

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
Пахомов Владислав Андреевич

Проверили:
ст. пр. Рязанов Юрий Дмитриевич
ст. пр. Бондаренко Татьяна Владими-
ровна

Белгород 2023 г.

Лабораторная работа №3.1

Отношения и их свойства

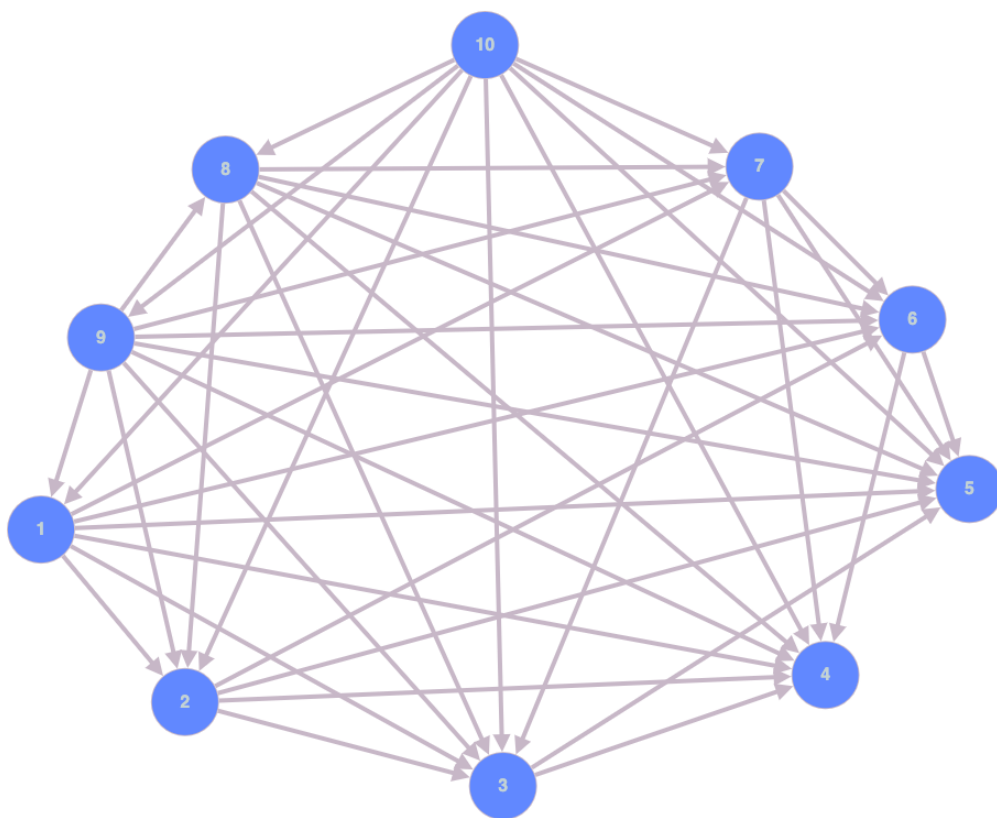
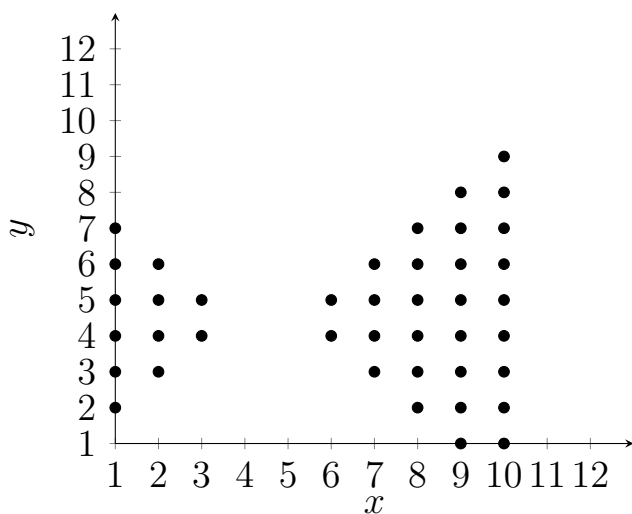
Вариант 10

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

Часть 1. Операции над отношениями

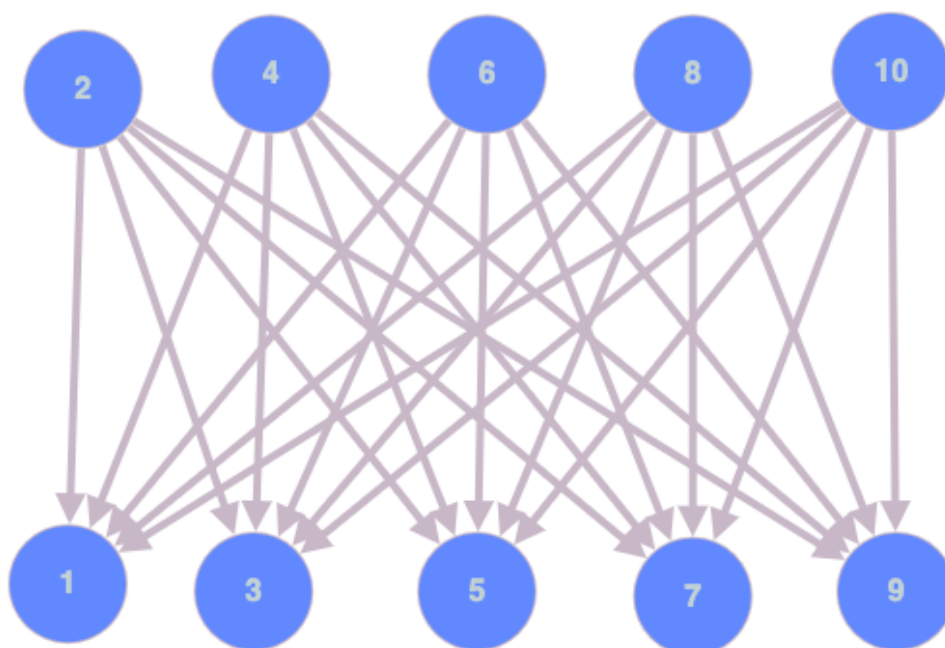
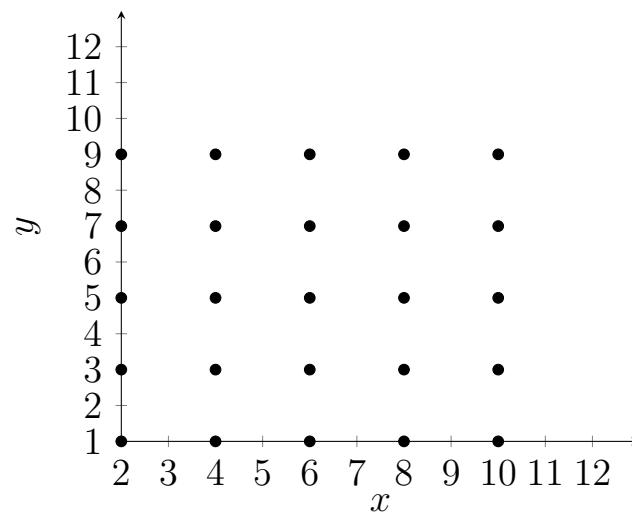
1.1. Представить отношения (см. "Варианты заданий", п.а) графиком, графом и матрицей.

$$A = \{(x, y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } (x < y < (9 - x) \text{ или } (9 - x) < y < x)\}$$



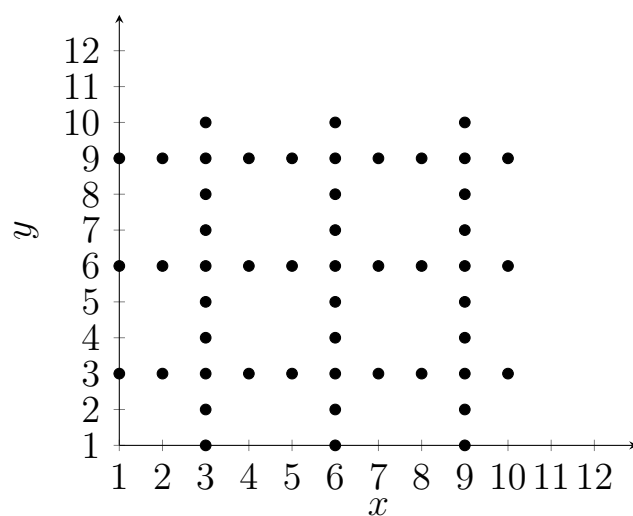
	1	2	3	4	5	6	7	8	9	10
1	0	1	1	1	1	1	1	0	0	0
2	0	0	1	1	1	1	0	0	0	0
3	0	0	0	1	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	0	0	0	0	0
7	0	0	1	1	1	1	0	0	0	0
8	0	1	1	1	1	1	1	0	0	0
9	1	1	1	1	1	1	1	1	0	0
10	1	1	1	1	1	1	1	1	1	0

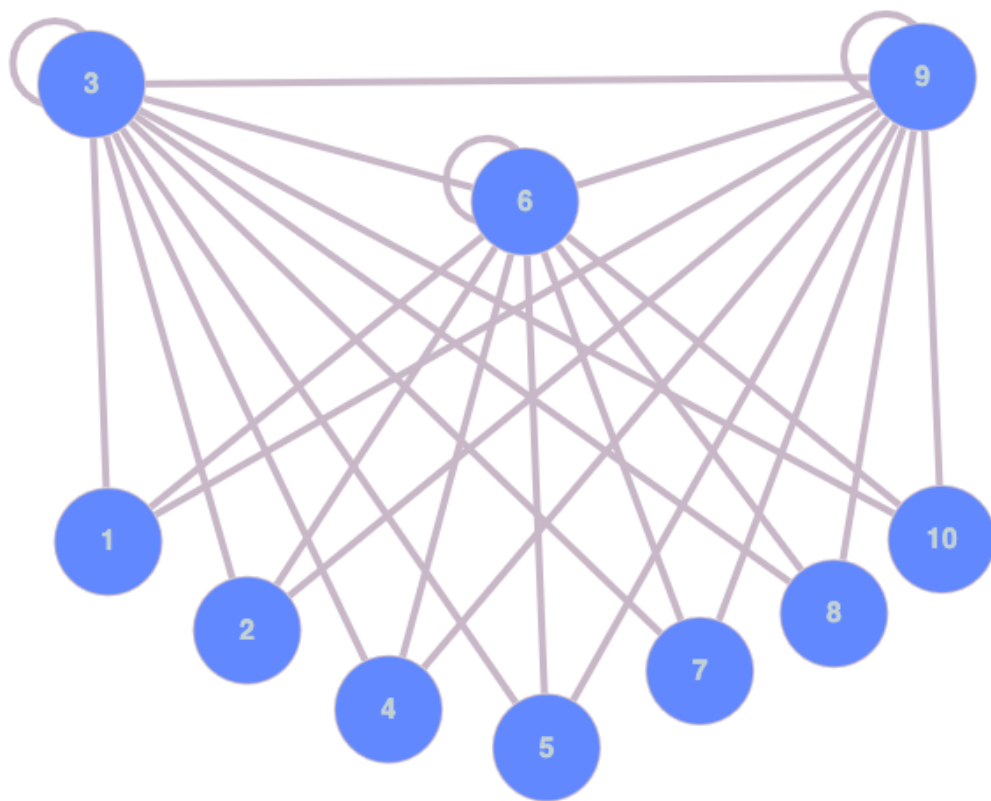
$$B = \{(x, y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } x - \text{чётно и } y - \text{нечётно}\}$$



	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

$$C = \{(x, y) | x \in N \text{ и } y \in N \text{ и } x < 11 \text{ и } y < 11 \text{ и } x \cdot y \text{ кратно трём}\}$$





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

1.2. Вычислить значение выражения (см. "Варианты заданий", п.б) при заданных отношениях (см. "Варианты заданий", п.а).

$$D = A \circ B^2 - \overline{C} \cup C^{-1}$$

$$D = A \overset{1}{\circ} B \overset{2}{\circ} B \overset{5}{-} \overset{3}{\overline{C}} \overset{6}{\cup} \overset{4}{C^{-1}}$$

$$1) A \circ B =$$

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

$$2) _1 \circ B =$$

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

$$3) \overline{C} =$$

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

$$4) C^{-1} =$$

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

5) $_2 - _3 =$

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

6) $_5 \cup _4 =$

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

$D =$

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

1.3. Написать программы, формирующие матрицы заданных отношений (см. "Варианты заданий", п.а).

main.cpp

```
#include "../libs/alg/alg.h"

bool predA(int x, int y) {
    return (x < y && y < (9 - x)) || ((9 - x) < y && y < x);
}

bool predB(int x, int y) {
    return x % 2 == 0 && y % 2 != 0;
```

```

}

bool predC(int x, int y) {
    return (x * y) % 3 == 0 ;
}

int main() {
    BoolMatrixRelation a(10, predA);
    std::cout << a << std::endl;

    BoolMatrixRelation b(10, predB);
    std::cout << b << std::endl;

    BoolMatrixRelation c(10, predC);
    std::cout << c << std::endl;
}

```

alg.h (объявление методов класса)

```

class BoolMatrixRelation
{
    private:
        std::vector<std::vector<bool>> data;
        int size;

    public:
        BoolMatrixRelation(const int size, bool (*pred)(int, int));
        ~BoolMatrixRelation();
        friend std::ostream& operator<<(std::ostream& out, BoolMatrixRelation &val) {
            out << std::setw(3) << "" << " ";
            for (int i = 1; i <= val.size; i++) {
                out << std::setw(3) << i << " ";
            }
            out << "\n";

            for (int x = 0; x < val.size; x++) {
                out << std::setw(3) << x + 1 << " ";
                for (int y = 0; y < val.size; y++) {
                    out << std::setw(3) << val.data[x][y] << " ";
                }

                out << "\n";
            }

            return out;
        }
};

```

task13.cpp (реализация методов класса)


```
#include "../alg.h"

BoolMatrixRelation::BoolMatrixRelation(const int size, bool (*pred)(int, int)) {
    this->size = size;

    for (int x = 1; x <= size; x++) {
        std::vector<bool> val;

        for (int y = 1; y <= size; y++) {
            val.push_back(pred(x, y));
        }

        this->data.push_back(val);
    }
}

BoolMatrixRelation::~BoolMatrixRelation() {}
```

Результат выполнения программы:

```
vlad@Mac-Pro-Vladislav bin % ./Users/vlad/Desktop/C/discrete_math/build/bin/lab7_task13
1 0 1 1 1 1 1 1 0 0 0
2 0 0 1 1 1 1 0 0 0 0
3 0 0 0 1 1 0 0 0 0 0
4 0 0 0 0 0 0 0 0 0 0
5 0 0 0 0 0 0 0 0 0 0
6 0 0 0 1 1 0 0 0 0 0
7 0 0 1 1 1 1 0 0 0 0
8 0 1 1 1 1 1 1 0 0 0
9 1 1 1 1 1 1 1 1 0 0
10 1 1 1 1 1 1 1 1 1 0

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

1.4. Программно реализовать операции над отношениями.

Немного модифицируем класс BoolMatrixRelation. *alg.h* (объявление методов класса)

```
class BoolMatrixRelation
{
private:
    std::vector<std::vector<bool>> data;
    int size;

    static BoolMatrixRelation getDefault() {
        return BoolMatrixRelation();
    }
}
```

```

public:
    BoolMatrixRelation(const int size, std::function<bool (int, int)> pred);
    BoolMatrixRelation() {
        this->size = 0;
    }
    ~BoolMatrixRelation();

    bool includes(BoolMatrixRelation b);
    bool equals(BoolMatrixRelation b);
    bool includesStrict(BoolMatrixRelation b);
    BoolMatrixRelation unite(BoolMatrixRelation b);
    BoolMatrixRelation intersect(BoolMatrixRelation b);
    BoolMatrixRelation diff(BoolMatrixRelation b);
    BoolMatrixRelation symDiff(BoolMatrixRelation b);
    BoolMatrixRelation non();
    BoolMatrixRelation transpose();
    BoolMatrixRelation compose(BoolMatrixRelation b);
    BoolMatrixRelation pow(int p);

    friend std::ostream& operator<<(std::ostream& out, BoolMatrixRelation &val) {
        out << std::setw(3) << "" << " ";
        for (int i = 1; i <= val.size; i++) {
            out << std::setw(3) << i << " ";
        }
        out << "\n";

        for (int x = 0; x < val.size; x++) {
            out << std::setw(3) << x + 1 << " ";
            for (int y = 0; y < val.size; y++) {
                out << std::setw(3) << val.data[x][y] << " ";
            }

            out << "\n";
        }

        return out;
    }
};

```

task13.cpp (реализация методов класса)

```

#include "../alg.h"

BoolMatrixRelation::BoolMatrixRelation(const int size, std::function<bool (int, int)> pred)
{
    this->size = size;

    for (int x = 1; x <= size; x++)
    {
        std::vector<bool> val;

        for (int y = 1; y <= size; y++)
        {

```

```

        val.push_back(pred(x, y));
    }

    this->data.push_back(val);
}

}

BoolMatrixRelation::~BoolMatrixRelation() {}

```

task14.cpp (реализация методов класса)

```

#include "../alg.h"

bool BoolMatrixRelation::includes(BoolMatrixRelation b)
{
    if (this->size != b.size) return false;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (data[i][j] && !b.data[i][j])
                return false;
        }
    }
    return true;
}

bool BoolMatrixRelation::equals(BoolMatrixRelation b)
{
    if (this->size != b.size) return false;

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (data[i][j] != b.data[i][j])
                return false;
        }
    }
    return true;
}

bool BoolMatrixRelation::includesStrict(BoolMatrixRelation b)
{
    return (*this).includes(b) && !(*this).equals(b);
}

BoolMatrixRelation BoolMatrixRelation::unite(BoolMatrixRelation b)
{
    if (this->size != b.size) return BoolMatrixRelation::getDefault();

    return BoolMatrixRelation(size, [this, &b](int x, int y) {
        return data[x - 1][y - 1] || b.data[x - 1][y - 1];
    });
}

BoolMatrixRelation BoolMatrixRelation::intersect(BoolMatrixRelation b)
{
    if (this->size != b.size) return BoolMatrixRelation::getDefault();
}

```

```

        return BoolMatrixRelation(size, [this, &b])(int x, int y) {
            return data[x - 1][y - 1] && b.data[x - 1][y - 1];
        });
    }

BoolMatrixRelation BoolMatrixRelation::diff(BoolMatrixRelation b)
{
    if (this->size != b.size) return BoolMatrixRelation::getDefault();

    return BoolMatrixRelation(size, [this, &b])(int x, int y) {
        return data[x - 1][y - 1] && !b.data[x - 1][y - 1];
    });
}

BoolMatrixRelation BoolMatrixRelation::symDiff(BoolMatrixRelation b)
{
    if (this->size != b.size) return BoolMatrixRelation::getDefault();

    return BoolMatrixRelation(size, [this, &b])(int x, int y) {
        return data[x - 1][y - 1] ^ b.data[x - 1][y - 1];
    });
}

BoolMatrixRelation BoolMatrixRelation::non()
{
    return BoolMatrixRelation(size, [this])(int x, int y) {
        return !data[x - 1][y - 1];
    });
}

BoolMatrixRelation BoolMatrixRelation::transpose()
{
    return BoolMatrixRelation(size, [this])(int x, int y) {
        return data[y - 1][x - 1];
    });
}

BoolMatrixRelation BoolMatrixRelation::compose(BoolMatrixRelation b)
{
    if (this->size != b.size) return BoolMatrixRelation::getDefault();

    return BoolMatrixRelation(size, [this, &b])(int x, int y) {
        for (int z = 0; z < size; z++) {
            if (data[x - 1][z] && b.data[z][y - 1])
                return true;
        }

        return false;
    });
}

BoolMatrixRelation BoolMatrixRelation::pow(int p)
{
    if (p < 0) return transpose();
    if (p == 0) return BoolMatrixRelation(size, [](int x, int y){return x == y;});
    if (p == 1) return *this;

    BoolMatrixRelation lowP = pow(p - 1);
    return compose(lowP);
}

```

```
}
```

- 1.5. Написать программу, вычисляющую значение выражения (см. “Варианты заданий”, п.б) и вычислить его при заданных отношениях (см. “Варианты заданий”, п.а).

main.cpp

```
#include "../libs/alg/alg.h"

bool predA(int x, int y) {
    return (x < y && y < (9 - x)) || ((9 - x) < y && y < x);
}

bool predB(int x, int y) {
    return x % 2 == 0 && y % 2 != 0;
}

bool predC(int x, int y) {
    return (x * y) % 3 == 0 ;
}

int main() {
    BoolMatrixRelation a(10, predA);
    BoolMatrixRelation b(10, predB);
    BoolMatrixRelation c(10, predC);
    BoolMatrixRelation d = ((a.compose(b)).compose(b)).diff(c.non()).unite(c.pow(-1));

    std::cout << d << std::endl;
}
```

Результат выполнения программы:

```
vlad@Mac-Pro-Vladislav bin % ./Users/vlad/Desktop/C/discrete_math/build/bin/lab7_task15
  1  2  3  4  5  6  7  8  9 10
1  0  0  1  0  0  1  0  0  1  0
2  0  0  1  0  0  1  0  0  1  0
3  1  1  1  1  1  1  1  1  1  1
4  0  0  1  0  0  1  0  0  1  0
5  0  0  1  0  0  1  0  0  1  0
6  1  1  1  1  1  1  1  1  1  1
7  0  0  1  0  0  1  0  0  1  0
8  0  0  1  0  0  1  0  0  1  0
9  1  1  1  1  1  1  1  1  1  1
10 0  0  1  0  0  1  0  0  1  0
```

Значение формулы D, вычисленное вручную, и результат выполнения программы совпали.

Часть 2. Свойства отношений

- 2.1. Определить основные свойства отношений (см. “Варианты заданий”, п.а).

	A	B	C
Рефлексивность			
Антирефлексивность	+	+	
Симметричность			+
Антисимметричность	+	+	
Транзитивность	+	+	
Антитранзитивность		+	
Полнота			

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

	A	B	C
Толерантно			
Эквивалентно			
Порядка	+	+	
Нестрогого порядка			
Строгого порядка	+	+	
Линейного порядка			
Нестрогого линейного порядка			
Строгого линейного порядка			

2.3. Написать программу, определяющую свойства отношения, в том числе толерантности, эквивалентности и порядка, и определить свойства отношений (см. "Варианты заданий", п.а). *main.cpp*

```
#include "../libs/alg/alg.h"

bool predA(int x, int y) {
    return (x < y && y < (9 - x)) || ((9 - x) < y && y < x);
}

bool predB(int x, int y) {
    return x % 2 == 0 && y % 2 != 0;
}

bool predC(int x, int y) {
    return (x * y) % 3 == 0;
}

void outputProperties(std::string name, BoolMatrixRelation a) {
    std::pair<int, int> failedAt;
    std::cout << "Properties for " << name << "\n";
    if (a.isReflexive(failedAt)) {
        std::cout << "Reflexive\n";
    } else {
        std::cout << "Non reflexive, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
    }
    if (a.isAntiReflexive(failedAt)) {
        std::cout << "Antireflexive\n";
    } else {
        std::cout << "Non antireflexive, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
    }
}
```

```

}

if (a.isSymmetric(failedAt)) {
    std::cout << "Symmetric\n";
} else {
    std::cout << "Non symmetric, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
}

if (a.isAntiSymmetric(failedAt)) {
    std::cout << "AntiSymmetric\n";
} else {
    std::cout << "Non antisymmetric, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
}

if (a.isTransitive(failedAt)) {
    std::cout << "Transitive\n";
} else {
    std::cout << "Non transitive, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
}

if (a.isAntiTransitive(failedAt)) {
    std::cout << "AntiTransitive\n";
} else {
    std::cout << "Non antitransitive, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
}

if (a.isFull(failedAt)) {
    std::cout << "Full\n";
} else {
    std::cout << "Non full, failed pair: (" << failedAt.first << ", " << failedAt.second << ")\n";
}

std::cout << (a.isTolerant() ? "Tolerant" : "Non tolerant") << "\n";
std::cout << (a.isEquivalent() ? "Equivalent" : "Non equivalent") << "\n";
std::cout << (a.isOrdered() ? "Ordered" : "Non ordered") << "\n";
std::cout << (a.isOrderedNonStrict() ? "Ordered non strict" : "Non ordered non strict") << "\n";
std::cout << (a.isOrderedStrict() ? "Ordered strict" : "Non ordered strict") << "\n";
std::cout << (a.isOrderedLinear() ? "Ordered linear" : "Non ordered linear") << "\n";
std::cout << (a.isOrderedLinearNonStrict() ? "Ordered linear non strict" : "Non ordered linear non strict")
↳ << "\n";
std::cout << (a.isOrderedLinearStrict() ? "Ordered linear strict" : "Non ordered linear strict") << "\n" <<
↳ std::endl;
}

int main() {
    BoolMatrixRelation a(10, predA);
    BoolMatrixRelation b(10, predB);
    BoolMatrixRelation c(10, predC);

    outputProperties("A", a);
    outputProperties("B", b);
    outputProperties("C", c);
}

```

```

class BoolMatrixRelation
{
private:
    std::vector<std::vector<bool>> data;
    int size;

    static BoolMatrixRelation getDefault() {
        return BoolMatrixRelation();
    }

public:
    BoolMatrixRelation(const int size, std::function<bool (int, int)> pred);
    BoolMatrixRelation() {
        this->size = 0;
    }
    ~BoolMatrixRelation();

    bool includes(BoolMatrixRelation b);
    bool equals(BoolMatrixRelation b);
    bool includesStrict(BoolMatrixRelation b);
    BoolMatrixRelation unite(BoolMatrixRelation b);
    BoolMatrixRelation intersect(BoolMatrixRelation b);
    BoolMatrixRelation diff(BoolMatrixRelation b);
    BoolMatrixRelation symDiff(BoolMatrixRelation b);
    BoolMatrixRelation non();
    BoolMatrixRelation transpose();
    BoolMatrixRelation compose(BoolMatrixRelation b);
    BoolMatrixRelation pow(int p);

    static BoolMatrixRelation getIdentity(int size);
    static BoolMatrixRelation getUniversum(int size);

    bool isEmpty();

    bool isReflexive(std::pair<int, int> &failed);
    bool isAntiReflexive(std::pair<int, int> &failed);
    bool isSymmetric(std::pair<int, int> &failed);
    bool isAntiSymmetric(std::pair<int, int> &failed);
    bool isTransitive(std::pair<int, int> &failed);
    bool isAntiTransitive(std::pair<int, int> &failed);
    bool isFull(std::pair<int, int> &failed);
    bool isTolerant();
    bool isEquivalent();
    bool isOrdered();
    bool isOrderedNonStrict();
    bool isOrderedStrict();
    bool isOrderedLinear();
    bool isOrderedLinearNonStrict();
    bool isOrderedLinearStrict();

    friend std::ostream& operator<<(std::ostream& out, BoolMatrixRelation &val) {
        out << std::setw(3) << " " << " ";
        for (int i = 1; i <= val.size; i++) {
            out << std::setw(3) << i << " ";

```



```

    }
    out << "\n";

    for (int x = 0; x < val.size; x++) {
        out << std::setw(3) << x + 1 << " ";
        for (int y = 0; y < val.size; y++) {
            out << std::setw(3) << val.data[x][y] << " ";
        }

        out << "\n";
    }

    return out;
}

};

```

task23.cpp

```

#include "../alg.h"

bool BoolMatrixRelation::isEmpty() {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            if (data[i][j]) return false;
        }
    }

    return true;
}

BoolMatrixRelation BoolMatrixRelation::getIdentity(int size) {
    return BoolMatrixRelation(size, [](int x, int y) {
        return x == y;
    });
}

BoolMatrixRelation BoolMatrixRelation::getUniversum(int size) {
    return BoolMatrixRelation(size, [](int x, int y) {
        return true;
    });
}

bool BoolMatrixRelation::isReflexive(std::pair<int, int> &failed)
{
    for (int i = 0; i < size; i++) {
        if (!data[i][i]) {
            failed = {i + 1, i + 1};
            return false;
        }
    }

    return true;
}

bool BoolMatrixRelation::isAntiReflexive(std::pair<int, int> &failed)

```

```

{
    for (int i = 0; i < size; i++) {
        if (data[i][i]) {
            failed = {i + 1, i + 1};
            return false;
        }
    }

    return true;
}

bool BoolMatrixRelation::isSymmetric(std::pair<int, int> &failed)
{
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (data[i][j] != data[j][i]) {
                failed = {i + 1, j + 1};
                return false;
            }
        }
    }

    return true;
}

bool BoolMatrixRelation::isAntiSymmetric(std::pair<int, int> &failed)
{
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (data[i][j] && data[j][i]) {
                failed = {i + 1, j + 1};
                return false;
            }
        }
    }

    return true;
}

bool BoolMatrixRelation::isTransitive(std::pair<int, int> &failed)
{
    //return ((*this).pow(2)).includes((*this));
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int z = 0; z < size; z++) {
                if ((data[i][z] && data[z][j]) && !data[i][j]) {
                    failed = {i + 1, j + 1};
                    return false;
                }
            }
        }
    }

    return true;
}

bool BoolMatrixRelation::isAntiTransitive(std::pair<int, int> &failed)
{

```

```

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            for (int z = 0; z < size; z++) {
                if ((data[i][z] && data[z][j]) && data[i][j]) {
                    failed = {i + 1, j + 1};
                    return false;
                }
            }
        }
    }

    return true;
}

bool BoolMatrixRelation::isFull(std::pair<int, int> &failed)
{
    for (int i = 0; i < size; i++) {
        for (int j = i + 1; j < size; j++) {
            if (!(data[i][j] || data[j][i])) {
                failed = {i + 1, j + 1};
                return false;
            }
        }
    }

    return true;
}

bool BoolMatrixRelation::isTolerant()
{
    std::pair<int, int> ignored;
    return isReflexive(ignored) && isSymmetric(ignored);
}

bool BoolMatrixRelation::isEquivalent()
{
    std::pair<int, int> ignored;
    return isReflexive(ignored) && isSymmetric(ignored) && isTransitive(ignored);
}

bool BoolMatrixRelation::isOrdered()
{
    std::pair<int, int> ignored;
    return isAntiSymmetric(ignored) && isTransitive(ignored);
}

bool BoolMatrixRelation::isOrderedNonStrict()
{
    std::pair<int, int> ignored;
    return isOrdered() && isReflexive(ignored);
}

bool BoolMatrixRelation::isOrderedStrict()
{
    std::pair<int, int> ignored;
    return isOrdered() && isAntiReflexive(ignored);
}

bool BoolMatrixRelation::isOrderedLinear()
{
    std::pair<int, int> ignored;

```

```

        return isOrdered() && isFull(ignored);
    }
    bool BoolMatrixRelation::isOrderedLinearNonStrict()
    {
        std::pair<int, int> ignored;
        return isOrderedNonStrict() && isFull(ignored);
    }
    bool BoolMatrixRelation::isOrderedLinearStrict()
    {
        std::pair<int, int> ignored;
        return isOrderedStrict() && isFull(ignored);
    }
}

```

Результат выполнения программы:

```

vlad@Mac-Pro-Vladislav bin % ./Users/vlad/Desktop/C/discrete_math/build/bin/lab7_task23
Properties for A
Non reflexive, failed pair: (1, 1)
Antireflexive
Non symmetric, failed pair: (1, 2)
Antisymmetric
Transitive
Non antitransitive, failed pair: (1, 3)
Non full, failed pair: (1, 8)
Non tolerant
Non equivalent
Ordered
Non ordered non strict
Ordered strict
Non ordered linear
Non ordered linear non strict
Non ordered linear strict

Properties for B
Non reflexive, failed pair: (1, 1)
Antireflexive
Non symmetric, failed pair: (1, 2)
Antisymmetric
Transitive
Antitransitive
Non full, failed pair: (1, 3)
Non tolerant
Non equivalent
Ordered
Non ordered non strict
Ordered strict
Non ordered linear
Non ordered linear non strict
Non ordered linear strict

Properties for C
Non reflexive, failed pair: (1, 1)
Non antireflexive, failed pair: (3, 3)
Symmetric
Non antisymmetric, failed pair: (1, 3)
Non transitive, failed pair: (1, 1)
Non antitransitive, failed pair: (1, 3)
Non full, failed pair: (1, 2)
Non tolerant
Non equivalent
Non ordered
Non ordered non strict
Non ordered strict
Non ordered linear
Non ordered linear non strict
Non ordered linear strict

```

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