Cover Sheet

CV Project 6: Thinning C++

Student: Fengzhang Du

Project Due Date: 04/09/2021

Algorithm Steps for Thinning:

```
step 0: inFile <- open input file from argv[1]
        numRows, numCols, minVal, maxVal <- read from inFile
        outFile1 <- open from argv [2]
        outFile2 <- open from argv [3]
        outFile1 <- write numRows, numCols, minVal, maxVal
        dynamically allocate aryOne of size numRows + 2 by numCols + 2.
        dynamically allocate aryTwo of size numRows + 2 by numCols + 2.
step 1: zeroFrame(aryOne)
        zeroFrame(aryTwo)
step 2: loadImage (inFile, aryOne)
step 3: cycleCount <- 0
step 4: reformatPrettyPrint (aryOne, outFile2)

step 5: changeFlag <- 0
step 6: NorthThinning (aryOne, aryTwo)
        copyArys ()
step 7: SouthThinning (aryOne, aryTwo)
        copyArys()
step 8: WestThinning (aryOne, aryTwo)
        copyArys()
step 9: EastThinning (aryOne, aryTwo)
        copyArys()
step 10: cycleCount ++
Step 11: reformatPrettyPrint (aryOne, outFile2)
Step 12: repeat step 5 to step 11 while changeFlag > 0

step 13: outFile1 <- output inside frame of aryOne from [1][1]
step 14: close all files
```

Source Code

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdlib>
using namespace std;
```

```
class Thinning{
  public:
  int numRows, numCols, minVal, maxVal, changeflag, cycleCount;
  int** aryOne;
  int** aryTwo;

  // consructor
  public:
  Thinning(ifstream &input){
      read_header(input);
      init2D(aryOne, 2);
      init2D(aryTwo, 2);
  }

  // methods
  void read_header(ifstream &input){
      input >> numRows >> numCols >> minVal >> maxVal;
  }

  void write_header(ofstream &w){
      w << numRows<< " " << numCols<< " " << minVal << " " << maxVal << endl;
  }

  // take cares of zeroFrame. p = 2, extra columns or rows
  void init2D(int**& arr, int p){
      arr = new int*[numRows+p];
      for (int i=0; i<numRows+p; i++){
          arr[i] = new int[numCols+p];
          for (int j=0; j<numCols+p; j++){
              arr[i][j] = 0;
          }
      }
  }

  void free_heap(){
      for (int i=0; i<numRows+2; i++){
          delete[] aryOne[i];
          delete[] aryTwo[i];
      }
      delete[] aryOne;
      delete[] aryTwo;
      cout << "Heap freed!" << endl;
  }

  void loadImage(ifstream &input){
      for (int i=1; i<=numRows; i++){
          for (int j=1; j<=numCols; j++){
              input >> aryOne[i][j];
          }
      }
  }
```

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

bool exceedNeighbors(int i, int j, int max){
    int count = 0;
    for(int k=i-1; k<=i+1; k++){
        for(int d=j-1; d<=j+1; d++){
            if(k == i && d == j) continue;
            if(count >= max) return true;
            if(aryOne[k][d]>0) count++;
        }
    }
    return false;
}

bool isConnector(int i, int j){
    // check for 6 cases
    int L = aryOne[i][j-1];
    int R = aryOne[i][j+1];
    int T = aryOne[i-1][j];
    int B = aryOne[i+1][j];
    int TL = aryOne[i-1][j-1];
    int TR = aryOne[i-1][j+1];
    int BL = aryOne[i+1][j-1];
    int BR = aryOne[i+1][j+1];
    // case 1
    if(L==0 && R==0){
        if((T==1 || TL==1 || TR==1) && (B==1 || BL==1 || BR==1)) return true;
    }
    // case 2
    if(T==0 && B==0){
        if((TL==1 || L==1 || BL==1) && (TR==1 || R==1 || BR==1)) return true;
    }
    // case alpha
    if(T==0 && L==0 && TL==1) return true;
    // case beta
    if(L==0 && B==0 && BL==1) return true;
    // case gamma
    if(T==0 && R==0 && TR==1) return true;
    // case delta
    if(R==0 && B==0 && BR==1) return true;

    return false;
}
```

```
void NorthThinning(){
    for (int i=1; i<=numRows; i++){
        for (int j=1; j<=numCols; j++){
            if(aryOne[i][j] > 0){
                // check 3 conditions
                if(aryOne[i-1][j]==0 && exceedNeighbors(i, j, 4) && !isConnector(i,j)){
                    aryTwo[i][j] = 0;
                    changeflag++;
                }else{
                    aryTwo[i][j] = 1;
                }
            }
        }
    }
    copyArys();
}

void SouthThinning(){
    for (int i=1; i<=numRows; i++){
        for (int j=1; j<=numCols; j++){
            if(aryOne[i][j] > 0){
                // check 3 conditions
                if(aryOne[i+1][j]==0 && exceedNeighbors(i, j, 4) && !isConnector(i,j)){
                    aryTwo[i][j] = 0;
                    changeflag++;
                }else{
                    aryTwo[i][j] = 1;
                }
            }
        }
    }
    copyArys();
}

void WestThinning(){
    for (int i=1; i<=numRows; i++){
        for (int j=1; j<=numCols; j++){
            if(aryOne[i][j] > 0){
                // check 3 conditions
                if(aryOne[i][j-1]==0 && exceedNeighbors(i, j, 3) && !isConnector(i,j)){
                    aryTwo[i][j] = 0;
                    changeflag++;
                }else{
                    aryTwo[i][j] = 1;
                }
            }
        }
    }
    copyArys();
}
```

```
    void EastThinning(){
        for (int i=1; i<=numRows; i++){
            for (int j=1; j<=numCols; j++){
                if(aryOne[i][j] > 0){
                    // check 3 conditions
                    if(aryOne[i][j+1]==0 && exceedNeighbors(i, j, 3) && !isConnector(i,j)){
                        aryTwo[i][j] = 0;
                        changeflag++;
                    }else{
                        aryTwo[i][j] = 1;
                    }
                }
            }
        }
        copyArys();
    }

    void reformatPrettyPrint(int**& arr, ofstream &w, string title){ // only print array one.

        if(title != "Final Result of Thinning: ") w << title << "Cycle - " << cycleCount << endl;
        else{
            w << title << endl;
            write_header(w);
        }
        for(int i=1; i<=numRows; i++){
            for(int j=1; j<=numCols; j++){
                if(arr[i][j] == 0){
                    w << "  ";
                }else{
                    w << arr[i][j] << " ";
                }
            }
            w << endl;
        }
        w << endl;
    }

};

int main(int argc, const char* argv[]){
    // step 0
    ifstream input;
    input.open(argv[1]);

    ofstream output1;
    output1.open(argv[2]);

    ofstream output2;
    output2.open(argv[3]);

    if (input.is_open() && output1.is_open() && output2.is_open()){
```

```
        // step 1
        Thinning* img = new Thinning(input);

        // step 2
        img->loadImage(input);

        // step 3
        img->cycleCount = 0;

        // step 4
        img->reformatPrettyPrint(img->aryOne, output2, "Image before Thinning: ");
        do{
            // step 5
            img->changeflag = 0;

            // step 6
            img->NorthThinning();

            // step 7
            img->SouthThinning();

            // step 8
            img->WestThinning();

            // step 9
            img->EastThinning();

            // step 10
            img->cycleCount++;

            // step 11
            img->reformatPrettyPrint(img->aryOne, output2, "Result of Thinning: ");

            // step 12 repeat 5-11
        }while(img->changeflag > 0);

        // step 13 -> output the final result to file 1.
        img->reformatPrettyPrint(img->aryTwo, output1, "Final Result of Thinning: ");

        img->free_heap();
    }else {
        cout << "Error: input or output file is not open!"<< endl;
    }
     // step 14
    input.close();
    output1.close();
    output2.close();
    return 0;
}
```

Program Output

image1_outFile1

```
Final Result of Thinning:
30 40 0 1
```

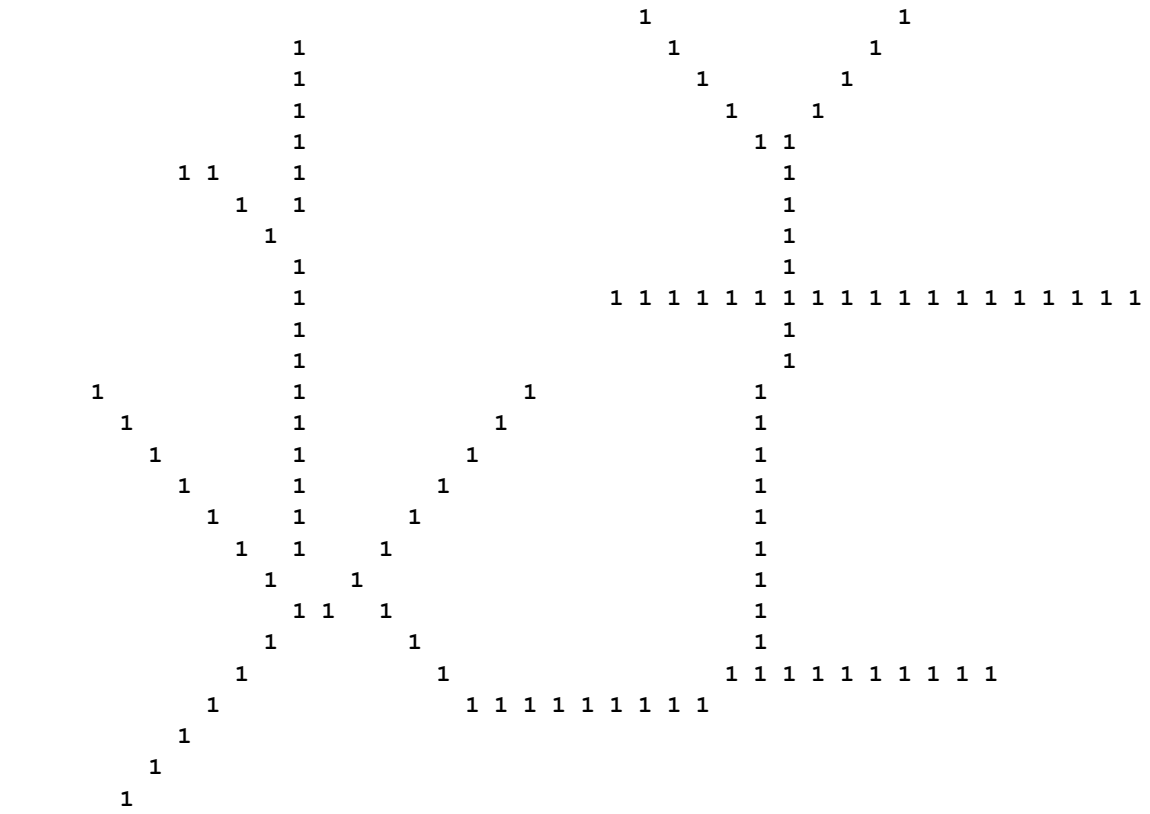
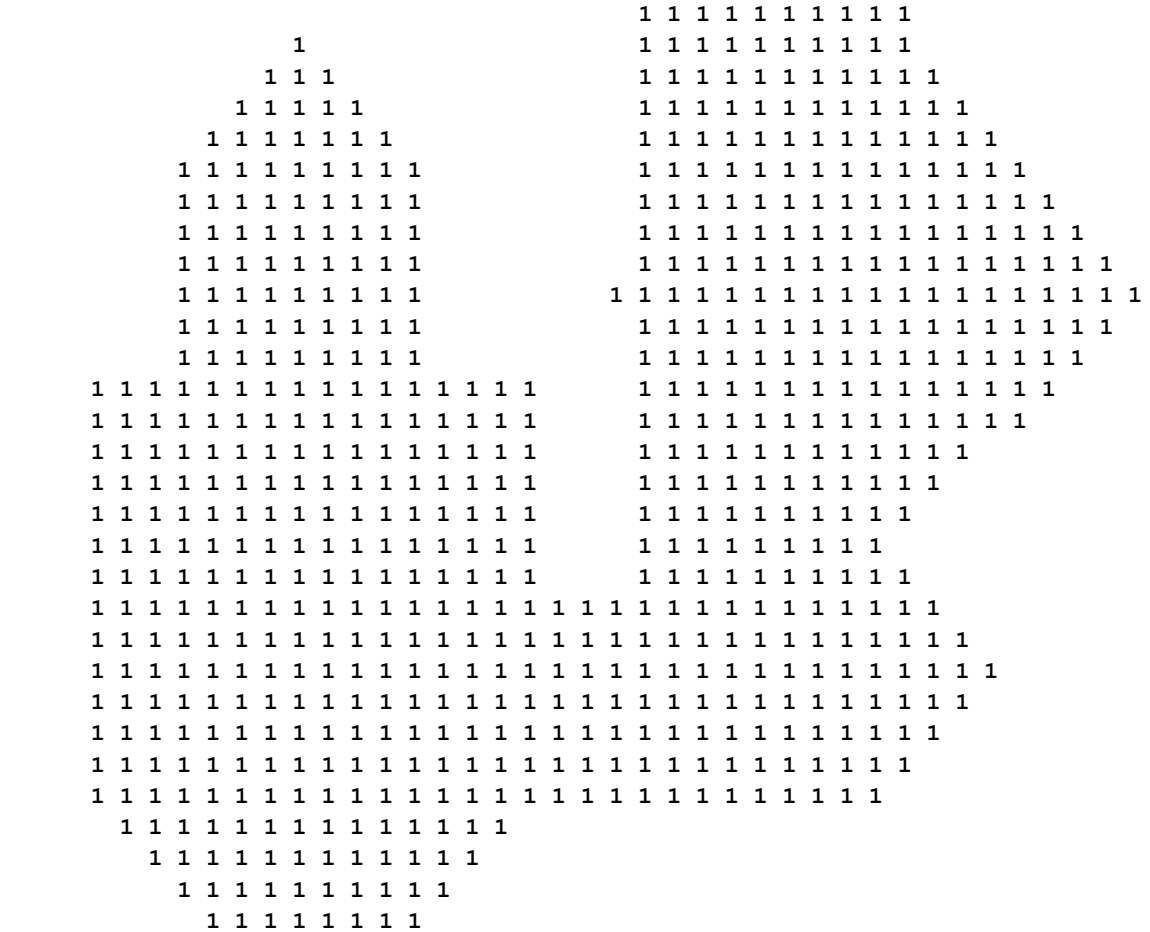
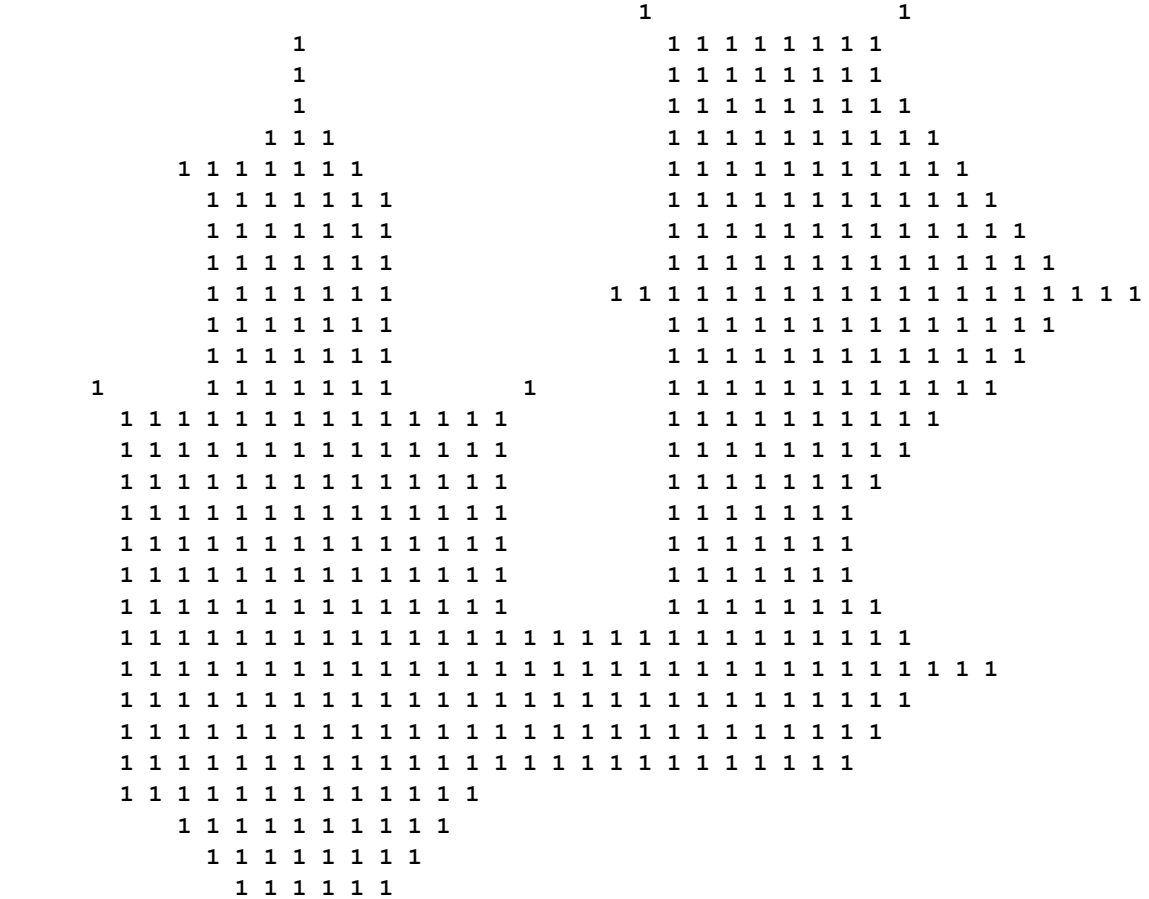


image1_outFile2

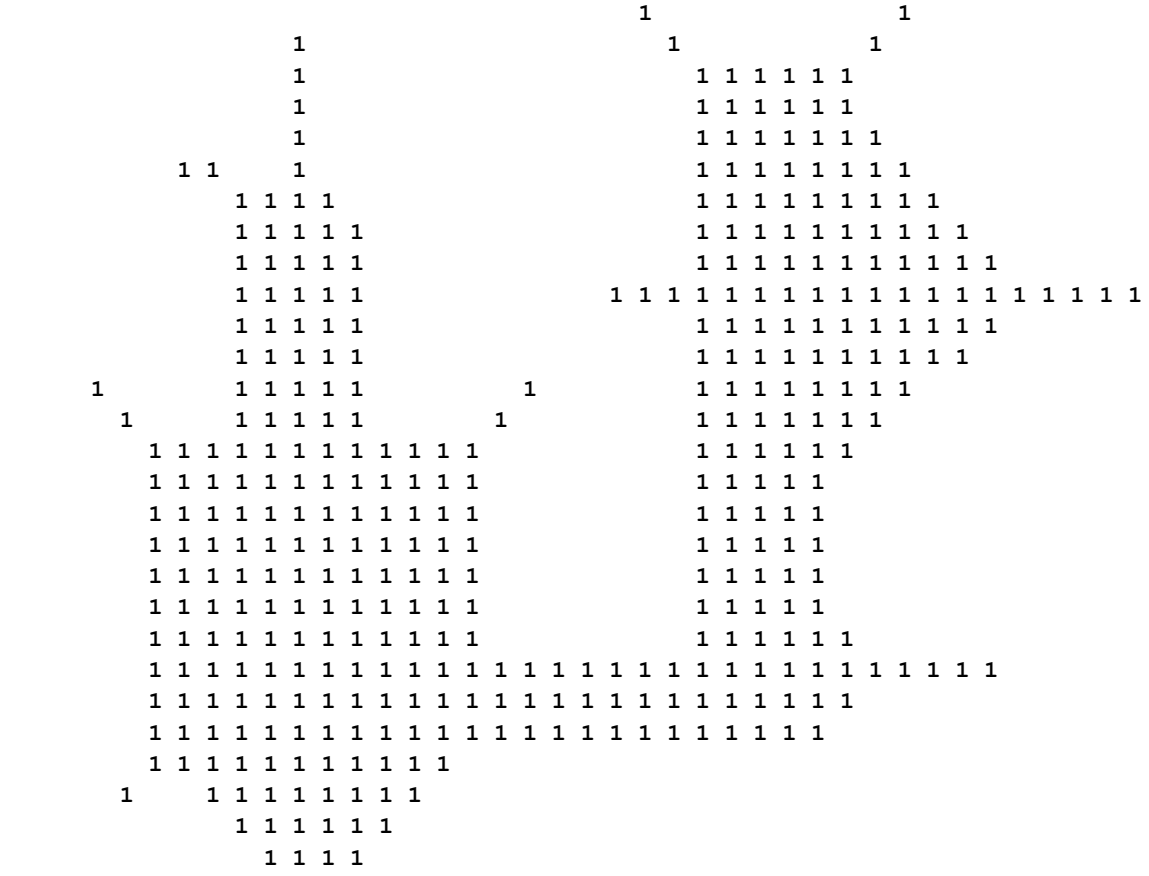
```
Image before Thinning: Cycle - 0
```



Result of Thinning: Cycle - 1

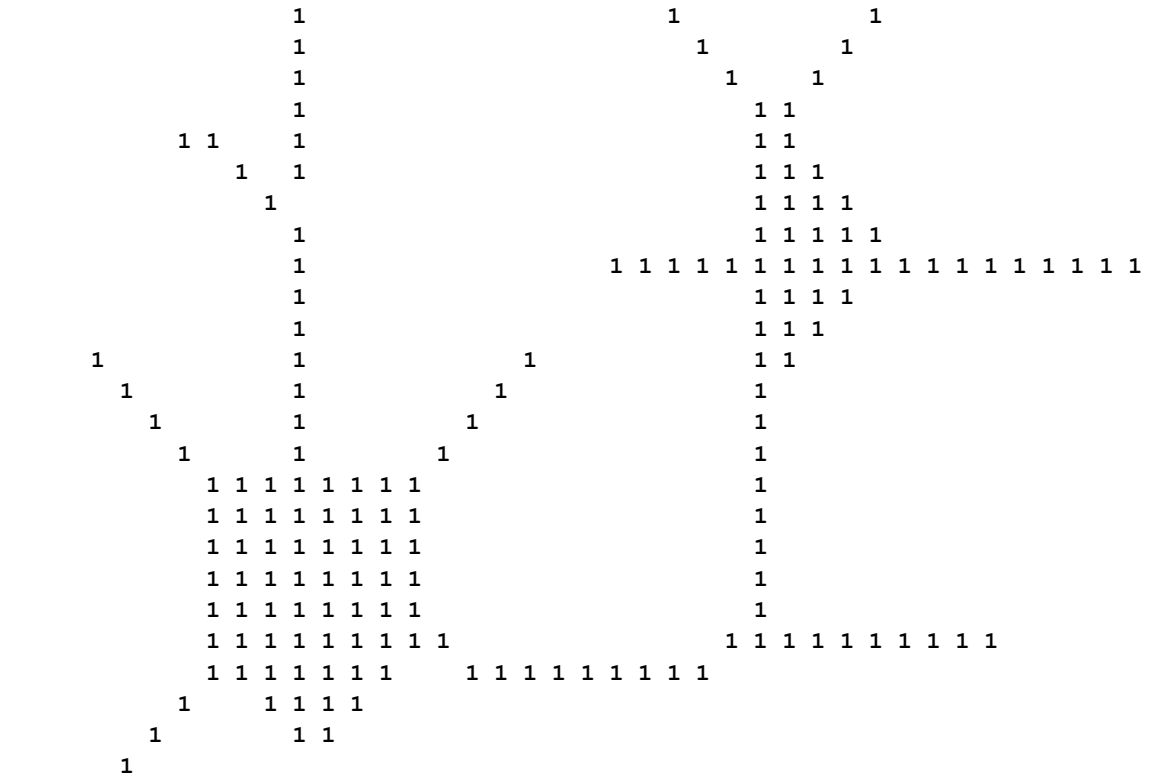


Result of Thinning: Cycle - 2

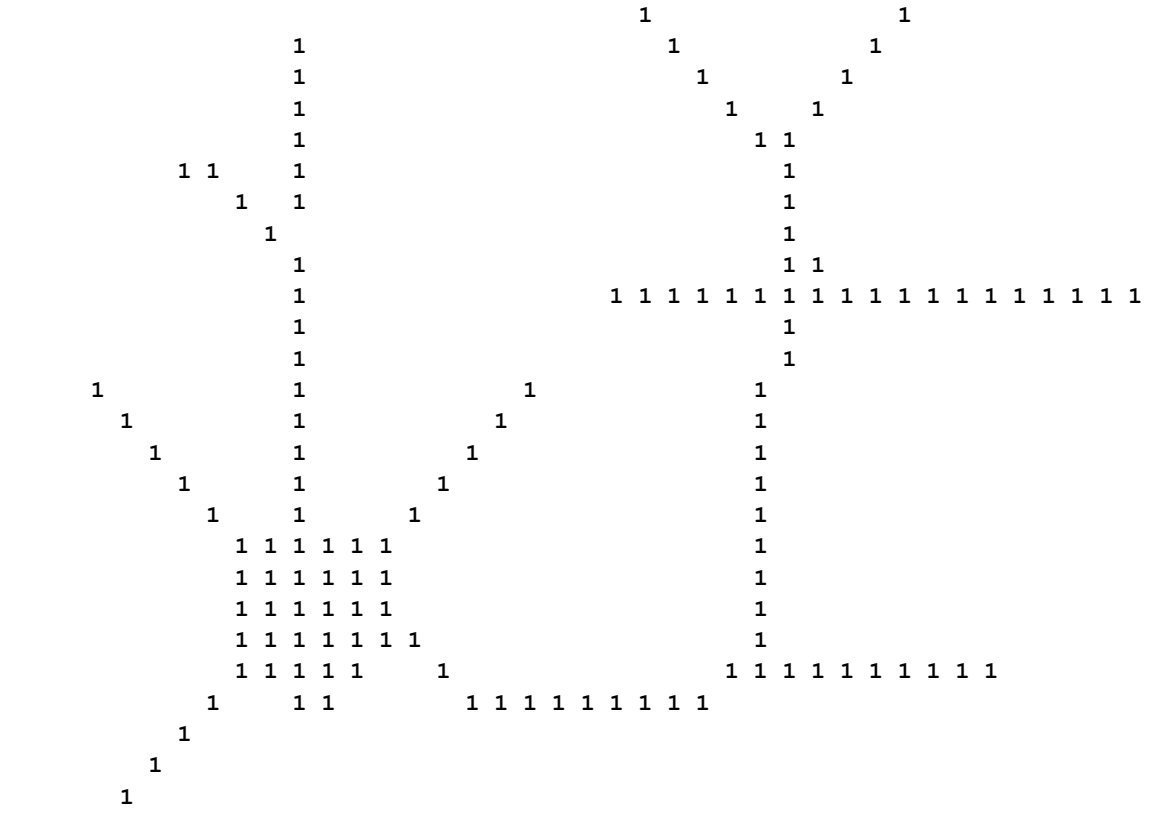


Result of Thinning: Cycle - 3

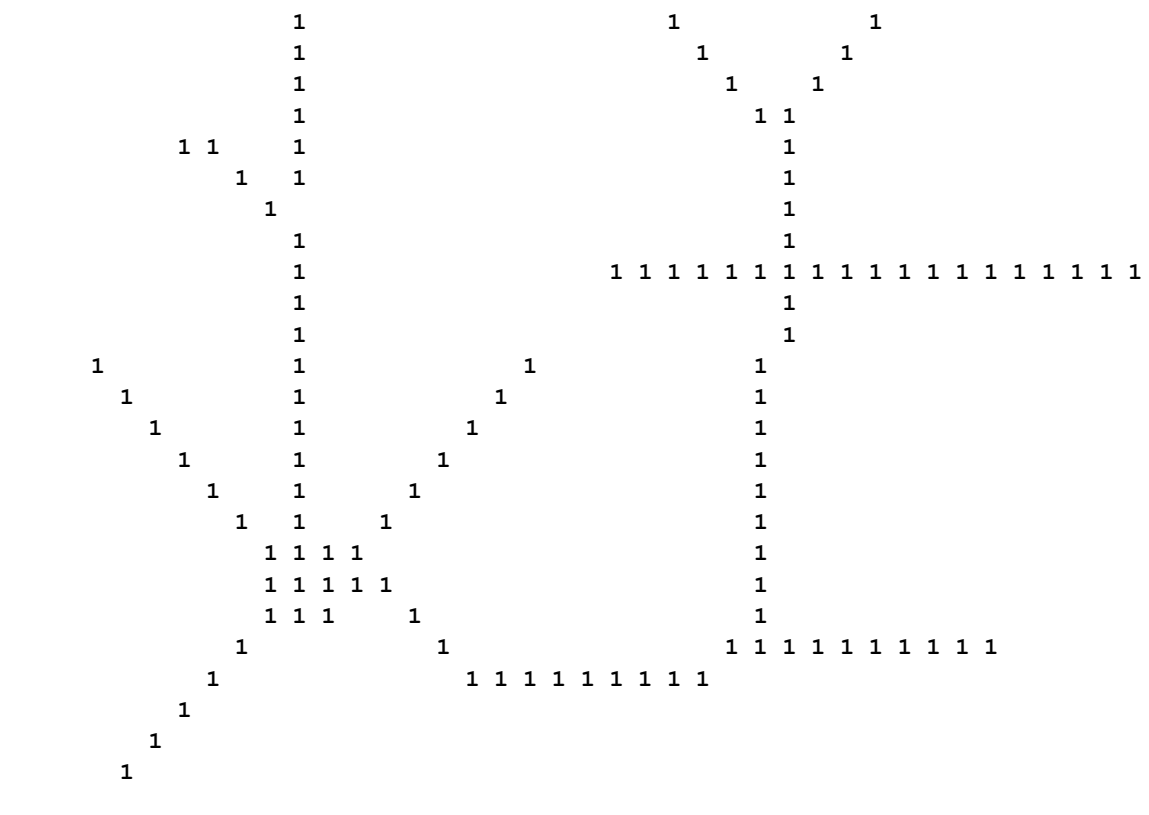
Result of Thinning: Cycle - 4



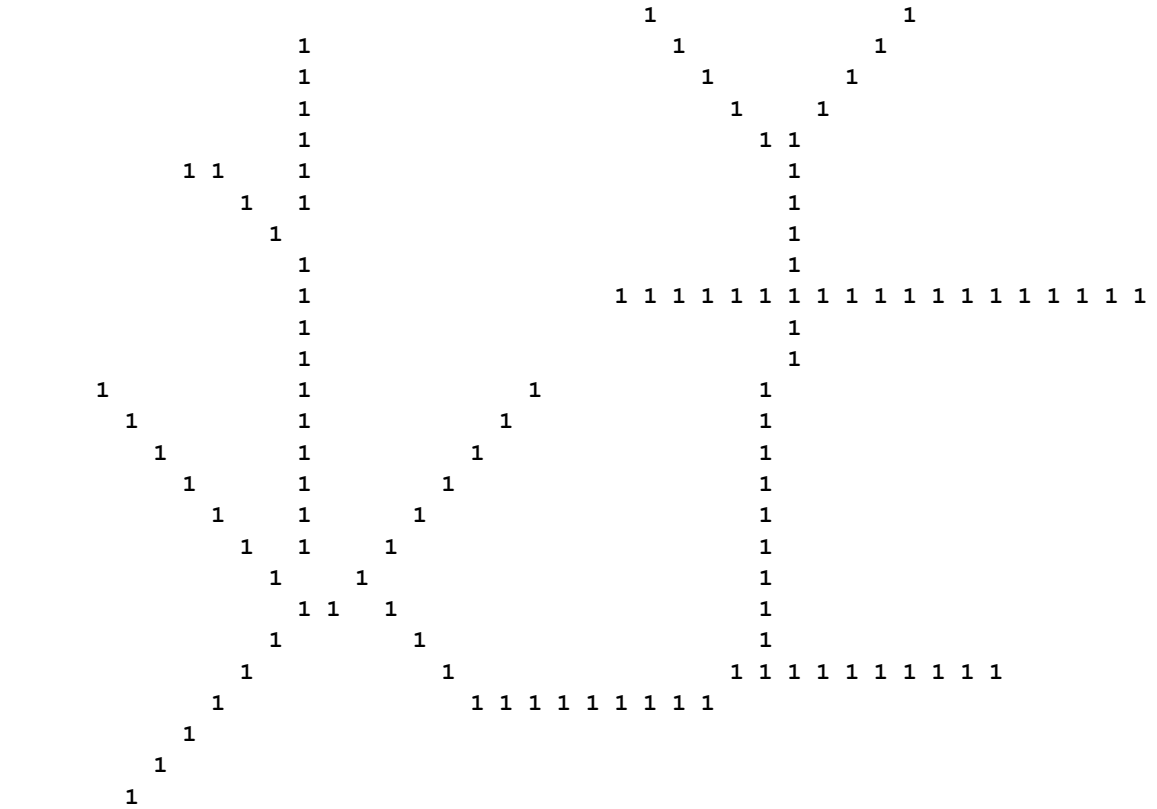
Result of Thinning: Cycle - 5



Result of Thinning: Cycle - 6



Result of Thinning: Cycle - 7



Result of Thinning: Cycle - 8

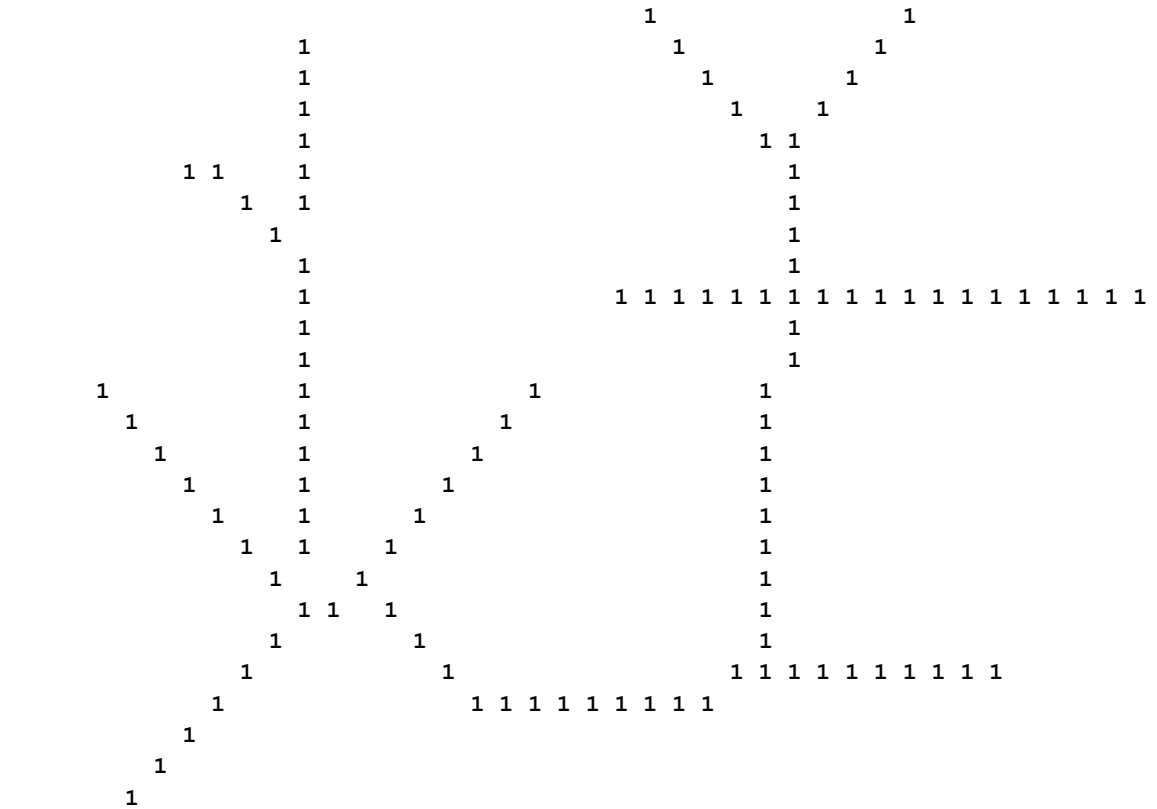


image2_outFile1

```
Final Result of Thinning:
45 64 0 1
```

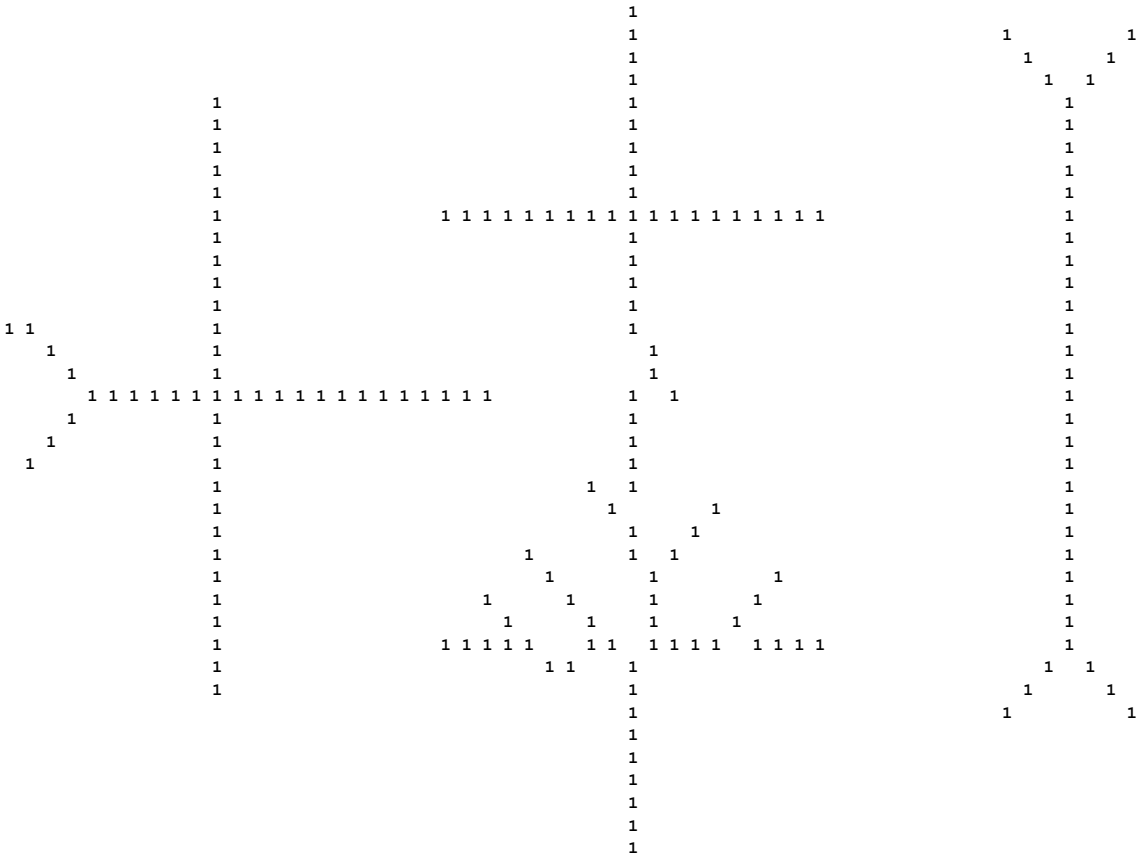
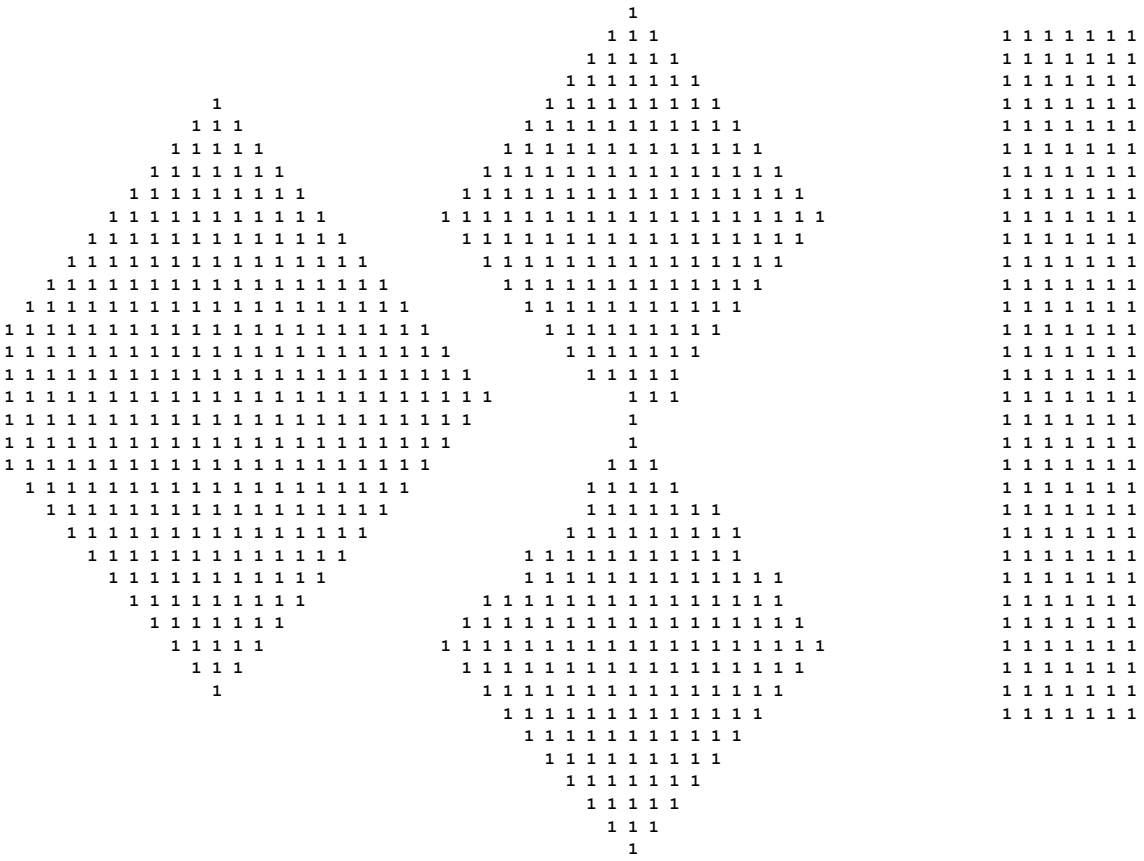
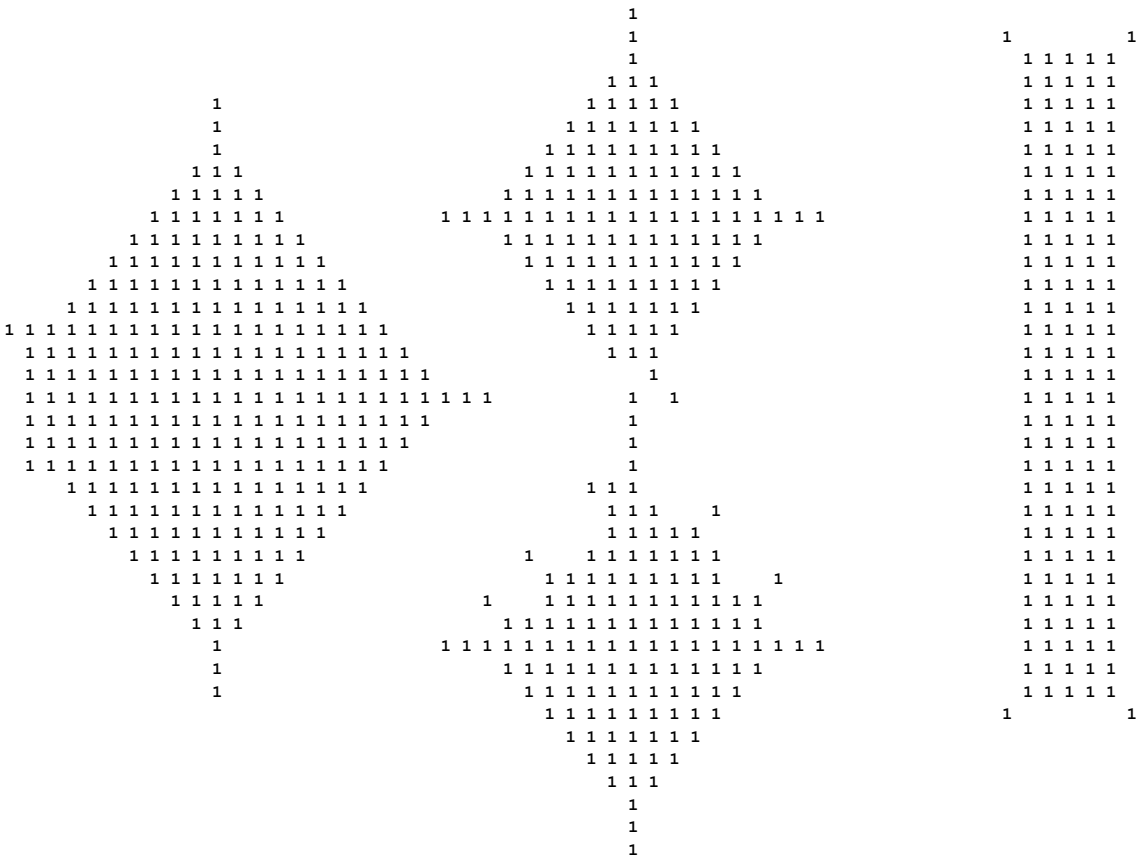


image2_outFile2

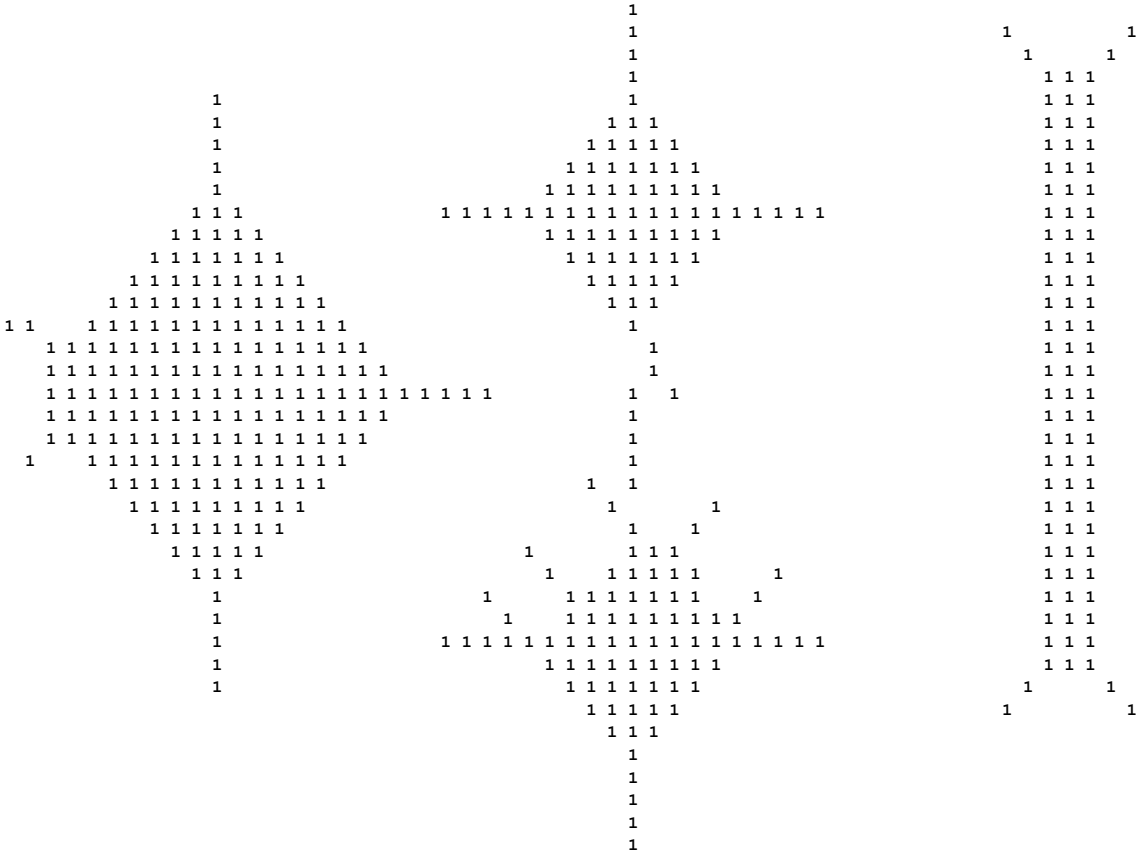
```
Image before Thinning: Cycle - 0
```



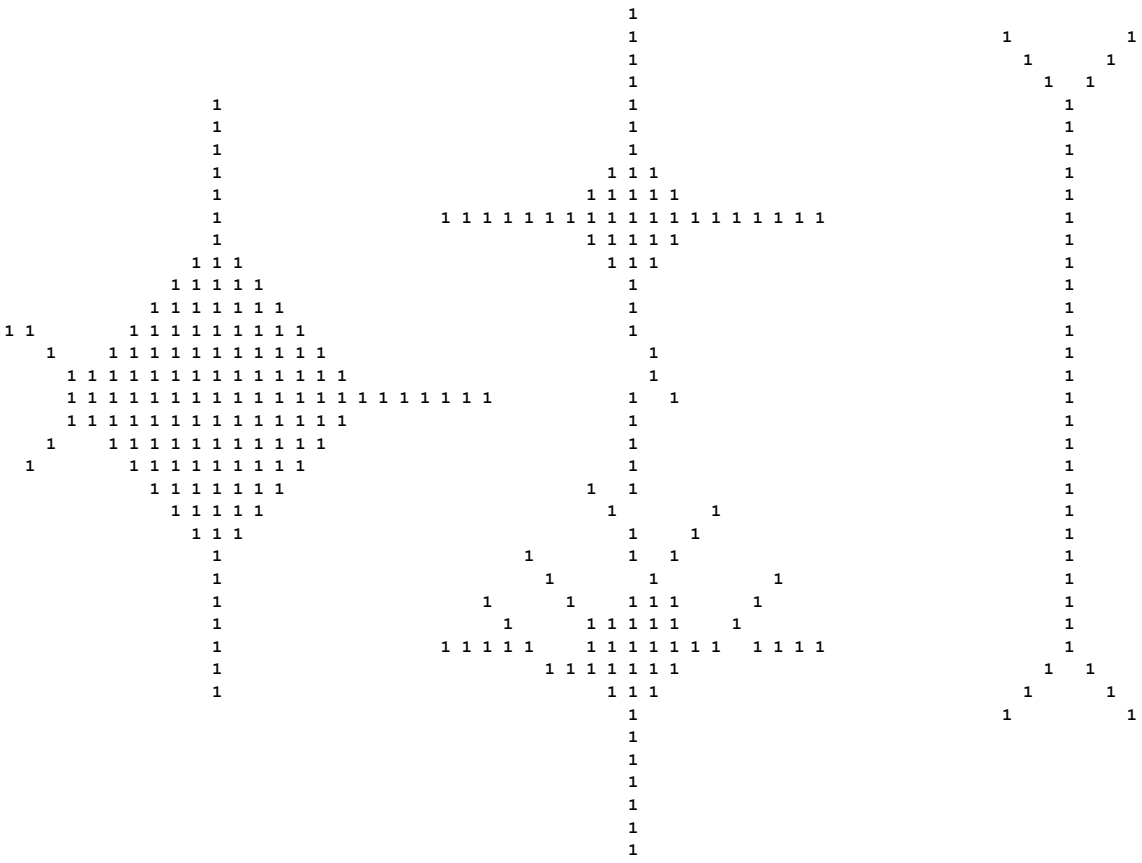
Result of Thinning: Cycle - 1



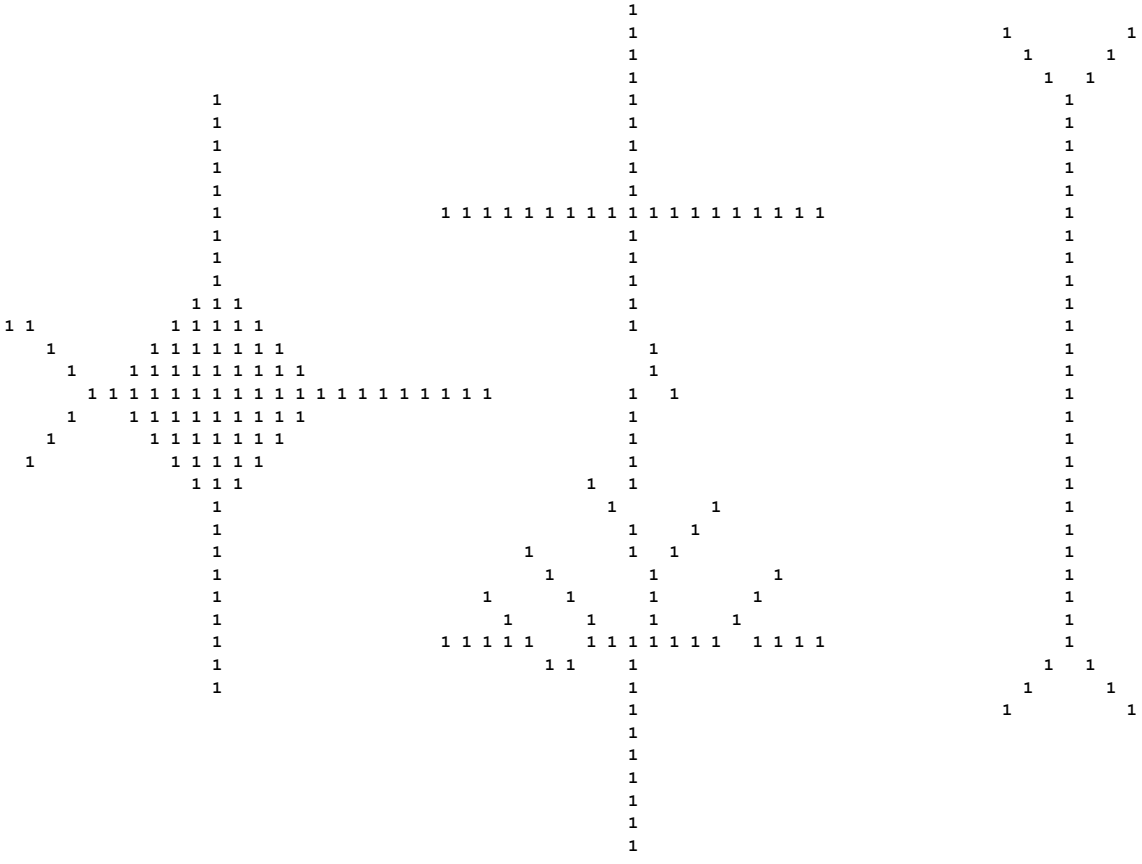
Result of Thinning: Cycle - 2



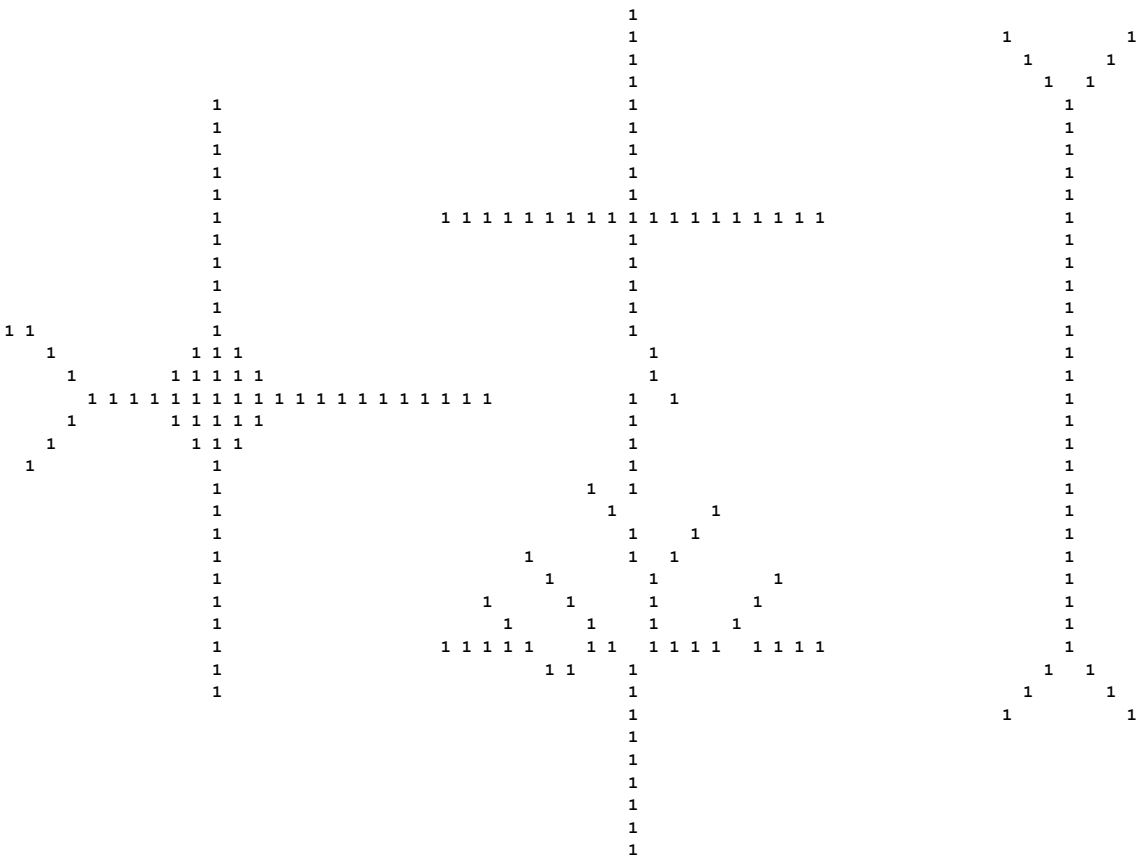
Result of Thinning: Cycle - 3



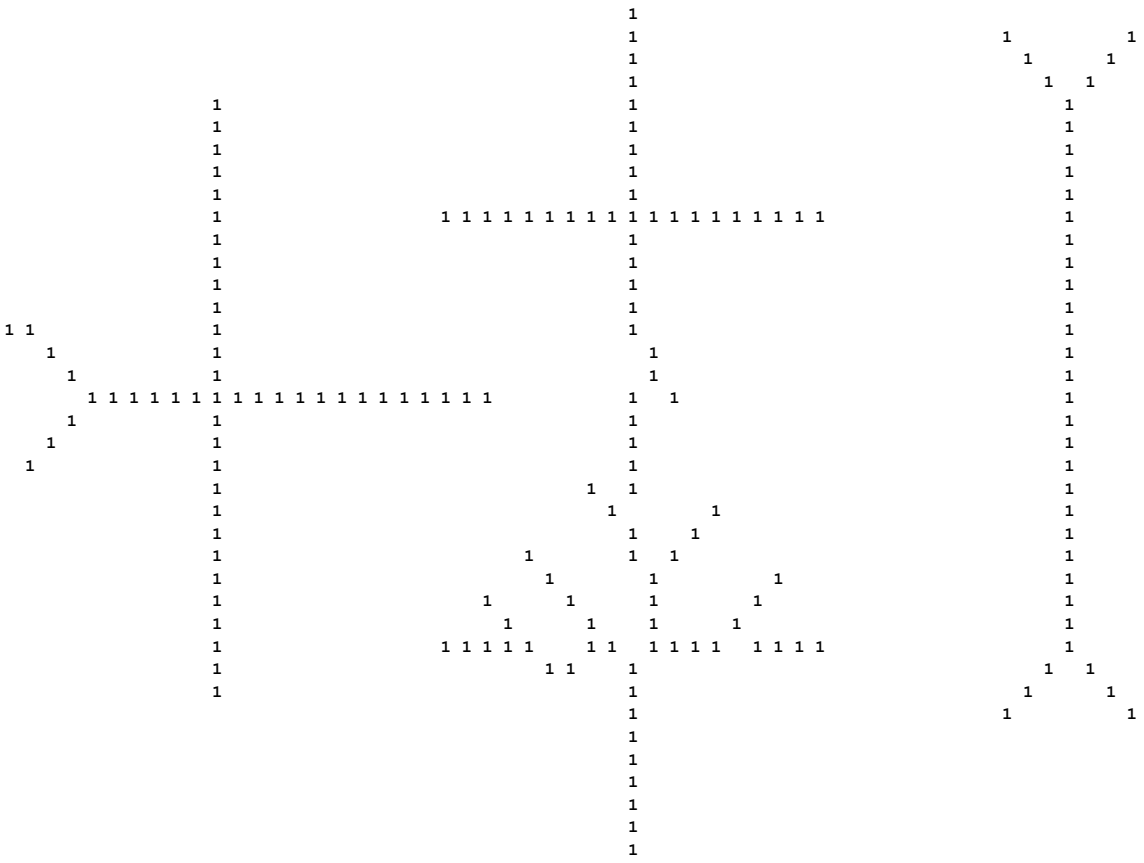
Result of Thinning: Cycle - 4



Result of Thinning: Cycle - 5



Result of Thinning: Cycle - 6



Result of Thinning: Cycle - 7