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

Норикова Павла Сергеевича

|  |  |  |
| --- | --- | --- |
| Преподаватель  профессор, д.ф.-м.н. | \_\_\_\_\_\_\_\_\_\_ | В. А. Молчанов |
|  | подпись, дата |  |

Саратов 2022

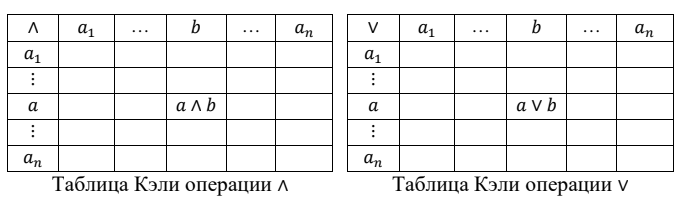
ВВЕДЕНИЕ

Цель данной работы — изучение основных понятий универсальной алгебры и операций над бинарными отношениями, а также разработка алгоритмов проверки свойств операций и алгоритмов выполнения операции над бинарными отношениями и матрицами.

АЛГЕБРАИЧЕСКАЯ ОПЕРАЦИЯ И КЛАССИФИКАЦИЯ СВОЙСТВ ОПЕРАЦИЙ

В решёточно упорядоченном множестве определены две бинарные операции: и . Операция ∧ называется пересечением и операция ∨ – объединением. Эти операции решетки взаимосвязаны с ее порядком по формулам: .

Для конечного множества такие операции определяются с помощью таблиц Кэли – квадратных таблиц размерности , в которых строки и столбцы помечены элементами множества и на пересечении строки и столбца находится результат применения соответствующей операции к паре .



1) , – идемпотентность операций ∧,∨;

2) , – коммутативность операций ∧,∨;

3) ,  – ассоциативность операций ∧,∨;

4)  Решетка называется дистрибутивной, если для любых выполняются равенства: ,

5) Операция называется обратимой, если уравнения и ( и ), где , имеют решение, причём единственное.

Отображение называется алгебраической n-арной операцией или просто алгебраической операцией на множестве A. При этом n называется порядком или арностью алгебраической операции f.

Алгоритм 1 – проверка свойства идемпотентности для операций .

Вход. Таблица Кэли v, количество элементов таблицы N.

Выход. "Операция обладает свойством идемпотентности" или "Операция НЕ обладает свойством идемпотентности".

Шаг 1. В цикле по i от 0 до N - 1 сравниваем значения v2[i][i] с i.

Шаг 2. Если все они равны, выводим "Операция обладает свойством идемпотентности", иначе "Операция НЕ обладает свойством идемпотентности".

Алгоритм 2 – проверка свойства ассоциативности для операций .

Вход. Таблица Кэли v, количество элементов таблицы N.

Выход. "Операция обладает свойством ассоциативности" или "Операция НЕ обладает свойством ассоциативности ".

Шаг 1. В цикле по i от 0 до N – 1

Шаг 1.1 В цикле по j от 0 до N – 1

Шаг 1.1.1 В цикле по k от 0 до N – 1 сравниваем v[v[i][j]][k] и v[i][v[j][k]].

Шаг 2. Если все они равны, выводим "Операция обладает свойством ассоциативности ", иначе "Операция НЕ обладает свойством ассоциативности ".

Алгоритм 3 – проверка свойства коммутативности для операций .

Вход. Таблица Кэли v, количество элементов таблицы N.

Выход. "Операция обладает свойством коммутативности" или "Операция НЕ обладает свойством коммутативности".

Шаг 1. В цикле по i от 0 до N – 1

Шаг 1.1 В цикле по j от 0 до N – 1 сравниваем v[i][j] и v[j][i].

Шаг 2. Если все они равны, выводим "Операция обладает свойством коммутативности", иначе "Операция НЕ обладает свойством коммутативности".

Алгоритм 4 – проверка свойства дистрибутивности для операций .

Вход. Таблица Кэли v1 операции , Таблица Кэли v2 операции , количество элементов таблицы N.

Выход. "Решётка является дистрибутивной" или "Решётка НЕ является дистрибутивной".

Шаг 1. В цикле по i от 0 до N – 1

Шаг 1.1 В цикле по j от 0 до N – 1

Шаг 1.1.1 В цикле по k от 0 до N – 1 сравниваем v2[v1[i][j]][k] и v1[[i][k]][v2[j][k]]

Шаг 2. Если все они равны, выводим "Решётка является дистрибутивной", иначе "Решётка НЕ является дистрибутивной".

ОСНОВНЫЕ ОПЕРАЦИИ НАД БИНАРНЫМИ ОТНОШЕНИЯМИ

Объединением бинарных отношений и называется бинарное отношение , определяющееся по формуле: или;

Пересечением бинарных отношений и называется бинарное отношение , определяющееся по формуле: и;

Дополнением бинарного отношения называется отношение , определяющееся по формуле: и;

Обратным для бинарного отношения называется бинарное отношение , определяющееся по формуле: ;

Композицией бинарных отношений и называется бинарное отношение , определяющееся по формуле: .

Алгоритм 5 – построение объединения множеств, представляющих бинарные отношения:

*Вход.* Матрица бинарного отношения v1, матрица бинарного отношения v2 на множестве А, количество элементов множества А - N.

*Выход.* Матрица объединения res.

Шаг 1. Создаём матрицу res и кладём в неё v1.

Шаг 2. В цикле по i от 0 до N – 1

Шаг 2.1 В цикле по j от 0 до N – 1 если v2[i][j] = 1, положим v2[i][j] = 1.

Шаг 3. Выводим res.

Алгоритм 6 – построение пересечения множеств, представляющих бинарные отношения:

*Вход.* Матрица бинарного отношения v1, матрица бинарного отношения v2 на множестве А, количество элементов множества А - N.

*Выход.* Матрица пересечения res.

Шаг 1. Создаём матрицу res.

Шаг 2. В цикле по i от 0 до N – 1

Шаг 2.1 В цикле по j от 0 до N – 1 положим res[i][j] = v1[i][j] \* v2[i][j].

Шаг 3. Выводим res.

Алгоритм 7 – построение обратного множества, представляющего бинарное отношение:

*Вход.* Матрица бинарного отношения v на множестве А, количество элементов множества А - N.

*Выход.* Обратная матрица res.

Шаг 1. Создаём матрицу res.

Шаг 2. В цикле по i от 0 до N – 1

Шаг 2.1 В цикле по j от 0 до N – 1 положим res[i][j] = v[j][i].

Шаг 3. Выводим res.

Алгоритм 8 – построение дополнения множества, представляющего бинарное отношение:

*Вход.* Матрица бинарного отношения v на множестве А, количество элементов множества А - N.

*Выход.* Матрица дополнения res.

Шаг 1. Создаём матрицу res.

Шаг 2. В цикле по i от 0 до N – 1

Шаг 2.1 В цикле по j от 0 до N – 1 если v[i][j] = 1, положим res[i][j] = 0, иначе положим res[i][j] = 1.

Шаг 3. Выводим res.

Алгоритм 9 – построение композиции множеств, представляющих бинарные отношения:

*Вход.* Матрица бинарного отношения v1, матрица бинарного отношения v2 на множестве А, количество элементов множества А - N.

*Выход.* Матрица композиции res.

Шаг 1. Создаём матрицу res.

Шаг 2. В цикле по i от 0 до N – 1

Шаг 2.1 В цикле по j от 0 до N – 1 заводим переменную count = 0

Шаг 2.1.1 В цикле по k от 0 до N – 1 count = count + v1[i][k] \* v2[k][j].

Шаг 2.2 Если count > 0, res[i][j] = 1, иначе res[i][j] = 0

Шаг 3. Выводим res.

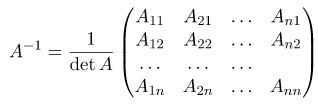
ОСНОВНЫЕ ОПЕРАЦИИ НАД МАТРИЦАМИ

*Суммой* двух матриц и одинаковой размерности называют матрицу такой же размерности, элементы которой равны сумме соответствующих элементов матриц и , то есть .

Если количество столбцов матрицы равно числу строк матрицы , то для них определена матрица размерности , которую называют её *произведением*. Элементы матрицы находятся по правилу: элемент равен сумме попарных произведений элементов i-той строки матрицы и j-того столбца матрицы : .

Если в матрице размерности заменить строки столбцами с соответствующими номерами, то получим *транспонированную матрицу* размерности .

Для невырожденной квадратной матрицы  существует *обратная матрица*  такая что . Найти обратную матрицу можно по формуле:

,

где  – алгебраические дополнения элементов  матрицы .

Следующие алгоритмы описаны для работы над конечным полем.

Алгоритм 10 – сложение матриц:

*Вход.* Матрица v1 и матрица v2 обе размерности .

*Выход.* Матрица сложения res.

Шаг 1. Создаём матрицу res.

Шаг 2. В цикле по i от 0 до n – 1

Шаг 2.1 В цикле по j от 0 до m – 1 положим res[i][j] = v1[i][j] + v2[i][j].

Шаг 3. Выводим res.

Алгоритм 11 – умножение матриц:

*Вход.* Матрица v1 размерности и матрица v2 размерности .

*Выход.* Матрица умножения res размерности .

Шаг 1. Создаём матрицу res размерности .

Шаг 2. В цикле по i от 0 до n – 1

Шаг 2.1 В цикле по j от 0 до m – 1 заводим переменную count = 0

Шаг 2.1.1 В цикле по k от 0 до l – 1 res[i][j] = res[i][j] + v1[i][k] \* v2[k][j].

Шаг 3. Выводим res.

Алгоритм 12 – транспонирование матрицы:

*Вход.* Матрица v размерности .

*Выход.* Транспонированная матрица res размерности .

Шаг 1. Создаём матрицу res размерности .

Шаг 2. В цикле по i от 0 до m – 1

Шаг 2.1 В цикле по j от 0 до n – 1 положим res[i][j] = v[j][i].

Шаг 3. Выводим res.

Алгоритм 13 – нахождение определителя матрицы:

*Вход.* Матрица v размерности .

*Выход.* Определитель матрицы res.

Шаг 1. Заводим переменные k = 1 и res = 0.

Шаг 2. Если n = 1, res = v[0][0].

Шаг 3. Если n = 2, res = v[0][0] \* v[1][1] - v[1][0] \* v[0][1].

Шаг 4. Иначе, в цикле по i от 0 до n – 1

Шаг 4.1 Создаём матрицу v2 размерности , которая является матрицей v без i-ой строки и i-го столбца.

Шаг 4.2 res = res + k \* v[0][i] \* Det(v2), где Det(matr) – наш алгоритм.

Шаг 4.3 k = k \* (-1).

Шаг 4. Возвращаем res.

Алгоритм 14 – обращение матрицы:

*Вход.* Матрица v размерности .

*Выход.* Обратная матрица res.

Шаг 1. Заводим переменную det = 0 и матрицу res размерности .

Шаг 2. det = 1/Det(v).

Шаг 3. В цикле по i от 0 до n – 1

Шаг 3.1 В цикле по j от 0 до n – 1

Шаг 3.1.1 Создаём матрицу v2 размерности , которая является матрицей v без i-ой строки и j-го столбца.

Шаг 3.1.2 res[i][j] = Det(v2).

Шаг 3.1.3 Если (i + j) - нечётное число, res[i][j] = res[i][j] \* (-1).

Шаг 4. Транспонируем матрицу res.

Шаг 5. Умножаем все элементы матрицы res на det.

Шаг 6. Выводим res.

ВРЕМЕННАЯ СЛОЖНОСТЬ АЛГОРИТМОВ

Проверка свойства идемпотентности - O(n)

Проверка свойства ассоциативности - O(n3)

Проверка свойства коммутативности - O(n3)

Проверка свойства дистрибутивности - O(n3)

Построение объединения множеств, представляющих бинарные отношения - O(n2)

Построение пересечения множеств, представляющих бинарные отношения - O(n2)

Построение обратного множества, представляющего бинарное отношение - O(n2)

Построение дополнения множества, представляющего бинарное отношение - O(n2)

Построение композиции множеств, представляющих бинарные отношения - O(n3)

Сложение матриц - O(n2)

Умножение матриц - O(n3)

Транспонирование матрицы - O(n2)

Нахождение определителя матрицы - O(n!)

Обращение матрицы - O(n2 \* n!)

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

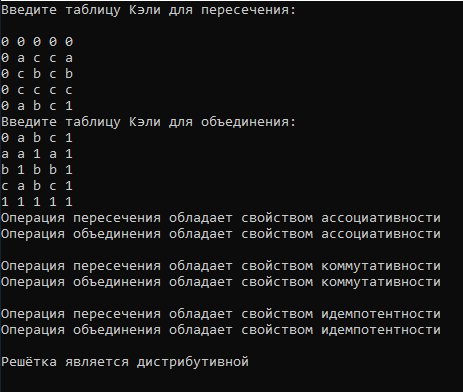


Рис.1 – первый тест

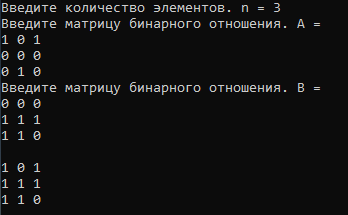


Рис.2 – второй тест

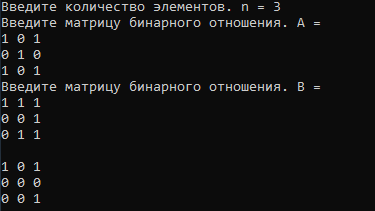


Рис.3 – третий тест

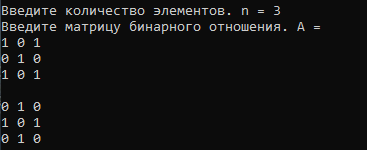


Рис.4 – четвёртый тест

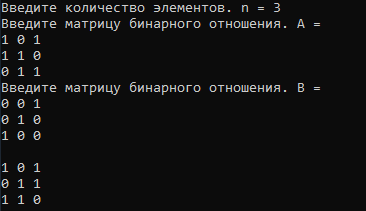


Рис.5 – пятый тест

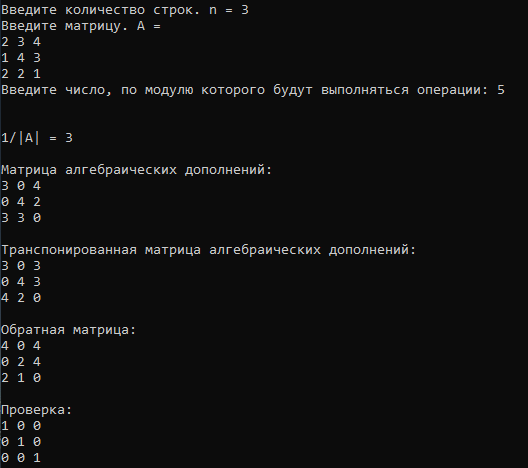
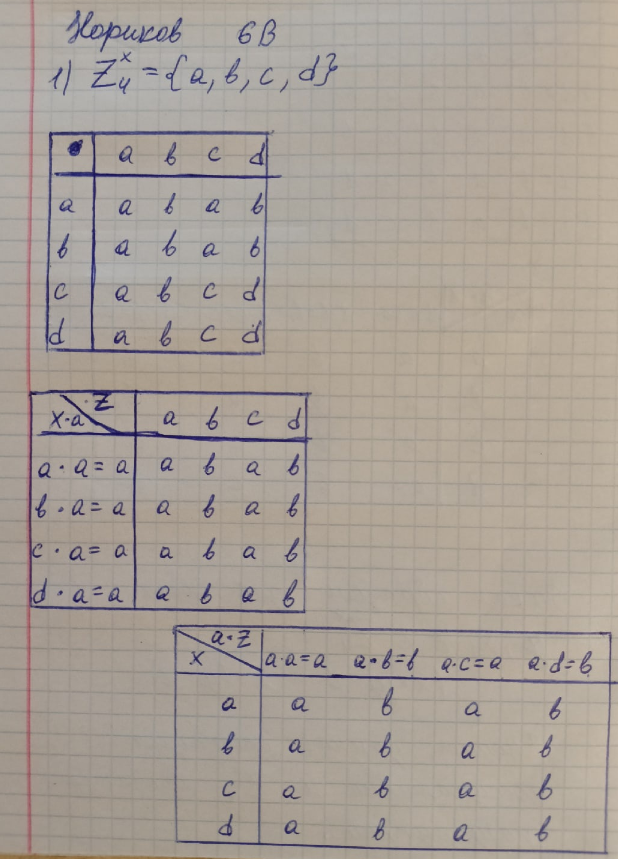
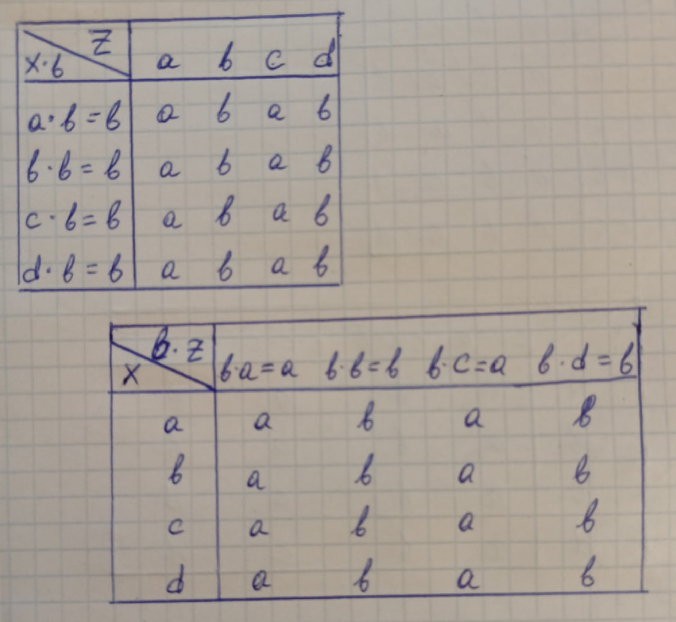
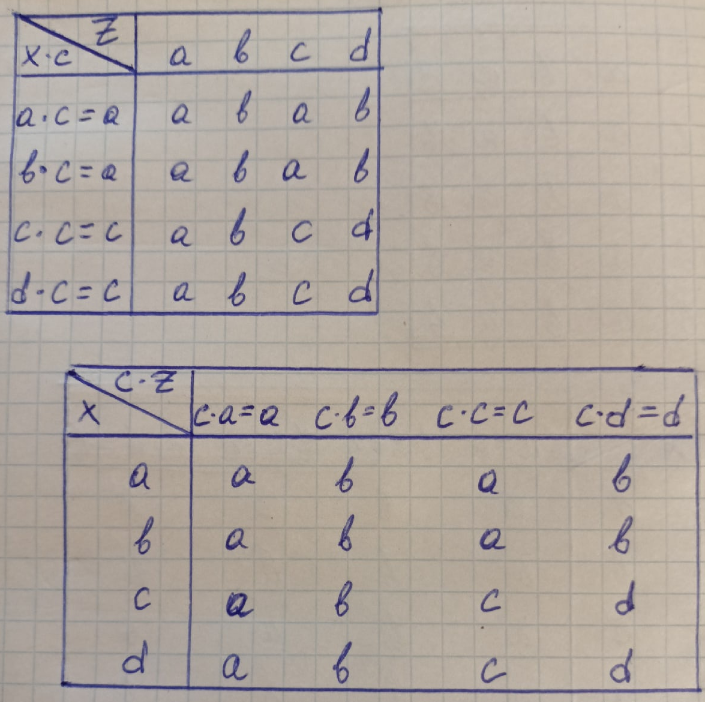


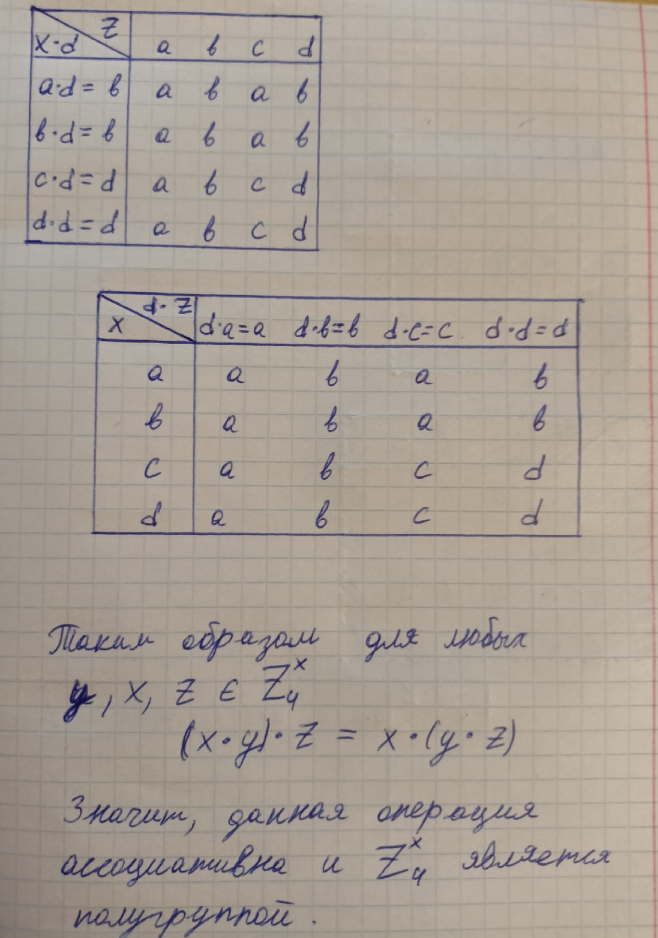
Рис.6 – шестой тест

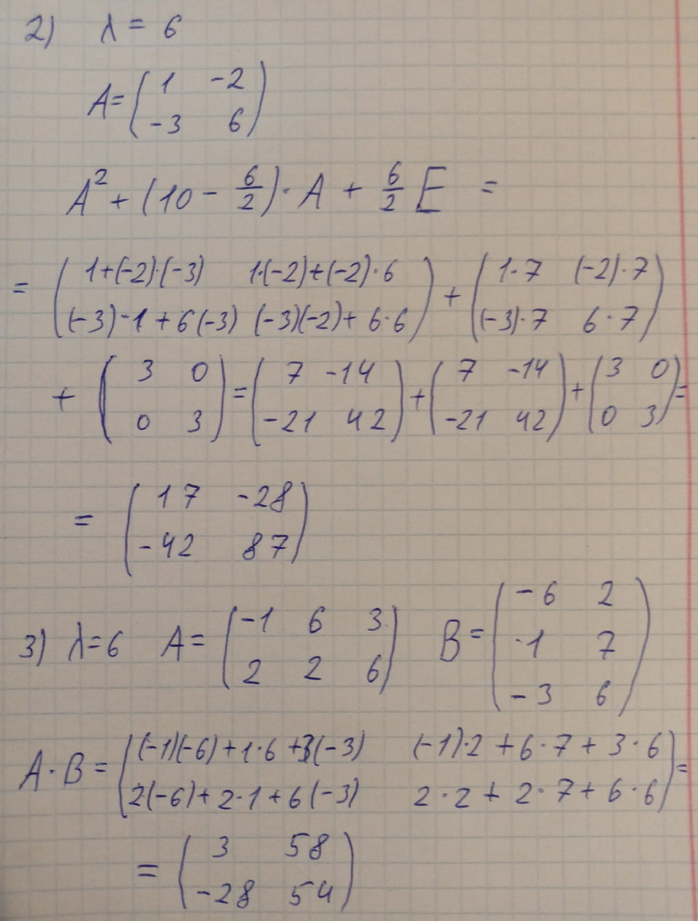
ПИСЬМЕННОЕ ЗАДАНИЕ











КОД ПРОГРАММЫ

#include <iostream>

using namespace std;

int find\_number(char\* els, int n, char x) {

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

if (els[i] == x)

return i;

}

}

bool assoc(char\*\* v, char\* els, int n) {

int checker = 0;

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

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

if (v[find\_number(els, n, v[i][j])][k] != v[i][find\_number(els, n, v[j][k])])

return false;

}

}

}

return true;

}

bool comm(char\*\* v, int n) {

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

for (int j = 0; j < n; j++) {

if (v[i][j] != v[j][i])

return false;

}

}

return true;

}

bool idemp(char\*\* v, char\* els, int n) {

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

if (v[i][i] != els[i])

return false;

}

return true;

}

bool distr(char\*\* v1, char\*\* v2, char\* els, int n) {

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

for (int j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

if (v2[find\_number(els, n, v1[i][j])][k] != v1[find\_number(els, n, v2[i][k])][find\_number(els, n, v2[j][k])] ||

v1[find\_number(els, n, v2[i][j])][k] != v2[find\_number(els, n, v1[i][k])][find\_number(els, n, v1[j][k])])

return false;

}

}

}

return true;

}

void my\_union(int\*\* v1, int\*\* v2, 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] = v1[i][j];

}

}

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

for (int j = 0; j < n; j++) {

if (v2[i][j])

res[i][j] = 1;

}

}

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

for (int j = 0; j < n; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

void my\_intersection(int\*\* v1, int\*\* v2, 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] = v1[i][j] \* v2[i][j];

}

}

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

for (int j = 0; j < n; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

void my\_reverse(int\*\* v, 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] = v[j][i];

}

}

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

for (int j = 0; j < n; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

void my\_addition(int\*\* v, 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++) {

if (v[i][j])

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;

}

}

void my\_multiplication(int\*\* v1, int\*\* v2, 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] = 0;

}

}

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

for (int j = 0; j < n; j++) {

int count = 0;

for (int k = 0; k < n; k++) {

count += v1[i][k] \* v2[k][j];

}

if (count > 0)

res[i][j] = 1;

}

}

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

for (int j = 0; j < n; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

int\*\* modul\_matr(int\*\* v1, int n, int m, int num) {

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] = v1[i][j] % num;

if (res[i][j] < 0)

res[i][j] += num;

}

}

return res;

}

void my\_multiplication\_matr(int\*\* v1, int\*\* v2, int n, int l, int m, int num) {

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 < m; j++) {

for (int k = 0; k < l; k++) {

res[i][j] += v1[i][k] \* v2[k][j];

}

}

}

res = modul\_matr(res, n, m, num);

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

for (int j = 0; j < m; j++) {

cout << res[i][j] << ' ';

}

cout << endl;

}

}

void my\_addition\_matr(int\*\* v1, int\*\* v2, int n, int m, int num) {

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] = v1[i][j] + v2[i][j];

}

}

res = modul\_matr(res, n, m, num);

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

for (int j = 0; j < m; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

void my\_transp\_matr(int\*\* v, int n, int m, int num) {

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] = v[j][i];

}

}

res = modul\_matr(res, m, n, num);

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++)

cout << res[i][j] << ' ';

cout << endl;

}

}

int\*\* minus\_stroka\_stolbec(int\*\* v, int n, int str, int stolb) {

int my\_i = 0, my\_j = 0;

int\*\* res;

res = new int\* [n - 1];

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

if (i != str) {

res[my\_i] = new int[n - 1];

for (int j = 0; j < n; j++) {

if (j != stolb) {

res[my\_i][my\_j] = v[i][j];

my\_j++;

}

}

my\_i++;

}

my\_j = 0;

}

return res;

}

int Det(int\*\* v, int n) {

int temp = 0;

int k = 1;

if (n == 1)

temp = v[0][0];

else if (n == 2)

temp = v[0][0] \* v[1][1] - v[1][0] \* v[0][1];

else {

int\*\* temp\_matr = new int\* [n - 1];

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

for (int j = 0; j < n - 1; j++)

temp\_matr[j] = new int[n - 1];

temp\_matr = minus\_stroka\_stolbec(v, n, 0, i);

temp += k \* v[0][i] \* Det(temp\_matr, n - 1);

k = -k;

}

}

return temp;

}

int\*\* my\_transp\_matr\_for\_obr(int\*\* v, int n, int num) {

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] = v[j][i];

}

}

return res;

}

void my\_obr\_matr(int\*\* v, int n, int num) {

double\*\* transp;

int det = Det(v, n);

det = det % num;

if (det == 0)

cout << endl << "Определитель равен нулю. Матрица вырожденная." << endl << endl;

else {

if (det < 0)

det += num;

int obr\_det = 0;

for (int i = 1; i <= num; i++)

if ((i \* det) % num == 1) {

obr\_det = i;

break;

}

cout << endl << "1/|A| = " << obr\_det << endl << endl;

int\*\* res1;

res1 = new int\* [n];

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

res1[i] = new int[n];

for (int j = 0; j < n; j++) {

res1[i][j] = Det(minus\_stroka\_stolbec(v, n, i, j), n - 1);

if ((i + j) % 2 == 1)

res1[i][j] \*= -1;

}

}

res1 = modul\_matr(res1, n, n, num);

cout << "Матрица алгебраических дополнений:" << endl;

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

for (int j = 0; j < n; j++)

cout << res1[i][j] << ' ';

cout << endl;

}

cout << endl;

int\*\* res2 = my\_transp\_matr\_for\_obr(res1, n, num);

res2 = modul\_matr(res2, n, n, num);

cout << "Транспонированная матрица алгебраических дополнений:" << endl;

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

for (int j = 0; j < n; j++) {

cout << res2[i][j] << ' ';

res2[i][j] \*= obr\_det;

}

cout << endl;

}

cout << endl;

res2 = modul\_matr(res2, n, n, num);

cout << "Обратная матрица:" << endl;

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

for (int j = 0; j < n; j++) {

cout << res2[i][j] << ' ';

}

cout << endl;

}

cout << endl;

cout << "Проверка:" << endl;

my\_multiplication\_matr(v, res2, n, n, n, num);

cout << endl;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

int w = 999;

while (w > 0) {

cout << endl;

cout << "1 - алгоритмы проверки свойств операций" << endl;

cout << "2 - алгоритмы выполнения операции над бинарными отношениями" << endl;

cout << "3 - алгоритмы выполнения операций над матрицами" << endl;

cout << "0 - выход" << endl;

cin >> w;

if (w == 1) {

int n;

cout << "Введите количество элементов. n = ";

cin >> n;

char\* els = new char[n];

cout << "Введите элементы: ";

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

cin >> els[i];

char\*\* v1;

v1 = new char\* [n];

cout << endl;

cout << "Введите таблицу Кэли для пересечения:" << endl;

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

v1[i] = new char[n];

for (int j = 0; j < n; j++) {

cin >> v1[i][j];

}

}

char\*\* v2;

v2 = new char\* [n];

cout << "Введите таблицу Кэли для объединения:" << endl;

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

v2[i] = new char[n];

for (int j = 0; j < n; j++) {

cin >> v2[i][j];

}

}

if (assoc(v1, els, n)) cout << "Операция пересечения обладает свойством ассоциативности" << endl;

else cout << "Операция пересечения НЕ обладает свойством ассоциативности" << endl;

if (assoc(v2, els, n)) cout << "Операция объединения обладает свойством ассоциативности" << endl;

else cout << "Операция объединения НЕ обладает свойством ассоциативности" << endl;

cout << endl;

if (comm(v1, n)) cout << "Операция пересечения обладает свойством коммутативности" << endl;

else cout << "Операция пересечения НЕ обладает свойством коммутативности" << endl;

if (comm(v2, n)) cout << "Операция объединения обладает свойством коммутативности" << endl;

else cout << "Операция объединения НЕ обладает свойством коммутативности" << endl;

cout << endl;

if (idemp(v1, els, n)) cout << "Операция пересечения обладает свойством идемпотентности" << endl;

else cout << "Операция пересечения НЕ обладает свойством идемпотентности" << endl;

if (idemp(v2, els, n)) cout << "Операция объединения обладает свойством идемпотентности" << endl;

else cout << "Операция объединения НЕ обладает свойством идемпотентности" << endl;

cout << endl;

if (distr(v1, v2, els, n)) cout << "Решётка является дистрибутивной" << endl;

else cout << "Решётка НЕ является дистрибутивной" << endl;

cout << endl;

}

if (w == 2) {

int ch = 0;

cout << "1 - объединение" << endl;

cout << "2 - пересечение" << endl;

cout << "3 - дополнение" << endl;

cout << "4 - произведение" << endl;

cout << "5 - нахождение обратного" << endl;

cin >> ch;

if (ch == 1 || ch == 2 || ch == 4) {

int n;

cout << "Введите количество элементов. n = ";

cin >> n;

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу бинарного отношения. А = " << endl;

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

v1[i] = new int[n];

for (int j = 0; j < n; j++) {

cin >> v1[i][j];

}

}

int\*\* v2;

v2 = new int\* [n];

cout << "Введите матрицу бинарного отношения. B = " << endl;

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

v2[i] = new int[n];

for (int j = 0; j < n; j++) {

cin >> v2[i][j];

}

}

cout << endl;

if (ch == 1)

my\_union(v1, v2, n);

if (ch == 2)

my\_intersection(v1, v2, n);

if (ch == 4)

my\_multiplication(v1, v2, n);

}

else {

int n;

cout << "Введите количество элементов. n = ";

cin >> n;

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу бинарного отношения. А = " << endl;

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

v1[i] = new int[n];

for (int j = 0; j < n; j++) {

cin >> v1[i][j];

}

}

cout << endl;

if (ch == 3)

my\_addition(v1, n);

if (ch == 5)

my\_reverse(v1, n);

}

cout << endl;

}

if (w == 3) {

int ch = 0;

cout << "1 - сложение" << endl;

cout << "2 - умножение" << endl;

cout << "3 - транспонирование" << endl;

cout << "4 - обращение" << endl;

cin >> ch;

if (ch == 1) {

int n, m;

cout << "Введите количество строк. n = ";

cin >> n;

cout << "Введите количество столбцов. m = ";

cin >> m;

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу. А = " << endl;

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

v1[i] = new int[m];

for (int j = 0; j < m; j++) {

cin >> v1[i][j];

}

}

int\*\* v2;

v2 = new int\* [n];

cout << "Введите матрицу. B = " << endl;

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

v2[i] = new int[m];

for (int j = 0; j < m; j++) {

cin >> v2[i][j];

}

}

cout << endl;

double num;

cout << "Введите число, по модулю которого будут выполняться операции: ";

cin >> num;

cout << endl;

my\_addition\_matr(v1, v2, n, m, num);

}

if (ch == 2) {

int n, k1, k2, m;

cout << "Введите количество строк первой матрицы. n = ";

cin >> n;

cout << "Введите количество столбцов первой матрицы. k1 = ";

cin >> k1;

cout << "Введите количество строк второй матрицы. k2 = ";

cin >> k2;

cout << "Введите количество столбцов второй матрицы. m = ";

cin >> m;

if (k1 == k2) {

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу. А = " << endl;

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

v1[i] = new int[k1];

for (int j = 0; j < k1; j++) {

cin >> v1[i][j];

}

}

int\*\* v2;

v2 = new int\* [k1];

cout << "Введите матрицу. B = " << endl;

for (int i = 0; i < k1; i++) {

v2[i] = new int[m];

for (int j = 0; j < m; j++) {

cin >> v2[i][j];

}

}

int num;

cout << "Введите число, по модулю которого будут выполняться операции: ";

cin >> num;

cout << endl;

my\_multiplication\_matr(v1, v2, n, k1, m, num);

}

else

cout << "Ошибка! Количество столбцов А должно быть равно количеству строк B" << endl;

}

if (ch == 3) {

int n, m;

cout << "Введите количество строк. n = ";

cin >> n;

cout << "Введите количество столбцов. m = ";

cin >> m;

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу. А = " << endl;

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

v1[i] = new int[m];

for (int j = 0; j < m; j++) {

cin >> v1[i][j];

}

}

int num;

cout << "Введите число, по модулю которого будут выполняться операции: ";

cin >> num;

cout << endl;

my\_transp\_matr(v1, n, m, num);

}

if (ch == 4) {

int n;

cout << "Введите количество строк. n = ";

cin >> n;

int\*\* v1;

v1 = new int\* [n];

cout << "Введите матрицу. А = " << endl;

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

v1[i] = new int[n];

for (int j = 0; j < n; j++) {

cin >> v1[i][j];

}

}

int num;

cout << "Введите число, по модулю которого будут выполняться операции: ";

cin >> num;

cout << endl;

my\_obr\_matr(v1, n, num);

}

cout << endl;

}

}

return 0;

}

ЗАКЛЮЧЕНИЕ

В ходе данной работы были изучены и реализованы алгоритмы проверки свойств операций : ассоциативность, коммутативность, идемпотентность, обратимость, дистрибутивность, а также алгоритмы выполнения операций над бинарными отношениями и матрицами.