

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ
компьютерной безопасности и
криптографии

Универсальные алгебры и алгебра отношений

ОТЧЁТ

ПО ДИСЦИПЛИНЕ

«ПРИКЛАДНАЯ УНИВЕРСАЛЬНАЯ АЛГЕБРА»

студентки 3 курса 331 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Шуликиной Анастасии Александровны

Преподаватель

профессор, д.ф.-м.н.

подпись, дата

В. А. Молчанов

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Цель работы и порядок её выполнения	4
2 Теория	5
2.1 Понятие алгеброической операции и классификация свойств операций	5
2.1.1 Алгоритм проверки свойства коммутативности	5
2.1.2 Алгоритм поиска номера элемента	6
2.1.3 Алгоритм проверки свойства ассоциативности	6
2.1.4 Алгоритм проверки свойства идемпотентность	7
2.1.5 Алгоритм проверки свойства дистрибутивности	7
2.2 Основные операции над бинарными отношениями	8
2.2.1 Алгоритм объединения бинарных отношений	8
2.2.2 Алгоритм пересечения бинарных отношений	9
2.2.3 Алгоритм произведения бинарных отношений	9
2.2.4 Алгоритм дополнения бинарных отношений	9
2.2.5 Алгоритм обратного бинарного отношения	10
2.3 Основные операции над матрицами	10
2.3.1 Алгоритм сложения матриц	11
2.3.2 Алгоритм умножения матриц	12
2.3.3 Алгоритм транспонирования матрицы	12
2.3.4 Алгоритм зачеркивания строк и столбцов	12
2.3.5 Алгоритм нахождения детерминанта	13
2.3.6 Алгоритм обращения матрицы	13
3 Решение задач	15
4 Программная реализация рассмотренных алгоритмов	18
4.1 Результаты тестирования программы	18
4.2 Код программы, на основе рассмотренных алгоритмов, на языке C++	19

ВВЕДЕНИЕ

В данной лабораторной работе поставлена задача рассмотрения понятия алгебраических операций и классификация свойств операций, основные операции над бинарными отношениями, основные операции над матрицами, написание алгоритмов для изученных тем и выполнение заданий, представленных в данной лабораторной работе.

1 Цель работы и порядок её выполнения

Цель работы – изучение основных понятий универсальной алгебры и операций над бинарными отношениями.

Порядок выполнения работы:

1. Рассмотреть понятие алгебраической операции и классификацию свойств операций. Разработать алгоритмы проверки свойств операций: ассоциативность, коммутативность, идемпотентность, обратимость, дистрибутивность.
2. Рассмотреть основные операции над бинарными отношениями. Разработать алгоритмы выполнения операции над бинарными отношениями.
3. Рассмотреть основные операции над матрицами. Разработать алгоритмы выполнения операций над матрицами.

2 Теория

2.1 Понятие алгебраической операции и классификация свойств операций

Пусть A – непустое множество и n – неотрицательное целое число.

Отображение $f : A^n \rightarrow A$ называется алгебраической n -арной операцией или просто алгебраической операцией на множестве A . При этом n называется порядком или арностью алгебраической операции f .

В случае $n = 0$ по определению $A^0 = \emptyset$ и, значит, 0-арная операция f просто выделяет в множестве A некоторый элемент $f(\emptyset) \in A$. В случае $n = 1$ операция f называется также унарной и в случае $n = 2$ - бинарной. Для унарных и бинарных операций, как правило, используется специальная символическая запись.

В решеточно упорядоченном множестве $A = (A, \leq)$ определены две бинарные операции: $a \wedge b = \inf(a, b)$ и $a \vee b = \sup(a, b)$. Операция \wedge называется пересечением, а операция \vee - объединением. Эти операции решетки взаимосвязаны с ее порядком по формулам: $a \leq b \Leftrightarrow a \wedge b = a$ и $a \leq b \Leftrightarrow a \vee b = b$.

Свойства операций решетки:

1. $a \wedge a = a$, $a \vee a = a$ - идемпотентность операций \wedge, \vee ;
2. $a \wedge b = b \wedge a$, $a \vee b = b \vee a$ - коммутативность операций \wedge, \vee ;
3. $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, $(a \vee b) \vee c = a \vee (b \vee c)$ - ассоциативность операций \wedge, \vee ;
4. $a \wedge (a \vee b) = a$, $a \vee (a \wedge b) = a$ - законы поглощения.

Решетка A называется дистрибутивной, если для любых $a, b, c \in A$ выполняются равенства

$$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c),$$

$$(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$$

2.1.1 Алгоритм проверки свойства коммутативности

Вход. Таблица Кэли A пересечения и объединения размерностью N и список элементов x .

Выход. «Операция пересечения обладает свойством коммутативности» или «Операция пересечения не обладает свойством коммутативности» или «Операция объединения обладает свойством коммутативности» или «Операция объединения не обладает свойством коммутативности»

Шаг 1. Создаем булеву переменную $chek = true$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до N .

Шаг 2.2. Если таблица $A_{(ij)} \neq A_{(ji)}$, то $chek = false$.

Шаг 3. Иначе, если $chek = true$, то выводим «Операция пересечения обладает свойством коммутативности» или «Операция объединения обладает свойством коммутативности».

Трудоемкость алгоритма $O(N^2)$

2.1.2 Алгоритм поиска номера элемента

Вход. Элемент a , список элементов x размерностью N .

Выход. Номер элемента a в списке x .

Шаг 1. Цикл i от 1 до N .

Шаг 1.1. Если $x[i] = a$, то $return i$

Трудоемкость алгоритма $O(N)$

2.1.3 Алгоритм проверки свойства ассоциативности

Вход. Таблица Кэли A пересечения и объединения размерностью N и список элементов x .

Выход. «Операция пересечения обладает свойством ассоциативности» или «Операция пересечения не обладает свойством ассоциативности» или «Операция объединения обладает свойством ассоциативности» или «Операция объединения не обладает свойством ассоциативности»

Шаг 1. Создаем булеву переменную $chek = true$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до N .

Шаг 2.2. Для каждого j цикл k от 1 до N .

Шаг 2.3. Для каждого k применяем алгоритм поиска номера элемента 2.1.2 для каждого $A_{(ij)}$ и $A_{(jk)}$ и сохраняем результаты в $i2$ и $j2$.

Шаг 3. Если $A_{i2,k} \neq A_{i,j2}$, то $chek = false$.

Шаг 4. Иначе, если $chek = true$, то выводим «Операция пересечения обладает свойством ассоциативности» или «Операция объединения обладает свойством ассоциативности».

Трудоемкость алгоритма $O(N^3)$

2.1.4 Алгоритм проверки свойства идемпотентности

Вход. Таблица Кэли A пересечения и объединения размерностью N и список элементов x .

Выход. «Операция пересечения обладает свойством идемпотентности» или «Операция пересечения не обладает свойством идемпотентности» или «Операция объединения обладает свойством идемпотентности» или «Операция объединения не обладает свойством идемпотентности»

Шаг 1. Создаем булеву переменную $chek = true$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i если $A_{ii} \neq x[i]$, то $chek = false$.

Шаг 3. Иначе, если $chek = true$, то выводим «Операция пересечения обладает свойством идемпотентности» или «Операция объединения обладает свойством идемпотентности».

Трудоемкость алгоритма $O(N)$

2.1.5 Алгоритм проверки свойства дистрибутивности

Вход. Таблица Кэли A пересечения и объединения размерностью N и список элементов x .

Выход. «Решетка объединения является дистрибутивной» или «Решетка объединения не является дистрибутивной» или «Решетка пересечения является дистрибутивной» или «Решетка пересечения не является дистрибутивной»

Шаг 1. Создаем булеву переменные проверки $chek = 0$, $chek2 = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до N .

Шаг 2.2. Для каждого j цикл k от 1 до N .

Шаг 2.3. Для каждого k применяем алгоритм поиска номера элемента 2.1.2 для каждого $A_{(ij)}$ (пересечения и объединения), $A_{(ik)}$ (пересечения и объединения) и $A_{(jk)}$ (пересечения и объединения) и сохраняем результаты в $i1$, $i2$ и $j2$.

Шаг 3. Если решетка объединения $A_{(i1,k)}$, в которой поиск номеров элементов находился при подаче решетки пересечения, равен решетке пересечения $A_{(i2,j2)}$, в которой поиск номеров элементов находился при подаче решетки объединения, то $chek++$.

Шаг 3.1. Если решетка пересечения $A_{(i1,k)}$, в которой поиск номеров элементов находился при подаче решетки объединения, равен решетке объединения $A_{(i2,j2)}$, в которой поиск номеров элементов находился при подаче решетки пересечения, то $chek2++$.

Шаг 4. Если $chek > 0$, то выводим «Решетка объединения является дистрибутивной», если $chek2 > 0$, то выводим «Решетка пересечения является дистрибутивной».

Трудоемкость алгоритма $O(N^3)$

2.2 Основные операции над бинарными отношениями

Бинарным отношением между элементами A и B называется любое подмножество ρ множества $A \times B$, то есть $\rho \subset A \times B$.

По определению, бинарным отношением называется множество пар. Если ρ – бинарное отношение (т.е. множество пар), то говорят, что параметры x и y связаны бинарным отношением ρ , если пара $\langle x, y \rangle$ является элементом ρ , т.е. $\langle x, y \rangle \in \rho$.

Основными операциями над бинарными отношениями является:

1. Объединение. Есть $\rho \subset A \times B$ и $s \subset A \times B$, их отношение $\rho \cup s = \{(a, b) : (a, b) \in \rho \mid (a, b) \in s\}$.
2. Пересечение. Есть $\rho \subset A \times B$ и $s \subset A \times B$, их отношение $\rho \cap s = \{(a, b) : (a, b) \in \rho \ \& \ (a, b) \in s\}$.
3. Дополнение. Дополнением $\rho \subset A \times B$ называется отношение $\rho_1 \subset A \times B$, если $\rho_1 = \{(a, b) : (a, b) \notin \rho\}$.
4. Композиция(произведение). Есть $\rho \subset A \times B$ и $s = B \times C$, их отношение $\rho \times s = \{(a, c) : (a, b) \in \rho \ \& \ (b, c) \in s \text{ для некоторого } b \in B\}$.
5. Обратное. Есть $\rho \subset A \times B$, отношение $\rho^{-1} = \{(a, b) : (b, a) \in \rho\}$.

2.2.1 Алгоритм объединения бинарных отношений

Вход. Две матрицы $a_{(ij)}$, $b_{(ij)}$ бинарного отношения размерностью $N \times M$.

Выход. Матрица объединения бинарного отношения res .

Шаг 1. Создаем переменную $res = a_{(ij)}$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до M .

Шаг 2.2. Для каждого j если $res[i][j] == 1 \parallel b[i][j] == 1$, то $res[i][j] =$

1.

Шаг 3. *return* $res_{(ij)}$.

Трудоемкость алгоритма $O(N^2)$

2.2.2 Алгоритм пересечения бинарных отношений

Вход. Две матрицы $a_{(ij)}$, $b_{(ij)}$ бинарного отношения размерностью $N \times M$.

Выход. Матрица пересечения бинарного отношения res .

Шаг 1. Создаем переменную $res = a_{(ij)}$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до M .

Шаг 2.2. Для каждого j если $res[i][j] == 1 \&\& b[i][j] == 1$, то $res[i][j] =$

1.

Шаг 3. *return* $res_{(ij)}$.

Трудоемкость алгоритма $O(N^2)$

2.2.3 Алгоритм произведения бинарных отношений

Вход. Две матрицы бинарного отношения $a_{(ij)}$ размерностью $N \times M$, $b_{(ij)}$ размерностью $M \times M2$.

Выход. Матрица произведения бинарного отношения res .

Шаг 1. Создаем переменную $res = a_{(ij)}$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до $M2$ и создает переменную $count = 0$.

Шаг 2.2. Для каждого j цикл k от 1 до M .

Шаг 2.3. Для каждого k $count + = a[i][k] * b[k][j]$, если $count > 0$, то $res[i][j] = 1$ иначе $res[i][j] = 0$.

Шаг 3. *return* $res_{(ij)}$.

Трудоемкость алгоритма $O(N^3)$

2.2.4 Алгоритм дополнения бинарных отношений

Вход. Матрица $a_{(ij)}$ бинарного отношения размерностью $N \times M$.

Выход. Матрица дополнения бинарного отношения res .

Шаг 1. Создаем переменную $res = a_{(ij)}$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до M .

Шаг 2.2. Для каждого j если $res[i][j] == 1$, то $res[i][j] = 0$, иначе $res[i][j] = 1$.

Шаг 3. *return* $res_{(ij)}$.

Трудоемкость алгоритма $O(N^2)$

2.2.5 Алгоритм обратного бинарного отношения

Вход. Матрица $a_{(ij)}$ бинарного отношения размерностью $N \times M$.

Выход. Матрица нахождения обратного бинарного отношения res .

Шаг 1. Создаем переменную $res = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до M .

Шаг 2.2. Для каждого j если $res[i][j] = a[j][i]$.

Шаг 3. *return* $res_{(ij)}$.

Трудоемкость алгоритма $O(N^2)$

2.3 Основные операции над матрицами

Матрица – математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), который представляет собой совокупность строк и столбцов, на пересечении которых находятся его элементы. Количество строк и столбцов задает размер матрицы. Если у матрицы одинаковое число строк и столбцов, ее называют квадратной.

Сами числа называют элементами матрицы и характеризуют их положением в матрице, задавая номер строки и номер столбца и записывая их в виде двойного индекса, причем вначале записывают номер строки, а затем столбца.

Главной диагональю квадратной матрицы называют элементы, имеющие одинаковые индексы, то есть те элементы, у которых номер строки совпадает с номером столбца. Побочная диагональ идет «перпендикулярно» главной диагонали.

Особую важность представляют собой так называемые единичные матрицы. Это квадратные матрицы, у которых на главной диагонали стоят 1, а все остальные числа равны 0. Обозначают единичные матрицы E . Матрицы

называют равными, если у них равны число строк, число столбцов, и все элементы, имеющие одинаковые индексы, равны. Матрица называется нулевой, если все ее элементы равны 0. Обозначается нулевая матрица O .

Основные операции над матрицами:

1. Сложение. Складывать можно только матрицы одинакового размера, то есть имеющие одинаковое число строк и одинаковое число столбцов. При сложении матриц соответствующие их элементы складываются, то есть имея $A = (a_{ij})$ и $B = (b_{ij})$ при сложении получаем $A + B = (a_{ij} + b_{ij})$.
2. Умножение. Умножать можно матрицы, если число столбцов первой матрицы равно числу строк второй матрицы. В результате получится матрица, число строк которой равно числу строк первой матрицы, а число столбцов равно числу столбцов второй матрицы. То есть, имея $A = (a_{ij})$ размерностью $(n \times m)$ и $B = (b_{ij})$ размерностью $(m \times l)$, при умножении матриц получаем матрицу размерности $(n \times l)$: $A_{ij} \times B_{ij} = a[i][1]b[1][j] + a[i][2]b[2][j] + \dots + a[i][m]b[m][j] = \sum_{n=1}^m a[n][m]b[m][n]$.
3. Транспонирование. При транспонировании у матрицы строки становятся столбцами и наоборот. Полученная матрица называется транспонированной и обозначается A^T . Имея матрицу $A = (a_{ij})$ получаем $A^T = (a_{ji})$.
4. Обращение матрицы. Рассмотрим квадратную матрицу A . Матрица A^{-1} называется обратной к матрице A , если их произведения равны единичной матрице $A \times A^{-1} = E$. Обратная матрица существует только для квадратных матриц. Обратная матрица существует, только если матрица A невырождена, то есть ее определитель $\det A$ не равен нулю. В противном случае обратную матрицу посчитать невозможно. Пусть имеем матрицу $A = (a_{ij})$ размерностью $n \times n$, тогда обратную матрицу

$$\text{найдем: } A^{-1} = \frac{1}{\det A} \times \begin{pmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{pmatrix}.$$

2.3.1 Алгоритм сложения матриц

Вход. Две матрицы $a_{(ij)}$, $b_{(ij)}$ размерностью $N \times M$ и простое число che , по модулю которого будет выполняться операция.

Выход. Результат сложения матриц res .

Шаг 1. Создаем переменную $res = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до M .

Шаг 2.2. Для каждого j : $res[i][j] = (a[i][j] + b[i][j]) \% che$.

Шаг 3. $return\ res_{(ij)}$.

Трудоемкость алгоритма $O(N * M)$

2.3.2 Алгоритм умножения матриц

Вход. Две матрицы $a_{(ij)}$, $b_{(ij)}$ размерностью $N \times M$ и $N2 \times M$ и простое число che , по модулю которого будет выполняться операция.

Выход. Результат умножения матриц res .

Шаг 1. Создаем переменную $res = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до L .

Шаг 2.2. Для каждого j цикл k от 1 до M .

Шаг 2.3. Для каждого k : $res[i][j] += a[i][k] + b[k][j]$, $res[i][j] = res[i][j] \% che$.

Шаг 3. $return\ res_{(ij)}$.

Трудоемкость алгоритма $O(N * M * L)$

2.3.3 Алгоритм транспонирования матрицы

Вход. Матрица $a_{(ij)}$ размерностью $N \times M$ и простое число che , по модулю которого будет выполняться операция.

Выход. Результат транспонирования матрицы res .

Шаг 1. Создаем переменную $res = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Для каждого i цикл j от 1 до N .

Шаг 2.2. Для каждого j : $res[i][j] = a[j][i]$, $res[i][j] = res[i][j] \% che$.

Шаг 3. $return\ res_{(ij)}$.

Трудоемкость алгоритма $O(N * M)$

2.3.4 Алгоритм зачеркивания строк и столбцов

Вход. Матрица $a_{(ij)}$ размерностью N , строки row и столбцы col , которые нужно вычеркнуть из матрицы.

Выход. Матрица res .

Шаг 1. Создаем переменные $ki = 0$, $kj = 0$.

Шаг 2. Цикл i от 1 до N .

Шаг 2.1. Если $i! = row$, то для каждого i цикл j от 1 до N .

Шаг 2.2. Если $j! = col$, то для каждого j : $res[ki][kj] = a[i][j]$.

Шаг 3. $return\ res_{(ki,kj)}$.

Трудоемкость алгоритма $O(N^2)$

2.3.5 Алгоритм нахождения детерминанта

Вход. Матрица $a_{(ij)}$ размерностью N и простое число che , по модулю которого будет выполняться операция.

Выход. Матрица res .

Шаг 1. Создаем переменные $res = 0$, $k = 1$.

Шаг 2. Если $N == 1$, то $res = a[0][0]$.

Шаг 3. Иначе, если $N == 2$, то $res = a[0][0] * a[1][1] - a[1][0] * a[0][1]$.

Шаг 4. Иначе цикл i от 1 до N .

Шаг 4.1. Для каждого i цикл j от 1 до $N - 1$.

Шаг 4.2. Производится вычеркивание строки и столбца матрицы по алгоритму 2.3.4 и возвращается новый результат res_2 .

Шаг 4.3. $res = res + k * a[0][i] * \text{прохождение данного алгоритма 2.3.5 заново, } k = -k$

Шаг 5. Пока $res < 0$ $(res + che) \% che$

Шаг 6. $return\ res \% che$.

Трудоемкость алгоритма $O(N!)$

2.3.6 Алгоритм обращения матрицы

Вход. Матрица $a_{(ij)}$ размерностью N и простое число che , по модулю которого будет выполняться операция.

Выход. Результат обращения матрицы res .

Шаг 1. Создаем переменную $obr = 0$, $invDet$.

Шаг 2. Получаем определитель матрицы det с помощью алгоритма 2.3.5.

Шаг 3. Пока $det < 0$, $det = det + che$.

Шаг 4. $det = det \% che$.

Шаг 5. Создаем цикл i от 2 до 99 000, если $(i * det) \% che == 1$, то $ind_{det} = i$.

Шаг 6. Если $\det == 0$, то матрица является вырожденной и обратную матрицу получить нельзя.

Шаг 7. Иначе создаем цикл i от 0 до N .

Шаг 7.1. Для каждого i создаем цикл j от 0 до N .

Шаг 7.2. Для каждого j создаем цикл k от 0 до $N - 1$.

Шаг 7.3. Отправляем на алгоритм 2.3.4.

Шаг 7.4. Создаем новую переменную $A = -1^{i+j+2} * \text{определитель}$, полученный из матрицы на шаге 7.4.

Шаг 7.5. Пока $A < 0$, $A = A + che$.

Шаг 7.6. $A = A \% che$.

Шаг 7.7. $obr = (A * invDet) \% che$.

Шаг 8. Транспонируем матрицу obr по алгоритму 2.3.3.

Трудоемкость алгоритма $O(N^3 * M!)$ M – минор матрицы.

3 Решение задач

Вариант 9.

Задание 1.

\cdot	a	b	c	d
a	b	b	a	a
b	b	b	b	b
c	a	b	d	c
d	a	b	c	d

$x \cdot a$ и z	a	b	c	d
$a \cdot a = b$	b	b	b	b
$b \cdot a = b$	b	b	b	b
$c \cdot a = a$	b	b	a	a
$d \cdot a = a$	b	b	a	a
x и $a \cdot z$	$a \cdot a = b$	$a \cdot b = b$	$a \cdot c = b$	$a \cdot d = b$
a	b	b	b	b
b	b	b	b	b
c	b	b	a	a
d	b	b	a	a
$x \cdot b$ и z	a	b	c	d
$a \cdot b = b$	b	b	b	b
$b \cdot b = b$	b	b	b	b
$c \cdot b = b$	b	b	b	b
$d \cdot b = b$	b	b	b	b
x и $b \cdot z$	$b \cdot a = b$	$b \cdot b = b$	$b \cdot c = b$	$b \cdot d = b$
a	b	b	b	b
b	b	b	b	b
c	b	b	b	b
d	b	b	b	b
$x \cdot c$ и z	a	b	c	d
$a \cdot c = a$	b	b	a	a
$b \cdot c = b$	b	b	b	b
$c \cdot c = d$	a	b	c	d
$d \cdot c = c$	a	b	d	c

x и $c \cdot z$	$c \cdot a = a$	$c \cdot b = b$	$c \cdot c = d$	$c \cdot d = c$
a	b	b	a	a
b	b	b	b	b
c	a	b	c	d
d	a	b	d	c
$x \cdot d$ и z	a	b	c	d
$a \cdot d = a$	b	b	a	a
$b \cdot d = b$	b	b	b	b
$c \cdot d = c$	a	b	d	c
$d \cdot d = d$	a	b	c	d
x и $d \cdot z$	$d \cdot a = a$	$d \cdot b = b$	$d \cdot c = c$	$d \cdot d = d$
a	b	b	a	a
b	b	b	b	b
c	a	b	d	c
d	a	b	c	d

Так $\forall x, y, z \in Z_4^\times$ выполняется $(x \cdot y)z = x(y \cdot z)$. Значит, данная операция ассоциативна и Z_4^\times является полугруппой.

Задание 2.

$$\lambda = 9$$

$$A^2 + (10 - 9/2) \times A + 9/2 \times E,$$

$$A = \begin{pmatrix} 1 & -2 \\ -3 & 9 \end{pmatrix},$$

$$\begin{pmatrix} 1 & -2 \\ -3 & 9 \end{pmatrix} \times \begin{pmatrix} 1 & -2 \\ -3 & 9 \end{pmatrix} + \begin{pmatrix} 5,5 & -11 \\ -16,5 & 49,5 \end{pmatrix} + \begin{pmatrix} 4,5 & 0 \\ 0 & 4,5 \end{pmatrix} =$$

$$\begin{pmatrix} 7 & -20 \\ -30 & 87 \end{pmatrix} + \begin{pmatrix} 5,5 & -11 \\ -16,5 & 49,5 \end{pmatrix} + \begin{pmatrix} 4,5 & 0 \\ 0 & 4,5 \end{pmatrix} = \begin{pmatrix} 17 & -31 \\ 46,5 & 141 \end{pmatrix}.$$

Задание 3.

$$A = \begin{pmatrix} -1 & 9 & 3 \\ 9/3 & 2 & 8 - \frac{9}{3} \end{pmatrix}$$

$$B = \begin{pmatrix} -9 & 2 \\ 1 & 10 - \frac{9}{2} \\ -3 & 9 \end{pmatrix}$$

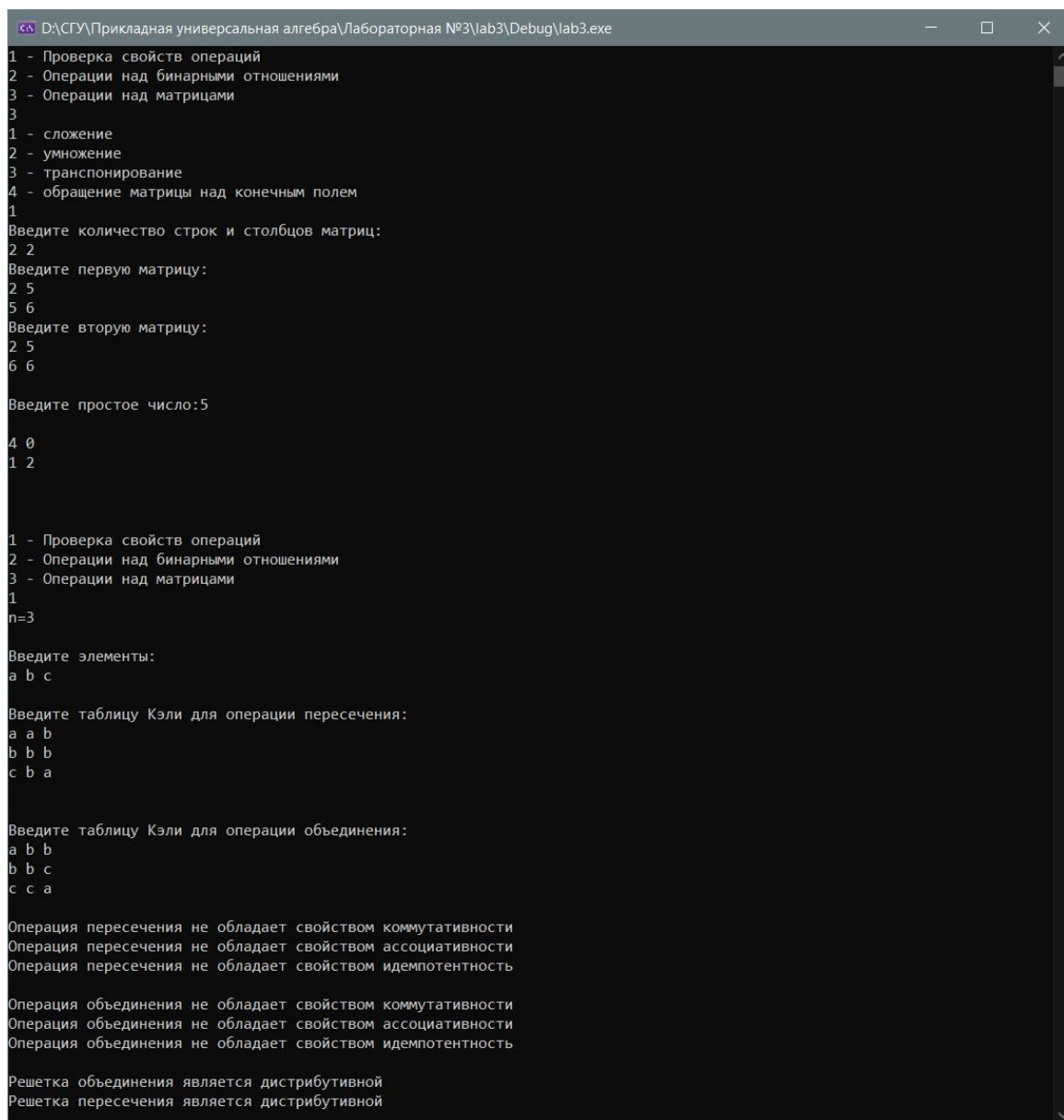
$$A \times B = \begin{pmatrix} -1 & 9 & 3 \\ 3 & 2 & 5 \end{pmatrix} \times \begin{pmatrix} -9 & 2 \\ 1 & 5,5 \\ -3 & 9 \end{pmatrix} =$$

$$\begin{pmatrix} 9 + 9 + (-9) & (-2) + 49,5 + 27 \\ (-27) + 2 + (-15) & 6 + 11 + 45 \end{pmatrix} = \begin{pmatrix} 9 & 74,5 \\ -40 & 62 \end{pmatrix}.$$

4 Программная реализация рассмотренных алгоритмов

4.1 Результаты тестирования программы

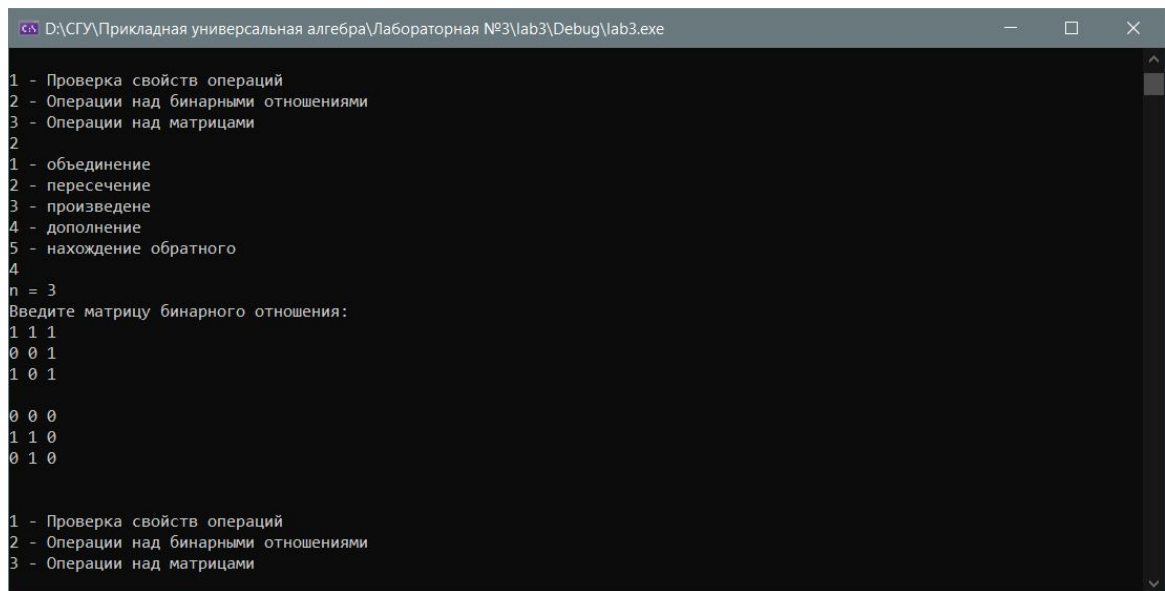
На рисунках 1-3 можно увидеть работу, реализуемой программы, по рассмотренным алгоритмам.



```
D:\СТУ\Прикладная универсальная алгебра\Лабораторная №3\lab3\Debug\lab3.exe
1 - Проверка свойств операций
2 - Операции над бинарными отношениями
3 - Операции над матрицами
3
1 - сложение
2 - умножение
3 - транспонирование
4 - обращение матрицы над конечным полем
1
Введите количество строк и столбцов матриц:
2 2
Введите первую матрицу:
2 5
5 6
Введите вторую матрицу:
2 5
6 6
Введите простое число:5
4 0
1 2

1 - Проверка свойств операций
2 - Операции над бинарными отношениями
3 - Операции над матрицами
1
n=3
Введите элементы:
a b c
Введите таблицу Кэли для операции пересечения:
a a b
b b b
c b a
Введите таблицу Кэли для операции объединения:
a b b
b b c
c c a
Операция пересечения не обладает свойством коммутативности
Операция пересечения не обладает свойством ассоциативности
Операция пересечения не обладает свойством идемпотентность
Операция объединения не обладает свойством коммутативности
Операция объединения не обладает свойством ассоциативности
Операция объединения не обладает свойством идемпотентность
Решетка объединения является дистрибутивной
Решетка пересечения является дистрибутивной
```

Рисунок 1



```
D:\СГ\Прикладная универсальная алгебра\Лабораторная №3\lab3\Debug\lab3.exe
1 - Проверка свойств операций
2 - Операции над бинарными отношениями
3 - Операции над матрицами
2
1 - объединение
2 - пересечение
3 - произведение
4 - дополнение
5 - нахождение обратного
4
n = 3
Введите матрицу бинарного отношения:
1 1 1
0 0 1
1 0 1

0 0 0
1 1 0
0 1 0

1 - Проверка свойств операций
2 - Операции над бинарными отношениями
3 - Операции над матрицами
```

Рисунок 2

4.2 Код программы, на основе рассмотренных алгоритмов, на языке C++

```
#include <iostream>
#include <vector>

using namespace std;

int search_num(char a, char* x, int n) {
    for (int i = 0; i < n; i++)
        if (x[i] == a)
            return i;
}

bool commutativity(char** mat, int n, char* x) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (mat[i][j] != mat[j][i])
                return false;
    return true;
}
```

```

bool associativity(char** mat, int n, char* x) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++)
                if (mat[search_num(mat[i][j], x, n)][k]
                    != mat[i][search_num(mat[j][k], x, n)])
                    return false;
    return true;
}

bool idempotivity(char** mat, int n, char* x) {
    for (int i = 0; i < n; i++)
        if (mat[i][i] != x[i])
            return false;
    return true;
}

void distributivity(char** peres,
char** uni, int n, char* x) {
    int chek = 0;
    int chek2 = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            for (int k = 0; k < n; k++) {
                if (uni[search_num(peres[i][j], x, n)][k] =
peres[search_num(uni[i][k], x, n)]
[search_num(uni[j][k], x, n)]) {
                    chek++;
                }
            }
    if (peres[search_num(uni[i][j], x, n)][k] =
uni[search_num(peres[i][k], x, n)]
[search_num(peres[j][k], x, n)]) {
        chek2++;
    }
}

```

```

}
if (chek > 0)
cout << "The union lattice is distributive\n";
else
cout << "The union lattice is not distributive\n";
if (chek2 > 0)
cout << "The intersection lattice is distributive\n";
else
cout << "The intersection lattice is not distributive\n";
}

void zad1() {
    int n;
    cout << "n=";
    cin >> n;

    cout << "\nEnter elements:\n";
    char* x = new char[n];
    for (int i = 0; i < n; i++)
        cin >> x[i];

    cout << "\nEnter the Cayley table
for the intersection operation:\n";
    char** peres = new char* [n];
    for (int i = 0; i < n; i++) {
        peres[i] = new char[n];
        for (int j = 0; j < n; j++) {
            cin >> peres[i][j];
        }
    }
    cout << endl;

    cout << "\nEnter the Cayley table
for the union operation:\n";

```

```

char** uni = new char* [n];
for (int i = 0; i < n; i++) {
    uni[i] = new char[n];
    for (int j = 0; j < n; j++) {
        cin >> uni[i][j];
    }
}
cout << endl;

if (commutativity(peres, n, x))
cout << "The operation of intersection
has the property of commutativity\n";
else cout << "The operation of
intersection does not have the
commutativity property\n";
if (associativity(peres, n, x))
cout << "The intersection operation
has the associativity property\n";
else cout << "The intersection operation
does not have the associativity property\n";
if (idenpotivity(peres, n, x))
cout << "The intersection operation has
the idempotency property\n\n";
else cout << "The intersection operation does
not have the idempotency property\n\n";

if (commutativity(uni, n, x))
cout << "The union operation has
the commutativity property\n";
else cout << "The union operation
does not have the commutativity property\n";
if (associativity(uni, n, x))
cout << "The union operation has the
associativity property\n";

```

```

        else cout << "The union operation
does not have the associativity property\n";
        if (idenpotivity(uni, n, x))
        cout << "The union operation has
the idempotent property\n\n";
        else cout << "The union operation
does not have the idempotent property\n\n";

        distributivity(peres, uni, n, x);
    }

```

```

void uni_bin(int** a, int** a2, int n) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[n];
        for (int j = 0; j < n; j++) {
            res[i][j] = a[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (res[i][j] == 1 || a2[i][j] == 1)
                res[i][j] = 1;
            cout << res[i][j] << ' ';
        }
        cout << endl;
    }
    //delete a, a2, res;
}

```

```

void pere_bin(int** a, int** a2, int n) {
    int** res;
    res = new int* [n];

```

```

    for (int i = 0; i < n; i++) {
        res[i] = new int[n];
        for (int j = 0; j < n; j++) {
            res[i][j] = a[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (res[i][j] == 1 && a2[i][j] == 1)
                res[i][j] = 1;
            else res[i][j] = 0;
            cout << res[i][j] << ' ';
        }
        cout << endl;
    }

    //delete a, a2, res;
}

void multi_bin(int** a, int** a2,
int n, int m, int n2, int m2) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[m];
        for (int j = 0; j < m; j++) {
            res[i][j] = a[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m2; j++) {
            int count = 0;
            for (int k = 0; k < n2; k++) {
                count += a[i][k] * a2[k][j];
            }
        }
    }
}

```



```

        }
        if (count > 0)
            res[i][j] = 1;
        else res[i][j] = 0;
        cout << res[i][j] << ' ';
    }
    cout << endl;
}

//delete a, a2, res;
}

void add_bin(int** a, int n) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[n];
        for (int j = 0; j < n; j++) {
            res[i][j] = a[i][j];
        }
    }
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (res[i][j] == 1)
                res[i][j] = 0;
            else res[i][j] = 1;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << res[i][j] << ' ';
        }
    }
}

```

```

        cout << endl;
    }
    //delete a, a2, res;
}

void reverse_bin(int** a, int n) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[n];
        for (int j = 0; j < n; j++) {
            res[i][j] = a[j][i];
            cout << res[i][j] << ' ';
        }
        cout << endl;
    }
    //delete a, a2, res;
}

```

```

void bin_relation() {
    int x = 0;
    cout << "1 - union
\n2 - intersection
\n3 - produced
\n4 - addition
\n5 - finding the reverse" << endl;
    cin >> x;
    if (x == 1 || x == 2 || x == 3) {
        int n;
        cout << "n = ";
        cin >> n;
    }
}

```

```

int** a;
a = new int* [n];
cout << "Input First
Binary Relation Matrix:" << endl;
for (int i = 0; i < n; i++) {
    a[i] = new int[n];
    for (int j = 0; j < n; j++) {
        cin >> a[i][j];
    }
}

int** a2;
a2 = new int* [n];
cout << "Input Second
Binary Relation Matrix:" << endl;
for (int i = 0; i < n; i++) {
    a2[i] = new int[n];
    for (int j = 0; j < n; j++) {
        cin >> a2[i][j];
    }
}
cout << endl;

if (x == 1)
    uni_bin(a, a2, n);
if (x == 2)
    pere_bin(a, a2, n);
if (x == 3)
    multi_bin(a, a2, n);
}

else {
    int n;
    cout << "n = ";

```

```

        cin >> n;

        int** a;
        a = new int* [n];
        cout << "Input Binary
        Relation Matrix:" << endl;
        for (int i = 0; i < n; i++) {
            a[i] = new int[n];
            for (int j = 0; j < n; j++) {
                cin >> a[i][j];
            }
        }
        cout << endl;

        if (x == 4)
            add_bin(a, n);
        if (x == 5)
            reverse_bin(a, n);
    }
    cout << endl;
}

void add_matr(int** a, int** a2, int n, int m, int che) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[m];
        for (int j = 0; j < m; j++) {
            res[i][j] =
                (a[i][j] + a2[i][j]) % che;
            cout << res[i][j] << ' ';
        }
        cout << endl;
    }
}

```

```

        //delete a, a2, res;
    }

void transponir(int** a, int n, int m, int che) {
    int** res;
    res = new int* [m];
    for (int i = 0; i < m; i++) {
        res[i] = new int[n];
        for (int j = 0; j < n; j++) {
            res[i][j] = a[j][i];
            cout << res[i][j] % che << ' ';
        }
        cout << endl;
    }

    //delete a, a2, res;
}

void multi_matr(int** a, int** a2,
int n, int m, int n2, int m2, int che) {
    int** res;
    res = new int* [n];
    for (int i = 0; i < n; i++) {
        res[i] = new int[m];
        for (int j = 0; j < m; j++) {
            res[i][j] = 0;
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m2; j++) {
            for (int k = 0; k < n2; k++) {
                res[i][j] +=
                a[i][k] * a2[k][j];
            }
        }
    }
}

```

```

        }
        cout << res[i][j] % che << ' ';
    }
    cout << endl;
}

//delete a, a2, res;
}

void transponir_obr(double** a, double** res, int n) {
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            res[j][i] = a[i][j];
}

void get_matr(vector<vector<int>> a,
int n, vector<vector<int>>& temp_matr,
int row, int col) {
    int ki = 0;
    for (int i = 0; i < n; i++)
        if (i != row) {
            int kj = 0;
            for (int j = 0; j < n; j++)
                if (j != col) {
                    temp_matr[ki][kj] = a[i][j];
                    kj++;
                }
            ki++;
        }
}

int Det (vector<vector<int>> a, int n, int che) {
    int temp = 0;
    int k = 1;

```

```

    if (n == 1)
    temp = a[0][0];
    else if (n == 2)
    temp = a[0][0] * a[1][1] - a[1][0] * a[0][1];
    else {
        for (int i = 0; i < n; i++) {
            int n2 = n - 1;
            vector<vector<int>> temp_matr(n2);
            for (int j = 0; j < n2; j++)
                temp_matr[j].resize(n2);
            get_matr(a, n, temp_matr, 0, i);
            temp = temp +
                k * a[0][i] * Det(temp_matr, n2, che);
            k = -k;
        }
    }
    while (temp < 0)
    temp += che;
    temp = temp % che;
    return temp;
}

void appeal(vector<vector<int>> a, int n, int che) {
    double** obr_matr = new double* [n];
    double** tobr_matr = new double* [n];

    for (int i = 0; i < n; i++) {
        obr_matr[i] = new double[n];
        tobr_matr[i] = new double[n];
    }

    int det = Det(a, n, che);
    while (det < 0)
    det = det + che;
}

```

```

det = det % che;

cout << "\nMatrix determinant = " << det << endl;

int inv_det;
for (int i = 1; i < 100000; ++i)
if ((i * det) % che == 1) {
inv_det = i;
break;
}

if (det) {
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
int n2 = n - 1;
vector<vector<int>> temp_matr(n2);
for (int k = 0; k < n2; k++)
temp_matr[k].resize(n2);
get_matr(a, n, temp_matr, i, j);
int A = pow(-1.0, i + j + 2)
* Det(temp_matr, n2, che);
while (A < 0)
A = A + che;
A = A % che;
obr_matr[i][j] = A * inv_det;
obr_matr[i][j] =
(int)obr_matr[i][j] % che;
}
}
}
else
cout << "The matrix is the degenerate " << endl;

if (det) {

```



```

        transponir_obr(obr_matr, tobr_matr, n);
        cout << "Inverse matrix:" << endl;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                cout << tobr_matr[i][j] << " ";
            cout << endl;
        }
    }
}

void matrix() {
    int che;
    int x = 0;
    cout << "1 - addition
\n2 - multiplication
\n3 - transposition
\n4 - inverse matrix" << endl;
    cin >> x;
    if (x == 1) {
        int n, m;
        cout << "Enter the number
        of rows and columns of matrices:" << endl;
        cin >> n >> m;

        int** a;
        a = new int* [n];
        cout << "Enter first matrix:" << endl;
        for (int i = 0; i < n; i++) {
            a[i] = new int[m];
            for (int j = 0; j < m; j++) {
                cin >> a[i][j];
            }
        }
    }
}

```

```

int** a2;
a2 = new int* [n];
cout << "Enter the second matrix:" << endl;
for (int i = 0; i < n; i++) {
    a2[i] = new int[m];
    for (int j = 0; j < m; j++) {
        cin >> a2[i][j];
    }
}
cout << endl;

cout << "Enter a prime number:";
cin >> che;
cout << endl;
add_matr(a, a2, n, m, che);
}
if (x == 2) {
    int n, m, n2, m2;
    cout << "Enter the number of
rows and columns of the first matrix:" << endl;
    cin >> n >> m;

    int** a;
    a = new int* [n];
    cout << "Enter first matrix:" << endl;
    for (int i = 0; i < n; i++) {
        a[i] = new int[m];
        for (int j = 0; j < m; j++) {
            cin >> a[i][j];
        }
    }

    cout << "Enter the number of rows
and columns of the second matrix:" << endl;

```

```

    cin >> n2 >> m2;

    int** a2;
    a2 = new int* [n2];
    cout << "Enter the second matrix:" << endl;
    for (int i = 0; i < n2; i++) {
        a2[i] = new int[m2];
        for (int j = 0; j < m2; j++) {
            cin >> a2[i][j];
        }
    }
    cout << endl;
    cout << "Enter a prime number:";
    cin >> che;
    cout << endl;
    if (m == n2)
        multi_matr(a, a2, n, m, n2, m2, che);
    else cout << "Unable to perform
matrix multiplication" << endl;
}
if (x == 3) {
    int n, m;
    cout << "Enter the number of rows
and columns of the matrix:" << endl;
    cin >> n >> m;

    int** a;
    a = new int* [n];
    cout << "Enter matrix:" << endl;
    for (int i = 0; i < n; i++) {
        a[i] = new int[m];
        for (int j = 0; j < m; j++) {
            cin >> a[i][j];
        }
    }
}

```

```

    }
    cout << endl;
    cout << "Enter a prime number:";
    int che;
    cin >> che;
    cout << endl;
    transponir(a, n, m, che);
}
cout << endl;
if (x == 4) {
    int n;
    cout << "Enter the number of rows and
columns of the matrix:" << endl;
    cin >> n;
    vector<vector<int>>> a(n);
    cout << "Enter matrix:" << endl;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j)
        {
            int m;
            cin >> m;
            a[i].push_back(m);
        }
    }
    cout << "Enter a prime number:";
    int che;
    cin >> che;
    cout << endl;
    appeal(a, n, che);
}
cout << endl;
}

int main()

```

```

{
setlocale (LC_ALL, "ru");
for (;;) {
    cout << "1 — Checking operation properties
\n2 — Operations on binary relations
\n3 — Matrix operations \n";
    int x;
    cin >> x;
    switch (x) {
        case 1:
            zad1();
            cout << endl;
            break;
        case 2:
            bin_relation();
            cout << endl;
            break;
        case 3:
            matrix();
            cout << endl;
            break;
    }
}
}

```

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были рассмотрены и реализованы основные понятия универсальной алгебры, операции над бинарными отношениями и операции над матрицами. Были рассмотрены понятия алгебраических операций и реализация алгоритмов проверки свойств операций: ассоциативности, коммутативности, идемпотентности, обратимости, дистрибутивности. Также изучены и реализованы основные операции над бинарными отношениями и основные операции над матрицами.