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»

студента 3 курса 331 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Никитина Арсения Владимировича

Преподаватель
профессор, д.ф.-м.н. ___________ В. А. Молчанов
подпись, дата

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В алгебрах могут быть заданы различные операции, которые могут иметь определенные свойства: ассоциативность, коммутативность, идемпотентность, обратимость и дистрибутивность относительно операции сложения. Операции могут быть заданы с помощью матриц, при этом матрично можно проверить их свойства. Существует понятие операций над бинарными отношениями. По определенным алгоритмам можно производить ряд операций над матрицами: объединение, пересечение, инверсию, обращение и композицию. Также существуют как унарные (то есть операции, определенные для одной матрицы) и бинарные операции над матрицами отношений: сложение, вычитание, произведение матрицы на константу, произведение двух матриц и транспонирование матрицы.

1 Цель работы и порядок ее 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 Теоретические сведения

2.1 Алгебраические операции

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

Далее для бинарной операции f по возможности будем использовать мультипликативную запись с помощью символа $\cdot$, то есть вместо $f(x, y)$ писать $x \cdot y$.

2.1.1 Свойства алгебраических операций

Бинарная операция $\cdot$ на множестве A называется:

1. *ассоциативной*, если $\forall x, y, z \in A$ выполняется $x \cdot (y \cdot z) = (x \cdot y) \cdot z$,
2. *коммутативной*, если $\forall x, a \in A$ выполняется $x \cdot y = y \cdot x$,
3. *идемпотентной*, если $\forall x \in A$ выполняется $x \cdot x = x$,
4. *обратимой*, если $\forall x \in A\ \exists a : x \cdot a = a \cdot x = 1$,
5. *дистрибутивной относительно операции* +, если $\forall x, y, z \in A$ выполняются равенства $x \cdot (y + z) = (x \cdot y) + (x \cdot z),\ (y + z) \cdot x = (y \cdot x) + (z \cdot x)$.

2.1.2 Алгоритм проверки операции на ассоциативность

Вход: Таблица Кэли операции $A = (a_{ij})$, размерности $n \times n$ и множество S размерности n, на котором задана операция.

Выход: «Операция является ассоциативной» или «Операция не является ассоциативной».

1. Цикл по i от 1 до n.
   а) Создать пустое множество S_1.
   б) Цикл по j от 1 до n.
      i. Добавить в S_1 значение $a[i][j]$.
   в) Создать пустое множество A_1 размерности $n \times n$ для таблицы Кэли.
   г) Цикл по j от 1 до n, цикл по k от 1 до n.
      i. Создать переменную ind и присвоить ей значение индекса $S_1[k]$ в множестве S.
      ii. Присвоить $A_1[j][k]$ значение $a[j][ind]$.
   д) Создать пустое множество S_2.
   е) Цикл по j от 1 до n.
      i. Добавить в S_2 значение $a[j][i]$.
   ж) Создать пустое множество A_2 размерности $n \times n$ для таблицы Кэли.

з) Цикл по j от 1 до n, цикл по k от 1 до n.

i. Создать переменную ind и присвоить ей значение индекса $S_1[j]$ в множестве S.

ii. Присвоить $A_2[j][k]$ значение $a[ind][k]$.

и) Цикл по j от 1 до n, цикл по k от 1 до n.

i. Если $A_1[j][k] \neq A_2[j][k]$, то ответ — «Операция не является ассоциативной».

2. Ответ — «Операция является ассоциативной».

Трудоемкость алгоритма $O(N^3)$.

2.1.3 Алгоритм проверки операции на коммутативность

Вход: Таблица Кэли операции $A = (a_{ij})$, размерности $n \times n$.

Выход: «Операция является коммутативной» или «Операция не является коммутативной».

1. Цикл по i от 1 до n, цикл по j от 1 до n.

   а) Если $a_{ij} \neq a_{ji}$, то ответ — «Операция не является коммутативной».

2. Ответ — «Операция является коммутативной».

Трудоемкость алгоритма $O(N^2)$.

2.1.4 Алгоритм проверки операции на идемпотентность

Вход: Таблица Кэли операции $A = (a_{ij})$, размерности $n \times n$ и множество S размерности n, на котором задана операция.

Выход: «Операция является идемпотентной» или «Операция не является идемпотентной».

1. Цикл по i от 1 до n.

   а) Если $a[i][i] \neq s[i]$, то ответ — «Операция не является идемпотентной».

2. Ответ — «Операция является идемпотентной».

Трудоемкость алгоритма $O(N)$.

2.1.5 Алгоритм проверки операции на обратимость

Вход: Таблица Кэли операции $A = (a_{ij})$, размерности $n \times n$ и множество S размерности n, на котором задана операция.

Выход: «Список обратных элементов матрицы B» или «Операция не является обратимой».

1. Создать пустое множество B.
2. Цикл по i от 1 до n, цикл по j от 1 до n.
   а) Если $a[i][j] = a[j][i] = 1$, то добавить $S[i]$ во множество B.
3. Если полученное множество B непустое, то ответ — «Список обратных элементов матрицы B», если же множество B пустое, то ответ — «Операция не является обратимой».
   Трудоемкость алгоритма $O(N^2)$.

2.1.6 Алгоритм проверки операции на дистрибутивность

Вход: таблица Кэли операции * $A = (a_{ij})$, размерности $n \times n$, таблица Кэли операции + $A' = (a'_{ij})$, относительно которой делается проверка на дистрибутивность, размерности $n \times n$, множество элементов S, размерности n, на котором заданы обе операции.

Выход: «Операция * является дистрибутивной относительно операции +» или «Операция * не является дистрибутивной относительно операции +».

1. Цикл по i от 1 до n, цикл по j от 1 до n, цикл по k от 1 до n.
   а) Создать переменную $d =$ индекс $a^{'}[j][k]$ во множестве S.
   б) Создать переменную $v =$ индекс $a[i][j]$ во множестве S.
   в) Создать переменную $h =$ индекс $a[i][k]$ во множестве S.
   г) Создать переменную $f =$ индекс $a^{'}[j][k]$
   д) Создать переменную $k =$ индекс $a[k][i]$ во множестве S.
   е) Если $a[i][d] \neq a'[v][h]$ или $a[f][i] \neq a'[t][k]$, то ответ — «Операция * не является дистрибутивной относительно операции +».
2. Ответ — «Операция * является дистрибутивной относительно операции +».
   Трудоемкость алгоритма $O(N^3)$.

2.2 Основные операции над бинарными отношениями

1. Теоретико-множественные операции ($\cup, \cap, \neg$).
2. Обращение бинарных отношений: *обратным* для бинарного отношения $\rho \subset A \times B$ называется бинарное отношение $\rho^{-1} \subset B \times A$, определяемое по формуле: $\rho^{-1} = \{(b, a) : (a, b) \in \rho\}$.
3. Композиция бинарных отношений: *композицией* бинарных отношений $\rho \subset A \times B$ и $\sigma \subset B \times C$ называется бинарное отношение $\rho\sigma = \{(a, c) : (a, b) \in \rho$ и $(b, c) \in \sigma$ для некоторого $b \in B\}$. Если бинарное отношение

$A = (a_{ij})$ размерности $n \times m$ и бинарное отношение $B = (b_{ij})$ размерности $m \times h$, то бинарное отношение композиции этих отношений будет иметь размерность $n \times h$.

2.2.1 Алгоритм построения объединения бинарных отношений

Вход: бинарное отношение $A = (a_{ij})$ размерности $n \times m$ и бинарное отношение $B = (b_{ij})$ размерности $n \times m$.

Выход: бинарное отношение $C = (c_{ij})$ размерности $n \times m$ объединения бинарных отношений A и B.

1. Создать матрицу C размерности $n \times m$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) $c[i][j]$ присвоить значение $a[i][j] \vee b[i][j]$.
3. Ответ — бинарное отношение $C = (c_{ij})$ размерности $n \times m$ объединения бинарных отношений A и B.

   Трудоемкость алгоритма $O(N^2)$.

2.2.2 Алгоритм построения пересечения бинарных отношений

Вход: бинарное отношение $A = (a_{ij})$ размерности $n \times m$ и бинарное отношение $B = (b_{ij})$ размерности $n \times m$.

Выход: бинарное отношение $C = (c_{ij})$ размерности $n \times m$ пересечения бинарных отношений A и B.

1. Создать матрицу C размерности $n \times m$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) $c[i][j]$ присвоить значение $a[i][j] \wedge b[i][j]$.
3. Ответ — бинарное отношение $C = (c_{ij})$ размерности $n \times m$ объединения бинарных отношений A и B.

   Трудоемкость алгоритма $O(N^2)$.

2.2.3 Алгоритм построения дополнения бинарного отношения

Вход: бинарное отношение $A = (a_{ij})$ размерности $n \times m$.

Выход: бинарное отношение $B = (b_{ij})$ размерности $n \times m$ дополения бинарного отношения A.

1. Создать матрицу B размерности $n \times n$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.

а) $b[i][j]$ присвоить значение $\neg a[i][j]$ (то есть если значение $a[i][j]$ было равно единице, то $b[i][j]$ присвоится значение 0, если же значение $a[i][j]$ было равно нулю, то $b[i][j]$ присвоится значение 1).

3. Ответ — бинарное отношение $B = (b_{ij})$ размерности $n \times m$ дополнения бинарного отношения A.

   Трудоемкость алгоритма $O(N^2)$.

2.2.4 Алгоритм построения композиции бинарных отношений

Вход: бинарное отношение $A = (a_{ij})$ размерности $n \times m$ и бинарное отношение $B = (b_{ij})$ размерности $m \times h$.

Выход: бинарное отношение $C = (c_{ij})$ размерности $n \times h$ композиции бинарных отношений A и B.

1. Создать матрицу C размерности $n \times h$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до n.

   а) c[i][j] присвоить $sgn(\sum_{p=1}^{n} a[i][p]b[p][j])$, где sgn — знак полученного числа: если число > 0, то значение функции станет равно 1, если число <= 0, то значение функции станет равно 0.

3. Ответ — бинарное отношение $C = (c_{ij})$ размерности $n \times h$ композиции бинарных отношений A и B.

   Трудоемкость алгоритма $O(n \times m \times h)$.

2.3 Основные операции над матрицами

2.3.1 Сложение и вычитание матриц

Суммой $A + B$ матриц $A_{n\times m} = (a_{ij})$ и $B_{n\times m} = (b_{ij})$ называется матрица $C_{n\times m} = (c_{ij})$, где $(c_{ij}) = (a_{ij}) + (b_{ij})$ для всех $i = \overline{1,n}$ и $j = \overline{1,m}$.

2.3.2 Алгоритм сложения матриц

Вход: Матрица $A = (a_{ij})$ размерности $n \times m$ и матрица $B = (b_{ij})$ размерности $n \times m$, характеристика поля X.

Выход: Матрица $C = (c_{ij})$ размерности $n \times m$ суммы матриц A и B.

1. Создать матрицу C размерности $n \times m$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.

   а) $c[i][j]$ присвоить значение $(a[i][j] + b[i][j])\ mod\ X$.

3. Ответ — матрица $C = (c_{ij})$ размерности $n \times m$ суммы матриц A и B.

   Трудоемкость алгоритма $O(n \times m)$.

Разностью $A - B$ матриц $A_{n \times m} = (a_{ij})$ и $B_{n \times m} = (b_{ij})$ называется матрица $C_{n \times m} = (c_{ij})$, где $(c_{ij}) = (a_{ij}) - (b_{ij})$ для всех $i = \overline{1, n}$ и $j = \overline{1, m}$.

2.3.3 Алгоритм вычитания матриц

Вход: Матрица $A = (a_{ij})$ размерности $n \times m$ и матрица $B = (b_{ij})$ размерности $n \times m$, характеристика поля X.

Выход: Матрица $C = (c_{ij})$ размерности $n \times m$ разности матриц A и B.

1. Создать матрицу C размерности $n \times m$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) $c[i][j]$ присвоить значение $(a[i][j] - b[i][j]) \ mod \ X$.
3. Ответ — матрица $C = (c_{ij})$ размерности $n \times m$ разности матриц A и B.

Трудоемкость алгоритма $O(n \times m)$.

2.3.4 Умножение матрицы на число

Произведением матрицы $A_{n \times m} = (a_{ij})$ на число α называется матрица $C_{n \times m} = (c_{ij})$, где $c_{ij} = \alpha a_{ij}$ для всех $i = \overline{1, n}$ и $j = \overline{1, m}$.

2.3.5 Алгоритм умножения матрицы на число

Вход: Матрица $A = (a_{ij})$ размерности $n \times m$, число α, характеристика поля X.

Выход: Матрица $C = (c_{ij})$ размерности $n \times m$ умноженной на число α матрицы A.

1. Создать матрицу C размерности $n \times m$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) $c[i][j]$ присвоить значение $\alpha a[i][j] \ mod \ X$.
3. Ответ — матрица $C = (c_{ij})$ размерности $n \times m$ умноженной на число α матрицы A.

Трудоемкость алгоритма $O(n \times m)$.

2.3.6 Произведение двух матриц

Произведением матрицы $A_{m \times n} = (a_{ij})$ на матрицу $B_{m \times p} = (b_{ij})$ называется матрица $C_{m \times p} = (c_{ij})$, где $c_{ij} = \sum_{p=1}^{n} a_{ip} b_{pj}$ для всех $i = \overline{1, m}$ и $j = \overline{1, n}$.

2.3.7 Алгоритм умножения матриц

Вход: Матрица $A = (a_{ij})$ размерности $n \times m$ и матрица $B = (b_{ij})$ размерности $p \times n$, характеристика поля X.

Выход: Матрица $C = (c_{ij})$ размерности $n \times p$ умножения матрицы A на матрицу B.

1. Создать матрицу C размерности $n \times p$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) Создать переменную cur и присвоить ей значение ноль.
   б) Цикл по p от 1 до n
      i. Переменной cur присвоить значение $(cur + a[i][p]b[p][j]) \, mod \, X$.
3. Ответ — матрица $C = (c_{ij})$ размерности $n \times p$ умножения матрицы A на матрицу B.

   Трудоемкость алгоритма $O(n \times m \times p)$.

2.3.8 Транспорирование матрицы

Транспонированной по отношению к матрице $A_{m \times n} = (a_{ij})$ называется матрица $A^T_{n \times m} = (a^T_{ij})$ для элементов которой $a^T_{ij} = a_{ji}$.

Вход Матрица $A = (a_{ij})$ размерности $n \times m$.

Выход Матрица $A^T = A' = (a'_{ij})$.

1. Создать матрицу C размерности $m \times n$ и заполнить ее нулями.
2. Цикл по i от 1 до n, цикл по j от 1 до m.
   а) c[j][i] присвоить $a[i][j]$.
3. Ответ — матрица $C = (c_{ij})$ размерности $m \times n$ транспонированной матрицы A.

   Трудоемкость алгоритма $O(n \times m)$.

2.3.9 Обращение матрицы

Обращенной по отношения к матрице $A_{n \times n} = (a_{ij})$ называется матрица $A^{-1}_{n \times n} = (a^{-1}_{ij})$, при умножении которой на исходную матрицу A получается единичная матрица E: $AA^{-1} = A^{-1}A = E$.

2.3.10 Алгоритм нахождения обратной матрицы в конечном поле

Алгоритм нахождения минора

Пусть функция нахождения минора матрицы будет называться get_minor

Вход Матрица $A = (a_{ij})$ размерности $n \times n$, номер столбца $column$.

Выход Минор матрицы $A = (a_{ij})$ относительно столбца $column$.

1. Создать пустой список $minor$ для хранения миноров матрицы.
2. Цикл по i от 1 до n.
   а) Создать пустой список row.
   б) Цикл по j от 1 до n.
      i. Если $j \neq column$, добавить $a[i][j]$ в список row.
   в) Добавить список row в список $minor$
3. Ответ — минор матрицы $A = (a_{ij})$.

Трудоемкость алгоритма $O(n \times n)$.

Алгоритм нахождения определителя матрицы

Пусть функция нахождения определителя матрицы будет называться get_det

Вход Матрица $A = (a_{ij})$ размерности $n \times n$.

Выход Определитель матрицы A.

1. Если размер матрицы равен единице, то ответ — $a[0][0]$.
2. Создать переменную $det = 0$.
3. Создать переменную $multiplier = 1$.
4. Цикл по i от 1 до n.
   а) Создать переменную $el = a[0][i]$.
   б) Если $el \neq 0$, то переменной det присвоить значение $det + multiplier * el * get\_det(get\_minor(i, A))$.
   в) Переменной $multiplier$ присвоить значение $multiplier * -1$
5. Ответ — определитель матрицы $A = (a_{ij})$.

Трудоемкость алгоритма $O(\frac{(n+1)\times n}{2} \times n \times n)$, так как сложность алгоритма нахождения минора составляет $O(n \times n)$ и внутри алгоритма нахождения определителя происходит обращение к данному алгоритму $O(\frac{(n+1)\times n}{2})$

Алгоритм нахождения обратного по модулю элемента

Модульное обратное для числа a по модулю m можно найти с помощью расширенного алгоритма Евклида.

Алгоритм Евклида определяет наибольший общий делитель (НОД) двух целых чисел, скажем a и m. Если a имеет обратное по модулю m число, этот НОД должен быть равен 1. Несколько последних равенств, получаемых в процессе работы алгоритма, могут быть решены для нахождения НОД. Затем, используя метод «обратной подстановки», может быть получено выражение, связывающее исходные параметры и НОД. Другими словами, могут быть найдены целые x и y, удовлетворяющие равенство Безу:

$$\alpha x + my = \text{НОД}(\alpha, m) = 1$$

Данное равенство можно переписать следующим образом:

$\alpha x - 1 = (-y)m$, то есть $\alpha x \equiv 1 \pmod{m}$ и модульное обратное числа a вычислено.

Более эффективной версией является расширенный алгоритм Евклида, который с помощью дополнительных равенств сокращает два прохода алгоритма (обратную подстановку можно понимать как прохождение алгоритма в обратном порядке) до одного.

Сложность алгоритма составляет $O(log(m)^2)$ из условия, что $|\alpha| < m$.

Пусть функция нахождения обратного по модулю будет называться $get\_extended$

Вход Число x, число y, обратный определитель матрицы $A = (a_{ij})$ $detInverse$, модуль $FieldOrder$, определитель a матрицы $A = (a_{ij})$, вспомогательная переменная b, изначально равная $FieldOrder$.

Выход Обратный по модулю $FieldOrder$ элемент.

1. Если $a = 0$, то ответ — 0.
2. Создать переменную $y\_1 = 0$.
3. Создать переменную $x\_1 = 0$.
4. Переменной $FieldOrder$ присвоить значение a.
5. Переменной a присвоить значение $b \; mod \; a$
6. Вызвать функцию $get\_extended$.
7. Переменной x присвоить значение $y\_1 - (\frac{b}{a} * x\_1)$.
8. Переменной y_1 присвоить значение x_1.
9. Ответ — обратный по модулю $FieldOrder$ элемент.

Трудоемкость алгоритма составляет $O(log(m)^2) = O(log(m))$.

Алгоритм нахождения расширенного минора матрицы

Пусть функция нахождения расширенного минора матрицы будет называться $get\_minor\_extended$.

Вход Матрица $A = (a_{ij})$ размерности $n \times n$, столбец $column$, строка row.

Выход Расширенный минор матрицы A по строке row и столбцу $column$.

1. Создать пустой список $minor$.
2. Цикл по i от 1 до n.
   а) Если $i = row$, перейти к следующей итерации.
   б) Создать пустой список row_temp.
   в) Цикл по j от 1 до n.
      i. Если $j = column$, перейти к следующей итерации.
      ii. Добавить в row_temp элемент $a[i][j]$.
   г) Добавить в список $minor$ список row_temp.
3. Ответ — расширенный минор матрицы A по строке row и столбцу $column$.

Трудоемкость алгоритма $O(n \times n)$.

Алгоритм нахождения сопряженной матрицы

Вход Матрица $A = (a_{ij})$ размерности $n \times n$.

Выход Сопряженная матрица матрице A.

1. Создать переменную $sign = 1$.
2. Создать переменную $flag$ и присвоить ей значение 1, если размерность матрицы четная. Если размерность матрицы нечетная, то присвоить переменной $flag$ значение 0.
3. Создать матрицу con_matrix размерности $n \times n$ и заполнить ее нулями.
4. Цикл по i от 1 до n, цикл по j от 1 до n.
   а) $con\_matrix[i][j]$ присвоить значение $sign * get\_det(get\_minor\_extended(A,$
   б) Переменной $sign$ присвоить значение $sign * (-1)$
   в) Перед каждой новой итерацией по i присваивать $sign$ значение $sign * (-1)$, если $flag = 1$.
5. Вызвать алгоритм транспонирования матрицы от полученной матрицы con_matrix.
6. Ответ — сопряженная матрица матрице A.

Внутри алгоритма нахождения сопряженной матрицы $n \times n$ раз вызывается алгоритм нахождения определителя $O(\frac{(n+1)\times n}{2})$, а также алгоритм нахождения расширенного минора матрицы $O(n \times n)$, затем вызывается алгоритм получения транспонированной матрицы, сложность которого составляет $O(n \times n)$. Поэтому, трудоемкость алгоритма составляет $O(n \times n \times n \times n \times \frac{(n+1)\times n}{2} + n \times n) = O(\frac{(n+1)\times n^5}{2} + n \times n) = O(n^6)$.

Алгоритм свертки матрицы по модулю

Вход Матрица $A = (a_{ij})$ размерности $n \times n$, обратный определитель матрицы $detInverse$, модуль $FieldOrder$.

Выход Обратная матрица к матрице A.

1. Создать матрицу $A_1 = (a_{1_{ij}})$ размерности $n \times n$ и заполнить ее нулями
2. Цикл по i от 1 до n, цикл по j от 1 до n.
   а) Элементу $a_1[i][j]$ присвоить значение $(a[i][j] * detInverse) mod FieldOrder$.
   б) Если $a_1[i][j] < 0$, то $a_1[i][j]$ присвоить значение $a_1[i][j] + FieldOrder$
   в) Перед каждой новой итерацией по i присваивать $sign$ значение $sign * (-1)$, если $flag = 1$.
3. Ответ — обратная матрица матрице A.

Трудоемкость алгоритма составляет $O(n \times n)$.

3 Практическая часть

Задание 1

Мн-во: {a, b, c, d}

4)

·	a	b	c	d
a	a	b	c	d
b	b	c	b	d
c	c	b	c	d
d	d	d	d	d

Для док-ва ассоц-ти этой операции по тесту Лайта фиксируем эл-т $a = a, b, c, d$ и проверяем вып-ть равенства: $(x \cdot a) \cdot z = x \cdot (a \cdot z)$ $\forall x, z \in Z_4$ путем сравн-я двух таблиц.

Для $a = a$

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

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

✓

Для $a = b$

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

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

✓

Для $a = c$

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

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

✓

Для $a = d$

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

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

✓

=> Операция ассоциативна и Z_4 является полугруппой

Задание 2

Найти:

$\lambda = 4 \% 6 = 4$

$A^2 + (10 - \lambda/2) \cdot A + \frac{\lambda}{2} E$

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

1) $A^2 = \begin{pmatrix} 1 & -2 \\ -3 & 4 \end{pmatrix} \begin{pmatrix} 1 & -2 \\ -3 & 4 \end{pmatrix} = \begin{pmatrix} 7 & -10 \\ -15 & 22 \end{pmatrix}$

2) $(10 - 4/2) \cdot A = 8A = 8 \begin{pmatrix} 1 & -2 \\ -3 & 4 \end{pmatrix} = \begin{pmatrix} 8 & -16 \\ -24 & 32 \end{pmatrix}$

3) $\frac{\lambda}{2} E = 2E = 2 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$

4) $\begin{pmatrix} 7 & -10 \\ -15 & 22 \end{pmatrix} + \begin{pmatrix} 8 & -16 \\ -24 & 32 \end{pmatrix} = \begin{pmatrix} 15 & -26 \\ -39 & 54 \end{pmatrix}$

5) $\begin{pmatrix} 15 & -26 \\ -39 & 54 \end{pmatrix} + \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 17 & -26 \\ -39 & 56 \end{pmatrix}$

Ответ: $\begin{pmatrix} 17 & -26 \\ -39 & 56 \end{pmatrix}$

Задание 3

Вычислить AB, где $A = \begin{pmatrix} -1 & \lambda & 3 \\ \lambda/3 & 2 & 8-\lambda/3 \end{pmatrix}$

$B = \begin{pmatrix} -\lambda & 2 \\ 1 & 10-\lambda/2 \\ -3 & \lambda \end{pmatrix}$

$AB = \begin{pmatrix} -1 & 4 & 3 \\ 4/3 & 2 & 20/3 \end{pmatrix} \cdot \begin{pmatrix} -4 & 2 \\ 1 & 8 \\ -3 & 4 \end{pmatrix} = \begin{pmatrix} -1\cdot(-4) + 4\cdot 1 + 3\cdot(-3) & -2+32+12 \\ -\frac{16}{3} + 2 + \frac{(-60)}{3} & \frac{8}{3} + 16 + \frac{80}{3} \end{pmatrix} =$

$= \begin{pmatrix} -1 & 42 \\ 70/3 & 136/3 \end{pmatrix}$

Ответ: $\begin{pmatrix} -1 & 42 \\ -70/3 & 136/3 \end{pmatrix}$

4 Программная реализация рассмотренных алгоритмов

4.1 Результаты тестирования программы

```
Выберете действие:
Проверить свойство алгебраической операции (1)
Выполнить операцию над бинарным отношением\отношениями (2)
Выполнить операцию над матрицей/матрицами (3)
1
Введите размер множества элементов: 4
Введите множество длины 4: a b c d
Введите значения таблицы Кэли некоторой операции:
  a b c d
a a b b c
b b c a c
c c c b d
d b d b d
Введите 1, чтобы проверить идемпотентность.
Введите 2, чтобы проверить коммутативность.
 Введите 3, чтобы проверить ассоциативность.
Введите 4, чтобы проверить обратимость.
 Введите 5, чтобы проверить дистрибутивность.
Введите число: 1
Операция не является идемпотентной.
Вы желаете продолжить? 1 - Да, 0 - Нет.1
Введите число: 2
Операция не является коммутативной.
Вы желаете продолжить? 1 - Да, 0 - Нет.1
Введите число: 3
Операция не является ассоциативной.
Вы желаете продолжить? 1 - Да, 0 - Нет.1
Введите число: 4
Обратимость операции не выполняется.
```

Рисунок 1

```
Введите число: 5
Для проверки дистрибутивности введите значения второй таблицы Кэли операции.
  a b c d
a a b c d
b b c d a
c c b a b
d d a b c
Операция * не является дистрибутивной относительно операции +
```

Рисунок 2

```
Выполнить операцию над матрицей/матрицами (3)
2
Выберете операцию из списка:
Введите 1, чтобы получить объединение бинарных отношений.
 Введите 2, получить пересечение бинарных отношений.
 Введите 3, получить дополнение бинарного отношения.
 Введите 4, чтобы получить транспонированное бинарное отношение.
 Введите 5, чтобы получить композицию бинарных отношений.
Введите число: 1
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
0 1 0
1 1 1
0 0 1
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
0 0 0
1 1 1
1 1 1
Объединение бинарных отношений:
[[(1, 2)], [(2, 1), (2, 2), (2, 3)], [(3, 1), (3, 2), (3, 3)]]
0 1 0
1 1 1
1 1 1
```

Рисунок 3

```
Введите число: 2
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
0 1 0
1 0 1
0 1 0
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 0 1
0 1 0
1 0 1
[[], [], []]
0 0 0
0 0 0
0 0 0
```

Рисунок 4

```
Введите число: 3
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 1 1
0 0 0
1 1 1
Дополнение бинарного отношения:
[[], [(2, 1), (2, 2), (2, 3)], []]
0 0 0
1 1 1
0 0 0
```

Рисунок 5

```
Введите число: 4
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 0 0
1 0 0
1 0 0
Обращение бинарного отношения:
[[(1, 1), (1, 2), (1, 3)], [], []]
1 1 1
0 0 0
0 0 0
```

Рисунок 6

```
Введите число: 5
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 0 1
1 0 1
1 0 1
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
0 1 0
0 1 0
0 1 0
Композиция бинарных отношений:
[[(1, 2)], [(2, 2)], [(3, 2)]]
0 1 0
0 1 0
0 1 0
```

Рисунок 7

```
Выполнить операцию над матрицей/матрицами (3)
3
Введите характеристику поля: 5
Сумма матриц (1),
Разность матриц (2),
Умножение матрицы на число (3),
Произведение двух матриц (4),
Транспонирование матрицы (5),
Обращение матрицы (6):
Введите число: 1
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 2 3
4 4 4
2 3 1
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
1 2 3
3 2 1
1 1 1
Сумма матриц:
2 4 1
2 1 0
3 4 2
```

Рисунок 8

```
Введите число: 2
Введите размерность матрицы: 2 3
Введите значения элементов матрицы построчно:
1 2 3
2 3 4
Введите размерность матрицы: 2 3
Введите значения элементов матрицы построчно:
3 2 1
4 4 4
Разность матриц:
3 0 2
3 4 0
```

Рисунок 9

```
Введите число: 3
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
4 4 4
3 3 3
2 2 2
Введите число:
4
Умножение матрицы на число:
1 1 1
2 2 2
3 3 3
```

Рисунок 10

```
Введите число: 4
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
3 2 1
1 2 3
3 3 3
Введите размерность матрицы: 3 3
Введите значения элементов матрицы построчно:
3 2 1
1 2 3
3 2 1
Произведение матриц:
4 2 0
4 2 0
1 3 0
```

Рисунок 11

```
Введите число: 5
Введите размерность матрицы: 4 4
Введите значения элементов матрицы построчно:
1 2 3 4
4 3 2 1
1 2 3 3
3 2 2 3
Транспонированная матрица:
1 4 1 3
2 3 2 2
3 2 3 2
4 1 3 3
```

Рисунок 12

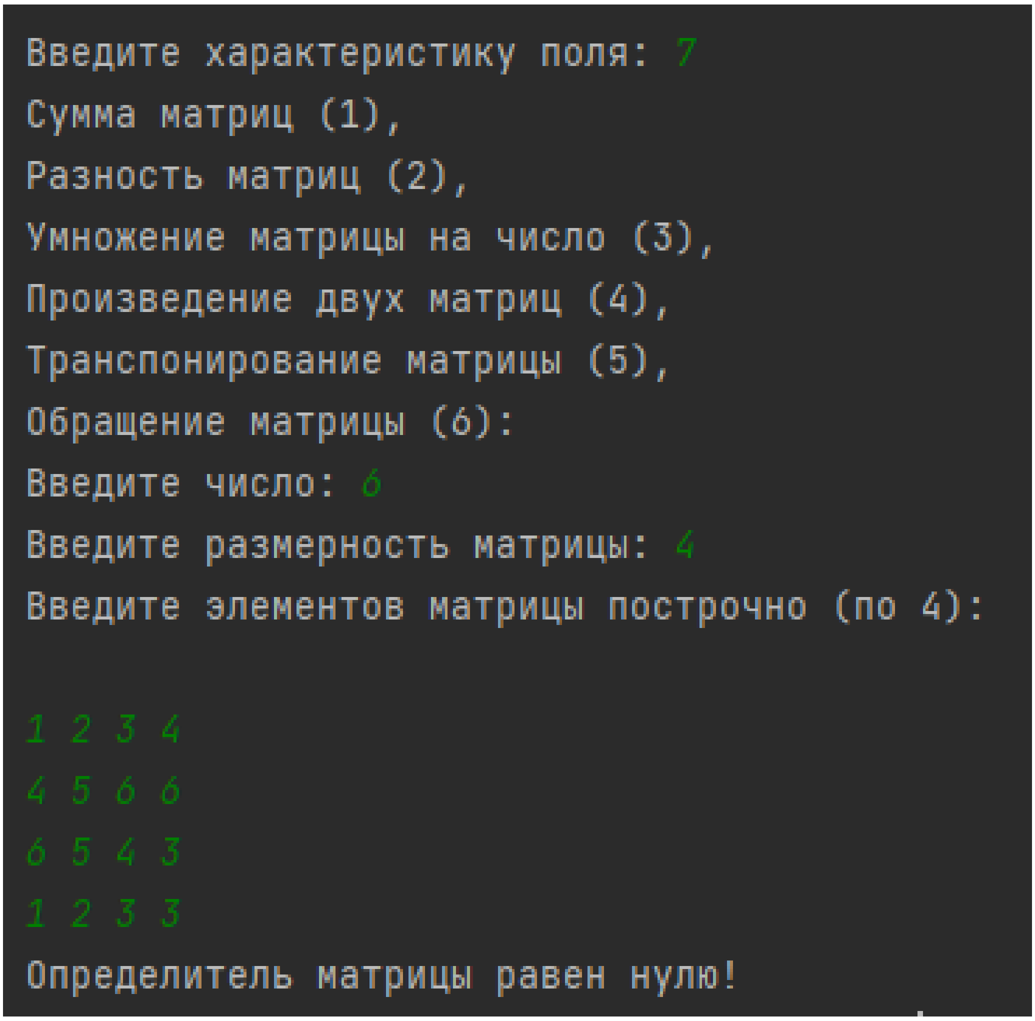

Рисунок 13

4.2 Коды программ, реализующих рассмотренные алгоритмы

4.2.1 Код программы, реализующей нахождение обратной матрицы в поле Галуа

```
import numpy as np
matrix = []
y = 0
a = 0
fieldOrder = 0
detInverse = 0

def get_minor(column, matrix):
    minor = []
    for i in range(1, len(matrix)):
        row = []
        for j in range(len(matrix)):
            if not j == column:
```

```
                row.append(matrix[i][j])
        minor.append(row)
    return minor


def get_det(matrix):
    if len(matrix) == 1:
        return matrix[0][0]
    det = 0
    mutiplier = 1
    for i in range(len(matrix)):
        el = matrix[0][i]
        if not el == 0:
            det += mutiplier * el * get_det(get_minor(i, matrix))
        mutiplier *= -1
    return det


def get_extended(x, y):
    global detInverse
    global a
    global fieldOrder
    if a == 0:
        x = 0
        y = 1
        return
    y_1 = 0
    x_1 = 0
    fieldOrder = a
    a = b % a
    get_extended(detInverse, y_1)
    x = y_1 - (b / a) * x_1
    y = x_1
    return


def get_minor_extended(matrix, row, column):
    minor = []
    for i in range(len(matrix)):
        if i == row:
            continue
```

```
        row = []
        for j in range(len(matrix)):
            if j == column:
                continue
            row.append(matrix[i][j])
        minor.append(row)
    return minor


def get_con_matrix(matrix):
    sign = 1
    flag = len(matrix) % 2 == 0
    con_matrix = [[0 for _ in range(len(matrix))] for _ in range(len(matrix))]
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            con_matrix[i][j] = sign * get_det(get_minor_extended(matrix, i, j))
            sign *= -1
        sign *= -1 if flag else 1
    return con_matrix.T


def reduce_matrix(fieldOrder, detInverse):
    global matrix
    for i in range(len(matrix)):
        for j in range(len(matrix)):
            matrix[i][j] *= detInverse
            matrix[i][j] %= fieldOrder
            if matrix[i][j] < 0:
                matrix[i][j] += fieldOrder


def main():
    fieldOrder = int(input('Введите значение поля: '))
    n, m = input('Введите размерность матрицы: ').split()
    n, m = int(n), int(m)
    print('Введите элементы матрцы построчно: ')
    global matrix

    matrix = [[int(val) for val in input().split()] for i in range(n)]
    check = 0
```

```
    a = get_det(matrix)
    if a == 0:
        print('Определитель матрицы равен нулю')
    else:
        while a < 0:
            a += fieldOrder
        get_extended(x, y)


if __name__ == "__main__":
    main()
```

4.2.2 Код программы, реализующей основные алгоритмы

```
import numpy as np


def make_set(a):
    print([[(i + 1, j + 1) for j in range(len(a[i])) if a[i][j]] for i in
    ↪ range(len(a))])


def print_matrix(a):
    for elems in a:
        print(*elems)


def get_ones(a):
    print_matrix([[1 if a[i][j] else 0 for j in range(len(a[i]))] for i in
    ↪ range(len(a))])


def get_matrix():
    n, m = input('Введите размерность матрицы: ').split()
    n, m = int(n),  int(m)
    print('Введите значения элементов матрицы построчно: ')
    matrix = [list(map(int, input().split())) for _ in range(n)]
    return matrix


def is_idempotent(st, a):
    for i in range(len(st)):
        if a[i][i] != st[i]:
```

```
            print('Операция не является идемпотентной.')
            return
    print('Операция является идемпоненетной.')
    return


def is_commutative(st, a):
    for i in range(len(st)):
        for j in range(len(st)):
            if a[i][j] != a[j][i]:
                print('Операция не является коммутативной.')
                return
    print('Операция является коммутативной.')
    return


def is_associative(st, a):
    for i in range(len(st)):
        for j in range(len(st)):
            for k in range(len(st)):
                if a[i][st.index(a[j][k])] != a[st.index(a[i][j])][k]:
                    print('Операция не является ассоциативной.')
                    return
    print('Операция является ассоцитивной.')
    return


def is_invertible(st, a):
    res = []
    for i in range(len(st)):
        flag = True
        for j in range(len(st)):
            if not(a[i][j] == a[j][i] == 1):
                flag = False
                break
        if flag:
            res.append(st[i])
    if res:
        print(f'Обратимость операции выполняется. Список обратных элементов {res}')
        return
```

```python
    else:
        print('Обратимость операции не выполняется.')
        return


def is_distributive(st, a):
    print('Для проверки дистрибутивности введите значения второй таблицы Кэли операции. ')
    # b = [[int(val) for val in input().split()] for _ in range(len(st))]
    print('  ', *st)
    b = [input(f'{st[i]}  ').split() for i in range(len(st))]
    for i in range(len(st)):
        for j in range(len(st)):
            for k in range(len(st)):
                if (a[i][st.index(b[j][k])] != b[st.index(a[i][j])][st.index(a[i][k])]) \
                        or (a[st.index(b[j][k])][i] != b[st.index(a[j][i])][st.index(a[k][i])]):
                    print('Операция * не является дистрибутивной относительно операции +')
                    return
    print('Операция * на матрице a является дистрибутивной относительно операции + на матрице b')
    return


def relation_properties():
    n = int(input('Введите размер множества элементов: '))
    # st = list(map(int, input().split()))
    st = input(f'Введите множество длины {n}: ').split()
    print("Введите значения таблицы Кэли некоторой операции: ")
    # a = [list(map(int, input().split())) for i in range(n)]
    print('  ', *st)
    a = [input(f'{st[i]}  ').split() for i in range(len(st))]
    print("Введите 1, чтобы проверить идемпотентность.\nВведите 2, чтобы проверить коммутативность.\n",
          "Введите 3, чтобы проверить ассоциативность.\nВведите 4, чтобы проверить обратимость.\n",
          "Введите 5, чтобы проверить дистрибутивность.")
    yes_or_no = 1
    while yes_or_no:
```

```
        bl = int(input('Введите число: '))
        if bl == 1:
            is_idempotent(st, a)
        elif bl == 2:
            is_commutative(st, a)
        elif bl == 3:
            is_associative(st, a)
        elif bl == 4:
            is_invertible(st, a)
        elif bl == 5:
            is_distributive(st, a)
        yes_or_no = int(input('Вы желаете продолжить? 1 - Да, 0 - Нет.'))


def relation_operation():
    print('Выберете операцию из списка:')
    print("Введите 1, чтобы получить объединение бинарных отношений. \n",
          "Введите 2, получить пересечение бинарных отношений. \n",
          "Введите 3, получить дополнение бинарного отношения. \n",
          "Введите 4, чтобы получить транспонированное бинарное отношение. \n",
          "Введите 5, чтобы получить композицию бинарных отношений.")
    yes_or_no = 1
    while yes_or_no:
        op = int(input('Введите число: '))
        if op == 1:
            a = np.array(get_matrix())
            b = np.array(get_matrix())
            print("Объединение бинарных отношений:")
            c = a + b
            make_set(c)
            get_ones(c)
        elif op == 2:
            a = np.array(get_matrix())
            b = np.array(get_matrix())
            c = a * b
            make_set(c)
            get_ones(c)
        elif op == 3:
            a = get_matrix()
            print("Дополнение бинарного отношения:")
```

```
            a_new = np.ones(np.array(a).shape) - a
            make_set(a_new)
            get_ones(a_new)
        elif op == 5:
            a = get_matrix()
            b = get_matrix()
            print("Композиция бинарных отношений:")
            c = np.matmul(np.array(a), np.array(b))
            make_set(c)
            get_ones(c)
        elif op == 4:
            a = get_matrix()
            print("Обращение бинарного отношения:")
            a_t = np.array(a).T
            make_set(a_t)
            print_matrix(a_t)
        yes_or_no = int(input('Вы желаете продолжить? 1 - Да, 0 - Нет. '))


def get_mod(a, lim):
    return [[a[i][j] % lim for j in range(len(a[i]))] for i in range(len(a))]


def matrix_operation():
    print('''Сумма матриц (1), \nРазность матриц (2), \nУмножение матрицы на число (3), \nПроизведение двух матриц (4),
Транспонирование матрицы (5), \nОбращение матрицы (6): ''')
    yes_or_no = 1
    lim = (int(input('Введите характеристику поля: ')))
    while yes_or_no:
        num = int(input('Введите число: '))
        if num == 1:
            a = get_matrix()
            b = get_matrix()
            print("Сумма матриц:")
            c = get_mod(np.array(a) + np.array(b), lim)
            print_matrix(c)
        elif num == 2:
            a = get_matrix()
            b = get_matrix()
            print("Разность матриц:")
```

```
            c = get_mod(np.array(a) - np.array(b), lim)
            print_matrix(c)
        elif num == 3:
            a = get_matrix()
            print("Введите число: ")
            alpha = int(input())
            print("Умножение матрицы на число:")
            a_alpha = get_mod((np.array(a) * alpha), lim)
            print_matrix(a_alpha)
        elif num == 4:
            a = get_matrix()
            b = get_matrix()
            print("Произведение матриц:")
            c = get_mod(np.matmul(np.array(a), np.array(b)), lim)
            print_matrix(c)
        elif num == 5:
            a = get_matrix()
            print("Транспонированная матрица:")
            a_t = np.array(a).T
            print_matrix(a_t)
        elif num == 6:
            import inverseMatrix as im
            a = get_matrix()
            print("Обращение матрицы:")
            a_inv = im.main()
        yes_or_no = int(input('Вы желаете продолжить? 1 - Да, 0 - Нет. '))


print('''Выберете действие: \n Проверить свойство алгебраической операции (1)
Выполнить операцию над бинарным отношением\отношениями (2) \n Выполнить операцию над матрицей/матрицами (3)''')
action = int(input())
if action == 1:
    relation_properties()
elif action == 2:
    relation_operation()
elif action == 3:
    matrix_operation()


# Задание 1
```

```
st = ['a', 'b', 'c', 'd']

a = [['a', 'a', 'a', 'a'],
     ['a', 'b', 'a', 'b'],
     ['a', 'a', 'c', 'c'],
     ['a', 'b', 'c', 'd']]

is_associative(st, a)

# Задание 2
a = np.array([[1, -2],
              [-3, 5]])

e = np.ones((2, 2))

for elems in np.matmul(a, a) + (10 - 5 / 2) * a + (5 / 2) * e:
    print(*elems)
print()

# Задание 3

a = np.array([[-1, 5, 3],
              [5 / 3, 2, 8 - 5 / 3]])

b = np.array([[-5, 2],
              [1, 10 - 5 / 2],
              [-3, 5]])

for elems in np.matmul(a, b):
    print(*elems)

# a b c d e
# b c d e a
# c d e a b
# d e a b c
# e a b c d
#
#
# a a a a a
# a b c d e
# a c e b d
```

```
# a d b e c
# a e d c b

#
# 0 1 1 0 1
# 1 0 0 0 0
# 0 1 0 1 1
# 0 0 1 1 0
# 1 1 0 0 0
# 0 1 0 1 0
#
#
# 1 0 0 1 0
# 0 1 1 1 1
# 1 0 1 0 0
# 1 1 0 0 1
# 0 0 1 1 1
# 1 0 1 0 1
```

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы были теоретические сведения об алгебраической операции и классификации свойств операций, основные операции над бинарными отношениями, а также основные операции над матрицами. На их основе были составлены алгоритмы проверки свойств операций: ассоциативность, коммутативность, идемпотентность, обратимость, дистрибутивность, алгоритмы выполнения операции над бинарными отношениями, а также алгоритмы выполнения операций над матрицами. Для всех алгоритмов произведена асимптотическая оценка.