МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №3.1

по дисциплине: Дискретная математика
по теме: «Отношения»

Выполнил: студент группы ПВ-223
Мелехов Артём Дмитриевич

Проверили:
доц. Рязанов Юрий Дмитриевич
ст. преп. Бондаренко Татьяна Владимировна

Белгород 2023 г.

Лабораторная работа №3.1

Цель работы: изучить способы задания отношений, операции над отношениями и свойства отношений, научиться программно реализовывать операции и определять свойства отношений.

Содержание отчета:

Решения заданий. Для каждого задания указаны:

– Название задания;

– Условие задания;

– Решение задания;

Вывод;

Полный листинг программы (с названиями директорий и файлов) и ссылка на репозиторий GitHub с выполненной работой.

Вариант №7.

а) $A = \{(x, y) | x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $x + y \vdots 3\}$

$B = \{(x, y) | x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $(2 < x < 8$ или $2 < y < 8)\}$

$C = \{(x, y) | x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $x^2 + y^2 < 100\}$

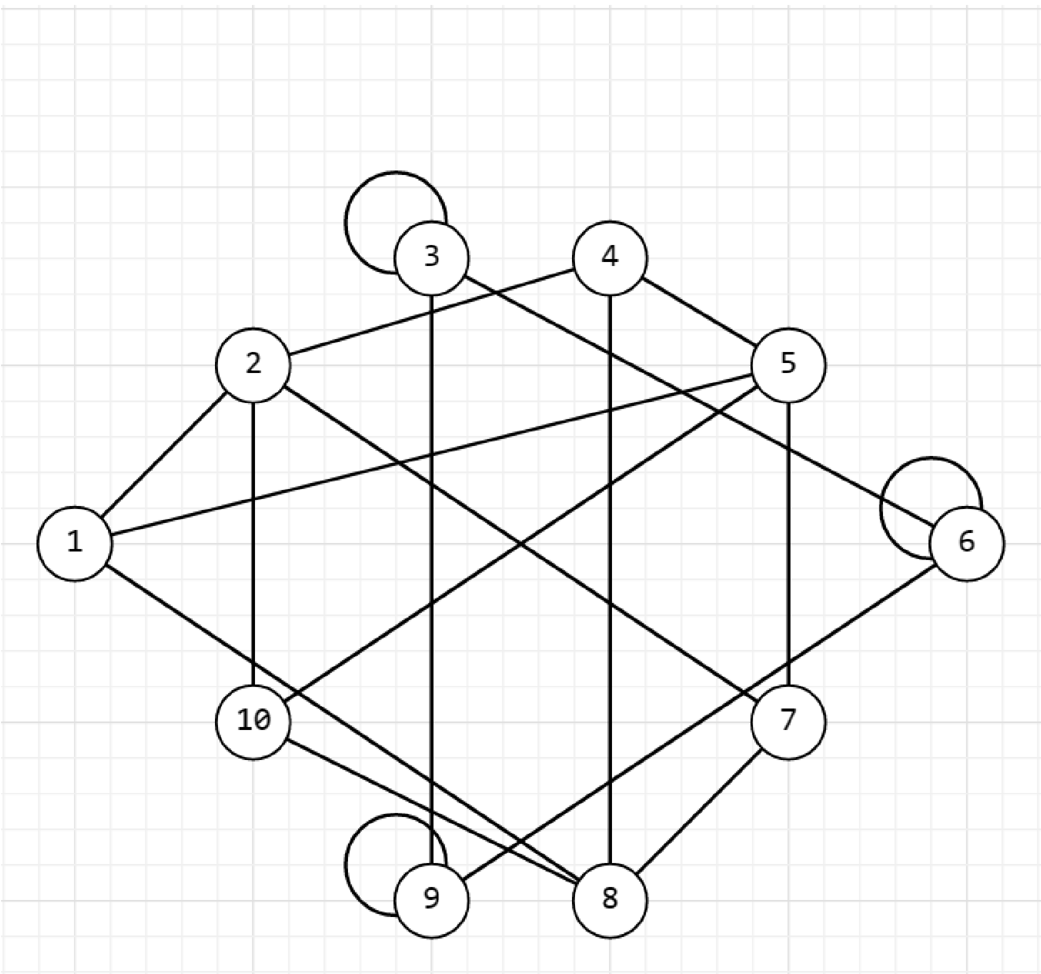
б) $D = A^2 - B \cup A^{-1} \circ C$

Задание №1.

Пункт 1:

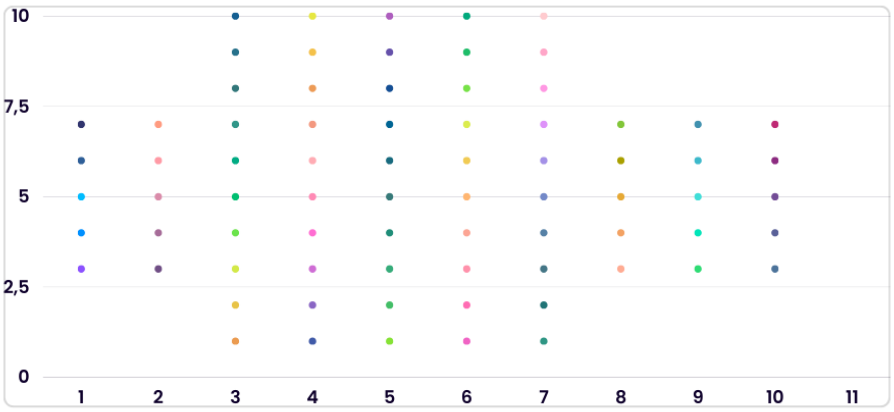
Представить отношения графиком, графом и матрицей.

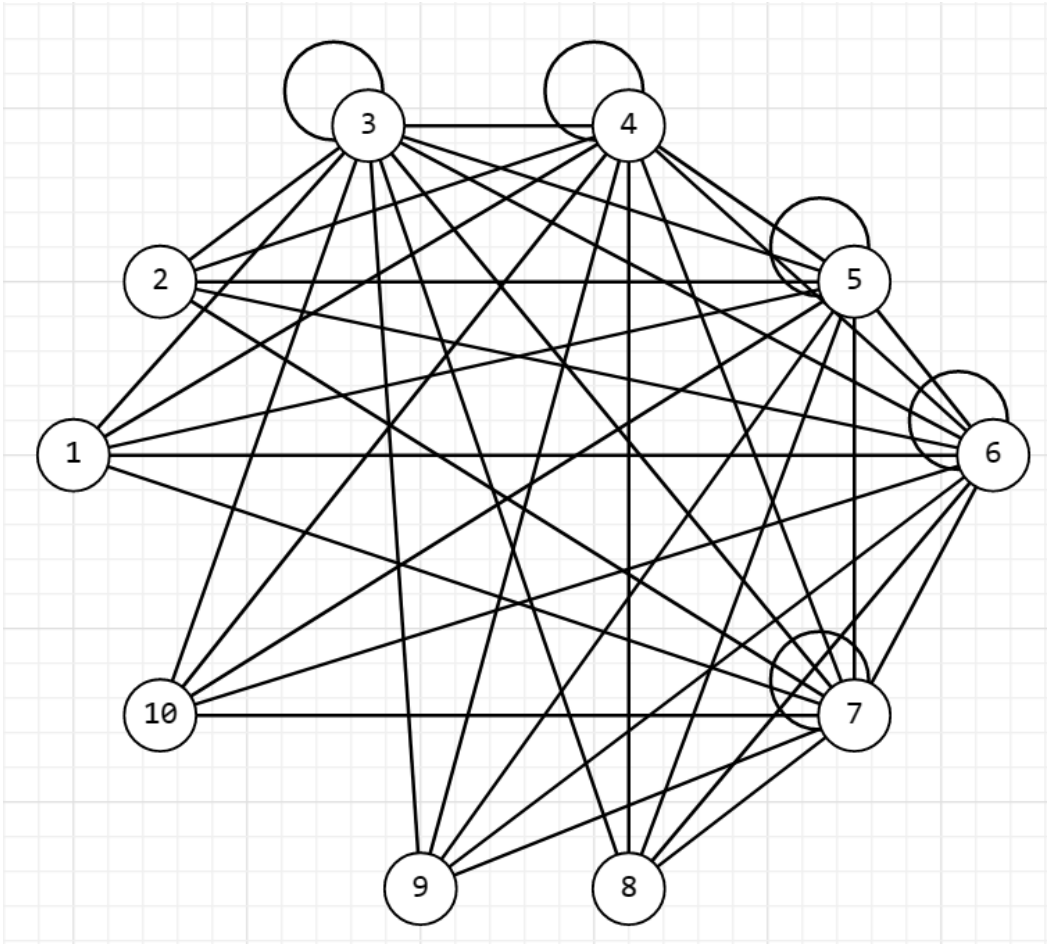
Решение:



$$A = \begin{vmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{vmatrix}$$

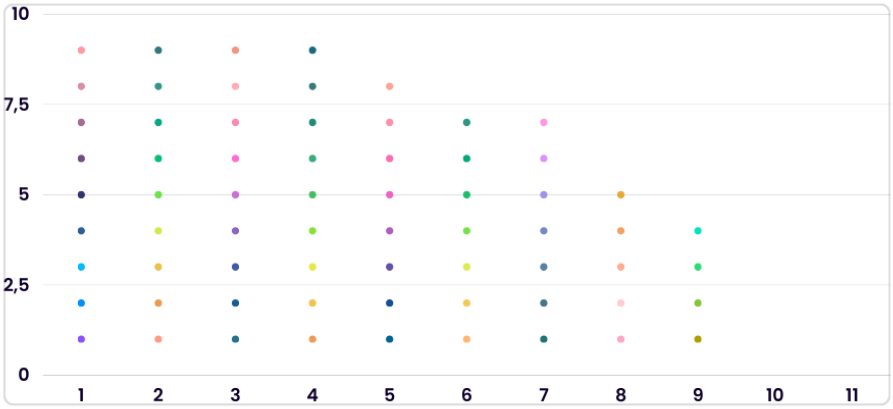
Отношение B

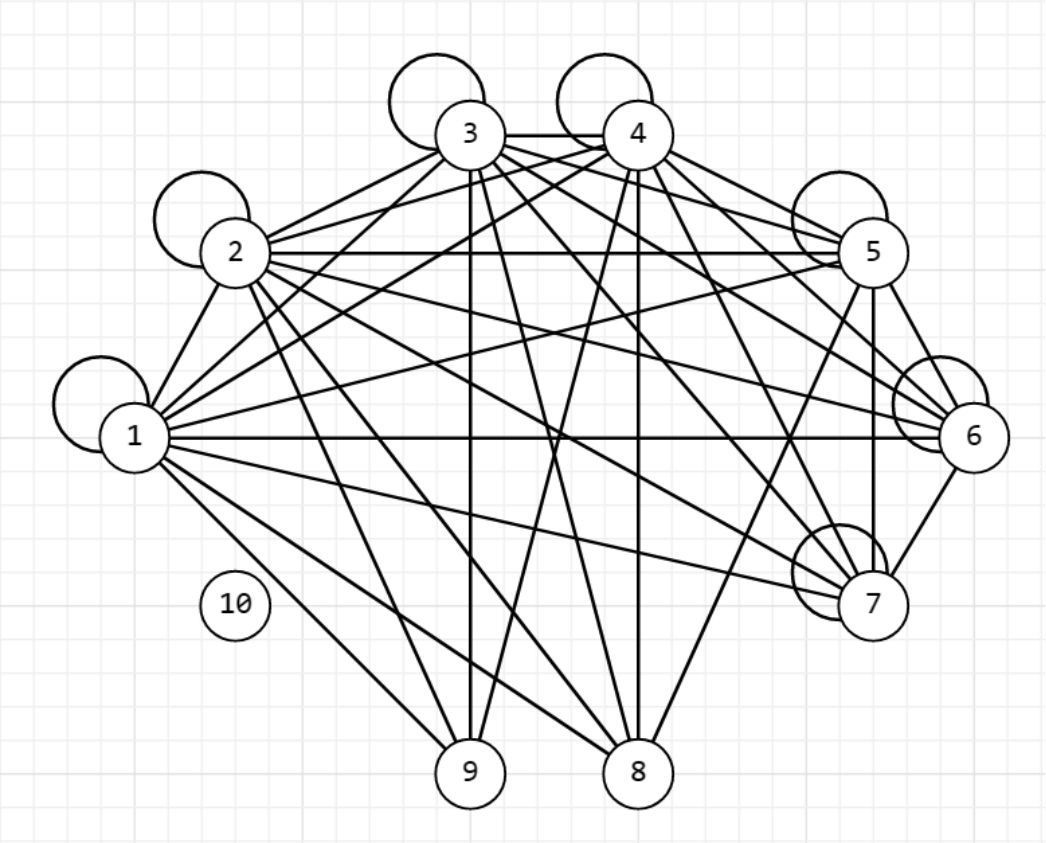




$$B = \begin{vmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{vmatrix}$$

Отношение C





$$C = \begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

Пункт 2:

Вычислить значения выражения при заданных отношениях.

Решение:

$$D = A^2 - B \cup A^{-1} \, o \, C$$

$$A^2 = \begin{vmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{vmatrix}$$

$$A^{-1} = \begin{vmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$A^2 - B = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{vmatrix}$$

$$A^{-1}\,o\,C \;=\begin{vmatrix} 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0 \end{vmatrix}$$

$$D\;=\begin{vmatrix} 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0\\ 1&1&1&1&1&1&1&1&1&0 \end{vmatrix}$$

Пункт 3:

Написать программы, формирующие матрицы заданных отношений.

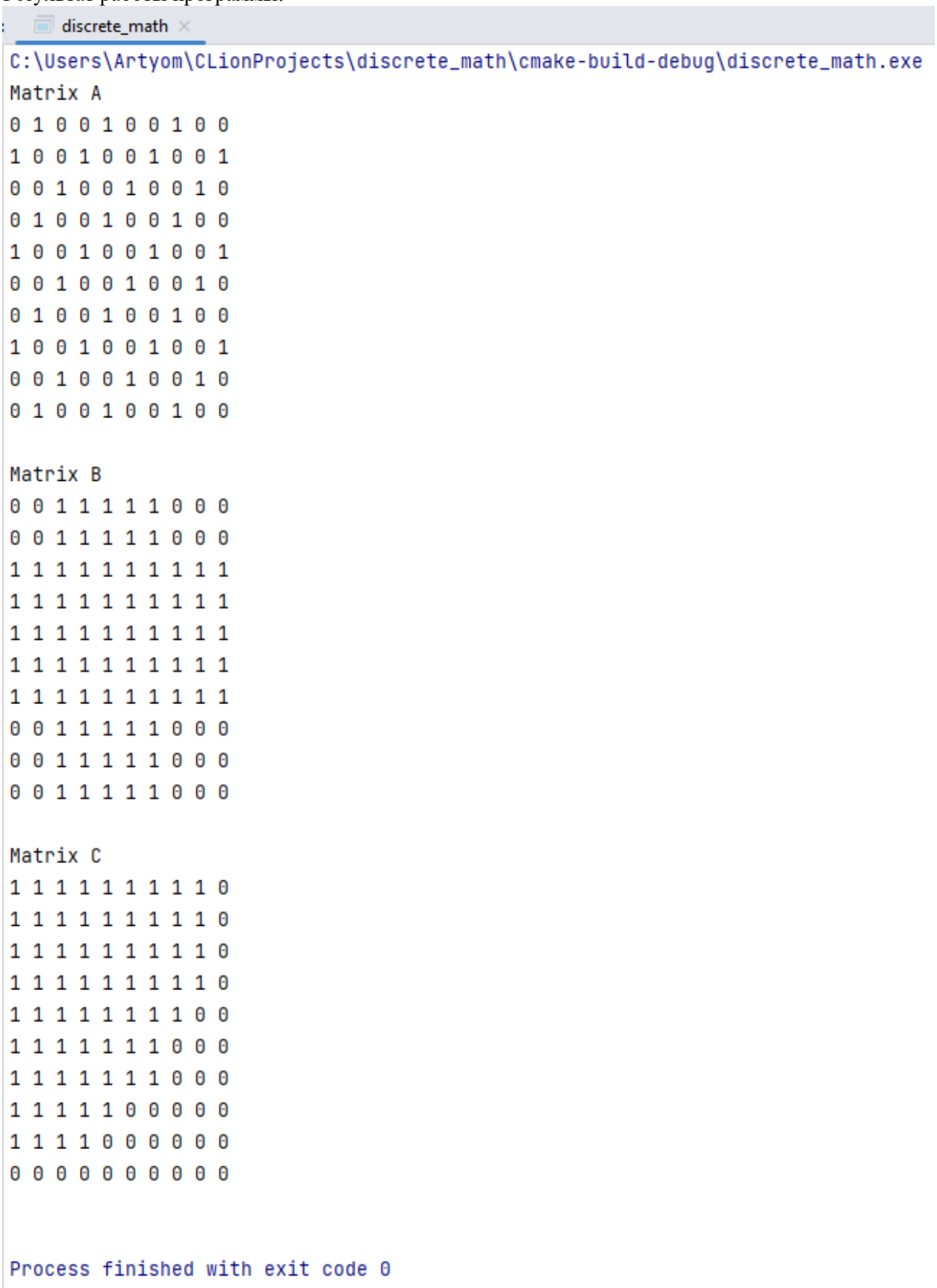
Код программы:

```
void matrix_a(Matrix a) {
    for (long long x = 0; x < a.n_cols; x++)
        for (long long y = 0; y < a.n_rows; y++)
            a.values[x][y] = !(((x + 1) + (y + 1)) % 3) ? 1 : 0;
}


void matrix_b(Matrix b) {
    for (long long x = 0; x < b.n_cols; x++)
        for (long long y = 0; y < b.n_rows; y++)
            b.values[x][y] = (2 < (x + 1) && (x + 1) < 8) || (2
< (y + 1) && (y + 1) < 8) ? 1 : 0;
}
```

```
void matrix_c(Matrix c) {
    for (long long x = 0; x < c.n_cols; x++)
        for (long long y = 0; y < c.n_rows; y++)
            c.values[x][y] = ((x + 1) * (x + 1) + (y + 1) * (y +
1)) < 100 ? 1 : 0;
}
```

Результат работы программы:



Пункт 4:

Программно реализовать операции над отношениями.

Код программы:

```
//
// Created by Artyom on 28.10.2023.
//

#include "matrix.h"

Matrix get_matrix(const long long n_rows, const long long
n_cols) {
    long long **a = (long long **) malloc(sizeof(long long *) *
n_rows);

    for (size_t i = 0; i < n_rows; i++)
        a[i] = (long long *) malloc(sizeof(long long) * n_cols);

    Matrix matrix = {
            a,
            n_rows,
            n_cols
    };
    return matrix;
}

void free_matrix(Matrix a) {
    for (size_t i = 0; i < a.n_rows; i++)
        free(a.values[i]);

    free(a.values);
}

void output_matrix(Matrix a) {
    for (size_t i = 0; i < a.n_rows; i++) {
        for (size_t j = 0; j < a.n_cols; j++)
            printf("%lld ", a.values[i][j]);

        printf("\n");
    }

    printf("\n");
}
```

```c
int equality(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            if (a->values[i][j] != b->values[i][j])
                return 0;

    return 1;
}

int inclusion(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            if (a->values[i][j] && !b->values[i][j])
                return 0;

    return 1;
}

int strict_inclusion(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    int is_strict = 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++) {
            if (a->values[i][j] && !b->values[i][j])
                return 0;

            if (!a->values[i][j] && b->values[i][j])
                is_strict = 1;
        }

    return is_strict;
}
```

```
Matrix unification(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] || b-
>values[i][j];

    return result;
}

Matrix intersection(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] && b-
>values[i][j];

    return result;
}

Matrix difference(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] && !b-
>values[i][j];

    return result;
}

Matrix symmetric_difference(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] ^ b-
>values[i][j];

    return result;
}
```

```
Matrix addition(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] + b-
>values[i][j];

    return result;
}

void appeal(Matrix *original_matrix, Matrix
*matrix_for_operations) {
    // Проверка на совместимость матриц
    if (original_matrix->n_rows != matrix_for_operations->n_cols
||
        original_matrix->n_cols != matrix_for_operations-
>n_rows) {
        printf("The matrices must be the same size\n");

        return;
    }

    for (long long x = 0; x < original_matrix->n_cols; x++)
        for (long long y = 0; y < original_matrix->n_rows; y++)
            matrix_for_operations->values[x][y] =
original_matrix->values[y][x];
}

void composition(Matrix *Matrix1, Matrix *Matrix2, Matrix
*MatrixForOperations) {
    // Проверка на совместимость матриц
    assert(Matrix1->n_cols == Matrix2->n_rows && Matrix1->n_rows
== MatrixForOperations->n_rows &&
           Matrix2->n_cols == MatrixForOperations->n_cols);

    for (long long x = 0; x < Matrix1->n_rows; x++)
        for (long long y = 0; y < Matrix2->n_cols; y++) {
            MatrixForOperations->values[x][y] = 0;

            for (long long z = 0; z < Matrix1->n_cols; z++)
                if ((MatrixForOperations->values[x][y] ||
(Matrix1->values[x][z] &&

Matrix2->values[z][y])))
                    MatrixForOperations->values[x][y] = 1;
                else
                    MatrixForOperations->values[x][y] = 0;
        }
}
```

Пункт 5:

Написать программу, вычисляющую значение выражения и вычислить его при заданных отношениях.

Код программы:

```
void matrix_d(Matrix a, Matrix b, Matrix c, Matrix *d) {
    Matrix new_a1 = copy_matrix(&a);
    Matrix new_b1 = copy_matrix(&b);
    Matrix a_2 = intersection(&new_a1, &new_b1);

    Matrix new_a2 = copy_matrix(&a);
    Matrix a_1 = get_matrix(a.n_cols, a.n_rows);
    appeal(&new_a2, &a_1);


    Matrix new_b2 = copy_matrix(&b);
    Matrix a_2_b = difference(&a_2, &new_b2);

    Matrix new_c1 = copy_matrix(&c);
    Matrix a_1_o_c = get_matrix(a.n_rows, a.n_cols);
    composition(&a_1, &new_c1, &a_1_o_c);

    *d = unification(&a_2_b, &a_1_o_c);

    free_matrix(new_a1);
    free_matrix(new_b1);
    free_matrix(new_a2);
    free_matrix(new_b2);
    free_matrix(new_c1);
}
```

Результат работы программы:

```
Matrix D
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0


Process finished with exit code 0
```

Задание №2.

Пункт 1, 2:

Определить основные свойства отношений. Определить, являются ли заданные отношения отношениями толерантности, эквивалентности и порядка.

Решение:

$$A = \{(x,y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } x + y \vdots 3\}$$

Не рефлексивно, симметрично, не транзитивно, не связно, не полно, не толерантно, не эквивалентно, не порядок, не строгий порядок, не линейный порядок, не строгий линейный порядок.

$$B = \{(x,y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } (2 < x < 8 \text{ или } 2 < y < 8)\}$$

Не рефлексивно, симметрично, не транзитивно, не связно, не полно, не толерантно, не эквивалентно, не порядок, не строгий порядок, не линейный порядок, не строгий линейный порядок.

$$C = \{(x,y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } x^2 + y^2 < 100\}$$

Не рефлексивно, симметрично, не транзитивно, не связно, не полно, не толерантно, не эквивалентно, не порядок, не строгий порядок, не линейный порядок, не строгий линейный порядок.

Пункт 3:

Написать программу, определяющую свойства отношения, в том числе толерантности, эквивалентности и порядка, и определить свойства отношений.

Решение:

```
bool is_reflexive(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        if (matrix->values[i][i] != 1)
            return 0;

    return 1;
}

bool is_symmetric(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            if (matrix->values[i][j] != matrix->values[j][i])
                return 0;

    return 1;
}
```

```
bool is_transitive(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            for (long long k = 0; k < matrix->n_cols; k++)
                if (matrix->values[i][j] && matrix->values[j][k]
&& !matrix->values[i][k])
                    return 0;

    return 1;
}

bool is_connected(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            // Проверка связности
            if (i != j && matrix->values[i][j] == 0 && matrix-
>values[j][i] == 0)
                return 0;

    return 1;
}

bool is_complete(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            if (i != j && matrix->values[i][j] == 0)
                return 0;

    return 1;
}

bool is_tolerant(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++) {
        if (matrix->values[i][i] != 1)
            return 0;

        for (long long j = 0; j < matrix->n_cols; j++)
            if (i != j && matrix->values[i][j] != 0 && matrix-
>values[j][i] != 1)
                return 0;
    }

    return 1;
}

bool is_equivalent(Matrix *matrix) {
    return is_reflexive(matrix) && is_symmetric(matrix) &&
is_transitive(matrix);
}
```

```
bool is_order(Matrix *matrix) {
    return is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix);
}

bool is_strict_order(Matrix *matrix) {
    return !is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix);
}

bool is_linear_order(Matrix *matrix) {
    return is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix) &&
           is_connected(matrix);
}

bool is_strict_linear_order(Matrix *matrix) {
    return !is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix) &&
           is_connected(matrix);
}

void all_relationship_properties(Matrix *m) {
    printf("Reflexive: %d\n", is_reflexive(m));
    printf("Symmetric: %d\n", is_symmetric(m));
    printf("Transitive: %d\n", is_transitive(m));
    printf("Connected: %d\n", is_connected(m));
    printf("Complete: %d\n", is_complete(m));
    printf("Tolerant: %d\n", is_tolerant(m));
    printf("Equivalent: %d\n", is_equivalent(m));
    printf("Order: %d\n", is_order(m));
    printf("Strict order: %d\n", is_strict_order(m));
    printf("Linear order: %d\n", is_linear_order(m));
    printf("Strict linear order: %d\n",
is_strict_linear_order(m));
}
```

Результат работы программы:

```
Matrix A
0 1 0 0 1 0 0 1 0 0
1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0
0 1 0 0 1 0 0 1 0 0
1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0
0 1 0 0 1 0 0 1 0 0
1 0 0 1 0 0 1 0 0 1
0 0 1 0 0 1 0 0 1 0
0 1 0 0 1 0 0 1 0 0

Reflexive: 0
Symmetric: 1
Transitive: 0
Connected: 0
Complete: 0
Tolerant: 0
Equivalent: 0
Order: 0
Strict order: 0
Linear order: 0
Strict linear order: 0
```

```
Matrix B
0 0 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 0 0
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
0 0 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 0 0
0 0 1 1 1 1 1 0 0 0

Reflexive: 0
Symmetric: 1
Transitive: 0
Connected: 0
Complete: 0
Tolerant: 0
Equivalent: 0
Order: 0
Strict order: 0
Linear order: 0
Strict linear order: 0
```

```
Matrix C
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0

Reflexive: 0
Symmetric: 1
Transitive: 0
Connected: 0
Complete: 0
Tolerant: 0
Equivalent: 0
Order: 0
Strict order: 0
Linear order: 0
Strict linear order: 0
```

Вывод: в ходе выполнения лабораторной работы были изучены способы задания отношений, операции над отношениями и свойства отношений. Также были программно реализовываны операции и определены свойства отношений.

Ссылка на GitHub (lab3_1): https://github.com/SStaryi/discrete_math

Полный листинг программы:

Файл `matrix/matrix.c`:

```c
//
// Created by Artyom on 28.10.2023.
//

#include "matrix.h"
```

```
Matrix get_matrix(const long long n_rows, const long long
n_cols) {
    long long **a = (long long **) malloc(sizeof(long long *) *
n_rows);

    for (size_t i = 0; i < n_rows; i++)
        a[i] = (long long *) malloc(sizeof(long long) * n_cols);

    Matrix matrix = {
            a,
            n_rows,
            n_cols
    };
    return matrix;
}

void free_matrix(Matrix a) {
    for (size_t i = 0; i < a.n_rows; i++)
        free(a.values[i]);

    free(a.values);
}

void output_matrix(Matrix a) {
    for (size_t i = 0; i < a.n_rows; i++) {
        for (size_t j = 0; j < a.n_cols; j++)
            printf("%lld ", a.values[i][j]);

        printf("\n");
    }

    printf("\n");
}

int equality(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            if (a->values[i][j] != b->values[i][j])
                return 0;

    return 1;
}
```

```
int inclusion(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            if (a->values[i][j] && !b->values[i][j])
                return 0;

    return 1;
}

int strict_inclusion(Matrix *a, Matrix *b) {
    if (a->n_rows != b->n_rows || a->n_cols != b->n_cols)
        return 0;

    int is_strict = 0;

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++) {
            if (a->values[i][j] && !b->values[i][j])
                return 0;

            if (!a->values[i][j] && b->values[i][j])
                is_strict = 1;
        }

    return is_strict;
}

Matrix unification(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] || b-
>values[i][j];

    return result;
}
```

```
Matrix intersection(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] && b-
>values[i][j];

    return result;
}

Matrix difference(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] && !b-
>values[i][j];

    return result;
}

Matrix symmetric_difference(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] ^ b-
>values[i][j];

    return result;
}

Matrix addition(Matrix *a, Matrix *b) {
    assert(a->n_rows == b->n_rows && a->n_cols == b->n_cols);

    Matrix result = get_matrix(a->n_rows, a->n_cols);

    for (long long i = 0; i < a->n_rows; i++)
        for (long long j = 0; j < a->n_cols; j++)
            result.values[i][j] = a->values[i][j] + b-
>values[i][j];

    return result;
}
```

```
void appeal(Matrix *original_matrix, Matrix
*matrix_for_operations) {
    // Проверка на совместимость матриц
    assert(original_matrix->n_rows == matrix_for_operations-
>n_cols &&
           original_matrix->n_cols == matrix_for_operations-
>n_rows);

    for (long long x = 0; x < original_matrix->n_cols; x++)
        for (long long y = 0; y < original_matrix->n_rows; y++)
            matrix_for_operations->values[x][y] =
original_matrix->values[y][x];
}

void composition(Matrix *matrix_1, Matrix *matrix_2, Matrix
*matrix_for_operations) {
    // Проверка на совместимость матриц
    assert(matrix_1->n_cols == matrix_2->n_rows && matrix_1-
>n_rows == matrix_for_operations->n_rows &&
           matrix_2->n_cols == matrix_for_operations->n_cols);

    for (long long x = 0; x < matrix_1->n_rows; x++)
        for (long long y = 0; y < matrix_2->n_cols; y++) {
            matrix_for_operations->values[x][y] = 0;

            for (long long z = 0; z < matrix_1->n_cols; z++)
                if ((matrix_for_operations->values[x][y] ||
(matrix_1->values[x][z] &&

matrix_2->values[z][y])))
                    matrix_for_operations->values[x][y] = 1;
                else
                    matrix_for_operations->values[x][y] = 0;
        }
}
```

```
Matrix copy_matrix(Matrix *original_matrix) {
    // Создание новой матрицы
    Matrix new_matrix = get_matrix(original_matrix->n_cols,
original_matrix->n_rows);

    new_matrix.n_rows = original_matrix->n_rows;
    new_matrix.n_cols = original_matrix->n_cols;
    new_matrix.values = malloc(new_matrix.n_rows * sizeof(long
long *));

    for (long long i = 0; i < new_matrix.n_rows; i++) {
        new_matrix.values[i] = malloc(new_matrix.n_cols *
sizeof(long long));

        for (long long j = 0; j < new_matrix.n_cols; j++)
            new_matrix.values[i][j] = original_matrix-
>values[i][j];
    }

    return new_matrix;
}
```

Файл matrix/matrix.h:

```
//
// Created by Artyom on 28.10.2023.
//

#ifndef DISCRETE_MATH_MATRIX_H
#define DISCRETE_MATH_MATRIX_H

#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

// Структура описывающая матрицу
typedef struct Matrix {
    long long **values; // значения
    long long n_rows;   // кол-во строк
    long long n_cols;   // кол-во столбцов
} Matrix;

// Выделение памяти для матрицы a размера m * n
Matrix get_matrix(const long long n_rows, const long long
n_cols);

// Освобождение памяти выделенной под матрицу
void free_matrix(Matrix a);

// Вывод матрицы a размера n * m
```

```
void output_matrix(Matrix a);
// Функция для проверки равенства двух матриц
int equality(Matrix *a, Matrix *b);

// Функция для проверки включения одной матрицы в другую
int inclusion(Matrix *a, Matrix *b);

// Функция для проверки строгого включения одной матрицы в
другую
int strict_inclusion(Matrix *a, Matrix *b);

// Функция для объединения двух матриц
Matrix unification(Matrix *a, Matrix *b);

// Функция для пересечения двух матриц
Matrix intersection(Matrix *a, Matrix *b);

// Функция для разности двух матриц
Matrix difference(Matrix *a, Matrix *b);

// Функция для симметричной разности двух матриц
Matrix symmetric_difference(Matrix *a, Matrix *b);

// Функция для сложения двух матриц
Matrix addition(Matrix *a, Matrix *b);

// Функция для обращения A^-1
void appeal(Matrix *original_matrix, Matrix
*matrix_for_operations);

// Функция для композиции
void composition(Matrix *matrix_1, Matrix *matrix_2, Matrix
*matrix_for_operations);

// Функция для копирования матриц
Matrix copy_matrix(Matrix *original_matrix);

#endif //DISCRETE_MATH_MATRIX_H
```

Файл lab3_1/lab3_1.c:

```
//
// Created by Artyom on 26.10.2023.
//

#include "lab3_1.h"

void matrix_a(Matrix a) {
    for (long long x = 0; x < a.n_cols; x++)
        for (long long y = 0; y < a.n_rows; y++)
            a.values[x][y] = !(((x + 1) + (y + 1)) % 3) ? 1 : 0;
}
```

```
void matrix_b(Matrix b) {
    for (long long x = 0; x < b.n_cols; x++)
        for (long long y = 0; y < b.n_rows; y++)
            b.values[x][y] = (2 < (x + 1) && (x + 1) < 8) || (2
< (y + 1) && (y + 1) < 8) ? 1 : 0;
}

void matrix_c(Matrix c) {
    for (long long x = 0; x < c.n_cols; x++)
        for (long long y = 0; y < c.n_rows; y++)
            c.values[x][y] = ((x + 1) * (x + 1) + (y + 1) * (y +
1)) < 100 ? 1 : 0;
}

void matrix_d(Matrix a, Matrix b, Matrix c, Matrix *d) {
    Matrix new_a1 = copy_matrix(&a);
    Matrix new_b1 = copy_matrix(&b);
    Matrix a_2 = intersection(&new_a1, &new_b1);

    Matrix new_a2 = copy_matrix(&a);
    Matrix a_1 = get_matrix(a.n_cols, a.n_rows);
    appeal(&new_a2, &a_1);


    Matrix new_b2 = copy_matrix(&b);
    Matrix a_2_b = difference(&a_2, &new_b2);

    Matrix new_c1 = copy_matrix(&c);
    Matrix a_1_o_c = get_matrix(a.n_rows, a.n_cols);
    composition(&a_1, &new_c1, &a_1_o_c);

    *d = unification(&a_2_b, &a_1_o_c);

    free_matrix(new_a1);
    free_matrix(new_b1);
    free_matrix(new_a2);
    free_matrix(new_b2);
    free_matrix(new_c1);
}

bool is_reflexive(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        if (matrix->values[i][i] != 1)
            return 0;

    return 1;
}
```

```
bool is_symmetric(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            if (matrix->values[i][j] != matrix->values[j][i])
                return 0;

    return 1;
}

bool is_transitive(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            for (long long k = 0; k < matrix->n_cols; k++)
                if (matrix->values[i][j] && matrix->values[j][k]
&& !matrix->values[i][k])
                    return 0;

    return 1;
}

bool is_connected(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            // Проверка связности
            if (i != j && matrix->values[i][j] == 0 && matrix-
>values[j][i] == 0)
                return 0;

    return 1;
}

bool is_complete(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++)
        for (long long j = 0; j < matrix->n_cols; j++)
            if (i != j && matrix->values[i][j] == 0)
                return 0;

    return 1;
}

bool is_tolerant(Matrix *matrix) {
    for (long long i = 0; i < matrix->n_rows; i++) {
        if (matrix->values[i][i] != 1)
            return 0;

        for (long long j = 0; j < matrix->n_cols; j++)
            if (i != j && matrix->values[i][j] != 0 && matrix-
>values[j][i] != 1)
                return 0;
```

```
    }

    return 1;
}
bool is_equivalent(Matrix *matrix) {
    return is_reflexive(matrix) && is_symmetric(matrix) &&
is_transitive(matrix);
}

bool is_order(Matrix *matrix) {
    return is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix);
}

bool is_strict_order(Matrix *matrix) {
    return !is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix);
}

bool is_linear_order(Matrix *matrix) {
    return is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix) &&
           is_connected(matrix);
}

bool is_strict_linear_order(Matrix *matrix) {
    return !is_reflexive(matrix) && !is_symmetric(matrix) &&
is_transitive(matrix) &&
           is_connected(matrix);
}

void all_relationship_properties(Matrix *m) {
    printf("Reflexive: %d\n", is_reflexive(m));
    printf("Symmetric: %d\n", is_symmetric(m));
    printf("Transitive: %d\n", is_transitive(m));
    printf("Connected: %d\n", is_connected(m));
    printf("Complete: %d\n", is_complete(m));
    printf("Tolerant: %d\n", is_tolerant(m));
    printf("Equivalent: %d\n", is_equivalent(m));
    printf("Order: %d\n", is_order(m));
    printf("Strict order: %d\n", is_strict_order(m));
    printf("Linear order: %d\n", is_linear_order(m));
    printf("Strict linear order: %d\n",
is_strict_linear_order(m));
}
```

Файл lab3_1/lab3_1.h:

```
//
// Created by Artyom on 26.10.2023.
//

#ifndef DISCRETE_MATH_LAB3_1_H
#define DISCRETE_MATH_LAB3_1_H
```

```c
#include <stdbool.h>

#include "../matrix/matrix.h"

void matrix_a(Matrix a);

void matrix_b(Matrix b);

void matrix_c(Matrix c);

void matrix_d(Matrix a, Matrix b, Matrix c, Matrix *d);

// Функция для проверки рефлексивности матрицы
bool is_reflexive(Matrix *matrix);

// Функция для проверки симметричности матрицы
bool is_symmetric(Matrix *matrix);

// Функция для проверки транзитивности матрицы
bool is_transitive(Matrix *matrix);

// Функция для проверки свойства связности
bool is_connected(Matrix *matrix);

// Функция для проверки полноты матрицы
bool is_complete(Matrix *matrix);

// Функция для проверки толерантности матрицы
bool is_tolerant(Matrix *matrix);

// Функция для проверки эквивалентности матрицы
bool is_equivalent(Matrix *matrix);

// Функция для проверки свойства отношения порядка
bool is_order(Matrix *matrix);

// Функция для проверки свойства строгого порядка
bool is_strict_order(Matrix *matrix);

// Функция для проверки свойства линейного порядка
bool is_linear_order(Matrix *matrix);

// Функция для проверки свойства строгого линейного порядка
bool is_strict_linear_order(Matrix *matrix);

void all_relationship_properties(Matrix *m);

#endif //DISCRETE_MATH_LAB3_1_H
```

Файл main.c:

```c
#include <stdio.h>

#include "lab3_1/lab3_1.h"
#include "matrix/matrix.h"

int main() {
    long long size = 10;

    Matrix a = get_matrix(size, size);
    matrix_a(a);
    printf("Matrix A\n");
    output_matrix(a);

    all_relationship_properties(&a);
    printf("\n");

    Matrix b = get_matrix(size, size);
    matrix_b(b);
    printf("Matrix B\n");
    output_matrix(b);

    all_relationship_properties(&b);
    printf("\n");

    Matrix c = get_matrix(size, size);
    matrix_c(c);
    printf("Matrix C\n");
    output_matrix(c);

    all_relationship_properties(&c);
    printf("\n");

    Matrix d = get_matrix(size, size);
    matrix_d(a, b, c, &d);
    printf("Matrix D\n");
    output_matrix(d);

    free_matrix(a);
    free_matrix(b);
    free_matrix(c);
    free_matrix(d);

    return 0;
}
```