Andrew Alleyne
Project 4:
Connected Components

Algorithms for connectedness

First pass:

1. Scan image left to right and top to bottom.
2. If current pixel is greater than zero begin looking at the neighbors
    a. In the case of 8 connectedness, we look at.
        i. a, b, c, d
    b. In the case of 4 connectedness, we look at:
        i. Every other pixel until the current pixel
3. If our neighboring pixels equal to zero, we assign a new label.
4. If our neighboring pixels all have the same label, we do not change the pixel value.
5. If some but not all our pixels have the same value do two things.
    a. Find the minimum of the neighbors.
    b. Assign the minimum to the current pixel.
6. In steps (3) and (5) update the equivalency table.
7. Repeat all steps until each pixel is processed.

Second Pass:

1. Scan the results of Pass1 left to right and top to bottom.
2. If the current pixel is greater than zero begin looking at the neighbors
3. Similarly,
    a. In the case of 8 connectedness, we look at.
        ii. e, f, g, h
    b. In the case of 4 connectedness, we look at:
        iii. Every other pixel after the current pixel

4. If our neighboring pixels equal to zero, we assign a new label.
5. If our neighboring pixels all have the same label including the current Pixel, we do not change the pixel value and it keeps its label.
6. If at least 2 amongst the neighbors have different labels excluding zero.
    a. Find the minimum of the neighbors.
    b. If the current pixel is larger than the minimum.
        i. Assign the Equivalency table at the current pixels value the minimum value.
        ii. Assign the current pixel the minimum.
7. Repeat all steps until each pixel is processed.

Third Pass

1. Assign property file struct array a dynamic size of the maximum label +1
2. Scan the results of Pass2 left to right and top to bottom.
3. If the current pixel is greater than zero.
    a. Current pixels value is equal to its value in the equivalency table.
    b. Update the pixel count for the current pixel.

    c. Display the minRow , minCol, maxRow and maxCol
4. Repeat step (2) and step (3) for each pixel.

Source Code

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

using namespace std;

class CClabel
{

    struct Property
    {
        int label;
        int numPixels;
        int minR;
        int minC;
        int maxR;
        int maxC;
    };

private:
    Property *property;
    int numRows;
    int numCols;
    int min;
    int max;
    int newLabel;
    int trueNumCC = 0;
    int minVal;
    int maxVl;
    int newMin;
    int newMax;
    int NonZeroNeighborAry[5];

    int counter = 0;
    int readLabel;

    int *EQAry;
    int **zeroFrameAry;
```

```
    string input;

public:
    CClabel(ifstream &inputFile)
    {
        newLabel = 0;

        if (inputFile)
        {
            inputFile >> this->numRows >> this->numCols >>
this->min >> this->max;
        }

        zeroFrameAry = new int *[numRows + 2];

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

        int size = (numRows * numCols) / 4;

        EQAry = new int[size];

        for (int i = 0; i < size; i++)
        {
            EQAry[i] = i;
        }
    }

    void zero2D()
    {
        for (int i = 0; i < numRows + 2; i++)
        {
            for (int j = 0; j < numCols + 2; j++)
            {
                zeroFrameAry[i][j] = 0;
            }
        }
    }
```

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

    // Connectness 4
    void connect4Pass1(ofstream &prettyPrint)
    {

        int minLabel;
        int currentLabel;

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

                if (zeroFrameAry[i][j] > 0)
                {

                    //look at the neigbors in the [i-j][j]  and
the [i][j - i] positon.

                    NonZeroNeighborAry[0] = zeroFrameAry[i -
1][j];
                    NonZeroNeighborAry[1] = zeroFrameAry[i][j -
i];

                    //First case where labels are equal to 0
                    if (NonZeroNeighborAry[0] == 0 &&
NonZeroNeighborAry[1] == 0)
```

```
                    {
                        newLabel++;
                        zeroFrameAry[i][j] = newLabel;

                        //Take current label and save it for
minimum update
                        currentLabel = zeroFrameAry[i][j];
                    }
                    else if (NonZeroNeighborAry[0] ==
NonZeroNeighborAry[1])
                    {
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                    }
                    else if (NonZeroNeighborAry[0] !=
NonZeroNeighborAry[1])
                    {
                        //Find the minimum between the two
neighbors
                        minLabel =
NeighborMinimum(NonZeroNeighborAry[0], NonZeroNeighborAry[1]);

                        for (int m = 0; m < 2; m++)
                        {
                            if (NonZeroNeighborAry[m] != 0)
                            {

                                NonZeroNeighborAry[m] =
minLabel;


                            }
                        }
                        //Assign the mininum to the current
pixel value
                        zeroFrameAry[i][j] = minLabel;

                        //Update the EQ table for neigbors
                        //updateEQ(currentLabel, i, j);
                    }
```

```
                }
            }
        }

        counter++;
        prettyPrint << "EquivalencyTable Pass  : " << counter
<< endl;
        printEQAry(prettyPrint);
    }

    int NeighborMinimum(int a, int b)
    {
        int min = NonZeroNeighborAry[0];

        for (int m = 0; m < 1; m++)
        {

            if (NonZeroNeighborAry[m] < NonZeroNeighborAry[m +
1])
            {

                min = NonZeroNeighborAry[m];
                if (min == 0)
                {
                    min = NonZeroNeighborAry[m + 1];
                }
            }
            else
            {

                min = NonZeroNeighborAry[m + 1];
                if (min == 0)
                {
                    min = NonZeroNeighborAry[m];
                }
            }
        }
        cout << min << endl;
        return min;
    }
```

```
    //Connected components pass 2
    void
    connect4Pass2(ofstream &prettyPrint)
    {

        for (int i = numRows; i >= 1; i--)
        {
            for (int j = numCols; j >= 1; j--)
            {
                if (zeroFrameAry[i][j] > 0)
                {
                    //Scan from current pixel [1][1]
                    NonZeroNeighborAry[0] = zeroFrameAry[i][j +
1];
                    NonZeroNeighborAry[1] = zeroFrameAry[i +
1][j];

                    int currentLabel = zeroFrameAry[i][j];

                    //First case where labels are equal to 0
keep the same on pass 2
                    if (NonZeroNeighborAry[0] == 0 &&
NonZeroNeighborAry[1] == 0)
                    {
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                        updateEQ(currentLabel, i, j);
                    }

                    //Second case where all/some of the pixels
have the same label. We do nothing

                    if (NonZeroNeighborAry[0] ==
NonZeroNeighborAry[1] && NonZeroNeighborAry[0] ==
zeroFrameAry[i][j])
                    {
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                    }
```

```
                    else if (NonZeroNeighborAry[0] !=
NonZeroNeighborAry[1])
                    {
                        //Find the minimum of the current pixel
along with c and d

                        int minLabel =
NeighborMinimumPass2(currentLabel);

                        updateEQ(currentLabel, i, j);
                        EQAry[currentLabel] = minLabel;
                        zeroFrameAry[i][j] = minLabel;

                        //update neighnors
                    }

                    zeroFrameAry[i][j] =
EQAry[zeroFrameAry[i][j]];
                }
            }
        }

        counter++;
        prettyPrint << "EquivalencyTable Pass  : " << counter
<< endl;
        printEQAry(prettyPrint);
    }

    int NeighborMinimumPass2(int currentLabel)
    {
        int min = NonZeroNeighborAry[0];
        NonZeroNeighborAry[2] = currentLabel;

        for (int m = 0; m < 3; m++)
        {
            if (min < NonZeroNeighborAry[m])
            {
                if (NonZeroNeighborAry[m] == 0)
                {
                    break;
```

```
            }
            else
            {
                min = NonZeroNeighborAry[m];
            }
        }
    }

    return min;
}

void updateEQ(int currentLabel, int i, int j)
{
    EQAry[currentLabel] = zeroFrameAry[i][j];
}

//Prints zeroframed array
void imgReformat(ofstream &prettyPrint)
{
    for (int i = 1; i <= numRows; i++)
    {
        for (int j = 1; j <= numCols; j++)
        {
            if (zeroFrameAry[i][j] == 0)
            {
                prettyPrint << "  ";
            }
            else
            {
                prettyPrint << zeroFrameAry[i][j] << "  ";
            }
        }
        prettyPrint << endl;
    }
    prettyPrint << "\n"
                << endl;
}

void printEQAry(ofstream &prettyPrint)
{
```

```
        for (int i = 0; i < newLabel; i++)
        {
            prettyPrint << i << " : " << EQAry[i] << endl;
        }

        prettyPrint << endl;
        prettyPrint << endl;
    }

    void manageEQAry(ofstream &prettyPrint)
    {
        readLabel = 0;
        // return true size of EQ table

        for (int i = 1; i < newLabel; i++)
        {

            if (i != EQAry[i])
            {
                EQAry[i] = EQAry[EQAry[i]];
            }
            else
            {
                readLabel++;
                EQAry[i] = readLabel;
            }
            //   cout << readLabel << endl;
        }

        prettyPrint << "EQ table  management" << endl;
        printEQAry(prettyPrint);
        prettyPrint << "" << endl;
    }

    void connectPass3(ofstream &prettyPrint, ofstream
&labelFile)
    {
        trueNumCC++;
```

```
        labelFile << numRows << " " << numCols << " " << min <<
" " << max << endl;
        labelFile << readLabel << endl;

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

        //PropertyFile
        property = new Property[readLabel + 1];
        //label
        int label = 1;
        int pixels = 0;

        while (label <= readLabel)
        {
            labelFile << "-----------------" << endl;
            pixels = 0;

            //label
            property[label].label = label;

            for (int i = 1; i <= numRows; i++)
            {
                for (int j = 1; j <= numCols; j++)
                {
                    if (zeroFrameAry[i][j] == label)
                    {

                        //pixelCount
                        pixels++;
                        property[label].numPixels = pixels;
```

```
                        if (pixels == 1)
                        { //minRow, minC of propfile
                            property[label].minR = i;
                            property[label].minC = j;
                            property[label].maxR = i;
                            property[label].maxC = j;
                        }
                        else
                        {

                            if (i < property[label].minR)
                            {
                                property[label].minR = i;
                            }

                            if (j < property[label].minC)
                            {
                                property[label].minC = j;
                            }

                            if (i > property[label].maxR)
                            {
                                property[label].maxR = i;
                            }

                            if (j > property[label].maxC)
                            {
                                property[label].maxC = j;
                            }
                        }
                    }
                }
            }

            labelFile << "" << property[label].label << endl;
            labelFile << property[label].numPixels << endl;
            labelFile << property[label].minR << "  " <<
property[label].minC << endl;
```

```
            labelFile << property[label].maxR << "   " <<
property[label].maxC << endl;
            labelFile << endl;

            label++;
        }
        counter++;

        prettyPrint << "EquivalencyTable Pass  : " << counter
<< endl;
        printEQAry(prettyPrint);
    }
    // Connectness 4

    // Connectness 8
    void connect8Pass1(ofstream &prettyPrint)
    {

        int minLabel;
        int currentLabel;

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

                if (zeroFrameAry[i][j] > 0)
                {

                    //look at the neigbors in the [i-j][j]  and
the [i][j - i] positon.

                    NonZeroNeighborAry[0] = zeroFrameAry[i -
1][j - 1];
                    NonZeroNeighborAry[1] = zeroFrameAry[i -
1][j];
                    NonZeroNeighborAry[2] = zeroFrameAry[i -
1][j + 1];
                    NonZeroNeighborAry[3] = zeroFrameAry[i][j -
1];
```

```
                    //First case where labels are equal to 0
                    if (NonZeroNeighborAry[0] == 0 &&
NonZeroNeighborAry[1] == 0 && NonZeroNeighborAry[2] == 0 &&
NonZeroNeighborAry[3] == 0)
                    {
                        newLabel++;
                        zeroFrameAry[i][j] = newLabel;

                        EQAry[newLabel] = newLabel;

                        //To update EQ table at currentV
                        currentLabel = zeroFrameAry[i][j];
                    }

                    // Second case where they all have the same
label
                    if (NonZeroNeighborAry[0] ==
NonZeroNeighborAry[1] && NonZeroNeighborAry[1] ==
NonZeroNeighborAry[2] && NonZeroNeighborAry[2] ==
NonZeroNeighborAry[3])
                    {
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                    }

                    //    Third case if they all have different
labels.
                    if (NonZeroNeighborAry[0] !=
NonZeroNeighborAry[1] || NonZeroNeighborAry[2] !=
NonZeroNeighborAry[3 || NonZeroNeighborAry[0] !=
NonZeroNeighborAry[2]])
                    {
                        //Find the minimum between the 8
neighbors
                        minLabel = NeighborMinimum8();

                        //Assign the mininum to the current
pixel value
                        updateEQv2(currentLabel, minLabel);
```

```
                        //update all neighbors to point to
minimum
                        for (int m = 0; i < 4; i++)
                        {
                            if (NonZeroNeighborAry[m] != 0)
                            {
                                EQAry[NonZeroNeighborAry[m]] =
minLabel;
                            }
                        }

                        zeroFrameAry[i][j] = minLabel;
                    }
                }
            }
        }

        counter++;
        prettyPrint << " Pass  : " << counter << endl;
        printEQAry(prettyPrint);
    }

    void updateEQv2(int currentLabel, int minLabel)
    {
        //update neighbors to point to minimum value
        EQAry[currentLabel] = minLabel;
    }

    int NeighborMinimum8()
    {
        int min = 0;
        // Look at neighbors and exclude zero neighbors from
min calculation
        for (int i = 0; i < 4; i++)
        {

            if (NonZeroNeighborAry[i] != 0)
            {
                min = NonZeroNeighborAry[i];
```

```
                if (NonZeroNeighborAry[i] < min)
                {
                    min = NonZeroNeighborAry[i];
                }
            }
        }
        return min;
    }

    void connect8Pass2(ofstream &prettyPrint)
    {
        int minLabel;
        int currentLabel;

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

                if (zeroFrameAry[i][j] > 0)
                {

                    //Array layout current to last pixel
neighbor

                    NonZeroNeighborAry[0] = zeroFrameAry[i][j +
1];
                    NonZeroNeighborAry[1] = zeroFrameAry[i +
1][j - 1];
                    NonZeroNeighborAry[2] = zeroFrameAry[i +
1][j];
                    NonZeroNeighborAry[3] = zeroFrameAry[i +
1][j + 1];

                    //First case where labels are equal to 0
                    if (NonZeroNeighborAry[0] == 0 &&
NonZeroNeighborAry[1] == 0 && NonZeroNeighborAry[2] == 0 &&
NonZeroNeighborAry[3] == 0)
                    {
```

```
                        //keeps label
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                    }

                    //Second case
                    if (NonZeroNeighborAry[0] ==
NonZeroNeighborAry[1] && NonZeroNeighborAry[1] ==
NonZeroNeighborAry[2] && NonZeroNeighborAry[2] ==
NonZeroNeighborAry[3])
                    {
                        zeroFrameAry[i][j] =
zeroFrameAry[i][j];
                    }
                    else
                    {
                        //Find the minimum between the 8
neighbors
                        minLabel =
NeighborMinimum8Pass2(currentLabel);

                        //If current lable is > minLabel we
update it
                        if (zeroFrameAry[i][j] > minLabel)
                        {
                            for (int m = 0; i < 4; i++)
                            {
                                //exclude zeros
                                if (NonZeroNeighborAry[m] != 0)
                                {


EQAry[NonZeroNeighborAry[m]] = minLabel;
                                }
                            }

                            //Assign the mininum to the current
pixel value
                            updateEQv2(currentLabel, minLabel);
                            zeroFrameAry[i][j] = minLabel;
```

```
                        }
                    }
                }
            }
        }
        counter++;
        prettyPrint << "Pass  : " << counter << endl;
        printEQAry(prettyPrint);
    }

    int NeighborMinimum8Pass2(int currentLabel)
    {
        int min = 0;

        NonZeroNeighborAry[4] = currentLabel;
        // Look at neighbors and exclude zero neighbors from
min calculation
        for (int i = 0; i < 5; i++)
        {

            if (NonZeroNeighborAry[i] != 0)
            {
                min = NonZeroNeighborAry[i];

                if (NonZeroNeighborAry[i] < min)
                {
                    min = NonZeroNeighborAry[i];
                }
            }
        }
        return min;
    }

    // Connectness 8
};

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

    if (argc < 3)
```

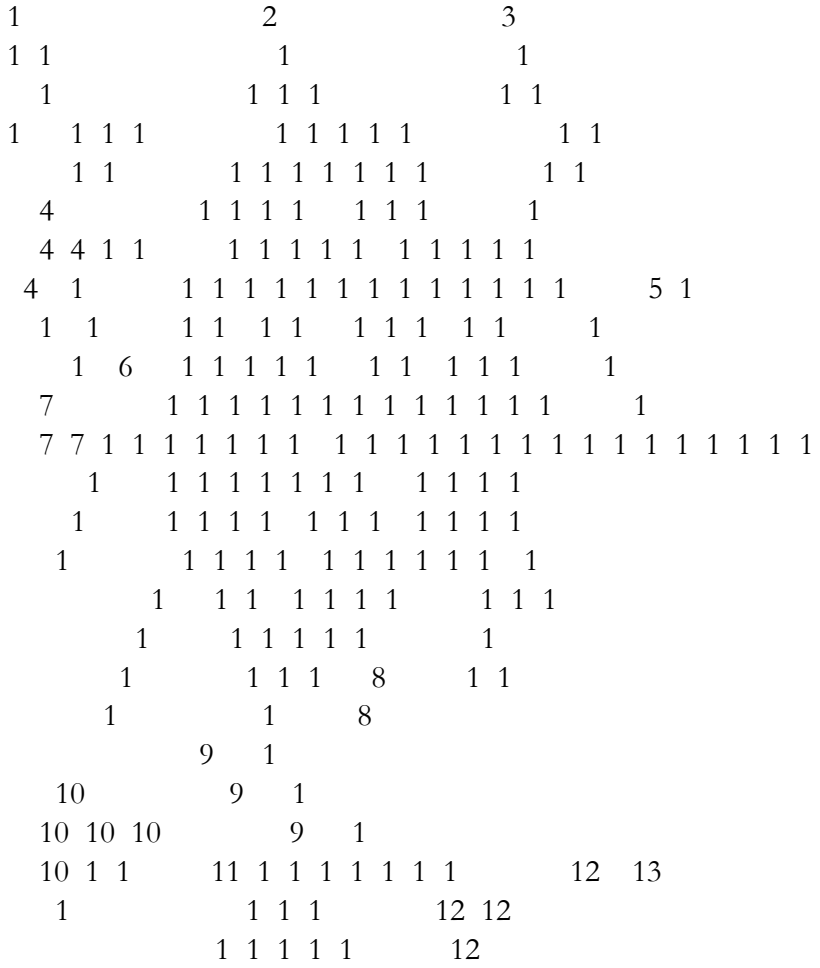
```
{
    cout << "Missing arguments. More needed." << endl;
}

string data = argv[1];
int connectness = stoi(argv[2]);

string rfPrettyPrint = argv[3];
string labelFile = argv[4];
string propertyFile = argv[5];

ifstream fs;
fs.open(data);

ofstream pp;
pp.open(rfPrettyPrint);

ofstream lf;
lf.open(labelFile);

ofstream pf;
pf.open(propertyFile);

CClabel cclabel(fs);
cclabel.zero2D();
cclabel.loadImage(fs);

if (connectness == 4)
{
    cclabel.connect4Pass1(pp);
    cclabel.imgReformat(pp);

    cclabel.connect4Pass2(pp);
    cclabel.imgReformat(pp);
}
else if (connectness == 8)
{
    cclabel.connect8Pass1(pp);
    cclabel.imgReformat(pp);
```

```
            cclabel.connect8Pass2(pp);
            cclabel.imgReformat(pp);
        }

        cclabel.manageEQAry(pp);
```

```
    cclabel.connectPass3(pp, pf);
    cclabel.imgReformat(lf);

    //  cclabel.drawBoxes(pp);
}
```

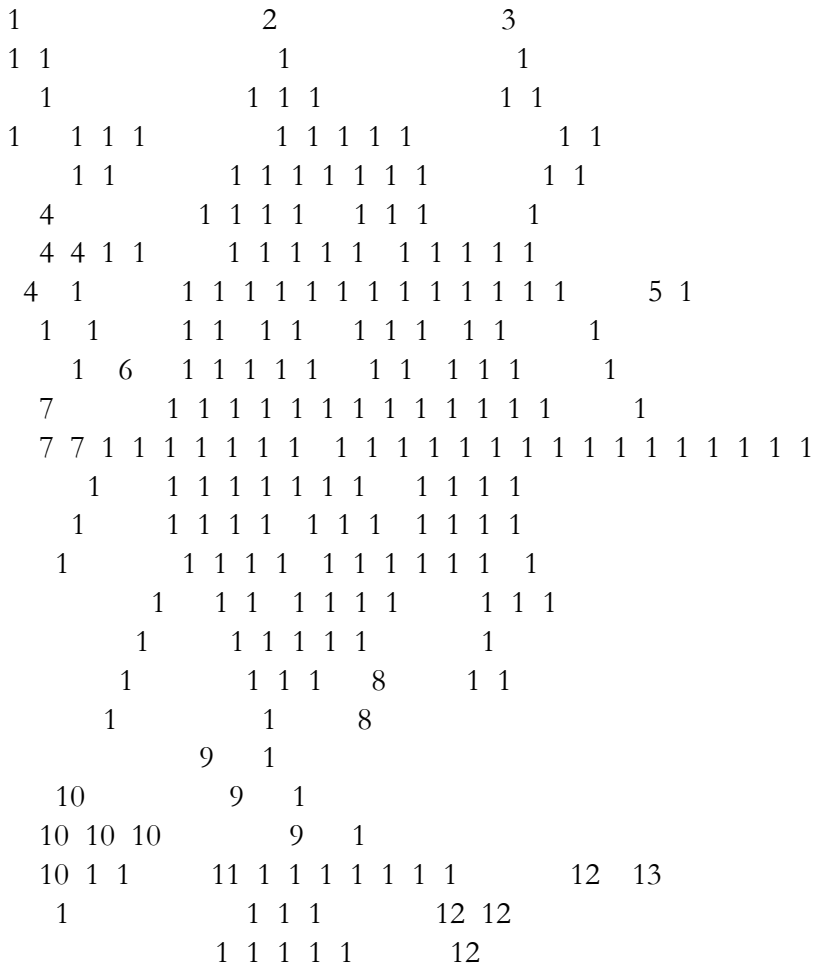
Output for 8 connectedness

Pass : 1
0 : 0
1 : 1
2 : 2
3 : 1
4 : 1
5 : 1
6 : 1
7 : 1
8 : 8
9 : 1
10 : 1
11 : 1
12 : 12



Pass : 2
0 : 0
1 : 1
2 : 2
3 : 1

4 : 1
5 : 1
6 : 1
7 : 1
8 : 8
9 : 1
10 : 1
11 : 1
12 : 12



EQ table management
0 : 0
1 : 1
2 : 2
3 : 1
4 : 1
5 : 1
6 : 1
7 : 1
8 : 3
9 : 1
10 : 1
11 : 1
12 : 4

Equivalency Table Pass : 3
0 : 0
1 : 1
2 : 2
3 : 1
4 : 1
5 : 1
6 : 1
7 : 1
8 : 3
9 : 1
10 : 1
11 : 1
12 : 4

Property File for 8 connectedness

25 31 0 1
4

1
230
1 1
25 30

2
1
1 16
1 16

3
2
18 20
19 21

4
4
23 24
25 25

Output for Label File

```
1                2              1
1 1               1              1
  1             1 1 1           1 1
1   1 1 1         1 1 1 1 1         1 1
    1 1        1 1 1 1 1 1 1       1 1
  1          1 1 1 1   1 1 1      1
  1 1 1 1     1 1 1 1 1  1 1 1 1 1
 1  1       1 1 1 1 1 1 1 1 1 1 1 1 1     1 1
  1  1      1 1  1 1   1 1 1  1 1     1
    1  1    1 1 1 1 1   1 1  1 1 1     1
  1        1 1 1 1 1 1 1 1 1 1 1 1 1     1
  1 1 1 1 1 1 1 1 1  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     1     1 1 1 1 1 1 1   1 1 1 1
    1      1 1 1 1  1 1 1  1 1 1 1
   1        1 1 1 1  1 1 1 1 1 1  1
         1    1 1  1 1 1 1     1 1 1
        1      1 1 1 1 1       1
       1        1 1 1   3     1 1
      1          1     3
             1   1
   1          1   1
  1 1 1         1   1
  1 1 1     1 1 1 1 1 1 1 1        4  12
   1            1 1 1       4 4
              1 1 1 1 1      4
```