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
тема: «Отношения и их свойства»

Выполнил ст. группы ПВ-223
Игнатьев Артур Олегович

Проверил:
Резанов Ю.Д.

Белгород, 2023 г.

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

Содержание отчета:

- Тема лабораторной работы;
- Цель лабораторной работы;
- Условия задач и их решение;
- Вывод.

Задания

Часть 1. Операции над отношениями

1.1. Представить отношения графиком, графом и матрицей.

1.2. Вычислить значение выражения при заданных отношениях.

1.3. Написать программы, формирующие матрицы заданных отношений.

1.4. Программно реализовать операции над отношениями.

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

Часть 2. Свойства отношений

2.1. Определить основные свойства отношений.

2.2. Определить, являются ли заданные отношения отношениями толерантности, эквивалентности и порядка.

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

$A = \{(x, y) \,|\, x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $(x > y$ или $y > 7)\}$

$B = \{(x, y) \,|\, x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и $10x + y$ кратно $3\}$

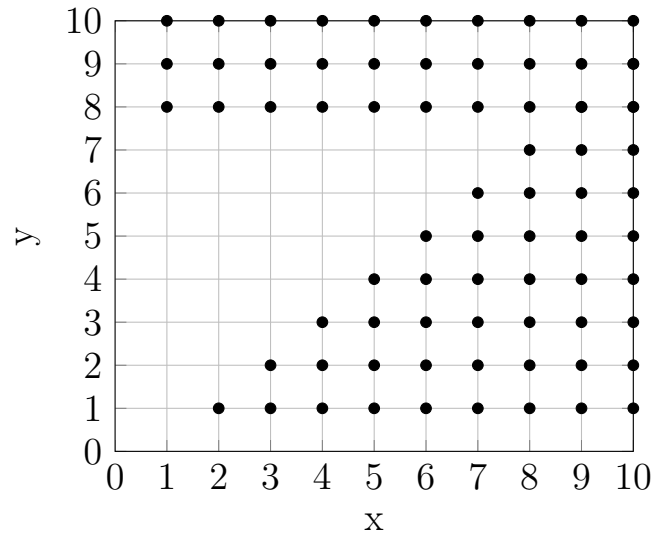
$C = \{(x, y) \,|\, x \in N$ и $y \in N$ и $x < 11$ и $y < 11$ и x- чётно и y- нечётно$\}$

б) $D = (A - B)^{-1} \cup (C \cap \overline{A})$

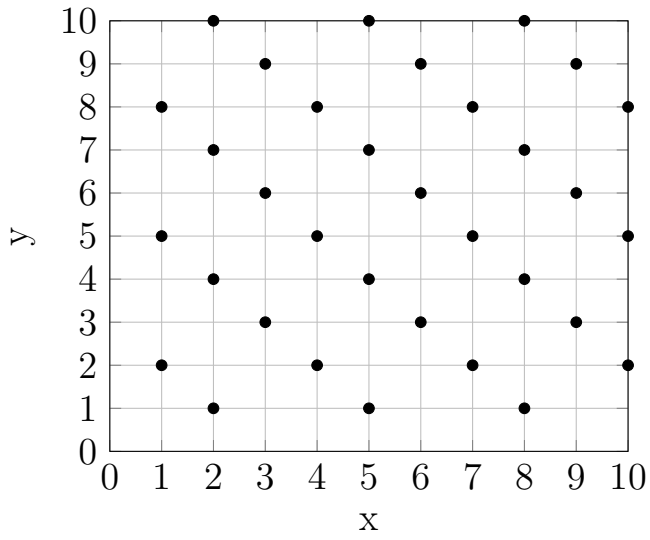
1.1. Представить отношения графиком, графом и матрицей.

Представление графиком:

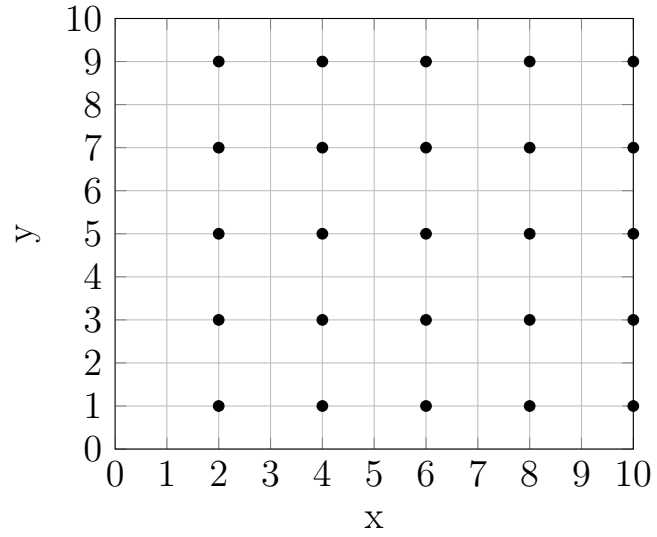
A



B

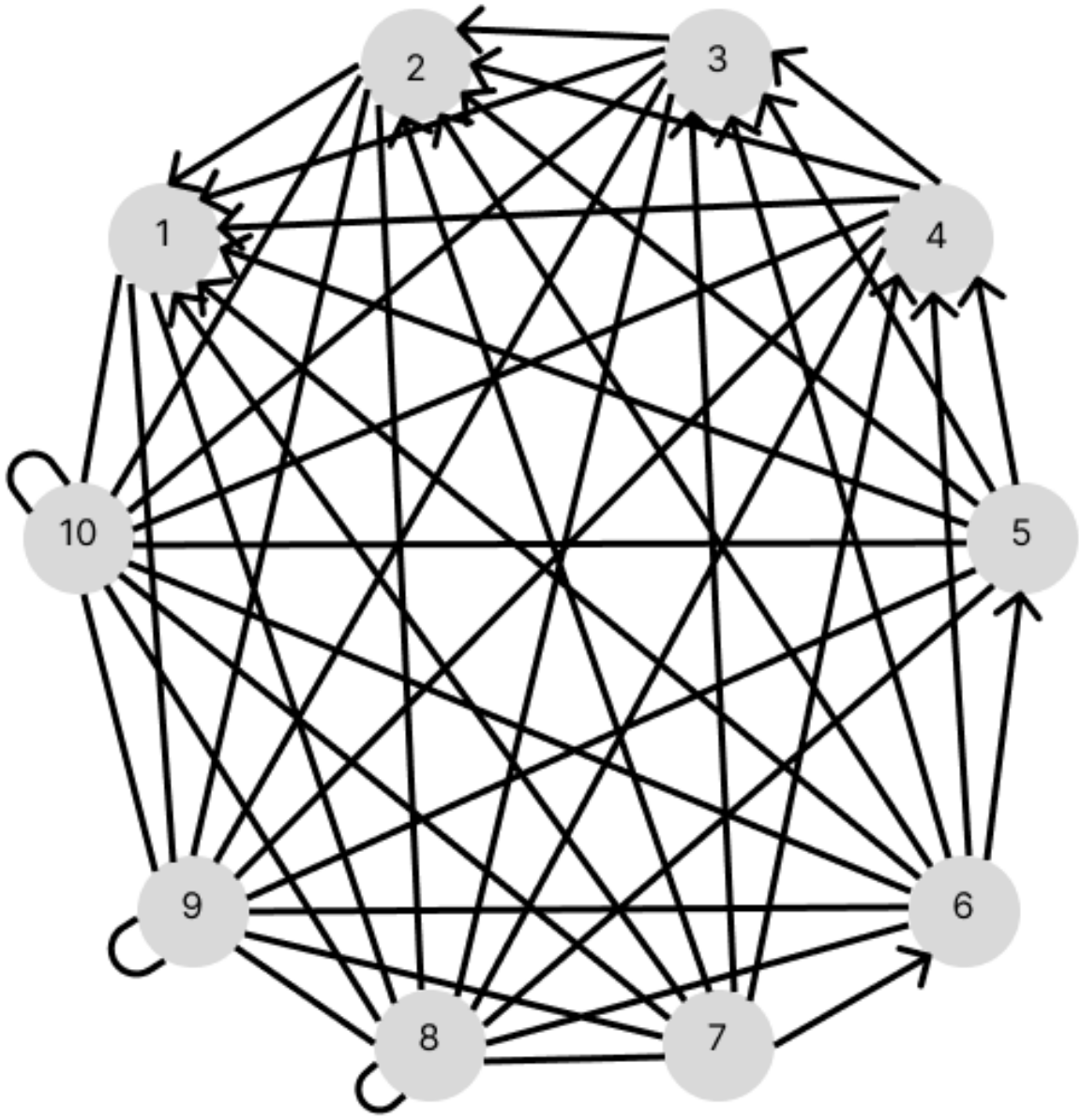


C

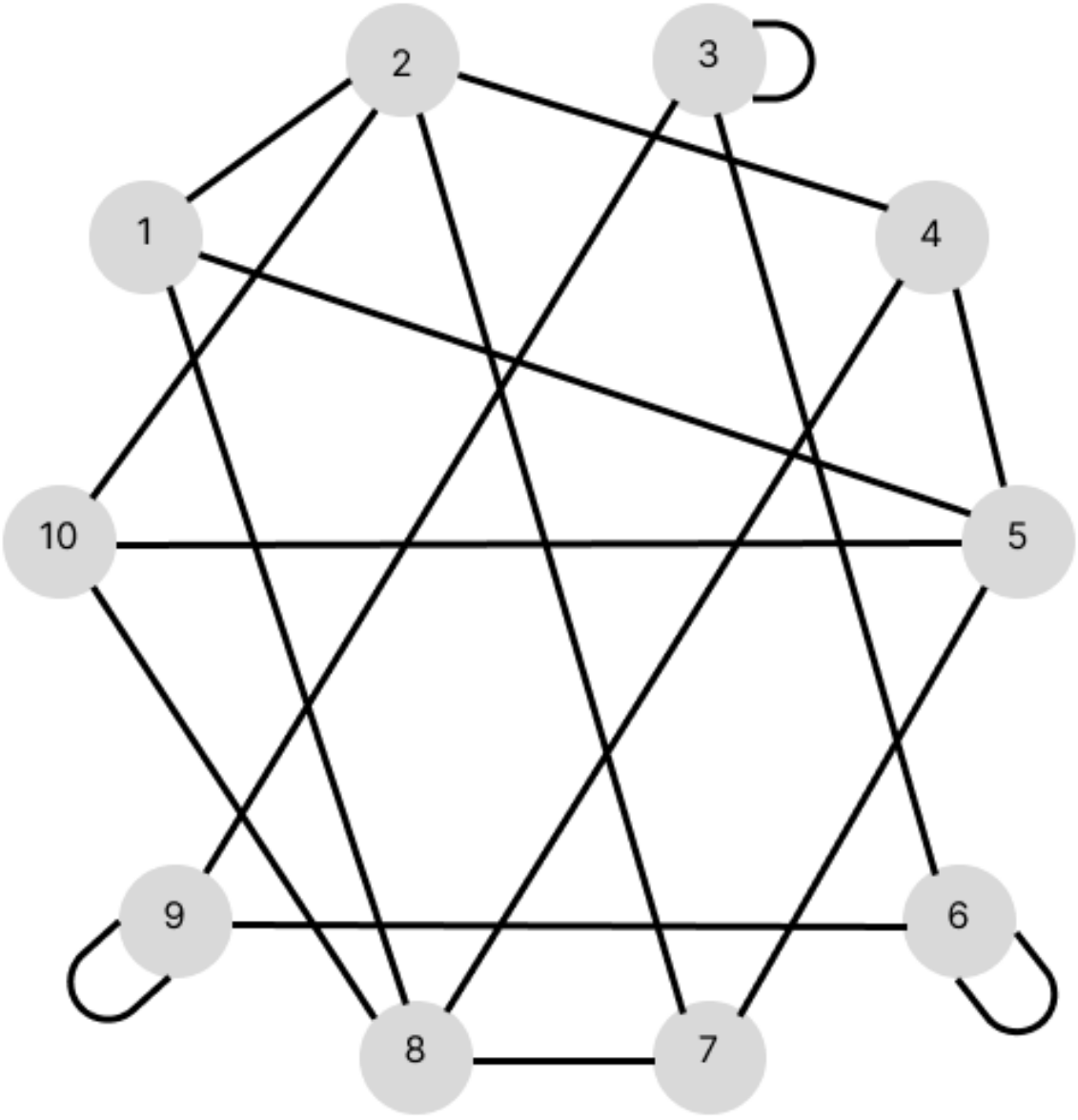


Представление графом:

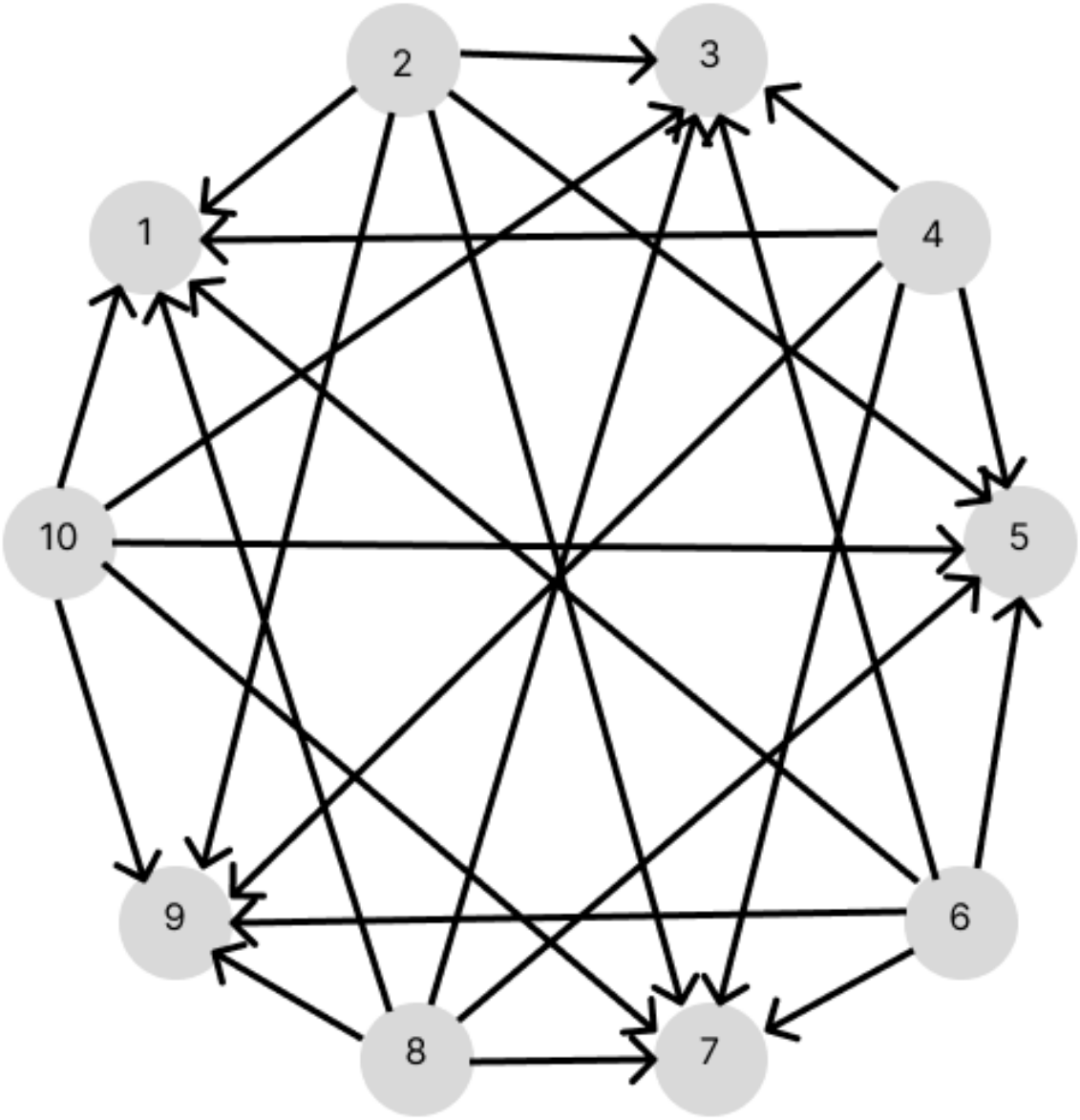
A



B



C



Представление матрицей:

A	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	1	1	1
2	1	0	0	0	0	0	0	1	1	1
3	1	1	0	0	0	0	0	1	1	1
4	1	1	1	0	0	0	0	1	1	1
5	1	1	1	1	0	0	0	1	1	1
6	1	1	1	1	1	0	0	1	1	1
7	1	1	1	1	1	1	0	1	1	1
8	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1

B	1	2	3	4	5	6	7	8	9	10
1	0	1	0	0	1	0	0	1	0	0
2	1	0	0	1	0	0	1	0	0	1
3	0	0	1	0	0	1	0	0	1	0
4	0	1	0	0	1	0	0	1	0	0
5	1	0	0	1	0	0	1	0	0	1
6	0	0	1	0	0	1	0	0	1	0
7	0	1	0	0	1	0	0	1	0	0
8	1	0	0	1	0	0	1	0	0	1
9	0	0	1	0	0	1	0	0	1	0
10	0	1	0	0	1	0	0	1	0	0

C	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	1	0	1	0
3	0	0	0	0	0	0	0	0	0	0
4	1	0	1	0	1	0	1	0	1	0
5	0	0	0	0	0	0	0	0	0	0
6	1	0	1	0	1	0	1	0	1	0
7	0	0	0	0	0	0	0	0	0	0
8	1	0	1	0	1	0	1	0	1	0
9	0	0	0	0	0	0	0	0	0	0
10	1	0	1	0	1	0	1	0	1	0

1.2. Вычислить значение выражения при заданных отношениях.

Выражение: $D = (A - B)^{-1} \cup (C \cap \overline{A})$

Ход решения:

$A - B$

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	0	1	1	0
3	1	1	0	0	0	0	0	1	0	1
4	1	0	1	0	0	0	0	0	1	1
5	0	1	1	0	0	0	0	1	1	0
6	1	1	0	1	1	0	0	1	0	1
7	1	0	1	1	0	1	0	0	1	1
8	0	1	1	0	1	1	0	1	1	0
9	1	1	0	1	1	0	1	1	0	1
10	1	0	1	1	0	1	1	0	1	1

$(A - B)^{-1}$

	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	1	1	0	1	1
2	0	0	1	0	1	1	0	1	1	0
3	0	0	0	1	1	0	1	1	0	1
4	0	0	0	0	0	1	1	0	1	1
5	0	0	0	0	0	1	0	1	1	0
6	0	0	0	0	0	0	1	1	0	1
7	0	0	0	0	0	0	0	0	1	1
8	0	1	1	0	1	1	0	1	1	0
9	1	1	0	1	1	0	1	1	0	1
10	1	0	1	1	0	1	1	0	1	1

$\overline{A}$

A	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	0	0	0
2	0	1	1	1	1	1	1	0	0	0
3	0	0	1	1	1	1	1	0	0	0
4	0	0	0	1	1	1	1	0	0	0
5	0	0	0	0	1	1	1	0	0	0
6	0	0	0	0	0	1	1	0	0	0
7	0	0	0	0	0	0	1	1	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

$C \cap \overline{A}$

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

$(A - B)^{-1} \cup (C \cap \overline{A})$

	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	1	1	0	1	1
2	0	0	1	0	1	1	0	1	1	0
3	0	0	0	1	1	0	1	1	0	1
4	0	0	0	0	0	1	1	0	1	1
5	0	0	0	0	1	1	0	1	1	0
6	0	0	0	0	0	0	1	1	0	1
7	0	0	0	0	0	0	0	0	1	1
8	0	1	1	0	1	1	0	1	1	0
9	1	1	0	1	1	0	1	1	0	1
10	1	0	1	1	0	1	1	0	1	1

Ответ:

D	1	2	3	4	5	6	7	8	9	10
1	0	0	1	1	0	1	1	0	1	1
2	0	0	1	0	1	1	0	1	1	0
3	0	0	0	1	1	0	1	1	0	1
4	0	0	0	0	0	1	1	0	1	1
5	0	0	0	0	1	1	0	1	1	0
6	0	0	0	0	0	0	1	1	0	1
7	0	0	0	0	0	0	0	0	1	1
8	0	1	1	0	1	1	0	1	1	0
9	1	1	0	1	1	0	1	1	0	1
10	1	0	1	1	0	1	1	0	1	1

1.3. Написать программы, формирующие матрицы заданных отношений.

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <malloc.h>
#include <windows.h>

// Функция для определения условия A
bool conditionA(int x, int y) {
    // Условие A: (x, y) | x  N и y  N и x < 11 и y < 11 и (x > y или y > 7)
    return x < 11 && y < 11 && (x > y || y > 7);
}

// Функция для определения условия B
bool conditionB(int x, int y) {
    // Условие B: (x, y) | x  N и y  N и x < 11 и y < 11 и 10x + y кратно 3
    return x < 11 && y < 11 && (10 * x + y) % 3 == 0;
}

// Функция для определения условия C
bool conditionC(int x, int y) {
    // Условие C: (x, y) | x  N и y  N и x < 11 и y < 11 и x - чётное и y - нечётное
    return x < 11 && y < 11 && x % 2 == 0 && y % 2 != 0;
}

// Функция для печати отношения
void printRelation(const bool **relation, int departureAreaSize, int arrivalAreaSize) {
    for (int x = 0; x < departureAreaSize; x++) {
        for (int y = 0; y < arrivalAreaSize; y++) {
            if (relation[x][y]) {
                printf("1 ");
            } else {
                printf("0 ");
            }
        }
        printf("\n");
    }
}

// Функция для создания отношения на основе заданного условия
bool **getRelationByCondition(bool (*func)(int, int), int departureAreaSize, int arrivalAreaSize)
↪  {
    // Выделяем память для двумерного массива
    bool **res = (bool **) malloc(departureAreaSize * sizeof(bool *));
```

```
    for (int i = 0; i < departureAreaSize; i++) {
        res[i] = (bool *) malloc(arrivalAreaSize * sizeof(bool));
    }

    // Заполняем массив в соответствии с заданным условием
    for (int x = 1; x <= departureAreaSize; x++) {
        for (int y = 1; y <= arrivalAreaSize; y++) {
            res[x - 1][y - 1] = func(x, y);
        }
    }

    return res;
}

int main() {
    // Устанавливаем кодировку вывода в UTF-8 для корректного отображения
    SetConsoleOutputCP(CP_UTF8);

    int departureAreaSize = 10;
    int arrivalAreaSize = 10;

    // Создаем отношения A, B и C на основе новых условий
    bool** A = getRelationByCondition(taskConditionA, departureAreaSize, arrivalAreaSize);
    bool** B = getRelationByCondition(taskConditionB, departureAreaSize, arrivalAreaSize);
    bool** C = getRelationByCondition(taskConditionC, departureAreaSize, arrivalAreaSize);

    // Выводим отношения на экран
    printf("Отношение A:\n");
    printRelation(A, departureAreaSize, arrivalAreaSize);

    printf("Отношение B:\n");
    printRelation(B, departureAreaSize, arrivalAreaSize);

    printf("Отношение C:\n");
    printRelation(C, departureAreaSize, arrivalAreaSize);

    // Освобождаем выделенную память для массивов
    for (int i = 0; i < departureAreaSize; i++) {
        free(A[i]);
        free(B[i]);
        free(C[i]);
    }
```

```
    free(A);
    free(B);
    free(C);

    return 0;
}
```

```
C:\Users\NTK\CLionProjects\DiscreteMath3.1
Отношение A:
0 0 0 0 0 0 0 1 1 1
1 0 0 0 0 0 0 1 1 1
1 1 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 1 1 1
1 1 1 1 0 0 0 1 1 1
1 1 1 1 1 0 0 1 1 1
1 1 1 1 1 1 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
Отношение B:
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
```

Рис. : Работа алгоритма. 1 часть.

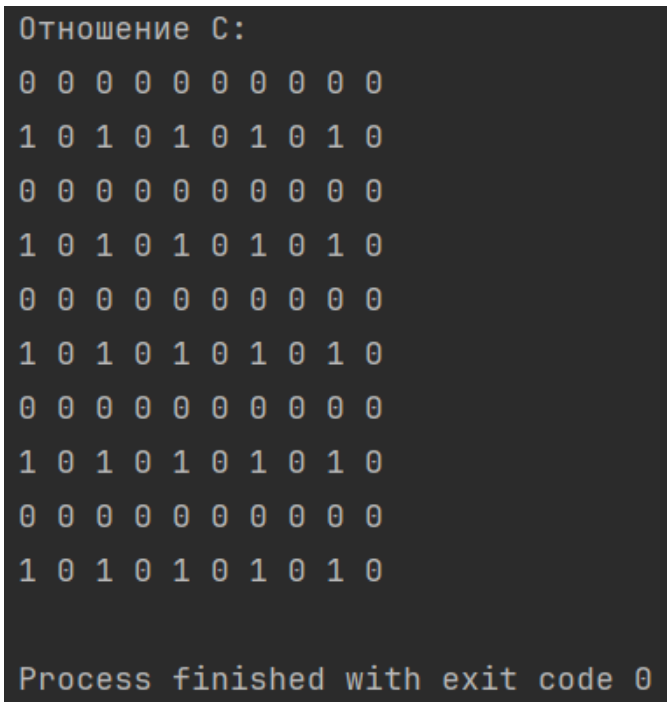


Рис. : Работа алгоритма. 2 часть.

1.4. Программно реализовать операции над отношениями.

```
// Функция для обновления размеров матрицы A и B до одинаковых размеров
void updateBetween(bool **A, int maxRows, int maxCols) {
    // Изменяем размеры матрицы A до maxRows x maxCols
    for (int i = 0; i < maxRows; i++) {
        A[i] = (bool *) realloc(A[i], maxCols * sizeof(bool));
        for (int j = 0; j < maxCols; j++) {
            if (j >= maxCols) {
                A[i][j] = 0; // Заполняем новые элементы нулями
            }
        }
    }

    // Освобождаем лишние строки в матрице A
    for (int i = maxRows; i < maxRows; i++) {
        free(A[i]);
    }

    // Добавляем новые строки к матрице A, если необходимо
    A = (bool **) realloc(A, maxRows * sizeof(bool *));
    for (int i = maxRows; i < maxRows; i++) {
        A[i] = (bool *) malloc(maxCols * sizeof(bool));
        for (int j = 0; j < maxCols; j++) {
            A[i][j] = 0; // Заполняем новые строки нулями
        }
```

```
    }
}

// Функция для обновления матрицы A до квадратной формы и возврата максимального размера
int updateToSquare(bool **A, int maxRows, int maxCols) {
    // Вычисляем максимум из maxRows и maxCols
    int maxMetric = maxRows > maxCols ? maxRows : maxCols;

    // Изменяем размеры матрицы A до maxMetric x maxMetric
    for (int i = 0; i < maxMetric; i++) {
        A[i] = (bool *) realloc(A[i], maxMetric * sizeof(bool));
        for (int j = maxCols; j < maxMetric; j++) {
            A[i][j] = 0; // Заполняем новые элементы нулями
        }
    }

    // Добавляем новые строки к матрице A, если необходимо
    for (int i = maxRows; i < maxMetric; ++i) {
        A = (bool **) realloc(A, (maxMetric + 1) * sizeof(bool *));
        A[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            A[i][j] = 0; // Заполняем новые строки нулями
        }
    }

    return maxMetric;
}

// Функция для операции включения (inclusion)
bool inclusionOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Проверяем, что каждый элемент A равен 1, если соответствующий элемент B равен 0
    for (int i = 0; i < maxRows; i++) {
        for (int j = 0; j < maxCols; j++) {
            if (A[i][j] == 1 && B[i][j] == 0)
                return false; // Найден элемент, который не соответствует условию
        }
    }
    return true; // Все элементы A удовлетворяют условию inclusion
}
```

```
// Функция для операции равенства (equality)
bool equalityOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Проверяем, что каждый элемент A равен соответствующему элементу B
    for (int i = 0; i < maxRows; i++) {
        for (int j = 0; j < maxCols; j++) {
            if (A[i][j] != B[i][j])
                return false; // Найден элемент, который не соответствует условию
        }
    }
    return true; // Все элементы A равны соответствующим элементам B
}

// Функция для операции строгого включения (strict inclusion)
bool strictInclusionOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);
    bool flag = false;

    // Проверяем, что каждый элемент A равен 1, если соответствующий элемент B равен 0,
    // и проверяем, есть ли хотя бы один различный элемент
    for (int i = 0; i < maxRows; i++) {
        for (int j = 0; j < maxCols; j++) {
            if (A[i][j] == 1 && B[i][j] == 0)
                return false; // Найден элемент, который не соответствует условию
            if (A[i][j] != B[i][j])
                flag = true;
        }
    }
    return flag; // Все элементы A удовлетворяют условию strict inclusion
}

// Функция для операции объединения (union)
bool **unionOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);
```

```
    // Создаем новую матрицу C
    int maxMetric = maxRows > maxCols ? maxRows : maxCols;
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = A[i][j] || B[i][j]; // Выполняем операцию объединения
        }
    }
    return C;
}

// Функция для операции пересечения (intersection)
bool **intersectionOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Создаем новую матрицу C
    int maxMetric = maxRows > maxCols ? maxRows : maxCols;
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = A[i][j] && B[i][j]; // Выполняем операцию пересечения
        }
    }
    return C;
}

// Функция для операции разности (difference)
bool **differenceOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Создаем новую матрицу C
    int maxMetric = maxRows > maxCols ? maxRows : maxCols;
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
```

```
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = A[i][j] && !B[i][j]; // Выполняем операцию разности
        }
    }
    return C;
}

// Функция для операции симметричной разности (symmetric difference)
bool **symmetricDifferenceOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Создаем новую матрицу C
    int maxMetric = maxRows > maxCols ? maxRows : maxCols;
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = (A[i][j] && !B[i][j]) || (!A[i][j] && B[i][j]); // Выполняем операцию
            ↪  симметричной разности
        }
    }
    return C;
}

// Функция для операции дополнения (complement)
bool **complementOperation(bool **A, int maxRows, int maxCols) {
    // Обновляем размеры матрицы A до квадратной формы и получаем maxMetric
    int maxMetric = updateToSquare(A, maxRows, maxCols);

    // Создаем новую матрицу C
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = !A[i][j]; // Выполняем операцию дополнения
        }
    }
    return C;
```

```
}

// Функция для операции инверсии (inversion)
bool **inversionOperation(bool **A, int maxRows, int maxCols) {
    // Обновляем размеры матрицы A до квадратной формы и получаем maxMetric
    int maxMetric = updateToSquare(A, maxRows, maxCols);

    // Создаем новую матрицу C
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = A[j][i]; // Выполняем операцию инверсии
        }
    }
    return C;
}

// Функция для операции композиции (composition)
bool **compositionOperation(bool **A, bool **B, int maxRows, int maxCols) {
    // Обновляем размеры матриц A и B до одинаковых
    updateBetween(A, maxRows, maxCols);
    updateBetween(B, maxRows, maxCols);

    // Получаем максимальный размер матрицы
    int maxMetric1 = updateToSquare(A, maxRows, maxCols);
    int maxMetric2 = updateToSquare(B, maxRows, maxCols);
    int maxMetric = maxMetric1 > maxMetric2 ? maxMetric1 : maxMetric2;

    // Создаем новую матрицу C
    bool **C = (bool **) malloc(maxMetric * sizeof(bool *));

    for (int i = 0; i < maxMetric; i++) {
        C[i] = (bool *) malloc(maxMetric * sizeof(bool));
        for (int j = 0; j < maxMetric; j++) {
            C[i][j] = 0; // Инициализируем матрицу C нулями
            for (int k = 0; k < maxMetric; k++) {
                C[i][j] = C[i][j] || (A[i][k] && B[k][j]); // Выполняем операцию композиции
            }
        }
    }
    return C;
```

```
}
```

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

```
int main() {
    // Устанавливаем кодировку вывода в UTF-8 для корректного отображения
    SetConsoleOutputCP(CP_UTF8);

    int departureAreaSize = 10;
    int arrivalAreaSize = 10;

    // Создаем отношения A, B и C на основе новых условий
    bool **A = getRelationByCondition(conditionA, departureAreaSize, arrivalAreaSize);
    bool **B = getRelationByCondition(conditionB, departureAreaSize, arrivalAreaSize);
    bool **C = getRelationByCondition(conditionC, departureAreaSize, arrivalAreaSize);

    // Выполняем операции над отношениями и выводим результат
    printf("Вывод:\n");
    printRelation(unionOperation(inversionOperation(differenceOperation( A, B, departureAreaSize,
    ↪  arrivalAreaSize),departureAreaSize, arrivalAreaSize),intersectionOperation(C,
    ↪  complementOperation(A, departureAreaSize, arrivalAreaSize), departureAreaSize,
    ↪  arrivalAreaSize), departureAreaSize, arrivalAreaSize), departureAreaSize,
    ↪  arrivalAreaSize);

    // Освобождаем выделенную память для массивов
    for (int i = 0; i < departureAreaSize; i++) {
        free(A[i]);
        free(B[i]);
        free(C[i]);
    }
    free(A);
    free(B);
    free(C);

    return 0;
}
```

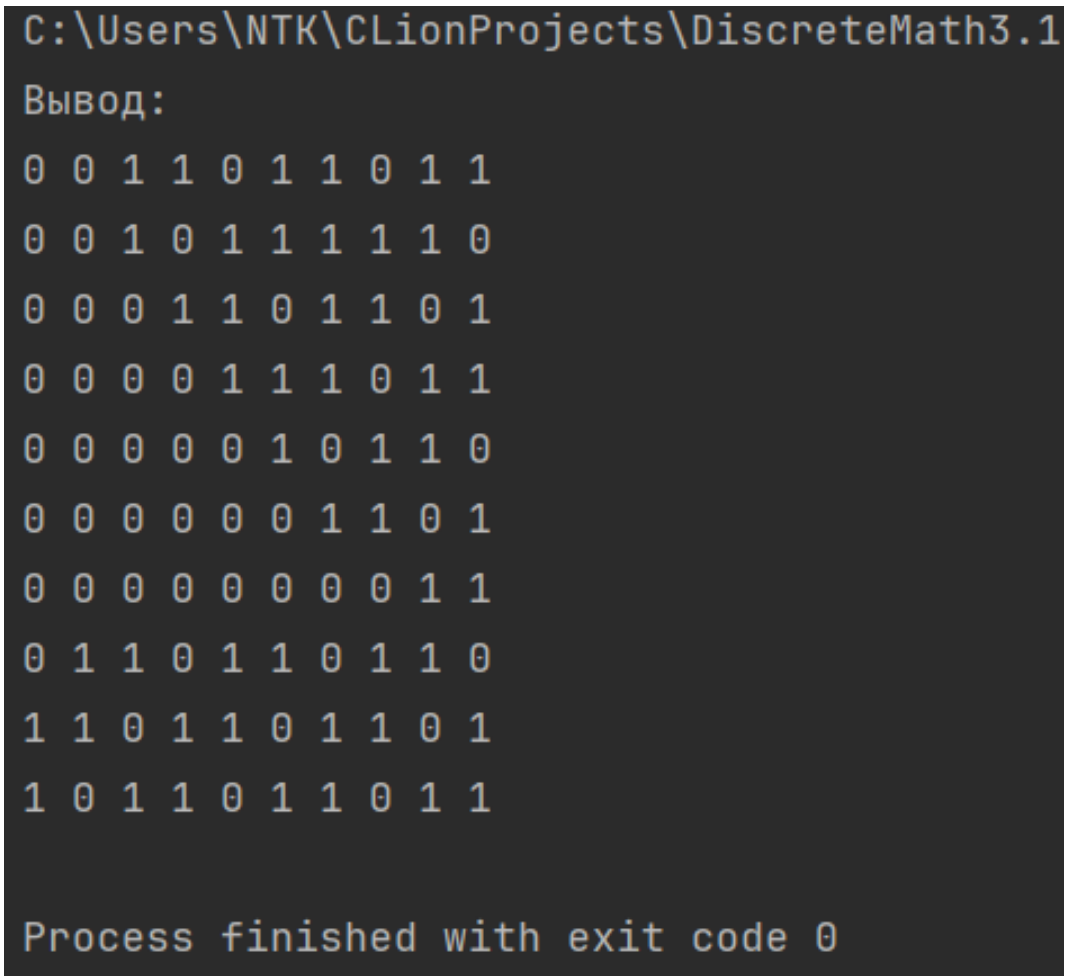


Рис. : Работа алгоритма.

2.1. Определить основные свойства отношений.

Свойство	A	B	C
Рефлексивно	-	-	-
Антирефлексивно	-	-	+
Симметрично	-	+	-
Антисимметрично	-	-	-
Транзитивно	-	-	+
Антитранзитивно	-	-	+
Полное	-	-	-

A: 1. не является рефлексивным, так как оно не включает в себя все пары вида (x, x), где x $<$ 11. В рефлексивном отношении все элементы должны иметь отношение с самим собой.

2. Не является антирефлексивным, так как оно позволяет существовать парам (x, x), где x $<$ 11. В антирефлексивном отношении не допускаются такие пары.

3. Не является симметричным, так как если (x, y) принадлежит A, то (y, x)

не обязательно принадлежит A. Симметричное отношение требует, чтобы если (x, y) принадлежит, то и (y, x) должно быть верным.

4. Не является антисимметричным, так как оно позволяет существовать парам (x, y) и (y, x), где $x \neq y$. В антисимметричном отношении не могут существовать пары, где $x \neq y$.

5. Не является транзитивным, так как транзитивность подразумевает, что если (x, y) и (y, z) принадлежат A, то (x, z) также должно принадлежать A. В случае отношения A это условие не выполняется.

6. Не является антитранзитивным, так как оно позволяет существовать парам (x, y) и (y, z), где (x, z) также принадлежит A. Антитранзитивное отношение подразумевает, что если (x, y) и (y, z) принадлежат, то (x, z) не должно принадлежать.

B: 1. Не является рефлексивным, так как оно не включает в себя все пары вида (x, x) (для $x < 11$). В рефлексивном отношении все элементы должны иметь отношение с самим собой.

2. Не является антирефлексивным, так как оно позволяет существовать парам (x, x), где $x < 11$. В антирефлексивном отношении не допускаются такие пары.

3. Является симметричным, так как (x, y) принадлежит B, то (y, x) также принадлежит B. Это свойство подразумевает, что отношение симметрично.

4. Не является антисимметричным, так как оно позволяет существовать парам (x, y) и (y, x), где $x \neq y$. В антисимметричном отношении не могут существовать пары, где $x \neq y$. Симметрия и антисимметрия исключают друг друга.

5. Не является транзитивным, так как транзитивность подразумевает, что если (x, y) и (y, z) принадлежат B, то (x, z) также должно принадлежать B. В случае отношения B это условие не выполняется.

6. Не является антитранзитивным, так как оно позволяет существовать парам (x, y) и (y, z), где (x, z) также принадлежит B. Антитранзитивное отношение подразумевает, что если (x, y) и (y, z) принадлежат, то (x, z) не должно принадлежать.

C: 1. Не является рефлексивным, так как оно не включает в себя все пары вида (x, x) (для $x < 11$). В рефлексивном отношении все элементы должны иметь отношение с самим собой.

2. Является антирефлексивным, так как оно не позволяет существовать парам (x, x), где $x < 11$. В антирефлексивном отношении не допускаются такие пары.

3. Не является симметричным, так как если (x, y) принадлежит C, то (y, x)

не обязательно принадлежит C. Симметричное отношение требует, чтобы если (x, y) принадлежит, то и (y, x) должно быть верным.

4. Не является антисимметричным, так как оно позволяет существовать парам (x, y) и (y, x), где $x \neq y$. В антисимметричном отношении не могут существовать пары, где $x \neq y$. Симметрия и антисимметрия исключают друг друга.

5. Является транзитивным, так как транзитивность подразумевает, что если (x, y) и (y, z) принадлежат C, то (x, z) также должно принадлежать C. В случае отношения C это условие выполняется.

6. Является антитранзитивным, так как оно не позволяет существовать парам (x, y) и (y, z), где (x, z) принадлежит C. Антитранзитивное отношение подразумевает, что если (x, y) и (y, z) принадлежат, то (x, z) не должно принадлежать.

2.2. Определить, являются ли заданные отношения отношениями толерантности, эквивалентности и порядка.

A: 1. Не имеет свойства толерантности, как как не обладает свойствам рефлексивности и симметричности.

2. Не является эквивалентным отношением, так как оно не обладает свойствами рефлексивности, симметричности и транзитивности.

3. Не является полным, так как оно не связывает все пары элементов (x, y), где x и y являются натуральными числами, $x < 11$ и $y < 11$. Множество A имеет ограничения в виде условий $x > y$ и $y > 7$, и, следовательно, не является полностью связанным.

B: 1. Не имеет свойства толерантности, как как не обладает свойствам рефлексивности и симметричности.

2. Не является эквивалентным отношением, так как оно не обладает свойствами рефлексивности, симметричности и транзитивности.

3. Не является полным, так как оно не связывает все пары элементов (x, y), где x и y являются натуральными числами, $x < 11$ и $y < 11$. Множество B имеет ограничения в виде условия $10x + y$ кратно 3, и, следовательно, не является полностью связанным.

C: 1. Не имеет свойства толерантности, как как не обладает свойствам рефлексивности и симметричности.

2. Не является эквивалентным отношением, так как оно не обладает свойствами рефлексивности, симметричности и транзитивности.

3. Не является полным, так как оно не связывает все пары элементов (x, y), где x и y являются натуральными числами, $x < 11$ и $y < 11$. Множество C

имеет ограничения в виде условий x - чётно и y - нечётно, и, следовательно, не является полностью связанным.

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

```
// Функция проверки рефлексивности
bool reflexivityOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что все диагональные элементы равны 1 (рефлексивность)
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        if (!A[i][i])
            return false;
    }

    return true;
}

// Функция проверки антирефлексивности
bool antiReflexivityOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что все диагональные элементы равны 0 (антирефлексивность)
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        if (A[i][i])
            return false;
    }

    return true;
}

// Функция проверки симметрии
bool symmetryOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что матрица симметрична относительно главной диагонали
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        for (int j = 0; j < ((maxRows + maxCols) / 2); j++) {
            if (A[i][j] != A[j][i])
```

```
                return false;
            }
        }

        return true;
}

// Функция проверки антисимметрии
bool antiSymmetryOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что матрица антисимметрична (A[i][j] и A[j][i] не могут быть оба равны 1)
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        for (int j = 0; j < ((maxRows + maxCols) / 2); j++) {
            if (A[i][j] == A[j][i])
                return false;
        }
    }

    return true;
}

// Функция проверки транзитивности
bool transitivityOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что матрица удовлетворяет транзитивному свойству
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        for (int j = 0; j < ((maxRows + maxCols) / 2); j++) {
            for (int k = 0; k < ((maxRows + maxCols) / 2); k++) {
                if (A[i][j] && A[j][k] && !A[i][k]) {
                    return false;
                }
            }
        }
    }

    return true;
}

// Функция проверки антитранзитивности
```

```
bool antiTransitivityOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что матрица не удовлетворяет транзитивному свойству
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        for (int j = 0; j < ((maxRows + maxCols) / 2); j++) {
            for (int k = 0; k < ((maxRows + maxCols) / 2); k++) {
                if (A[i][j] && A[j][k] && A[i][k])
                    return false;
            }
        }
    }

    return true;
}

// Функция проверки толерантности
bool toleranceOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам рефлексивности и симметрии
    return reflexivityOperation(A, maxRows, maxCols) && symmetryOperation(A, maxRows, maxCols);
}

// Функция проверки эквивалентности
bool equivalenceOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам рефлексивности, симметрии и транзитивности
    return reflexivityOperation(A, maxRows, maxCols) && symmetryOperation(A, maxRows, maxCols) &&
           transitivityOperation(A, maxRows, maxCols);
}

// Функция проверки полного порядка
bool fullyOperation(bool **A, int maxRows, int maxCols) {
    // Приводим матрицу A к квадратной форме
    updateToSquare(A, maxRows, maxCols);

    // Проверяем, что матрица удовлетворяет свойствам полного порядка
    for (int i = 0; i < ((maxRows + maxCols) / 2); i++) {
        for (int j = 0; j < ((maxRows + maxCols) / 2); j++) {
            if (!(A[i][j] && !A[j][i] || !A[i][j] && A[j][i]))
                return false;
        }
    }
```

```
    return true;
}

// Функция проверки порядка
bool orderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам антисимметрии и транзитивности
    return antiSymmetryOperation(A, maxRows, maxCols) && transitivityOperation(A, maxRows,
    ↪  maxCols);
}

// Функция проверки слабого порядка
bool looseOrderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам порядка и рефлексивности
    return orderOperation(A, maxRows, maxCols) && reflexivityOperation(A, maxRows, maxCols);
}

// Функция проверки строгого порядка
bool strictOrderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам порядка и антирефлексивности
    return orderOperation(A, maxRows, maxCols) && antiReflexivityOperation(A, maxRows, maxCols);
}

// Функция проверки линейного порядка
bool linearOrderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам порядка и полного порядка
    return orderOperation(A, maxRows, maxCols) && fullyOperation(A, maxRows, maxCols);
}

// Функция проверки нестрогого линейного порядка
bool nonStrictLinearOrderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам слабого порядка и полного порядка
    return looseOrderOperation(A, maxRows, maxCols) && fullyOperation(A, maxRows, maxCols);
}

// Функция проверки строгого линейного порядка
bool strictLinearOrderOperation(bool **A, int maxRows, int maxCols) {
    // Проверяем, что матрица удовлетворяет свойствам строгого порядка и полного порядка
    return strictOrderOperation(A, maxRows, maxCols) && fullyOperation(A, maxRows, maxCols);
}


int main() {
    // Устанавливаем кодировку вывода в UTF-8 для корректного отображения
```

```
SetConsoleOutputCP(CP_UTF8);

int departureAreaSize = 10;
int arrivalAreaSize = 10;

// Создаем отношения A, B и C на основе новых условий
bool **A = getRelationByCondition(conditionA, departureAreaSize, arrivalAreaSize);
bool **B = getRelationByCondition(conditionB, departureAreaSize, arrivalAreaSize);
bool **C = getRelationByCondition(conditionC, departureAreaSize, arrivalAreaSize);

//1.3
// Выводим отношения на экран
printf("Отношение A:\n");
printRelation(A, departureAreaSize, arrivalAreaSize);

printf("Отношение B:\n");
printRelation(B, departureAreaSize, arrivalAreaSize);

printf("Отношение C:\n");
printRelation(C, departureAreaSize, arrivalAreaSize);

//1.5
// Выполняем операции над отношениями и выводим результат
printf("Вывод:\n");
printRelation(unionOperation(inversionOperation(differenceOperation( A, B, departureAreaSize,
↪  arrivalAreaSize),departureAreaSize, arrivalAreaSize),intersectionOperation(C,
↪  complementOperation(A, departureAreaSize, arrivalAreaSize), departureAreaSize,
↪  arrivalAreaSize), departureAreaSize, arrivalAreaSize), departureAreaSize,
↪  arrivalAreaSize);

//2.3
printf("A:\nРефлексивность: %d\nАнтирефлексивность: %d\nСимметричность: %d\n"
       "Антисимметричность: %d\nТранзитивность: %d\nАнтитранзитивность: %d\n"
       "Толерантность: %d\nЭквивалентность: %d\nПолнота: %d\n",
       reflexivityOperation(A, departureAreaSize, arrivalAreaSize),
       antiReflexivityOperation(A, departureAreaSize, arrivalAreaSize),
       symmetryOperation(A, departureAreaSize, arrivalAreaSize),
       antiSymmetryOperation(A, departureAreaSize, arrivalAreaSize),
       transitivityOperation(A, departureAreaSize, arrivalAreaSize),
       antiTransitivityOperation(A, departureAreaSize, arrivalAreaSize),
       toleranceOperation(A, departureAreaSize, arrivalAreaSize),
       equivalenceOperation(A, departureAreaSize, arrivalAreaSize),
       fullyOperation(A, departureAreaSize, arrivalAreaSize));
```

```
    printf("B:\nРефлексивность: %d\nАнтирефлексивность: %d\nСимметричность: %d\n"
           "Антисимметричность: %d\nТранзитивность: %d\nАнтитранзитивность: %d\n"
           "Толерантность: %d\nЭквивалентность: %d\nПолнота: %d\n"
           "Порядок: %d\n,
           reflexivityOperation(B, departureAreaSize, arrivalAreaSize),
           antiReflexivityOperation(B, departureAreaSize, arrivalAreaSize),
           symmetryOperation(B, departureAreaSize, arrivalAreaSize),
           antiSymmetryOperation(B, departureAreaSize, arrivalAreaSize),
           transitivityOperation(B, departureAreaSize, arrivalAreaSize),
           antiTransitivityOperation(B, departureAreaSize, arrivalAreaSize),
           toleranceOperation(B, departureAreaSize, arrivalAreaSize),
           equivalenceOperation(B, departureAreaSize, arrivalAreaSize),
           fullyOperation(B, departureAreaSize, arrivalAreaSize));
    printf("C:\nРефлексивность: %d\nАнтирефлексивность: %d\nСимметричность: %d\n"
           "Антисимметричность: %d\nТранзитивность: %d\nАнтитранзитивность: %d\n"
           "Толерантность: %d\nЭквивалентность: %d\nПолнота: %d\n",
           reflexivityOperation(C, departureAreaSize, arrivalAreaSize),
           antiReflexivityOperation(C, departureAreaSize, arrivalAreaSize),
           symmetryOperation(C, departureAreaSize, arrivalAreaSize),
           antiSymmetryOperation(C, departureAreaSize, arrivalAreaSize),
           transitivityOperation(C, departureAreaSize, arrivalAreaSize),
           antiTransitivityOperation(C, departureAreaSize, arrivalAreaSize),
           toleranceOperation(C, departureAreaSize, arrivalAreaSize),
           equivalenceOperation(C, departureAreaSize, arrivalAreaSize),
           fullyOperation(C, departureAreaSize, arrivalAreaSize));

    // Освобождаем выделенную память для массивов
    for (int i = 0; i < departureAreaSize; i++) {
        free(A[i]);
        free(B[i]);
        free(C[i]);
    }
    free(A);
    free(B);
    free(C);

    return 0;
}
```

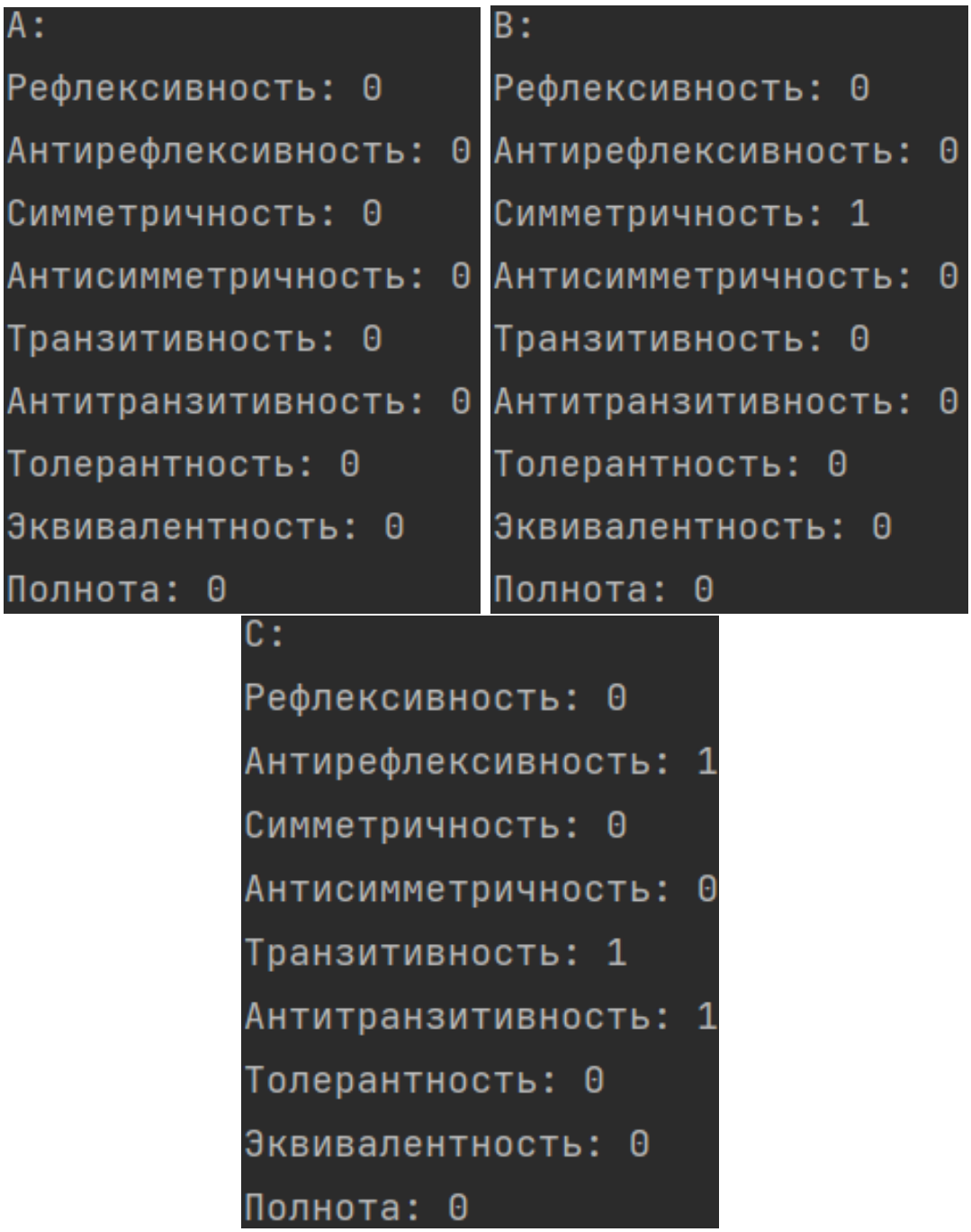


Рис. : Работа алгоритма.

Вывод: В ходе выполнения лабораторной работы изучили способы задания отношений, операции над отношениями и свойства отношений, научились программно реализовывать операции и определять свойства отношений.