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
Зиминой Ирины Олеговны

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

Саратов 2022

СОДЕРЖАНИЕ

1. Цель работы и порядок 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. Алгебраическая операция и классификация свойств операций

Алгебраическая операция: 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.

Также можно использовать мультипликативную запись с помощью символа · то есть запись $f(x, y)$ будет выглядеть как: $x \cdot y$.

Классификация свойств операций.

Бинарная операция · на множестве A называется:

- ассоциативной, если $\forall x, y, z \in A$ выполняется равенство:
$$x \cdot (y \cdot z) = (x \cdot y) \cdot z;$$
- коммутативной, если $\forall x, y \in A$ выполняется равенство:
$$x \cdot y = y \cdot x;$$
- идемпотентной, если $\forall x \in A$ выполняется равенство:
$$x \cdot x = x;$$
- обратимой, если $\forall x, y \in A$ уравнения $x \cdot a = y$ и $b \cdot x = y$ имеют решение, причем единственное;
- дистрибутивной относительно операции +, если $\forall x, y, z \in A$ выполняются равенства:
$$x \cdot (y + z) = (x \cdot y) + (x \cdot z),$$
$$(y + z) \cdot x = (y \cdot x) + (z \cdot x);$$

3. Алгоритмы проверки свойств операций.

3.1 Проверка ассоциативности.

Входные данные: Таблица Кэли операции ·, множество элементов над операцией · (двумерная матрица размерностью $N \times N$ и элементами $cayleytable[i][j]$).

Выходные данные: результат проверки на ассоциативность: «Ассоциативность выполняется» или «Ассоциативность не выполняется».

Описание алгоритма:

1) Запускается цикл прохода по индексу x от 0-го до $N - 1$.
2) Внутри цикла запускается цикл прохода по индексу y от 0-го до $N - 1$.
3) Внутри цикла запускается цикл прохода по индексу z от 0-го до $N - 1$.
4) Внутри циклов проверяется $\forall\, x, y, z$ элементов множества элементы $cayleytable[i][j]$ таблицы Кэли по условию: если $cayleytable[i][t] = cayleytable[p][k]$ (где $t = cayleytable[j][k], p = cayleytable[i][j]$ для всех $0 \le i, j, k \le n - 1$, где i соответствует номеру строки таблицы Кэли для элемента y, а j и k для x и z соответсвенно).
5) Если данное условие выполняется, то на выход идет сообщение: «Ассоциативность выполняется», иначе на выход идет сообщение: «Ассоциативность не выполняется».

Сложность алгоритма: $O(n^3)$.

3.2 Проверка коммутативности.

Входные данные: Таблица Кэли операции ·, множество элементов над операцией · (двумерная матрица размерностью $N \times N$ и элементами $cayleytable[i][j]$).

Выходные данные: результат проверки на коммутативность: «Коммутативность выполняется» или «Коммутативность не выполняется».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N - 1$.

2) Внутри цикла запускается цикл прохода по индексу j от 0-го до $N-1$.
3) Проверяется условие $cayleytable[i][j] \neq cayleytable[j][i]$.
4) Если данное условие выполняется, то на выход идет сообщение: «Коммутативность не выполняется», иначе на выход идет сообщение: «Коммутативность выполняется».

Сложность алгоритма: $O(n^2)$.

3.3 Проверка идемпотентности.

Входные данные: Таблица Кэли операции ·, множество элементов над операцией · (двумерная матрица размерностью $N \times N$ и элементами $cayleytable[i][j]$).

Выходные данные: результат проверки на идемпотентность: «Идемпотентность выполняется» или «Идемпотентность не выполняется».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Проверяется условие $cayleytable[i][i] \neq i+1$.
3) Если данное условие выполняется, то на выход идет сообщение: «Идемпотентность не выполняется», иначе на выход идет сообщение: «Идемпотентность выполняется».

Сложность алгоритма: $O(n)$.

3.4 Проверка обратимости.

Входные данные: Таблица Кэли операции ·, множество элементов над операцией · (двумерная матрица размерностью $N \times N$ и элементами $cayleytable[i][j]$).

Выходные данные: результат проверки на обратимость: «Обратимость выполняется» или «Обратимость не выполняется».

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до $N-1$.
2) Внутри цикла запускается цикл прохода по индексу j от 0-го до $N-1$.
3) Проверяется условие $cayleytable[i][j] = cayleytable[j][i] = 1$).

4) Если данное условие выполняется, то на выход идет сообщение: «Обратимость выполняется», иначе на выход идет сообщение: «Обратимость не выполняется».

Сложность алгоритма: $O(n^2)$.

3.5 Проверка дистрибутивности.

Входные данные: Таблица Кэли операции $\cdot$, множество элементов над операцией $\cdot$ (двумерная матрица размерностью $N \times N$ и элементами $cayleytable[i][j]$) и дополнительная таблица Кэли операции $+$ (двумерная матрица размерностью $N \times N$ и элементами $anothercayleytable[i][j]$). Относительно дополнительной таблицы Кэли осуществляется проверка дистрибутивности.

Выходные данные: результат проверки на дистрибутивность: «Дистрибутивность выполняется» или «Дистрибутивность не выполняется».

Описание алгоритма:

1) Запускается цикл прохода по индексу x от 0-го до $N - 1$.
2) Внутри цикла запускается цикл прохода по индексу y от 0-го до $N - 1$.
3) Внутри цикла запускается цикл прохода по индексу z от 0-го до $N - 1$.
4) Внутри циклов проверяется $\forall\, x, y, z$ элементов множества элементы $cayleytable[i][j]$ и $anothercayleytable[i][j]$ таблиц Кэли по условию: если $cayleytable[i][t] = anothercayleytable\,[p][q]$ и $cayleytable[t][i] = anothercayleytable\,[g][h]$ (где $t = anothercayleytable[j][k], p = cayleytable[i][j],\ q = cayleytable[i][k],\ g = cayleytable[j][i], h = cayleytable[k][i]$, для всех $0 \le i, j, k \le n - 1$).
5) Если данное условие выполняется, то на выход идет сообщение: «Дистрибутивность выполняется», иначе на выход идет сообщение: «Дистрибутивность не выполняется».

Сложность алгоритма: $O(n^3)$.

4. Основные операции над бинарными отношениями

– **Теоретико-множественные операции.**

Объединение (∪), пересечение (∩), дополнение (¬);

– **Обращение бинарных отношений.**

Обратным для бинарного отношения $\rho \subset A \times B$ является бинарное отношение $\rho^{-1} \subset B \times A$, которое определяется по формуле:

$$\rho^{-1} = \{(b, a) : (a, b) \in \rho\};$$

– **Композиция бинарных отношений.**

Композицией бинарных отношений $\rho \subset A \times B$ и $\sigma \subset B \times C$ является бинарное отношение $\rho\sigma \subset A \times C$, которое определяется по формуле:

$$\rho\sigma = \{(a, c) : (a, b) \in \rho \text{ и } (b, c) \in \sigma \text{ для некоторого } b \in B\}.$$

5. Алгоритмы операций над бинарными отношениями

5.1 Объединение бинарных отношений

Входные данные: матрицы $matrixA$ и $matrixB$ бинарных отношений с размерностью $N \times M$.

Выходные данные: результат объединения бинарных отношений.

Описание алгоритма:

1) Создается вектор пар $result\_union$.
2) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
3) Внутри цикла запускается цикл прохода по индексу j от 0-го до $M - 1$.
4) Внутри циклов выполняется условие логического ИЛИ для элементов $matrixA[i][j]$ и $matrixB[i][j]$.
5) Результат добавляется в вектор пар $result\_union$ в формате первый элемент равен $i + 1$, второй элемент равен $j + 1$.
6) Выводится построенный вектор пар $result\_union$.

Сложность алгоритма: $n * m$.

5.2 Пересечение бинарных отношений

Входные данные: матрицы $matrixA$ и $matrixB$ бинарных отношений с размерностью $N \times M$.

Выходные данные: результат пересечения бинарных отношений.

Описание алгоритма:

1) Создается вектор пар $result\_intersection$.
2) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
3) Внутри цикла запускается цикл прохода по индексу j от 0-го до $M - 1$.
4) Внутри циклов выполняется условие логического И для элементов $matrixA[i][j]$ и $matrixB[i][j]$.
5) Результат добавляется в вектор пар $result\_intersection$ в формате первый элемент равен $i + 1$, второй элемент равен $j + 1$.
6) Выводится построенный вектор пар $result\_intersection$.

Сложность алгоритма: $n * m$.

5.3 Дополнение бинарных отношений

Входные данные: матрица $matrix$ бинарного отношения с размерностью $N \times M$.

Выходные данные: результат дополнения бинарного отношения.

Описание алгоритма:

1) Создается вектор пар $result\_additions$.
2) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
3) Внутри цикла запускается цикл прохода по индексу j от 0-го до $M - 1$.
4) Внутри циклов проверяется условие если $matrix[i][j] = 0$.
7) Если условие выполняется, то $matrix[i][j]$ добавляется в вектор $result\_additions$ в формате первый элемент равен $i + 1$, второй элемент равен $j + 1$.
5) Выводится построенный вектор пар $result\_additions$.

Сложность алгоритма: $n * m$.

5.4 Обращение бинарных отношений

Входные данные: матрица $matrix$ бинарного отношения с размерностью $N \times M$.

Выходные данные: результат обращения бинарного отношения.

Описание алгоритма:

1) Создается транспонированная матрица $matrix\_transposed$ бинарного отношения ρ с размерностью $M \times N$.
2) Создается вектор пар $result\_reversal$.
3) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
4) Внутри цикла запускается цикл прохода по индексу j от 0-го до $M - 1$.
5) Внутри циклов проверяется условие если $matrix\_transposed[i][j] = 1$.
6) Если условие выполняется, то $matrix[i][j]$ добавляется в вектор $result\_reversal$ в формате первый элемент равен $i + 1$, второй элемент равен $j + 1$.
7) Выводится построенный вектор пар $result\_reversal$.

Сложность алгоритма: $n * m$.

5.5 Композиция бинарных отношений

Входные данные: матрицы $matrixA$ бинарных отношений с размерностью $N \times M$ и $matrixB$ бинарных отношений с размерностью $R \times T$.

Выходные данные: результат композиции бинарных отношений.

Описание алгоритма:

1) Создается матрица $matrixC$ размером $N \times T$.
2) Матрица $matrixC$ является матричным произведением $matrixA$ и $matrixB$.
3) Создается вектор пар $result\_composition$.
4) Запускается цикл прохода по индексу i от 0-го до $N - 1$.
5) Внутри цикла запускается цикл прохода по индексу j от 0-го до $M - 1$.
6) Внутри циклов проверяется условие если $matrixC\ [i][j] = 1$.

7) Если условие выполняется, то $matrixC\ [i][j]$ добавляется в вектор $result\_composition$ в формате первый элемент равен $i + 1$, второй элемент равен $j + 1$.
8) Выводится построенный вектор пар$result_composition$.

Сложность алгоритма: $n * t * m$.

6. Основные операции над матрицами

– **Сложение двух матриц.**

Суммой матрицы A (размерностью $m \times n$ и элементами a_{ij}) и матрицы B (размерностью $m \times n$ и элементами b_{ij}) является матрица C (размерностью $m \times n$ и элементами c_{ij}), где $c_{ij} = a_{ij} + b_{ij}$ для всех $i = \overline{1,m}$ и $j = \overline{1,n}$.

– **Умножение матрицы на число.**

Произведение матрицы A (размерностью $m \times n$ и элементами a_{ij}) на число α является матрица C (размерностью $m \times n$ и элементами c_{ij}), где $c_{ij} = \alpha a_{ij}$ для всех $i = \overline{1,m}$ и $j = \overline{1,n}$.

– **Умножение двух матриц.**

Произведением матрицы A (размерностью $m \times n$ и элементами a_{ij}) на матрицу B (размерностью $m \times n$ и элементами b_{ij}) является матрица C (размерностью $m \times n$ и элементами c_{ij}), где $c_{ij} = \sum_{\rho-1}^{n} a_{ij} b_{ij}$ для всех $i = \overline{1,m}$ и $j = \overline{1,n}$.

– **Транспонирование матрицы.**

Транспонированной матрицей от матрицы A (размерностью $m \times n$ и элементами a_{ij}) является матрица A^T(размерностью $m \times n$ и элементами a_{ij}), где a_{ij} транспонированной матрицы = a_{ji} исходной матрицы.

– **Обращение матрицы.**

Обращение матрицы A (размерностью $m \times n$ и элементами a_{ij}) это получение матрицы A^{-1}, матрицы, при умножении которой на исходную матрицу A получается единичная матрица E.эта матрица, удовлетворяет равенству: $AA^{-1} = A^{-1}A = E$.

7. Алгоритмы основных операций над матрицами

7.1 Сложение двух матриц над конечным поле.

Входные данные: матрицы $matrixA$ с размерностью $N \times M$ и $matrixB$ с размерностью $R \times T$ и порядок поля $order$.

Выходные данные: матрица $matrixC$ – результат сложения двух матриц над конечным полем или сообщение «Сложение матриц невозможно».

Описание алгоритма:

1) Проверяется условие возможности сложения матриц: если $N \neq R$ ИЛИ $M \neq T$, то выводится сообщение «Сложение матриц невозможно».
2) Иначе запускается цикл прохода по индексу i от 0-го до N-1.
3) Внутри цикла запускается цикл прохода по индексу j от 0-го до M-1.
4) Внутри циклов составляется матрица $matrixC$: $matrixC[i][j] = matrixA[i][j] + matrix[i][j]$.
5) При выводе $matrixC$ – результата сложения матриц $matrixA$ и $matrixB$ выводится $matrixC[i][j]\ mod\ order$.

Сложность алгоритма: $n * m$.

7.2 Умножение матрицы на число над конечным полем.

Входные данные: матрица $matrix$ с размерностью $N \times M$, порядок поля $order$ и число x, на которое умножается матрица.

Выходные данные: результат умножения матрицы на число над конечным полем.

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до N-1.
2) Внутри цикла запускается цикл прохода по индексу j от 0-го до M-1.
3) Внутри цикла происходит вывод элементов матрицы $matrix$, где каждый элемент умножается на число x и $(matrix[i][j] * x) mod\ order$.

Сложность алгоритма: $n * m$.

7.3 Умножение двух матриц над конечным полем.

Входные данные: матрицы $matrixA$ с размерностью $N \times M$ и $matrixB$ с размерностью $R \times T$ и порядок поля $order$.

Выходные данные: матрица $matrixD$ – результат умножения двух матриц над конечным полем или сообщение «Умножение матриц невозможно».

Описание алгоритма:

1) Проверяется условие возможности сложения матриц: если $M = R$, то запускается цикл прохода по индексу i от 0-го до N-1.
2) Внутри цикла запускается цикл прохода по индексу j от 0-го до T-1.
3) Внутри цикла запускается цикл прохода по индексу q от 0-го до R-1.
4) Внутри циклов составляется матрица $matrixD$: $matrixD[i][j] = matrixD[i][j] + matrixA[i][q] * matrix[q][j]$.
5) Иначе выводится сообщение «Умножение матриц невозможно».
6) При выводе $matrixD$ – результата умножения матриц $matrixA$ и $matrixB$ выводится $matrixD[i][j]\ mod\ order$.

Сложность алгоритма: $n * t * r$.

7.4 Транспонирование матрицы над конечным полем.

Входные данные: матрица $matrix$ с размерностью $N \times M$ и порядок поля $order$.

Выходные данные: транспонированная матрица $matrix$ с размерностью $M \times N$ над конечным полем.

Описание алгоритма:

1) Запускается цикл прохода по индексу i от 0-го до M-1.
2) Внутри цикла запускается цикл прохода по индексу j от 0-го до N-1.
3) Внутри цикла происходит формирование матрицы $matrixtransposed$: $matrixtransposed[i][j] = matrix[j][i]$.
4) При выводе $matrixtransposed$ – результата транспонирования матрицы $matrix$ выводится $matrixtransposed[i][j]\ mod\ order$.

Сложность алгоритма: $n * m$.

7.5 Обращение матрицы над конечным полем.

Входные данные: матрицы $matrix$ с размерностью $N \times N$ и порядок поля $order$.

Выходные данные: обращенная матрица с размерностью $N \times N$ над конечным полем или сообщение «Матрица является вырожденной».

Описание алгоритма:

1) Вычисляется определитель матрицы.
2) Проводится проверка матрицы на возможность нахождения обратной матрицы: если при делении определителя на порядок поля остаток отделения равен нулю, то на выход идёт сообщение «Матрица является вырожденной» и выполнение программы прекращается.
3) Иначе продолжается вычисление обращенной матрицы.
4) Формируется матрица $algdop$ – матрица алгебраических дополнений для матрицы $matrix$.
5) Формируется матрица $algdoptransposed$ – результат транспонирования матрицы $algdop$.
6) Вычисляется $detinverse$ – обратный элемент в конце по модулю.
7) Преобразуется матрица $algdop\_transposed$: запускается проход по матрице от элемента $algdoptransposed[0][0]$ до элемента $algdoptransposed[N-1][N-1]$ по индексу i от 0-го до $N-1$ и по индексу j от 0-го до $N-1$ и каждый $algdoptransposed[i][j]$ умножается на $detinverse$.
8) Запускается цикл прохода по индексу i от 0-го до $N-1$.
9) Внутри цикла запускается цикл прохода по индексу j от 0-го до $N-1$.
10) Внутри циклов проверяется условие: если $algdoptransposed[i][j] \geq 0$, то $algdoptransposed[i][j] = algdoptransposed[i][j]\ mod\ order$.
11) Иначе $algdoptransposed[i][j] = (algdoptransposed[i][j]\ mod\ order) + order$.
12) На выход идёт матрица $algdoptransposed$.

Сложность алгоритма: n^4.

8. Реализация рассмотренных алгоритмов

8.1 Алгоритмы для проверки свойств операций.

Программный код:

```
#include <iostream>
#include <windows.h>
#include <vector>

using namespace std;

bool associative(vector<vector<int>>& cayley_table) {
    for (int x = 0; x < size(cayley_table); ++x) {
        for (int y = 0; y < size(cayley_table); ++y) {
            for (int z = 0; z < size(cayley_table); ++z) {
                if (cayley_table[y][cayley_table[x][z] - 1]
                != cayley_table[cayley_table[y][x] - 1][z]) {
                    return false;
                }
            }
        }
    }
    return true;
}

bool commutativity(vector<vector<int>>& cayley_table) {
    for (int i = 0; i < size(cayley_table); ++i) {
        for (int j = 0; j < size(cayley_table); ++j) {
            if (cayley_table[i][j] != cayley_table[j][i]) {
                return false;
            }
        }
    }
    return true;
}

bool idempotency(vector<vector<int>>& cayley_table) {
    for (int i = 0; i < size(cayley_table); ++i) {
        if (cayley_table[i][i] != i + 1) {
            return false;
        }
    }
    return true;
}

bool invertibility(vector<vector<int>>& cayley_table) {
    for (int i = 0; i < size(cayley_table); ++i) {
        for (int j = 0; j < size(cayley_table); ++j) {
            if (!(cayley_table[i][j] == cayley_table[j][i] && cayley_table[i][j]
== 1)){
                return false;
            }
        }
    }
    return true;
}

bool distributivity(vector<vector<int>>& cayley_table) {

    cout << "Другая таблица Кэли:" << endl;
```

```
    vector<vector<int>> another_cayley_table(size(cayley_table),
vector<int>(size(cayley_table)));
    for (int i = 0; i < size(another_cayley_table); ++i) {
        for (int j = 0; j < size(another_cayley_table); ++j) {
            cin >> another_cayley_table[i][j];
        }
    }

    for (int x = 0; x < size(another_cayley_table); ++x) {
        for (int y = 0; y < size(another_cayley_table); ++y) {
            for (int z = 0; z < size(another_cayley_table); ++z) {
                int p = another_cayley_table[y][z] - 1;
                int q = cayley_table[x][y] - 1;
                int m = cayley_table[x][z] - 1;
                int g = another_cayley_table[y][z] - 1;
                int h = cayley_table[y][x] - 1;
                int u = cayley_table[z][x] - 1;
                if (cayley_table[x][p] != another_cayley_table[q][m] ||
                    cayley_table[g][x] != another_cayley_table[h][u]) {
                    return false;
                }
            }
        }
    }
    return true;
}

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Количество элементов" << endl;
    int n;
    cin >> n;

    cout << "Таблица Кэли:" << endl;
    vector<vector<int>> cayley_table(n, vector<int> (n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> cayley_table[i][j];
        }
    }

    if (associative(cayley_table)) {
        cout << "Ассоциативность выполянется" << endl;
    } else {
        cout << "Ассоциативность не выполняется" << endl;
    }

    if (commutativity(cayley_table)) {
        cout << "Коммутативность выполянется" << endl;
    } else {
        cout << "Коммутативность не выполняется" << endl;
    }

    if (idempotency(cayley_table)) {
        cout << "Идемпотентность выполняется" << endl;
    } else {
        cout << "Идемпотентность не выполняется" << endl;
    }

    if (invertibility(cayley_table)) {
        cout << "Обратимость выполняется" << endl;
```

```
    } else {
        cout << "Обратимость не выполняется" << endl;
    }

    if (distributivity(cayley_table)) {
        cout << "Дистрибутивность выполняется" << endl;
    } else {
        cout << "Дистрибутивность не выполняется" << endl;
    }

    return 0;
}
```

Результаты тестирования:

```
Количество элементов
5
Таблица Кэли:
1 2 3 4 5
2 3 4 5 1
3 4 5 1 2
4 5 1 2 3
5 1 2 3 4
Ассоциативность выполянется
Коммутативность выполянется
Идемпотентность не выполняется
Обратимость не выполняется
Другая таблица Кэли:
1 2 3 1 3
1 3 2 3 4
1 1 1 4 5
4 3 2 1 1
4 3 5 2 3
Дистрибутивность не выполняется
```

8.2 Алгоритмы операций над бинарными отношениями.

– **Объединение и пересечение бинарных отношений.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>

using namespace std;
```

```
int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матриц A и B (n m)" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица A:" << endl;
    vector<vector<int>> matrix_A(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix_A[i][j];
        }
    }

    cout << "Матрица B:" << endl;
    vector<vector<int>> matrix_B(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix_B[i][j];
        }
    }

    vector<pair<int, int>> result_union;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (matrix_A[i][j] || matrix_B[i][j]) {
                result_union.push_back({i + 1, j + 1});
            }
        }
    }

    vector<pair<int, int>> result_intersection;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            if (matrix_A[i][j] && matrix_B[i][j]) {
                result_intersection.push_back({i + 1, j + 1});
            }
        }
    }

    cout << "Результат объединения бинарных отношений: " << endl;
    cout << "[ ";
    for (auto q : result_union) {
        cout << "(" << q.first << ", " << q.second << ")" << ", ";
    }
    cout << "]" << endl;

    cout << "Результат пересечения бинарных отношений: " << endl;
    cout << "[ ";
    for (auto q : result_intersection) {
        cout << "(" << q.first << ", " << q.second << ")" << ", ";
    }
    cout << "]" << endl;

    return 0;
}
```

Результаты тестирования:

```
Размерность матриц A и B (n m)
3 2
Матрица A:
1 0
1 1
0 1
Матрица B:
1 1
0 0
0 0
Результат объединения бинарных отношений:
[ (1, 1), (1, 2), (2, 1), (2, 2), (3, 2), ]
Результат пересечения бинарных отношений:
[ (1, 1), ]
```

– **Дополнение и обращение бинарных отношений.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>

using namespace std;

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы (n m):" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица: " << endl;
    vector<vector<int>> matrix(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix[i][j];
        }
    }

    vector<vector<int>> matrix_transposed(m, vector<int> (n));
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j){
            matrix_transposed[i][j] = matrix[j][i];
        }
    }

    vector<pair<int, int>> result_additions;
    for (int i = 0; i < n; ++i) {
```

```
            for (int j = 0; j < m; ++j) {
                if (!matrix[i][j]) {
                    result_additions.push_back({i + 1, j + 1});
                }
            }
        }

        vector<pair<int, int>> result_reversal;
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if(matrix_transposed[i][j]){
                    result_reversal.push_back({i + 1, j + 1});
                }
            }
        }

        cout << "Результат дополнения бинарного отношения: " << endl;
        cout << "[ ";
        for (auto q : result_additions) {
            cout << "(" << q.first << ", " << q.second << ")" << ", ";
        }
        cout << "]" << endl;

        cout << "Результат обращения бинарного отношения: " << endl;
        cout << "[ ";
        for (auto q : result_reversal) {
            cout << "(" << q.first << ", " << q.second << ")" << ", ";
        }
        cout << "]" << endl;

        return 0;
}
```

Результаты тестирования:

```
Размерность матрицы (n m):
3 4
Матрица:
0 1 0 1
1 1 0 0
0 0 1 1
Результат дополнения бинарного отношения:
[ (1, 1), (1, 3), (2, 3), (2, 4), (3, 1), (3, 2), ]
Результат обращения бинарного отношения:
[ (1, 2), (2, 1), (2, 2), (3, 3), (4, 1), (4, 3), ]
```

– **Композиция бинарных отношений.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>
```

```cpp
using namespace std;

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы A (n m)" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица A:" << endl;
    vector<vector<int>> matrix_A(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix_A[i][j];
        }
    }

    cout << "Размерность матрицы B (r t)" << endl;
    int r, t;
    cin >> r >> t;

    cout << "Матрица B:" << endl;
    vector<vector<int>> matrix_B(r, vector<int> (t));
    for (int i = 0; i < r; ++i) {
        for (int j = 0; j < t; ++j) {
            cin >> matrix_B[i][j];
        }
    }

    vector<vector<int>> matrix_C(n, vector<int> (t));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < t; ++j) {
            matrix_C[i][j] = 0;
            for (int w = 0; w < m; ++w) {
                matrix_C[i][j] += matrix_A[i][w] * matrix_B[w][j];
            }
            if (matrix_C[i][j] > 1) {
                matrix_C[i][j] = 1;
            }
        }
    }

    vector<pair<int, int>> result_composition;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < t; ++j) {
            if (matrix_C[i][j]) {
                result_composition.push_back({i + 1, j + 1});
            }
        }
    }

    cout << "Результат композиции бинарных отношений: " << endl;
    if (m == r) {
        cout << "[ ";
        for (auto q : result_composition) {
            cout << "(" << q.first << ", " << q.second << ")" << ", ";
        }
        cout << "]" << endl;
    } else {
        cout << "Произведение матриц невозможно" << endl;
    }
```

```
    return 0;
}
```

Результаты тестирования:

```
Размерность матрицы A (n m)
2 4
Матрица A:
1 0 0 1
0 0 1 1
Размерность матрицы B (r t)
4 3
Матрица B:
0 1 1
1 0 1
1 1 1
0 0 1
Результат композиции бинарных отношений:
[ (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), ]
```

8.3 Алгоритмы основных операций над матрицами.

– **Сложение и умножение двух матриц над конечным полем.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>

using namespace std;

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы A (n m)" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица A:" << endl;
    vector<vector<int>> matrix_A(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix_A[i][j];
        }
    }

    cout << "Размерность матрицы B (r t)" << endl;
    int r, t;
```

```
    cin >> r >> t;

    cout << "Матрица B:" << endl;
    vector<vector<int>> matrix_B(r, vector<int> (t));
    for (int i = 0; i < r; ++i) {
        for (int j = 0; j < t; ++j) {
            cin >> matrix_B[i][j];
        }
    }

    cout << "Введите порядок поля:" << endl;
    int order;
    cin >> order;

    vector<vector<int>> matrix_C(n, vector<int> (m));

    if (n != r || m != t) {
        cout << "Сложение матриц невозможно" << endl;
    } else {
        for (int i = 0; i < n; ++i){
            for (int j = 0; j < m; ++j) {
                matrix_C[i][j] = matrix_A[i][j] + matrix_B[i][j];
            }
        }

        cout << "Результат сложения матриц A и B:" << endl;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < m; ++j) {
                cout << setw(4) << matrix_C[i][j] % order << ' ';
            }
            cout << endl;
        }

    }

    vector<vector<int>> matrix_D(n, vector<int> (t));

    if (m == r) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < t; ++j) {
                matrix_D[i][j] = 0;
                for (int q = 0; q < r; ++q) {
                    matrix_D[i][j] += matrix_A[i][q] * matrix_B[q][j];
                }
            }
        }
        cout << "Результат умножения матриц A и B: " << endl;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < t; ++j) {
                cout << setw(4) << matrix_D[i][j] % order << ' ';
            }
            cout << endl;
        }
    } else {
        cout << "Умножение матриц невозможно" << endl;
    }

    return 0;
}
```

Результаты тестирования:

```
Размерность матрицы A (n m)
3 3
Матрица A:
1 2 3
4 5 6
7 8 9
Размерность матрицы B (r t)
3 3
Матрица B:
9 8 7
6 5 4
3 2 1
Введите порядок поля:
7
Результат сложения матриц A и B:
   3    3    3
   3    3    3
   3    3    3
Результат умножения матриц A и B:
   2    3    4
   0    6    5
   5    2    6
```

– **Умножение матрицы на число над конечным полем.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>

using namespace std;

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы (n m):" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица: " << endl;
    vector<vector<int>> matrix(n, vector<int> (m));
```

```
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix[i][j];
        }
    }

    cout << "Число, на которое умножается матрица: " << endl;
    int x;
    cin >> x;

    cout << "Введите порядок поля:" << endl;
    int order;
    cin >> order;

    cout << endl << "Результат умножения матрицы на число:" << endl;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cout << setw(4) << matrix[i][j] * x % order<< ' ';
        }
        cout << endl;
    }

    return 0;
}
```

Результаты тестирования:

```
Размерность матрицы (n m):
2 4
Матрица:
3 5 6 2
8 4 5 3
Число, на которое умножается матрица:
2
Введите порядок поля:
6

Результат умножения матрицы на число:
   0    4    0    4
   4    2    4    0
```

– **Транспонирование матрицы над конечным полем.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>

using namespace std;

int main() {
```

```
    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы (n m):" << endl;
    int n, m;
    cin >> n >> m;

    cout << "Матрица: " << endl;
    vector<vector<int>> matrix(n, vector<int> (m));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < m; ++j) {
            cin >> matrix[i][j];
        }
    }

    cout << "Введите порядок поля:" << endl;
    int order;
    cin >> order;

    vector<vector<int>> matrix_transposed(m, vector<int> (n));
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j){
            matrix_transposed[i][j] = matrix[j][i];
        }
    }

    cout << endl << "Транспонированная матрица:" << endl;
    for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
            cout << setw(4) << matrix_transposed[i][j] % order << ' ';
        }
        cout << endl;
    }

    return 0;
}
```

Результаты тестирования:

```
Размерность матрицы (n m):
2 4
Матрица:
32 11 43 12
26 72 9 10
Введите порядок поля:
5

Транспонированная матрица:
   2    1
   1    2
   3    4
   2    0
```

– **Обращение матрицы над конечным полем.**

Программный код:

```
#include <iostream>
#include <iomanip>
#include <windows.h>
#include <vector>
#include <cmath>

using namespace std;

int Get_minor(vector<vector<int>>& matrix,vector<vector<int>>& matrix_alg, int
x, int y, int size) {
    int xCount = 0;
    int yCount = 0;

    for (int i = 0; i < size; ++i) {
        if (i != x) {
            yCount = 0;
            for (int j = 0; j < size; ++j) {
                if (j != y) {
                    matrix_alg[xCount][yCount] = matrix[i][j];
                    ++yCount;
                }
            } ++xCount;
        }
    }
    return 0;
}

int Find_determinant (vector<vector<int>>& matrix, int size) {
    if (size == 1) {
        return matrix[0][0];
    } else {

        int det = 0;
        vector<vector<int>> minor(size - 1, vector<int> (size - 1));
        for (int i = 0; i < size; ++i) {
            Get_minor(matrix, minor, 0, i, size);
            det += pow(-1, i) * matrix[0][i] * Find_determinant(minor, size -
1);
        }

        return det;
    }

}


void Find_alg_dop(vector<vector<int>>& matrix, int size, vector<vector<int>>&
matrix_alg) {
    int det = Find_determinant(matrix, size);
    if (det > 0) {
        det = -1;
    } else {
        det = 1;
    }

    vector<vector<int>> minor(size - 1, vector<int> (size - 1));
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
```

```
            Get_minor(matrix, minor, i, j, size);
            if ((i + j) % 2 == 0) {
                matrix_alg[i][j] = -det * Find_determinant(minor, size - 1);
            } else {
                matrix_alg[i][j] = det * Find_determinant(minor, size - 1);
            }
        }
    }
}

void gcdExtended (int a, int b, int& x, int& y)
{
    if (a == 0) { x = 0; y = 1; return; }

    int x_1, y_1;
    gcdExtended (b % a, a, x_1, y_1);

    x = y_1 - (b / a) * x_1;
    y = x_1;

    return;
}

int main() {

    SetConsoleOutputCP(CP_UTF8);

    cout << "Размерность матрицы для обращения (n): " << endl;
    int n;
    cin >> n;

    cout << "Матрица для обращения: " << endl;
    vector<vector<int>> matrix(n, vector<int> (n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> matrix[i][j];
        }
    }

    int det = Find_determinant(matrix, n);
    int det_inverse = 0;
    int rrr = 0;

    cout << "Введите порядок поля:" << endl;
    int order;
    cin >> order;

    if (det % order == 0) {
        cout << "Матрица является вырожденной" << endl;
        return 0;
    }

    vector<vector<int>> alg_dop(n, vector<int> (n));
    Find_alg_dop(matrix, n, alg_dop);

    vector<vector<int>> alg_dop_transposed(n, vector<int> (n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            alg_dop_transposed[i][j] = alg_dop[j][i];
        }
    }

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

```
        for (int j = 0; j < n; ++j) {
            alg_dop_transposed[i][j] = alg_dop_transposed[i][j] % order;
        }
    }

    gcdExtended (abs(det), order, det_inverse, rrr);

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            alg_dop_transposed[i][j] = alg_dop_transposed[i][j] * det_inverse;
        }
    }

    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (alg_dop_transposed[i][j] >= 0) {
                alg_dop_transposed[i][j] = alg_dop_transposed[i][j] % order;
            } else {
                alg_dop_transposed[i][j] = (alg_dop_transposed[i][j] % order) +
order;
            }
        }
    }

    cout << "ОБращенная матрица по модулю " << order << " : " << endl;

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

    return 0;
}
```

Результат тестирования:

```
Размерность матрицы для обращения (n):
3
Матрица для обращения:
6 24 1
13 16 10
20 17 15
Введите порядок поля:
26
ОБращенная матрица по модулю 26 :
8 5 10
21 8 21
21 12 8
```

9. Решение задач по варианту 3.

Вариант 3.

Задание 1.

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

Решение:

Мн-во {a,b,c,d}. Проходясь по всей таблице смотрим выполнение условия: $\forall x,y,z \in T_c: x(y \cdot z) = (x \cdot y) \cdot z$

1) $(a \cdot a) \cdot a = b \cdot a = b$

2) $(a \cdot b) \cdot a = b \cdot a = b$
$a \cdot (b \cdot a) = a \cdot b = b$

3) $(a \cdot c) \cdot a = a \cdot a = b$
$a \cdot (c \cdot a) = a \cdot a = b$

4) $(a \cdot d) \cdot a = a \cdot a = b$
$a \cdot (d \cdot a) = a \cdot a = b$

5) $(b \cdot a) \cdot a = b \cdot a = b$
$b \cdot (a \cdot a) = b \cdot b = b$

6) $(b \cdot c) \cdot a = b \cdot a = b$
$b \cdot (c \cdot a) = b \cdot a = b$

7) $(b \cdot d) \cdot a = b \cdot a = b$
$b \cdot (d \cdot a) = b \cdot a = b$

8) $(c \cdot a) \cdot a = a \cdot a = b$
$c \cdot (a \cdot a) = c \cdot b = b$

9) $(c \cdot b) \cdot a = b \cdot a = b$
$c \cdot (b \cdot a) = c \cdot b = b$

10) $(c \cdot d) \cdot a = c \cdot a = a$
$c \cdot (d \cdot a) = c \cdot a = a$

11) $(d \cdot a) \cdot a = a \cdot a = b$
$d \cdot (a \cdot a) = d \cdot b = b$

12) $(d \cdot b) \cdot a = b \cdot a = b$
$d \cdot (b \cdot a) = d \cdot b = b$

13) $(d \cdot d) \cdot a = d \cdot a = a$
$d \cdot (d \cdot a) = d \cdot a = a$

14) $(b \cdot b) \cdot a = b \cdot a = b$
$b \cdot (ba) = b \cdot b = b$

15) $(c \cdot c) \cdot a = d \cdot a = a$
$c \cdot (c \cdot a) = c \cdot a = a$

и т.д. В итоге после проверки всех возможных вариаций элементов выяснилось, что все комбинации удовлетворяют условию, значит проверяемая операция ассоциативна.

Задание 2.

Найти $A^2+(10-\frac{3}{2})\cdot A+\frac{3}{2}\cdot E$, где $A=\begin{pmatrix}1 & -2\\ -3 & 3\end{pmatrix}$, E-единичная матрица

Решение:

1) $A^2=\begin{pmatrix}1 & -2\\ -3 & 3\end{pmatrix}\cdot\begin{pmatrix}1 & -2\\ -3 & 3\end{pmatrix}=\begin{pmatrix}7 & -8\\ -12 & 15\end{pmatrix}$

2) $(10-3|2)\cdot A=8{,}5\cdot\begin{pmatrix}1 & -2\\ -3 & 3\end{pmatrix}=\begin{pmatrix}8{,}5 & -17\\ -25{,}5 & 25{,}5\end{pmatrix}$

3) $\frac{3}{2}\cdot E=1{,}5\cdot\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}=\begin{pmatrix}1{,}5 & 0\\ 0 & 1{,}5\end{pmatrix}$

4) $\begin{pmatrix}7 & -8\\ -12 & 15\end{pmatrix}+\begin{pmatrix}8{,}5 & -17\\ -25{,}5 & 25{,}5\end{pmatrix}=\begin{pmatrix}15{,}5 & -25\\ -37{,}5 & 40{,}5\end{pmatrix}$

5) $\begin{pmatrix}15{,}5 & -25\\ -37{,}5 & 40{,}5\end{pmatrix}+\begin{pmatrix}1{,}5 & 0\\ 0 & 1{,}5\end{pmatrix}=\begin{pmatrix}17 & -25\\ -37{,}5 & 42\end{pmatrix}$

Ответ: $\begin{pmatrix}17 & -25\\ -37{,}5 & 42\end{pmatrix}$

Задание 3

$A\cdot B-?$ $A=\begin{pmatrix}-1 & 3 & 3\\ 1 & 2 & 7\end{pmatrix}$ $B=\begin{pmatrix}-3 & 2\\ 1 & 8{,}5\\ -3 & 3\end{pmatrix}$

Решение:

$$\begin{pmatrix}-1 & 3 & 3\\ 1 & 2 & 7\end{pmatrix}\cdot\begin{pmatrix}-3 & 2\\ 1 & 8{,}5\\ -3 & 3\end{pmatrix}=\begin{pmatrix}-1\cdot-3+3\cdot1+3\cdot-3 & -1\cdot2+3\cdot8{,}5+3\cdot3\\ 1\cdot-3+2\cdot1+7\cdot-3 & 1\cdot2+2\cdot8{,}5+7\cdot3\end{pmatrix}=\begin{pmatrix}3+3+(-9) & -2+25{,}5+9\\ -3+2+(-21) & 2+17+21\end{pmatrix}$$

$$=\begin{pmatrix}-3 & 32{,}5\\ -22 & 40\end{pmatrix}$$

Ответ: $\begin{pmatrix}-3 & 32{,}5\\ -22 & 40\end{pmatrix}$

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были рассмотрены алгебраические операции и классификация свойств операций, операции над бинарными отношениями и выполнение операций над матрицами над конечным полем.

Результатом работы является: алгоритмы для проверки свойств операции, алгоритмы для выполнения операций над бинарными отношениями и алгоритмы для выполнения операции над матрицами над конечным полем.

Была осуществлена программная реализация описанных алгоритмов и проведено тестирование программ.