Quinn Roemer

CISP - 440

Assignment 6

10/25/2018

Part 0 - Relation Implementation

Description:

The goal for this assignment was to write code that would be able examine a given relation and determine if it is reflexive, symmetric, and/or transitive. If it fulfilled all 3 criteria we must print out its equivalence classes. We were given binary files that represented relations to use as our inputs. Seven in total.

Source Code:

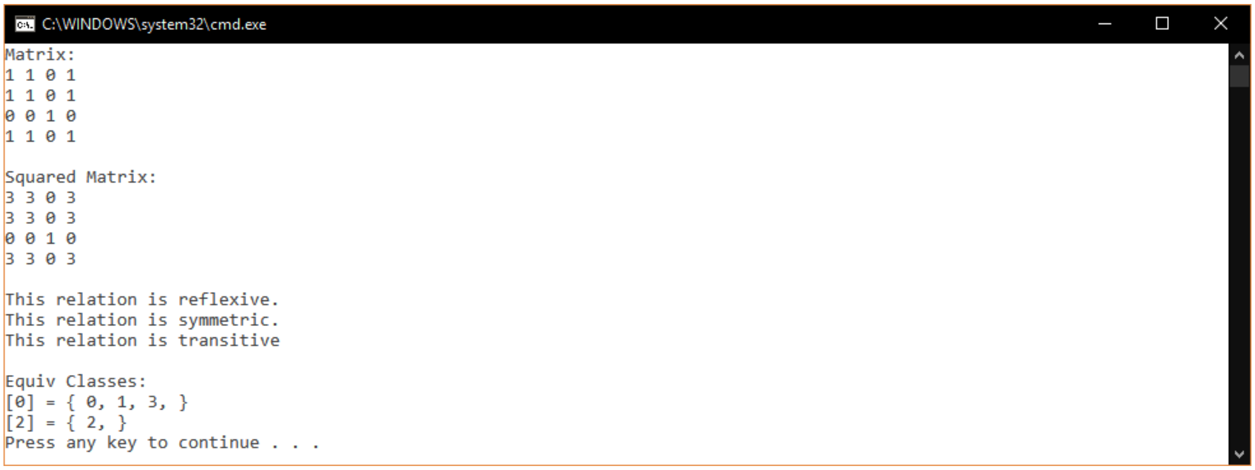
```
//Code written by Quinn Roemer, based on code by Professor Ross. 10/23/2018

#include <iostream>
#include <fstream>

using namespace std;

//Definitions
#define MAX 40

//Global data.
int squaredRelation[MAX][MAX];
int eClasses[MAX] = { 0 };
int relations[MAX][MAX];
int eCheck[3] = { 0 };
int mSize = 9;

//Function prototypes.
void printMatrix(bool);
void squareMatrix();
void equivClasses();
void printClasses();
void transitive();
void reflexive();
void readFile();
void symmetry();

//Main function to execute.
int main()
{
    readFile();

    cout << "Matrix:" << endl;
    printMatrix(true);
    squareMatrix();

    cout << "\nSquared Matrix:" << endl;
    printMatrix(false);
    cout << endl;

    reflexive();
    symmetry();
    transitive();

    //If the matrix represents an Equivalence Relation this code executes.
    if (eCheck[0] == 1 && eCheck[1] == 1 && eCheck[2] == 1)
    {
        cout << "\nEquiv Classes:" << endl;

        equivClasses();
        printClasses();
    }
}

//This function reads a matrix from a file.
void readFile()
{
    char value;

    //Opening file.
    ifstream file("R1.bin", ios_base::binary);

    //Checking to see if the file opened correctly.
    if (!file)
    {
        cerr << "Input file could not be opened." << endl;
        exit(1);
    }

    //Getting the matrix size.
    file.read(&value, 1);
    mSize = value;

    //Filling the matrix from the file.
    for (int count = 0; count < mSize; count++)
    {
        for (int index = 0; index < mSize; index++)
        {
                file.read(&value, 1);
                relations[count][index] = value;
        }
    }

    //Close the file.
    file.close();
}
```

```
//This function prints the matrix.
void printMatrix(bool flag)
{
    //Note, the flag determines if the square or reg relation is to be printed.

    if (flag == true)
    {
        for (int count = 0; count < mSize; count++)
        {

                for (int index = 0; index < mSize; index++)
                {
                        cout << relations[count][index] << " ";
                }
                cout << endl;
        }
    }
    else
    {
        for (int count = 0; count < mSize; count++)
        {

                for (int index = 0; index < mSize; index++)
                {
                        cout << squaredRelation[count][index] << " ";
                }
                cout << endl;
        }
    }
}

//This function checks a matrix for reflexivity.
void reflexive()
{
    for (int count = 0; count < mSize; count++)
    {
        if (relations[count][count] != 1)
        {
                cout << "This relation is NOT reflexive." << endl;
                return;
        }
    }

    cout << "This relation is reflexive." << endl;
    eCheck[0] = 1;
}

//This function checks a matrix for symmetry.
void symmetry()
{
    for (int count = 0; count < mSize; count++)
    {
        for (int index = 0; index < mSize; index++)
        {
               if (relations[count][index] == 1)
               {
                      if (relations[index][count] != 1)
                      {
                             cout << "This relation is NOT symmetric." << endl;
                             return;
                      }
               }
        }
    }

    cout << "This relation is symmetric." << endl;
    eCheck[1] = 1;
}

//This function checks a matrix for transivity.
void transitive()
{
    //Comparing the relation against its square.
    for (int count = 0; count < mSize; count++)
    {
        for (int index = 0; index < mSize; index++)
        {
               if (squaredRelation[count][index] >= 1)
               {
                      if (relations[count][index] == 0)
                      {
                             cout << "This relation is NOT transitive." << endl;
                             return;
                      }
               }
        }
    }

    cout << "This relation is transitive" << endl;
    eCheck[2] = 1;
}

//This function squares a matrix.
void squareMatrix()
{
    int sum = 0;

    //This loop keeps track of the row.
    for (int count = 0; count < mSize; count++)
    {
        //This loop keeps track of the column.
        for (int index = 0; index < mSize; index++)
        {
                //This loop iterates through all numbers on a row or column.
                for (int num = 0; num < mSize; num++)
                {
                        sum = sum + (relations[count][num] * relations[num][index]);
                }

                //Saving and resetting the necessary values.
                squaredRelation[count][index] = sum;
                sum = 0;
        }
    }
}

//This function finds the equivalence classes of an ER relation.
void equivClasses()
{
    /*For debugging purposes.
    cout << "\nMatrix:" << endl;
    printMatrix(true);
    cout << endl;*/

    for (int count = 0; count < mSize; count++)
    {
        for (int index = count; index >= 0; index--)
        {
                //For debugging purposes.
                //cout << relations[count][index] << " ";

                //If the value currently being looked is 1 this code executes.
                if (relations[count][index] == 1)
                {
                        //Setting the correct captain.
                        eClasses[index] = 1;

                        //This code looks back to the beginning of the iteration and resets the
captains in
                        //front of the current captain.
                        for (int number = count; number > index; number--)
                        {
                                if (relations[count][number] == 1)
                                {
                                        eClasses[number] = 0;
                                }
                        }
                }
        }
    }
```

```
}

//This function prints the equivalence classes of a given ER relation.
void printClasses()
{
    for (int count = 0; count < mSize; count++)
    {
        if (eClasses[count] == 1)
        {
                cout << "[" << count << "] = { ";

                for (int index = 0; index < mSize; index++)
                {
                        if (relations[count][index] == 1)
                        {
                                cout << index << ", ";
                        }
                }
                cout << "}" << endl;
        }
    }
}
```

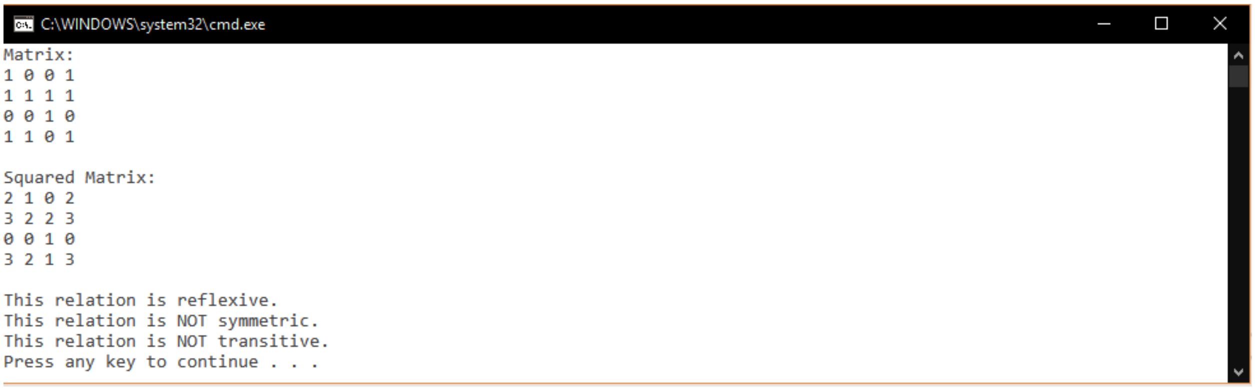
Output:

The following outputs are for R1.bin through R7.bin



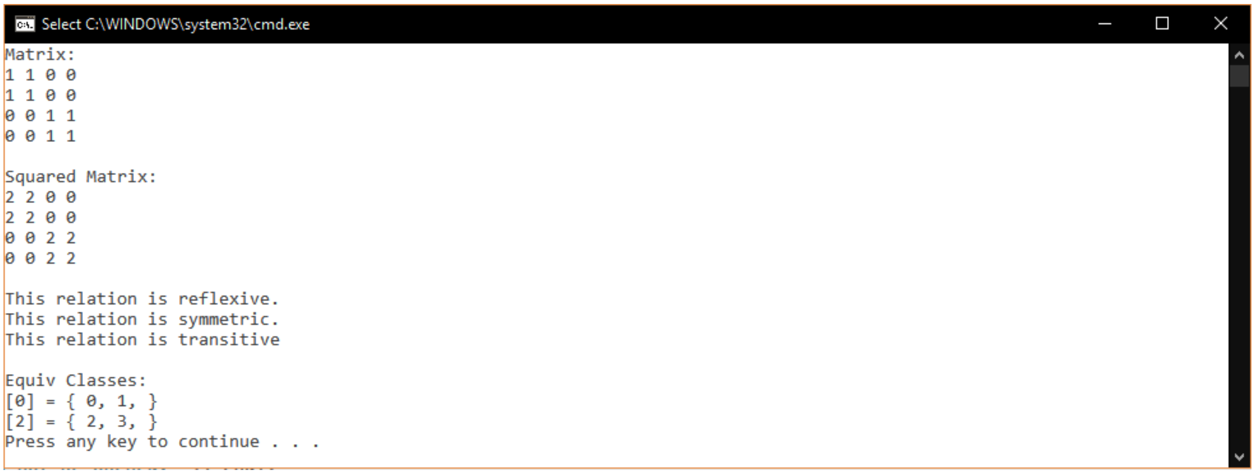
```
Matrix:
1 1 0 1
1 1 0 1
0 0 1 0
1 1 0 1

Squared Matrix:
3 3 0 3
3 3 0 3
0 0 1 0
3 3 0 3

This relation is reflexive.
This relation is symmetric.
This relation is transitive

Equiv Classes:
[0] = { 0, 1, 3, }
[2] = { 2, }
Press any key to continue . . .
```

R1.bin



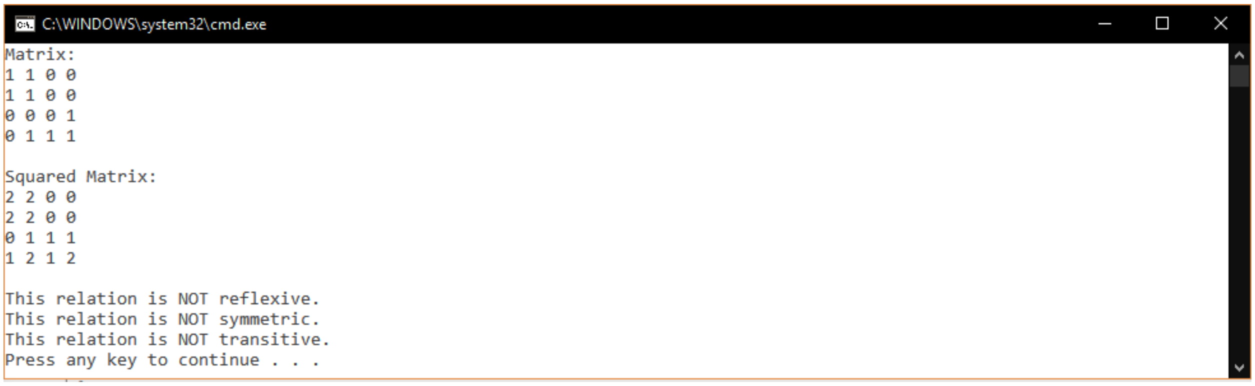
```
Matrix:
1 0 0 1
1 1 1 1
0 0 1 0
1 1 0 1

Squared Matrix:
2 1 0 2
3 2 2 3
0 0 1 0
3 2 1 3

This relation is reflexive.
This relation is NOT symmetric.
This relation is NOT transitive.
Press any key to continue . . .
```

R2.bin



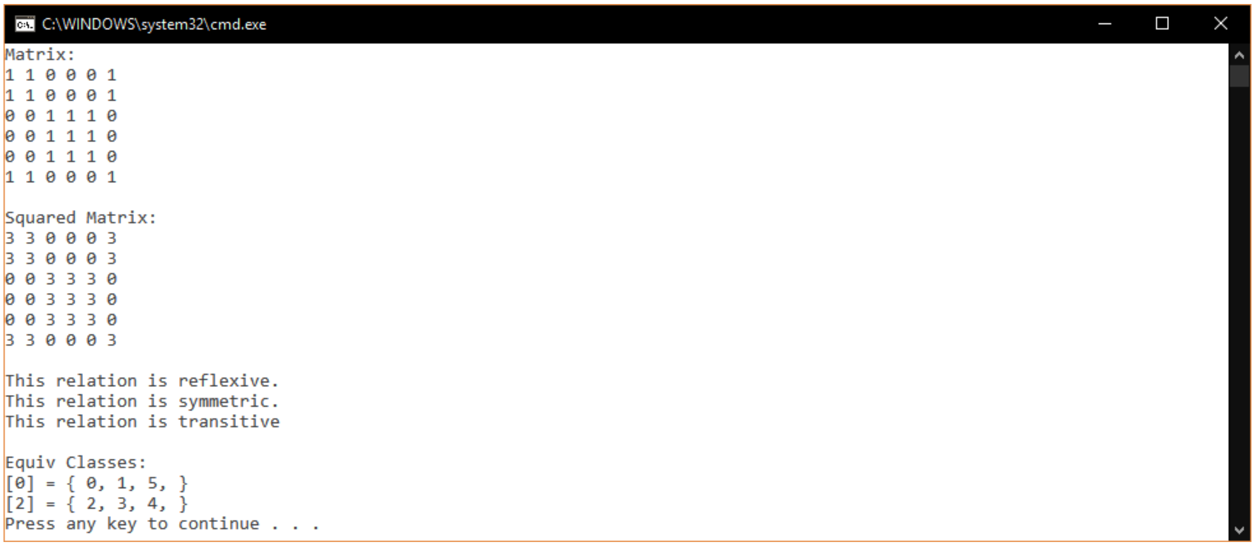
```
Matrix:
1 1 0 0
1 1 0 0
0 0 1 1
0 0 1 1

Squared Matrix:
2 2 0 0
2 2 0 0
0 0 2 2
0 0 2 2

This relation is reflexive.
This relation is symmetric.
This relation is transitive

Equiv Classes:
[0] = { 0, 1, }
[2] = { 2, 3, }
Press any key to continue . . .
```

R3.bin



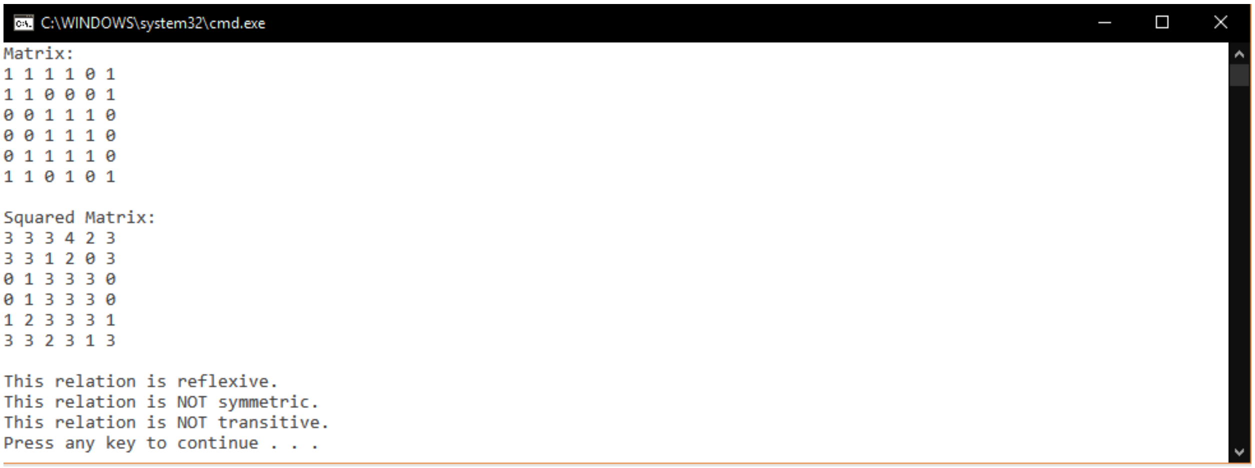
```
Matrix:
1 1 0 0
1 1 0 0
0 0 0 1
0 1 1 1

Squared Matrix:
2 2 0 0
2 2 0 0
0 1 1 1
1 2 1 2

This relation is NOT reflexive.
This relation is NOT symmetric.
This relation is NOT transitive.
Press any key to continue . . .
```

R4.bin



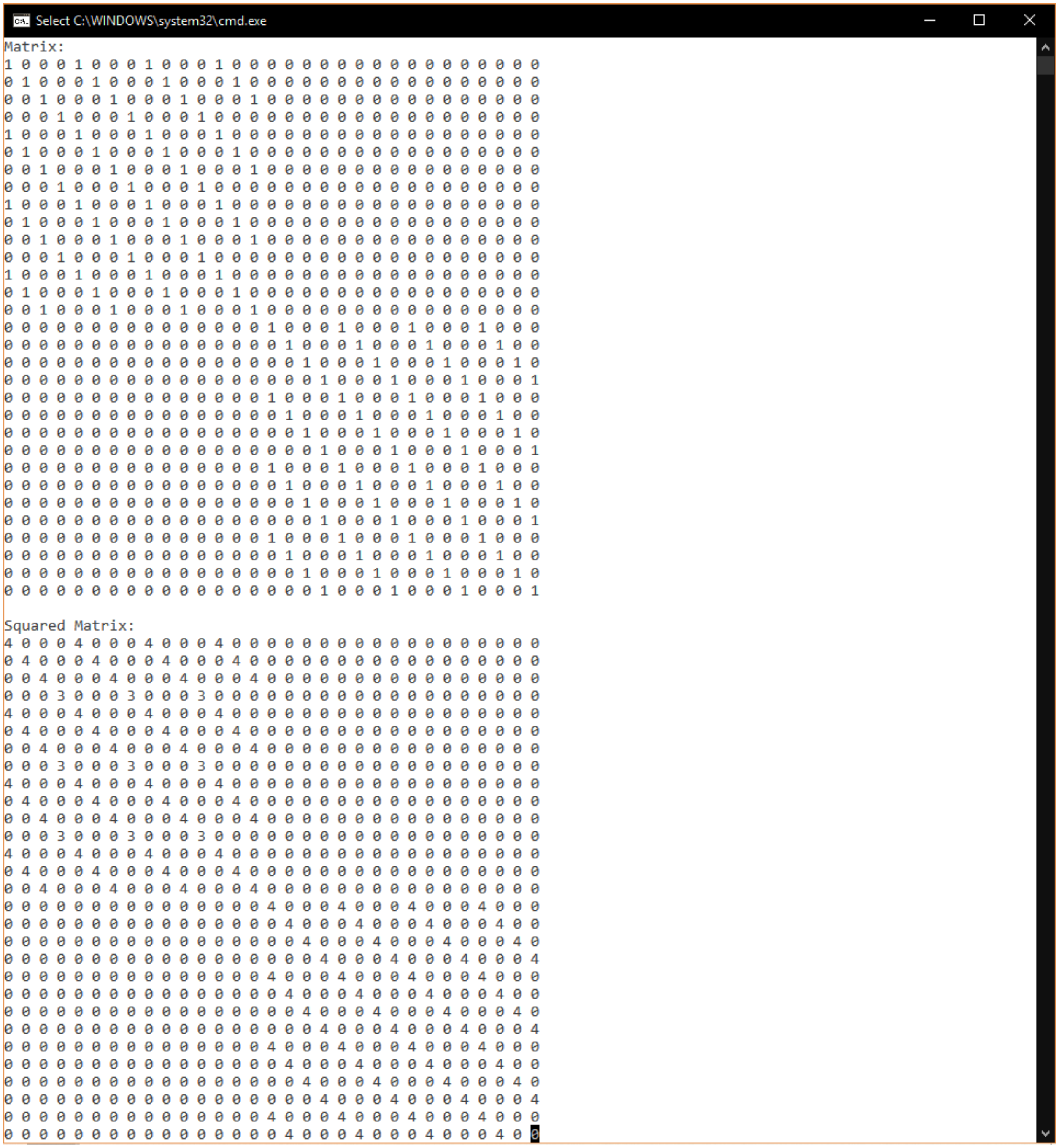
```
Matrix:
1 1 0 0 0 1
1 1 0 0 0 1
0 0 1 1 1 0
0 0 1 1 1 0
0 0 1 1 1 0
1 1 0 0 0 1

Squared Matrix:
3 3 0 0 0 3
3 3 0 0 0 3
0 0 3 3 3 0
0 0 3 3 3 0
0 0 3 3 3 0
3 3 0 0 0 3

This relation is reflexive.
This relation is symmetric.
This relation is transitive

Equiv Classes:
[0] = { 0, 1, 5, }
[2] = { 2, 3, 4, }
Press any key to continue . . .
```

R5.bin



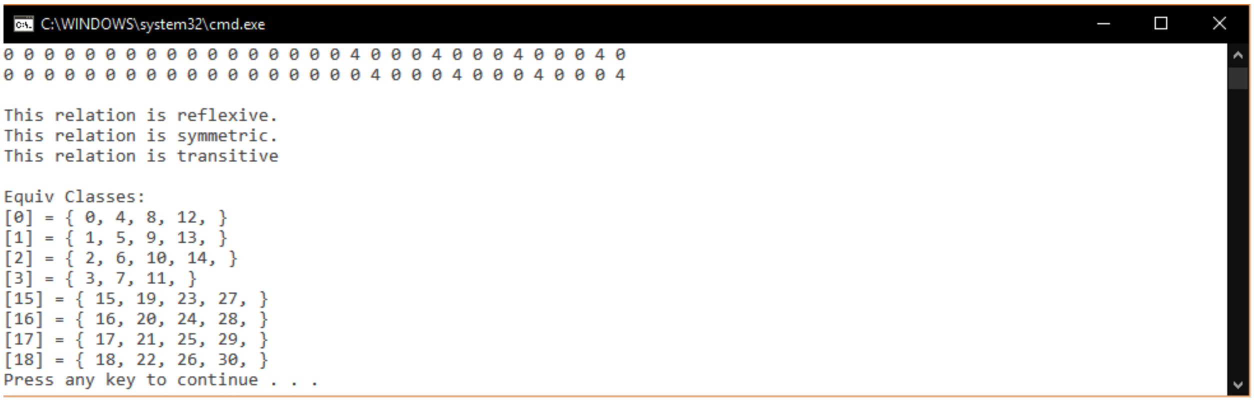
```
Matrix:
1 1 1 1 0 1
1 1 0 0 0 1
0 0 1 1 1 0
0 0 1 1 1 0
0 1 1 1 1 0
1 1 0 1 0 1

Squared Matrix:
3 3 3 4 2 3
3 3 1 2 0 3
0 1 3 3 3 0
0 1 3 3 3 0
1 2 3 3 3 1
3 3 2 3 1 3

This relation is reflexive.
This relation is NOT symmetric.
This relation is NOT transitive.
Press any key to continue . . .
```

R6.bin



```
Matrix:
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1

Squared Matrix:
4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 0 0 0 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 0 0 0 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 3 0 0 0 3 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0 0
```

R7.bin (part 1 of 2)



```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 4 0 0 0 4 0 0 0 4

This relation is reflexive.
This relation is symmetric.
This relation is transitive

Equiv Classes:
[0] = { 0, 4, 8, 12, }
[1] = { 1, 5, 9, 13, }
[2] = { 2, 6, 10, 14, }
[3] = { 3, 7, 11, }
[15] = { 15, 19, 23, 27, }
[16] = { 16, 20, 24, 28, }
[17] = { 17, 21, 25, 29, }
[18] = { 18, 22, 26, 30, }
Press any key to continue . . .
```

R7.bin (part 2 of 2)

Part 1 - Extra Credit

Description:

In addition to the above assignment the option to perform some extra credit was presented. To do so, we had to implement code that would be able to generate equivalence relations. To do this, I decided to write a function that generated equivalence classes and then work my way up from there writing the relation from the generated classes.

Note: This code is able to generate an equivalence relation ranging from size 2 through 40. In addition, the max number of equivalence classes per relation has been capped at 8 due to processing time.

Source Code:

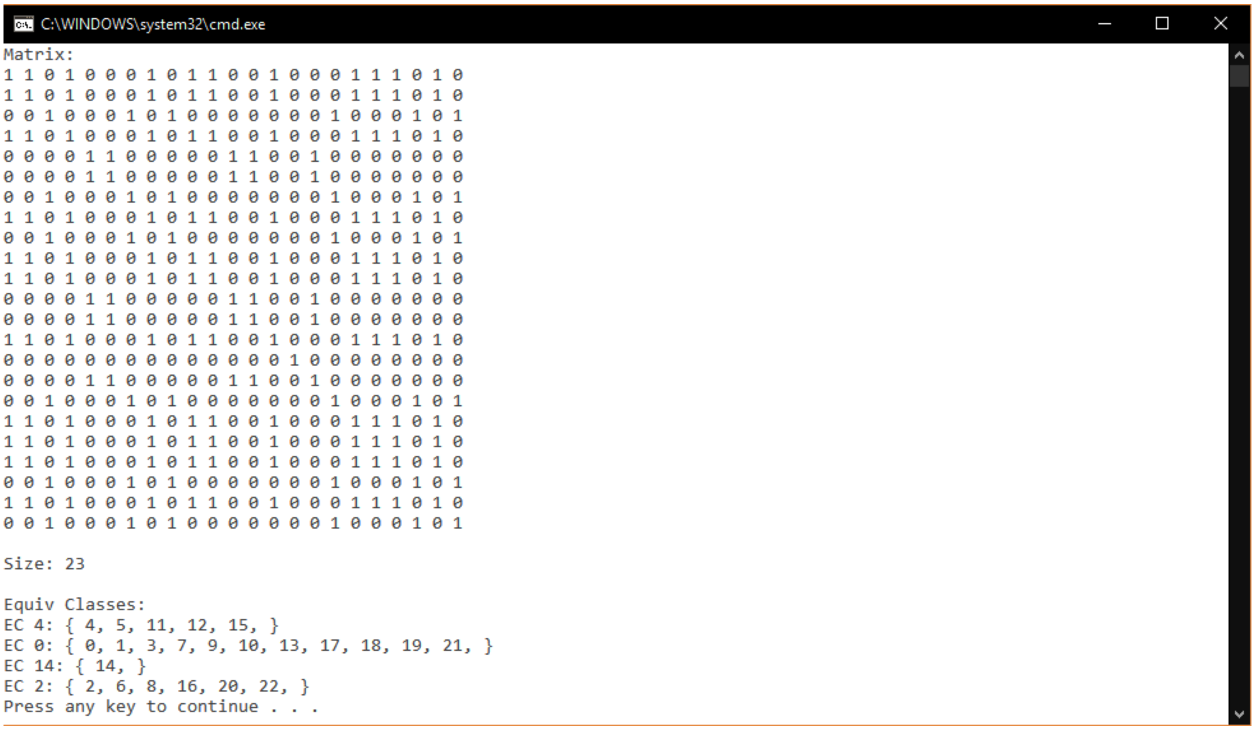
```
//Code written by Quinn Roemer 10/23/2018

#include <iostream>
#include <time.h>

using namespace std;

//Definitions.
#define MAX 40

//Global Data.
int eClasses[MAX][MAX] = { 0 };
int relation[MAX][MAX] = { 0 };
int equivSize[MAX] = { 0 };
int mSize;
int ecNum;

//Function prototypes.
void fillRelation();
void printClasses();
void printMatrix();
void erGen();

//Main function to execute.
int main()
{
    //Intilize a random seed.
    srand(time(NULL));

    //Generate equivalence classes and relation.
    erGen();
    fillRelation();
    //Print the matrix.
    cout << "Matrix:" << endl;
    printMatrix();

    //Print the size.
    cout << "\nSize: " << mSize << endl;

    //Print the classes.
    cout << "\nEquiv Classes:" << endl;
    printClasses();
}

//This function generates equivalence classes.
void erGen()
{
    int sum;
    int ecSize;
    int random;

    //Generate the size of the relation. No smaller than 2 and no bigger than 40.
    mSize = rand() % MAX;

    while (mSize < 2)
    {
        mSize = rand() % MAX;
    }

    //For debugging purposes.
    //cout << "Size: " << mSize << endl;

    //Generate number of equivalence relations no bigger than 8 but smaller than size and not
0;
    ecNum = rand() % mSize;

    while (ecNum == 0 || ecNum > 8)
    {
        ecNum = rand() % mSize;
    }

    //For debugging purposes.
    //cout << "ECNUM: " << ecNum << endl;

    //Generate the size of all ec classes. Must equal the orignal size.
    if (ecNum == 1)
    {
        equivSize[0] = mSize;
    }
    else
    {
```

```
    do
    {
           sum = 0;

           for (int count = 0; count < ecNum; count++)
           {
                  ecSize = rand() % mSize;

                  while (ecSize == 0)
                  {
                         ecSize = rand() % mSize;
                  }

                  sum = sum + ecSize;
                  equivSize[count] = ecSize;
           }

    } while (sum != mSize);
}

//For debugging purposes.
//for (int count = 0; count < ecNum; count++)
//{
//    cout << "EC: " << equivSize[count] << endl;
//}

//Make a new array to hold used values and intilize.
int* usedNum = new int[mSize];

for (int count = 0; count < mSize; count++)
{
    usedNum[count] = 0;
}

//Fill the equivalence classes.
for (int count = 0; count < ecNum; count++)
{
    for (int index = 0; index < equivSize[count]; index++)
    {
           random = rand() % mSize;

           while (usedNum[random] == 1)
           {
                  random = rand() % mSize;
           }

           eClasses[count][index] = random;
           usedNum[random] = 1;
    }
}
```

```
    delete(usedNum);
}

//This function fills a relation give equivalence classes.
void fillRelation()
{
    for (int number = 0; number < ecNum; number++)
    {
        for (int count = 0; count < equivSize[number]; count++)
        {
               for (int index = 0; index < equivSize[number]; index++)
               {
                       relation[eClasses[number][count]][eClasses[number][index]] = 1;
               }
        }
    }
}

//This function sorts and prints equivalence classes.
void printClasses()
{
    int temp;

    //Sort the generated classes (using bubble sort #lazy).
    for (int row = 0; row < ecNum; row++)
    {
        for (int count = 0; count < equivSize[row] - 1; count++)
        {
               for (int index = 0; index < equivSize[row] - 1; index++)
               {
                       if (eClasses[row][index] > eClasses[row][index + 1])
                       {
                              temp = eClasses[row][index];
                              eClasses[row][index] = eClasses[row][index + 1];
                              eClasses[row][index + 1] = temp;
                       }
               }
        }
    }

    //Print eClasses.
    for (int count = 0; count < ecNum; count++)
    {

        cout << "EC " << eClasses[count][0] << ": { ";

        for (int index = 0; index < equivSize[count]; index++)
        {
               cout << eClasses[count][index] << ", ";
        }
        cout << "}" << endl;
    }
}

//This function prints the matrix.
void printMatrix()
{
    for (int count = 0; count < mSize; count++)
    {

        for (int index = 0; index < mSize; index++)
        {
               cout << relation[count][index] << " ";
        }
        cout << endl;
    }
}
```

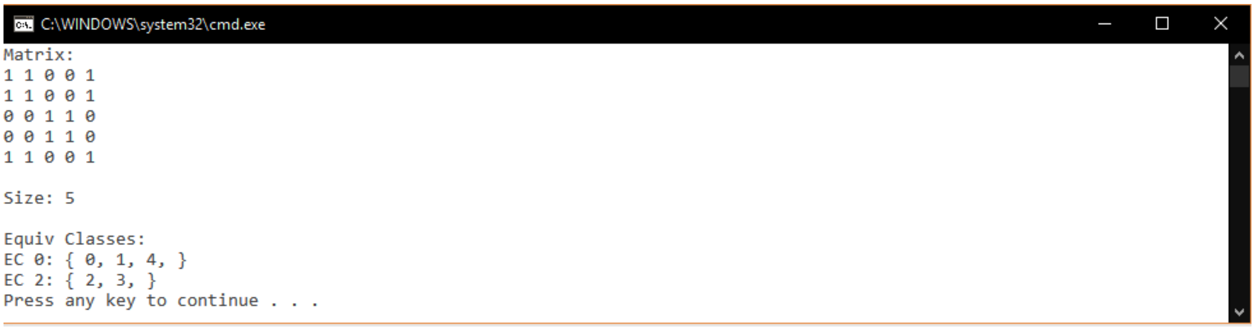
Output:

The following outputs are randomly generated equivalence relations of varying size.



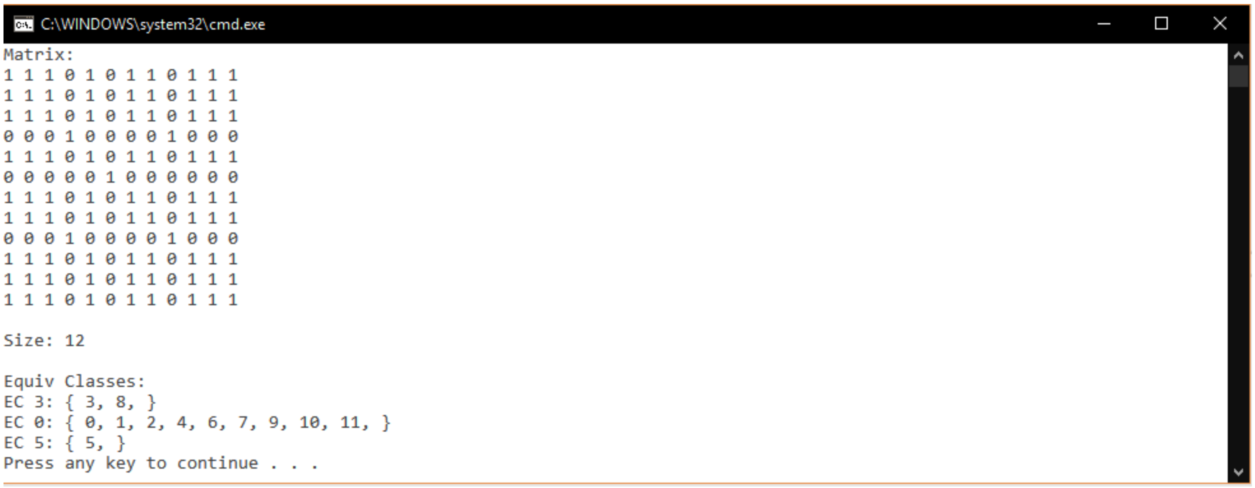
```
Matrix:
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1
1 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 1 1 1 0 1 0
0 0 1 0 0 0 1 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0 1

Size: 23

Equiv Classes:
EC 4: { 4, 5, 11, 12, 15, }
EC 0: { 0, 1, 3, 7, 9, 10, 13, 17, 18, 19, 21, }
EC 14: { 14, }
EC 2: { 2, 6, 8, 16, 20, 22, }
Press any key to continue . . .
```

(1 of 5)



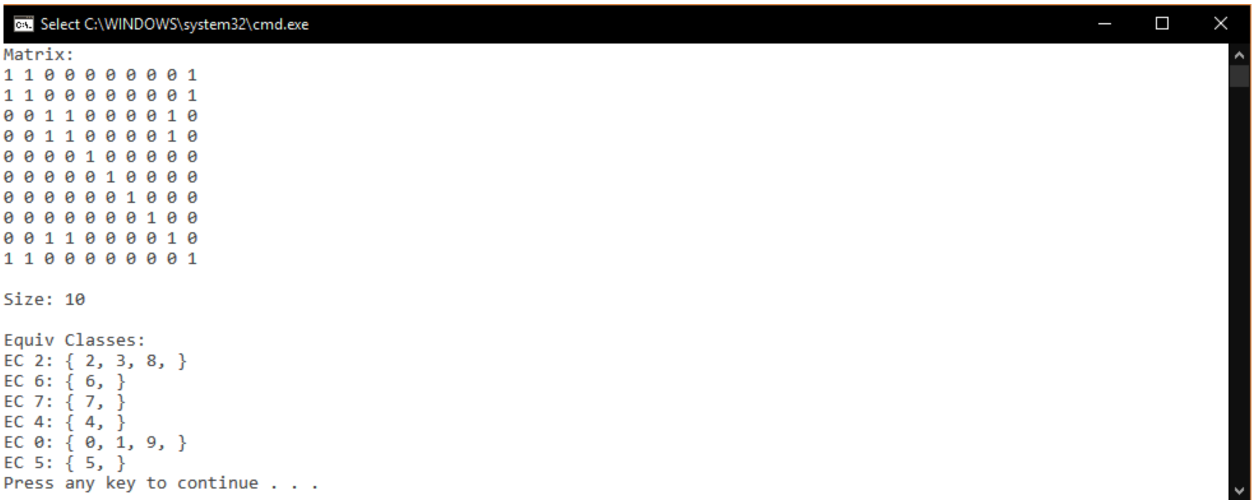
```
Matrix:
1 1 0 0 1
1 1 0 0 1
0 0 1 1 0
0 0 1 1 0
1 1 0 0 1

Size: 5

Equiv Classes:
EC 0: { 0, 1, 4, }
EC 2: { 2, 3, }
Press any key to continue . . .
```

(2 of 5)



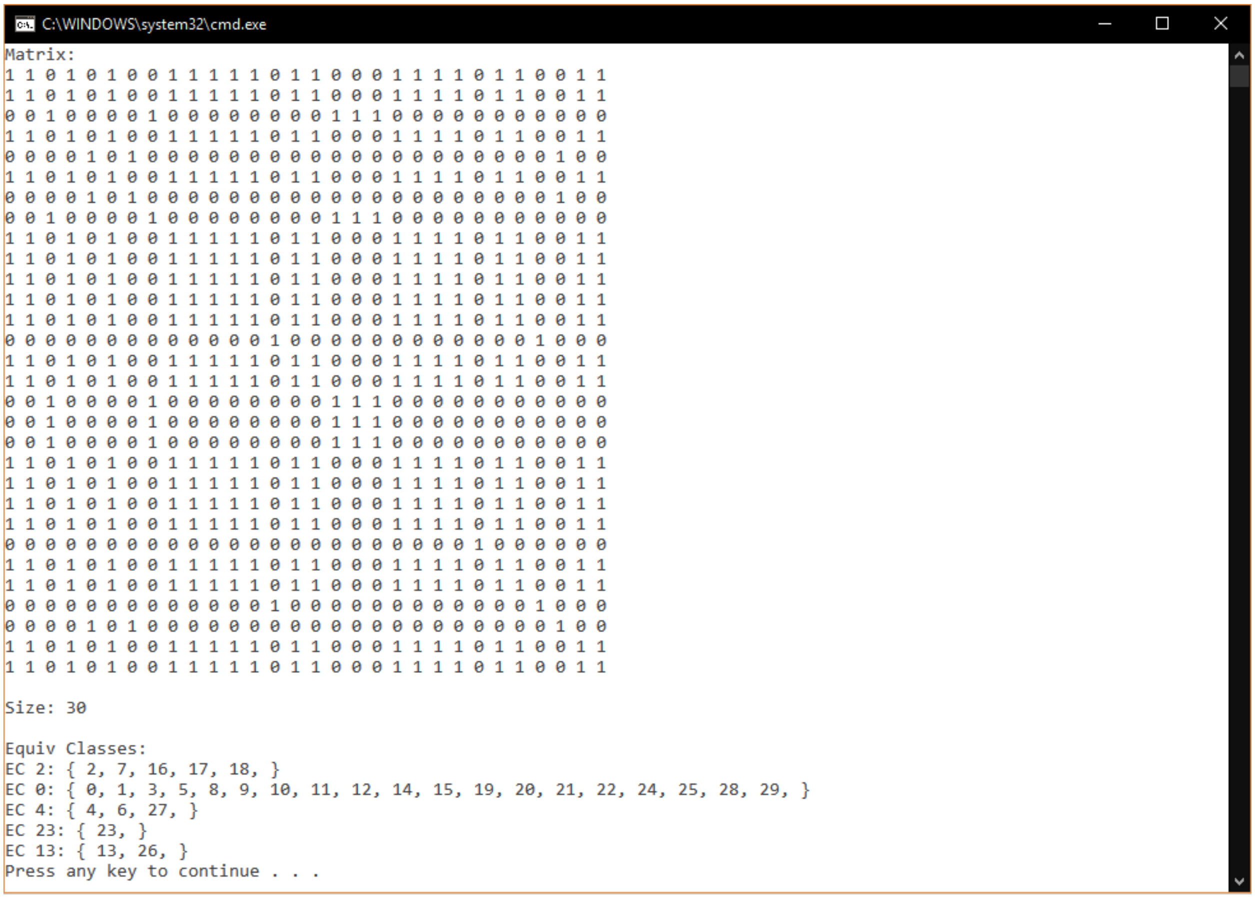
```
Matrix:
1 1 1 0 1 0 1 1 0 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1
0 0 0 1 0 0 0 0 1 0 0 0
1 1 1 0 1 0 1 1 0 1 1 1
0 0 0 0 0 1 0 0 0 0 0 0
1 1 1 0 1 0 1 1 0 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1
0 0 0 1 0 0 0 0 1 0 0 0
1 1 1 0 1 0 1 1 0 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1
1 1 1 0 1 0 1 1 0 1 1 1

Size: 12

Equiv Classes:
EC 3: { 3, 8, }
EC 0: { 0, 1, 2, 4, 6, 7, 9, 10, 11, }
EC 5: { 5, }
Press any key to continue . . .
```

(3 of 5)



```
Matrix:
1 1 0 0 0 0 0 0 0 1
1 1 0 0 0 0 0 0 0 1
0 0 1 1 0 0 0 0 1 0
0 0 1 1 0 0 0 0 1 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 1 1 0 0 0 0 1 0
1 1 0 0 0 0 0 0 0 1

Size: 10

Equiv Classes:
EC 2: { 2, 3, 8, }
EC 6: { 6, }
EC 7: { 7, }
EC 4: { 4, }
EC 0: { 0, 1, 9, }
EC 5: { 5, }
Press any key to continue . . .
```

(4 of 5)



```
Matrix:
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1

Size: 30

Equiv Classes:
EC 2: { 2, 7, 16, 17, 18, }
EC 0: { 0, 1, 3, 5, 8, 9, 10, 11, 12, 14, 15, 19, 20, 21, 22, 24, 25, 28, 29, }
EC 4: { 4, 6, 27, }
EC 23: { 23, }
EC 13: { 13, 26, }
Press any key to continue . . .
```

(5 of 5)

Conclusion

Programming is such a joy! Also, challenging at times. The first part of this assignment was relatively simple and I implemented it in only a couple hours. However, the extra credit assignment proved perplexing. I tried several different methods of attack until settling on the one seen in this assignment. Can’t wait to see what's next!