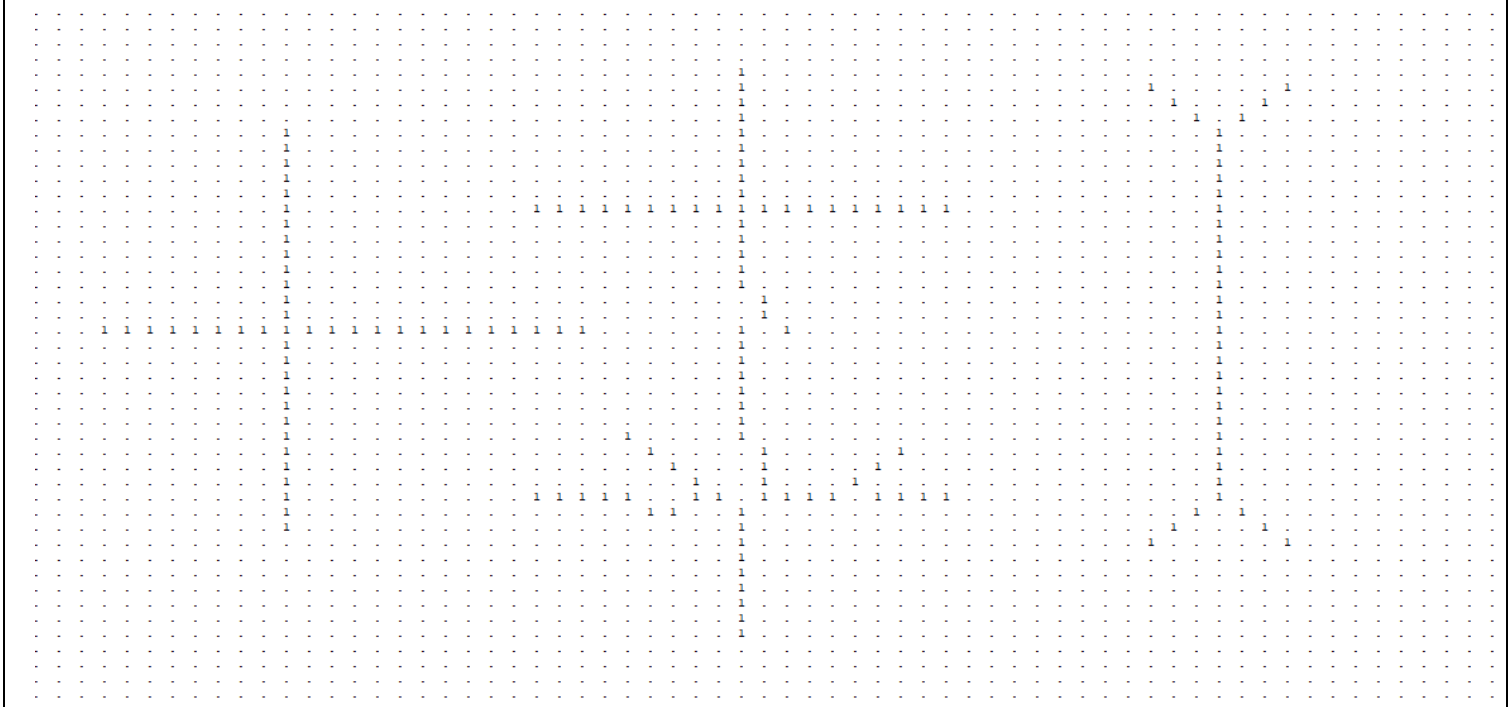
Results of thinning cycle = 3



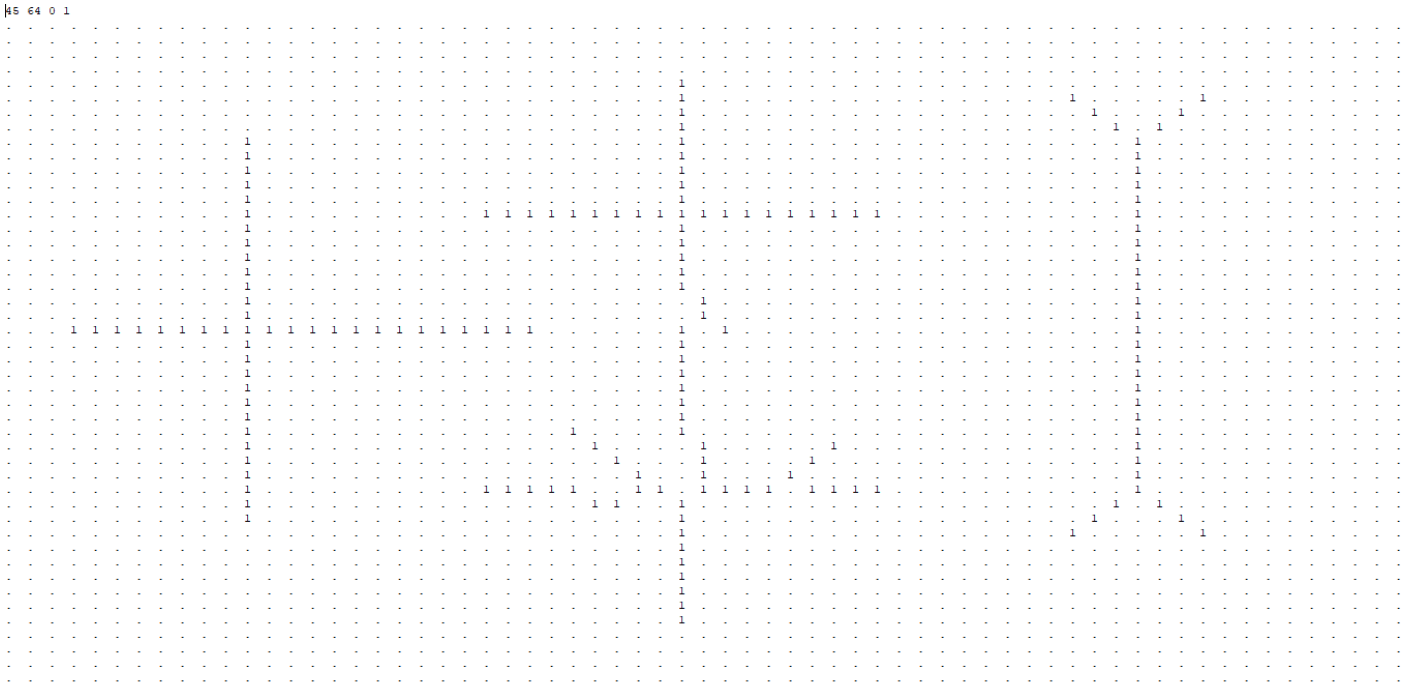
Results of thinning cycle - 5



Results of thinning cycle = 6



Results of thinning cycle = 7


45 64 0 1