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

Алексеева Александра Александровича

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

Саратов 2022

1 Цель работы и порядок её выполнения­

Цель работы – изучение основных свойств бинарных отношений и операций замыкания бинарных отношений.

Порядок выполненных работы:

1. Рассмотреть понятие алгебраической операции и классификацию свойств [1-3], [6]. Разобрать алгоритмы проверки свойств операции: ассоциативность, коммутативность, идемпотентность и дистрибутивность.

2. Рассмотреть основные операции над бинарными отношениями [1]. Разобрать алгоритмы выполнения операции над бинарными отношениями.

3. Рассмотреть основные операции над матрицами [1-3], [6]. Разобрать алгоритмы выполнения операций над матрицами

2 Теория

2.1 Понятие алгебраической операции и классификация свойств операций

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

*Определение*. Отображение *f*: *An* → *А* называется алгебраической *n*-арной операцией или просто алгебраической операцией на множестве *А*. При этом *n* называется порядком или арностью алгебраической операции *f*.

В случае *n* = 0 по определению *A*0 = {} и, значит, 0-арная операция *f* просто выделяет в множестве *А* некоторый элемент *f*() *A.* В случае *n* = 1 операция *f* называется также унарной и в случае *n* = 2 – бинарной. Для унарных и бинарных операций, как правило, используется специальная символическая запись.

Классификация свойств операций:

1) (a ∧ b) ∧ c = a ∧ (b ∧ c), (a ∨ b) ∨ c = a ∨ (b ∨ c) – ассоциативность операций ∧, ∨.

2) a ∧ b = b ∧ a, a ∨ b = b ∨ a – коммутативность операций ∧, ∨.

3) a ∧ a = a, a ∨ a = a – идемпотентность операций ∧, ∨.

4) (a ∧ b) ∨ c = (a ∨ c) ∧ (b ∨ c), (a ∨ b) ∧ c = (a ∧ c) ∨ (b ∧ c) – дистрибутивность операций ∧, ∨.

2.2 Алгоритм поиска номера элемента в списке элементов

*Вход*. Элемент *word*, Список элементов *el* и его размер *N*.

*Выход*. Номер, под которым стоит элемент *word* в списке *el*.

Шаг 1. Цикл по *i* от 1 до *N*. Если *el*[*i*] = *word*, то выводим *i*.

2.3 Алгоритм проверки свойства ассоциативности

*Вход*. Таблица Кэли mas, размер таблицы N и список элементов el.

*Выход*. «Операция обладает свойством ассоциативности» или «Операция не обладает свойством ассоциативности».

Шаг 1. Цикл по *a* от 1 до *N*. Для каждого a цикл по *b* от 1 до *N*. Для каждого *b* цикл по *c* от 1 до *N*. Для каждого *c* применяем алгоритм поиска номера элемента в списке элементов от *mas*[*a*][*b*], *N*, *el* и от *mas*[*b*][*c*], *N*, *el* и сохраняем полученные результаты в *ab* и *bc* соответственно.

Шаг 2. Если для любого c *mas*[*ab*][*c*] не равно *mas*[*a*][*bc*], то возвращаем «Операция не обладает свойством ассоциативности».

Шаг 3. Иначе «Операция обладает свойством ассоциативности»

2.4 Алгоритм проверки свойства коммутативности

*Вход*. Таблица Кэли mas и размер таблицы N.

*Выход*. «Операция обладает свойством коммутативности» или «Операция не обладает свойством коммутативности».

Шаг 1. Цикл по *a* от 1 до *N*. Для каждого a цикл по *b* от 1 до *N*. Для каждого *b* если *mas*[*a*][*b*] не равно *mas*[*b*][*a*], то возвращаем «Операция не обладает свойством коммутативности».

Шаг 2. Иначе «Операция обладает свойством коммутативности».

2.5 Алгоритм проверки свойства идемпотентности

*Вход*. Таблица Кэли *mas*, размер таблицы *N* и список элементов *el*.

*Выход*. «Операция обладает свойством идемпотентности» или «Операция не обладает свойством идемпотентности».

Шаг 1. Цикл по *a* от 1 до *N*. Для каждого a если *mas*[*a*][*a*] не равно *el*[*a*], возвращаем «Операция не обладает свойством идемпотентности».

Шаг 2. Иначе «Операция обладает свойством идемпотентности».

2.6 Алгоритм проверки свойства дистрибутивности

*Вход*. Таблица Кэли пересечения *pere*, таблица Кэли объединения *obed*, размер таблиц *N* и список элементов *el*.

*Выход*. «Решётка обладает свойством дистрибутивности» или «Решётка не обладает свойством дистрибутивности».

Шаг 1. Цикл по *a* от 1 до *N*. Для каждого *a* цикл по *b* от 1 до *N*. Для каждого *b* цикл по *c* от 1 до *N*. Для каждого *c* применяем алгоритм поиска элемента в списке элементов от (*pere*[*a*][*b*], *el*, *N*), (*obed*[*a*][*c*], *el*, *N*), (*obed*[*b*][*c*], *el*, *N*), (*obed*[*a*][*b*], *el*, *N*), (pe*re*[*a*][*c*], *el*, *N*), (*pere*[*b*][*c*], *el*, *N*) и сохраняем результаты в *pereAB*, *obedAC*, *obedBC*, *obedAB*, *pereAC* и *pereBC* соответственно.

Шаг 2. Для любого *c* если *obed*[*pereAB*][*c*] не равно *pere*[*obedAC*][*obedBC*] или *pere*[*obedAB*][*c*] не равно *obed*[*pereAC*][*pereBC*], возвращаем «Решётка не обладает свойством дистрибутивности».

Шаг 3. Иначе «Решётка обладает свойством дистрибутивности».

2.7 Основные операции над бинарными отношениями

1. *Объединением* бинарных отношений ρ ⊂ A × B и δ ⊂ A × B называется отношение ρδ ⊂ A × B, которое определяется по формуле

*ρδ* = {(a, b): (a, b) ∈ *ρ* ∨ (a, b) ∈ *δ*}

2. *Пересечением* бинарных отношений ρ ⊂ A × B и δ ⊂ A × B называется отношение *ρδ* ⊂ A × B, которое определяется по формуле

*ρδ* = {(a, b): (a, b) ∈ *ρ* ∧ (a, b) ∈ *δ*}

3. *Дополнением* бинарного отношения *ρ* ⊂ A × B называется отношение *ρ'* ⊂ A × B, которое определяется по формуле

*ρ'* = {(a, b): (a, b) ∉ *ρ*}

4. *Композицией* бинарных отношений ρ ⊂ A × B и δ ⊂ B × С называется отношение *ρδ* ⊂ A × С, которое определяется по формуле

*ρδ* = {(a, b): (a, b) ∈ *ρ* ∧ (b, c) ∈ *δ* для некоторого b ∈ B}

5. *Обратным* бинарным отношением для *ρ* ⊂ A × B называется отношение *ρ-1* ⊂ B × A, которое определяется по формуле

*ρ'* = {(b, a): (a, b) ∉ *ρ*}

2.8 Алгоритм объединения бинарных отношений

*Вход*. Размер матриц бин. отношений *N*, матрица 1-го бин. отношения *firstMatrix* и матрица 2-го бин. отношения *secondMatrix*.

*Выход*. Матрица объединения бин. отношений *res*.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*. Для каждого *j* сохраняем результат вычисления (*firstMatrix*[*i*][*j*] + *secondMatrix*[*i*][*j*] + 1) / 2 в *res*[*i*][*j*].

Шаг 2. Выводим *res*.

2.9 Алгоритм пересечения бинарных отношений

*Вход*. Размер матриц бин. отношений *N*, матрица 1-го бин. отношения *firstMatrix* и матрица 2-го бин. отношения *secondMatrix*.

*Выход*. Матрица объединения бин. отношений *res*.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*. Для каждого *j* сохраняем результат вычисления *firstMatrix*[*i*][*j*] · *secondMatrix*[*i*][*j*] в *res*[*i*][*j*].

Шаг 2. Выводим *res*.

2.10 Алгоритм дополнения бинарного отношения

*Вход*. Размер матриц бин. отношений *N* и матрица бин. отношения m*atrix*.

*Выход*. Матрица дополнения бин. отношений *res*.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*. Для каждого *j* сохраняем остаток от деления (*matrix*[*i*][*j*] *+* 1) / 2 в *res*[*i*][*j*].

Шаг 2. Выводим *res*.

2.11 Алгоритм композиции бинарных отношений

*Вход*. Размер матриц бин. отношений *N*, матрица 1-го бин. отношения *firstMatrix* и матрица 2-го бин. отношения *secondMatrix*.

*Выход*. Матрица объединения бин. отношений *res*.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*. Для каждого *j* создаём переменную *help* = 0 и запускаем цикл по *a* от 1 до *N*. Для каждого *а* увеличиваем *help* на *firstMatrix*[*i*][*a*] · *secondMatrix*[*a*][*j*].

Шаг 2. После цикла по *а* если *help* > 0, сохраняем в *res*[*i*][*j*] значение 1. Иначе сохраняем значение 0.

Шаг 3. Выводим *res*.

2.12 Алгоритм нахождения обратного бинарного отношения

*Вход*. Размер матриц бин. отношений *N* и матрица бин. отношения m*atrix*.

*Выход*. Матрица дополнения бин. отношений *res*.

Шаг 1. Цикл по *i* от 1 до *N*. Для каждого *i* цикл по *j* от 1 до *N*. Для каждого *j* сохраняем значение *matrix*[*j*][*i*] в *res*[*i*][*j*].

Шаг 2. Выводим *res*.

2.13 Основные операции матриц над конечным полем

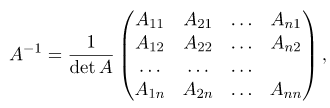
1. *Сложением* двух матриц A = (aij) и B = (bij) одинакового порядка называют матрицу C = (cij) такого же порядка, элементы которой равны сумме соответствующих элементов матриц A и B, то есть cij = aij + bij.

2. *Умножением* матриц A = (aij) размерности (k × n) и B = (bij) размерности (n × p) называют матрицу C = (cij) размерности (k × p), элементы которой находятся по правилу: элемент cij равен сумме попарных произведений элементов i-той строки матрицы A и j-того столбца матрицы B:

c*ij* = a*i*1b1*j* + a*i*2b2*j* + … + a*in*b*nj* = .

3. *Транспонированием* матрицы A = (a*ij*) размерности (m × n) называется матрица A*T* = (a*ji*) размерности (m × n), в которой строки начальной матрицы заменены столбцами с соответствующими номерами.

4. *Обращением* невырожденной квадратной матрицы A (detA ≠ 0) называют обратную матрицу A-1, такая что A · A-1 = A-1 · A = E. Найти обратную матрицу можно по формуле



где A*ij*– алгебраические дополнения элементов a*ij* матрицы А.

2.14 Алгоритм сложения матриц

*Вход*. 1-я матрица *firstMatrix*, 2-я матрица *secondMatrix*, количество их строк *row* и количество их столбцов *column*, порядок поля *field*.

*Выход*. Матрица сложения *res*.

Шаг 1. Цикл по *i* от 1 до *row*. Для каждого *i* цикл по *j* от 1 до *column*. Для каждого *j* остаток от деления (*firstMatrix*[*i*][*j*] + *secondMatrix*[*i*][*j*]) на *field* сохраняем в *res*[*i*][*j*].

Шаг 2. Выводим *re*s.

2.15 Алгоритм умножения матриц

*Вход*. 1-я матрица *firstMatrix* и 2-я матрица *secondMatrix*, порядок поля *field*.

*Выход*. Матрица сложения *res* или «Умножение данных матриц невозможно».

Шаг 1. Если количество строк 1-й матрицы не равно количеству столбцов 2-й матрицы, возвращаем «Умножение данных матриц невозможно».

Шаг 2. Сохраняем количество строк 1-й матрицы в *row1* и запускаем цикл по *i* от 1 до *row1*. Для каждого *i* сохраняем количество столбцов 2-й матрицы в *column2* и запускаем цикл по *j* от 1 до *column2*. Для каждого *j* создаём переменную *help* = 0, сохраняем количество строк 2-й матрицы в *row2* и запускаем цикл по *a* от 1 до *row2*. Для каждого *a* увеличиваем *help* на *firstMatrix*[*i*][*a*] · *secondMatrix*[*a*][*i*]. После цикла по *а* сохраняем остаток от деления *help* на *field* в *res*[*i*][*j*].

Шаг 3. Выводим *re*s.

2.16 Алгоритм транспонирования матрицы

*Вход*. Матрица *matrix*, количество её строк *row* и количество её столбцов *column*.

*Выход*. Транспонированная матрица *res*.

Шаг 1. Цикл по *i* от 1 до *column*. Для каждого *i* цикл по *j* от 1 до *row*. Сохраняем значение *matrix*[*j*][*i*] в *res*[*i*][*j*].

Шаг 2. Выводим *res*.

2.17 Алгоритм вычисления определителя матрицы

*Вход*. Матрица *matrix*, её размер *N* и множитель *count*.

*Выход*. Определитель матрицы *res*.

Шаг 1. Если *N* = 2, возвращаем определитель матрицы 2-го порядка *matrix*[1][1] · *matrix*[2][2] – *matrix*[1][2] · *matrix*[2][1]. Иначе если N = 3, возвращаем определитель матрицы 3-го порядка (*matrix*[1][1] · *matrix*[2][2] · *matrix*[3][3] + *matrix*[1][2] · *matrix*[2][3] · *matrix*[3][0] + *matrix*[2][1] · *matrix*[3][2] · *matrix*[1][3]) – (*matrix*[1][3] · *matrix*[2][2] · *matrix*[3][1] + *matrix*[1][2] · *matrix*[2][1] · *matrix*[3][3] + *matrix*[1][1] · *matrix*[2][3] · *matrix*[3][2]).

Шаг 2. Иначе цикл по *i* от 1 до *N*. Для каждого *i* создаём *res* = 0, домножаем *count* на -1, создаём список списков *detA* и запускаем цикл по *j* от 1 до *N*. Для каждого *j* создаём список *row* и запускаем цикл по *k* от 1 до *N*. Для каждого *k* если *j* не равно 0 и *k* не равно *i*, кладём в список *row* значение *matrix*[*j*][*k*]. После цикла по *k* если список *row* не пустой, кладём его в *detA*.

Шаг 3. После цикла по *j* увеличиваем *res* на *count* · *matrix*[0][*i*] · на значение алгоритма вычисления определителя матрицы от (*detA*, *N* - 1, -1).

Шаг 4. После цикла по *i* возвращаем *res*.

2.18 Алгоритм нахождения обратной матрицы

*Вход*. Матрица *matrix* и её размер *N*, порядок поля *field*.

*Выход*. Обратная матрица *res* или «Определитель матрицы равен 0 и обратной матрицы не существует».

Шаг 1. Применяем алгоритм вычисления определителя матрицы от (*matrix*, *N*, -1) и результат сохраняем в *determinantMatrix*. Пока *determinantMatrix* < 0, выполняем *determinantMatrix* = *determinantMatrix* + *field*. После цикла *determinantMatrix* = остатку от деления *determinantMatrix* на *field*. Если *determinantMatrix* = 0, возвращаем «Определитель матрицы равен 0 и обратной матрицы не существует»

Шаг 2. Инициализируем *inv\_det*. Цикл по *i* от 1 до *field*. Для каждого *i* если остаток от деления (*i* · *determinantMatrix*) на *field* = 1, присваиваем *inv\_det* = *i* и выходим из цикла.

Шаг 3. Создаём список списков *res*.

Шаг 4. Цикл по *i* от 1 до *N*. Для каждого *i* создаём список *help* и запускаем цикл по *j* от 1 до *N*. Для каждого *j* создаём *number* = 1. Если остаток от деления (*i* + *j*) / 2 = 1, присваиваем *number* = -1. Затем создаём список списков *det* и запускаем цикл по *a* от 1 до *N*. Для каждого *a* создаём список *row* и запускаем цикл по *b* от 1 до *N*. Для каждого *b* если *a* не равно *i* и *b* не равно *j*, кладём в список *row* значение *matrix*[*a*][*b*]. После цикла по *b* если список *row* не пустой, кладём *row* в *det*. После цикла по *a* сохраняем в *det\_dop* = *number* · алгоритм вычисления определителя матрицы от (*det*, *det*.*size*(), -1). Пока *det\_dop* < 0, *det\_dop* = *det\_dop* + *field*. Затем *det\_dop* = остаток от деления *det\_dop* на *field*. В *help* кладём остаток от деления (*det\_dop* · *inv\_det*) на *field*. После цикла по *j* кладём в *res* значение *help*.

Шаг 5. После цикла по *i* создаём список списков *resT* и заново запускаем цикл по *i* от 1 до *N*. Для каждого *i* запускаем цикл по *j* от 1 до *N*. Для каждого *j* значение *res*[*j*][*i*] присваиваем *resT*[*i*][*j*].

Шаг 6. После цикла по *i* выводим *resТ*.

3 Результаты работы

3.1 Оценка временной сложности алгоритмов

Поиск номера элемента в списке элементов – О(N);

Проверка свойства ассоциативности – O(N3);

Проверка свойства коммутативности – O(N2);

Проверка свойства идемпотентности – O(N);

Проверка свойства дистрибутивности – O(N3);

Объединение бинарных отношений – O(N2);

Пересечение бинарных отношений – O(N2);

Дополнение бинарного отношения – O(N2);

Композиция бинарных отношений – O(N3);

Нахождение обратного бинарного отношения – O(N2);

Сложение матриц – O(N · M), где N – количество строк матриц, M – количество столбцов матриц;

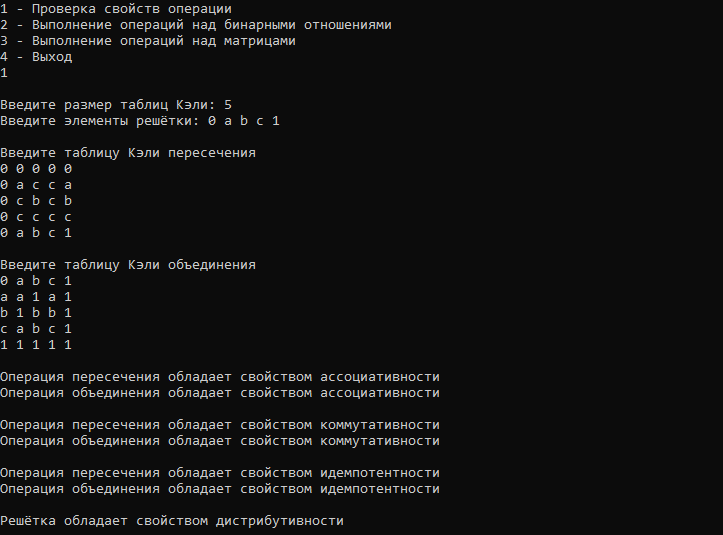
Умножение матриц – O(N · M · L), где N – количество строк 1-й матрицы, M – количество столбцов 2-й матрицы, L – количество столбцов 1-матрицы (или строк 2-й матрицы);

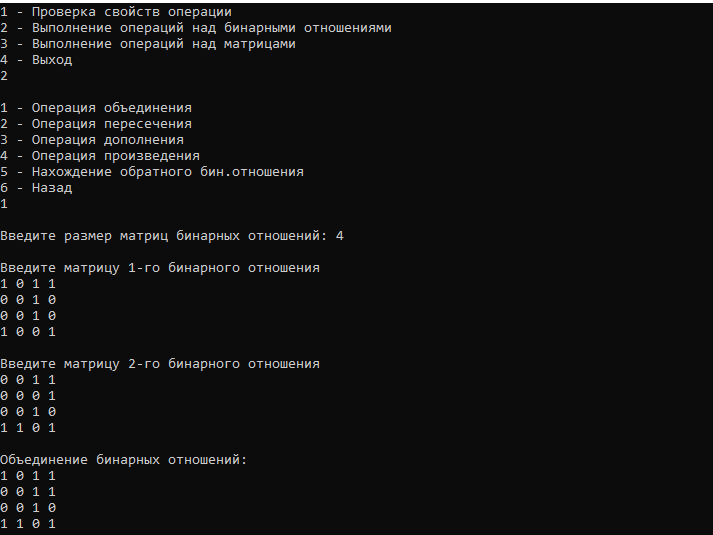
Транспонирование матрицы – O(N · M), где N – количество строк матрицы, M – количество столбцов матрицы;

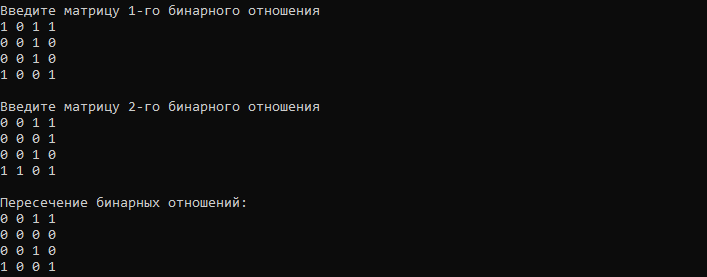
Вычисление определителя матрицы – O(N! – 3!) ~ O(N!);

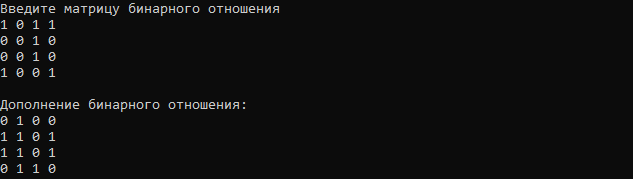
Нахождение обратной матрицы – O(N3 · M!), N – размер матрицы, M – размер вычисленного минора;

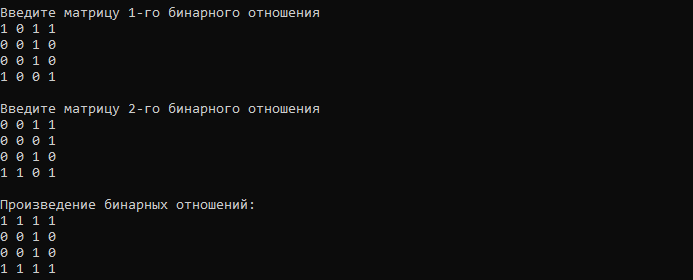
3.2 Результаты тестирования программы

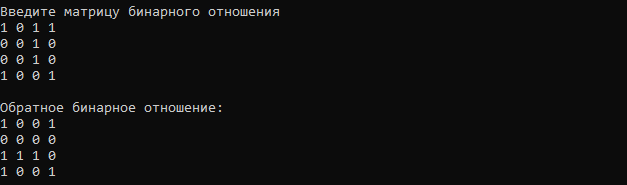


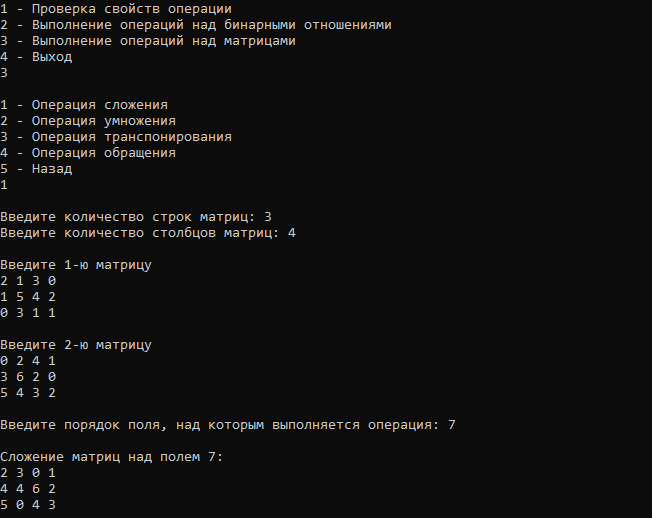


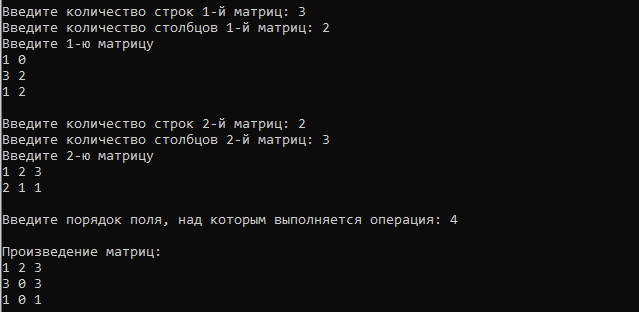


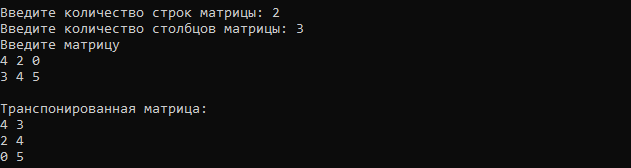


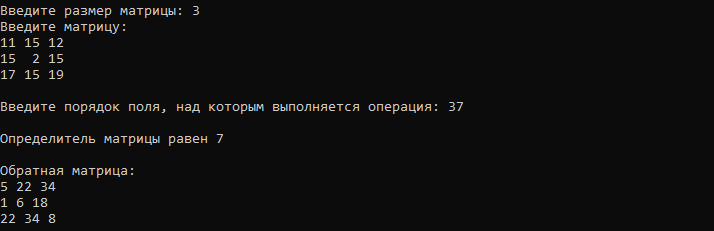












3.3 Код программы

#include "iostream"

#include "vector"

using namespace std;

int findNumber(char word, char\* el, int N) { //Поиск номера буквы в элементах решётки

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

if (el[i] == word)

return i;

}

bool associativity(char\*\* mas, int N, char\* el) { //Проверка свойства ассоциативности

for (int a = 0; a < N; a++)

for (int b = a; b < N; b++)

for (int c = 0; c < N; c++)

if (mas[findNumber(mas[a][b], el, N)][c] != mas[a][findNumber(mas[b][c], el, N)])

return false;

return true;

}

bool commutativity(char\*\* mas, int N) { //Проверка свойства коммутативности

for (int a = 0; a < N; a++)

for (int b = 0; b < N; b++)

if (mas[a][b] != mas[b][a])

return false;

return true;

}

bool idempotency(char\*\* mas, int N, char\* el) { //Проверка свойства идемпотентности

for (int a = 0; a < N; a++)

if (mas[a][a] != el[a])

return false;

return true;

}

bool distributivity(char\*\* pere, char\*\* obed, int N, char\* el) { //Проверка свойства дистрибутивности

for (int a = 0; a < N; a++)

for (int b = 0; b < N; b++)

for (int c = 0; c < N; c++)

if (obed[findNumber(pere[a][b], el, N)][c] != pere[findNumber(obed[a][c], el, N)][findNumber(obed[b][c], el, N)] ||

pere[findNumber(obed[a][b], el, N)][c] != obed[findNumber(pere[a][c], el, N)][findNumber(pere[b][c], el, N)])

return false;

return true;

}

void checkFeatures() { //Проверка свойств операции

int N;

cout << "\nВведите размер таблиц Кэли: ";

cin >> N;

cout << "Введите элементы решётки: ";

char\* el = new char[N];

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

cin >> el[i];

cout << "\nВведите таблицу Кэли пересечения \n";

char\*\* peresechenie = new char\* [N];

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

peresechenie[i] = new char[N];

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

cin >> peresechenie[i][j];

}

cout << endl;

cout << "Введите таблицу Кэли объединения \n";

char\*\* obedinenie = new char\* [N];

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

obedinenie[i] = new char[N];

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

cin >> obedinenie[i][j];

}

cout << endl;

if (associativity(peresechenie, N, el)) cout << "Операция пересечения обладает свойством ассоциативности \n";

else cout << "Операция пересечения не обладает свойством ассоциативности \n";

if (associativity(obedinenie, N, el)) cout << "Операция объединения обладает свойством ассоциативности \n\n";

else cout << "Операция объединения не обладает свойством ассоциативности \n\n";

if (commutativity(peresechenie, N)) cout << "Операция пересечения обладает свойством коммутативности \n";

else cout << "Операция пересечения не обладает свойством коммутативности \n";

if (commutativity(obedinenie, N)) cout << "Операция объединения обладает свойством коммутативности \n\n";

else cout << "Операция объединения не обладает свойством коммутативности \n\n";

if (idempotency(peresechenie, N, el)) cout << "Операция пересечения обладает свойством идемпотентности \n";

else cout << "Операция пересечения не обладает свойством идемпотентности \n";

if (idempotency(obedinenie, N, el)) cout << "Операция объединения обладает свойством идемпотентности \n\n";

else cout << "Операция объединения не обладает свойством идемпотентности \n\n";

if (distributivity(peresechenie, obedinenie, N, el)) cout << "Решётка обладает свойством дистрибутивности \n";

else cout << "Решётка не обладает свойством дистрибутивности \n\n";

}

void binObed() { //Объединение бинарных отношений

int N;

cout << "\nВведите размер матриц бинарных отношений: ";

cin >> N;

cout << "\nВведите матрицу 1-го бинарного отношения \n";

int\*\* firstMatrix = new int\* [N];

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

firstMatrix[i] = new int[N];

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

cin >> firstMatrix[i][j];

}

cout << "\nВведите матрицу 2-го бинарного отношения \n";

int\*\* secondMatrix = new int\* [N];

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

secondMatrix[i] = new int[N];

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

cin >> secondMatrix[i][j];

}

cout << "\nОбъединение бинарных отношений: \n";

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

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

cout << (firstMatrix[i][j] + secondMatrix[i][j] + 1) / 2 << " ";

cout << endl;

}

}

void binPere() { //Пересечение бинарных отношений

int N;

cout << "\nВведите размер матриц бинарных отношений: ";

cin >> N;

cout << "\nВведите матрицу 1-го бинарного отношения \n";

int\*\* firstMatrix = new int\* [N];

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

firstMatrix[i] = new int[N];

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

cin >> firstMatrix[i][j];

}

cout << "\nВведите матрицу 2-го бинарного отношения \n";

int\*\* secondMatrix = new int\* [N];

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

secondMatrix[i] = new int[N];

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

cin >> secondMatrix[i][j];

}

cout << "\nПересечение бинарных отношений: \n";

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

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

cout << firstMatrix[i][j] \* secondMatrix[i][j] << " ";

cout << endl;

}

}

void binDop() { //Дополнение бинарного отношнения

int N;

cout << "\nВведите размер матрицы бинарного отношения: ";

cin >> N;

cout << "\nВведите матрицу бинарного отношения \n";

int\*\* matrix = new int\* [N];

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

matrix[i] = new int[N];

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

cin >> matrix[i][j];

}

cout << "\nДополнение бинарного отношения: \n";

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

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

cout << (matrix[i][j] + 1) % 2 << " ";

cout << endl;

}

}

void binMult() { //Произведение бинарных отношений

int N;

cout << "\nВведите размер матриц бинарных отношений: ";

cin >> N;

cout << "\nВведите матрицу 1-го бинарного отношения \n";

int\*\* firstMatrix = new int\* [N];

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

firstMatrix[i] = new int[N];

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

cin >> firstMatrix[i][j];

}

cout << "\nВведите матрицу 2-го бинарного отношения \n";

int\*\* secondMatrix = new int\* [N];

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

secondMatrix[i] = new int[N];

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

cin >> secondMatrix[i][j];

}

cout << "\nПроизведение бинарных отношений: \n";

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

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

int res = 0;

for (int a = 0; a < N; a++)

res += firstMatrix[i][a] \* secondMatrix[a][j];

if (res > 0) cout << 1 << " ";

else cout << 0 << " ";

}

cout << endl;

}

}

void binReverse() { //Обратное бинарное отношение

int N;

cout << "\nВведите размер матрицы бинарного отношения: ";

cin >> N;

cout << "\nВведите матрицу бинарного отношения \n";

int\*\* matrix = new int\* [N];

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

matrix[i] = new int[N];

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

cin >> matrix[i][j];

}

cout << "\nОбратное бинарное отношение: \n";

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

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

cout << matrix[j][i] << " ";

cout << endl;

}

}

void binRelations() { //Бинарные отношения

for (;;) {

cout << "\n1 - Операция объединения \n2 - Операция пересечения \n3 - Операция дополнения \n";

cout << "4 - Операция произведения \n5 - Нахождение обратного бин.отношения \n6 - Назад\n";

int x;

cin >> x;

switch (x) {

case 1:

binObed();

break;

case 2:

binPere();

break;

case 3:

binDop();

break;

case 4:

binMult();

break;

case 5:

binReverse();

break;

case 6:

return;

default:

cout << "Incorrect. Try again! \n";

}

}

}

void matrAdd() { //Сложение матриц

cout << "\nВведите количество строк матриц: ";

int row;

cin >> row;

cout << "Введите количество столбцов матриц: ";

int column;

cin >> column;

cout << "\nВведите 1-ю матрицу \n";

int\*\* firstMatrix = new int\* [row];

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

firstMatrix[i] = new int[column];

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

cin >> firstMatrix[i][j];

}

cout << "\nВведите 2-ю матрицу \n";

int\*\* secondMatrix = new int\* [row];

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

secondMatrix[i] = new int[column];

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

cin >> secondMatrix[i][j];

}

cout << "\nСложение матриц: \n";

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

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

cout << firstMatrix[i][j] + secondMatrix[i][j] << " ";

cout << endl;

}

}

void matrMult() { //Умножение матриц

cout << "\nВведите количество строк 1-й матриц: ";

int row1;

cin >> row1;

cout << "Введите количество столбцов 1-й матриц: ";

int column1;

cin >> column1;

cout << "Введите 1-ю матрицу \n";

int\*\* firstMatrix = new int\* [row1];

for (int i = 0; i < row1; i++) {

firstMatrix[i] = new int[column1];

for (int j = 0; j < column1; j++)

cin >> firstMatrix[i][j];

}

cout << "\nВведите количество строк 2-й матриц: ";

int row2;

cin >> row2;

cout << "Введите количество столбцов 2-й матриц: ";

int column2;

cin >> column2;

cout << "Введите 2-ю матрицу \n";

int\*\* secondMatrix = new int\* [row2];

for (int i = 0; i < row2; i++) {

secondMatrix[i] = new int[column2];

for (int j = 0; j < column2; j++)

cin >> secondMatrix[i][j];

}

if (row2 != column1) {

cout << "Умножение данных матриц невозможно!\n";

return;

}

cout << "\nПроизведение матриц: \n";

for (int i = 0; i < row1; i++) {

for (int j = 0; j < column2; j++) {

int res = 0;

for (int a = 0; a < row2; a++)

res += firstMatrix[i][a] \* secondMatrix[a][j];

cout << res<< " ";

}

cout << endl;

}

}

void matrTransposed() { //Транспонированная матрица

cout << "\nВведите количество строк матрицы: ";

int row;

cin >> row;

cout << "Введите количество столбцов матрицы: ";

int column;

cin >> column;

cout << "Введите матрицу \n";

int\*\* matrix = new int\* [row];

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

matrix[i] = new int[column];

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

cin >> matrix[i][j];

}

cout << "\nТранспонированная матрица: \n";

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

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

cout << matrix[j][i] << " ";

cout << endl;

}

}

double detMatr(vector <vector <double> > matrix, int N, double count) { //Определитель матрицы

if (N == 2)

return (matrix[0][0] \* matrix[1][1] - matrix[0][1] \* matrix[1][0]);

else if (N == 3)

return ((matrix[0][0] \* matrix[1][1] \* matrix[2][2] + matrix[0][1] \* matrix[1][2] \* matrix[2][0] + matrix[1][0] \* matrix[2][1] \* matrix[0][2]) -

(matrix[0][2] \* matrix[1][1] \* matrix[2][0] + matrix[0][1] \* matrix[1][0] \* matrix[2][2] + matrix[0][0] \* matrix[1][2] \* matrix[2][1]));

else {

double res = 0;

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

count \*= -1;

vector< vector <double> > detA;

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

vector <double> row;

for (int k = 0; k < N; k++)

if (j != 0 && k != i)

row.push\_back(matrix[j][k]);

if (!row.empty())

detA.push\_back(row);

}

res += count \* matrix[0][i] \* detMatr(detA, N - 1, -1);

}

return res;

}

}

void matrReverse() { //Обратная матрица

cout << "\nВведите размер матрицы: ";

int N;

cin >> N;

cout << "Введите матрицу: \n";

vector <vector <double> > matrix;

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

vector <double> row(N);

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

cin >> row[j];

matrix.push\_back(row);

}

cout << "\nВведите порядок поля, над которым выполняется операция: ";

int field;

cin >> field;

int determinantMatrix = detMatr(matrix, N, -1);

while (determinantMatrix < 0)

determinantMatrix += field;

determinantMatrix %= field;

if (determinantMatrix == 0) {

cout << "\nОпределитель матрицы равен 0 и обратной матрицы не существует\n";

return;

}

else

cout << "\nОпределитель матрицы равен " << determinantMatrix << "\n\n";

int inv\_det = 0;

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

if ((i \* determinantMatrix) % field == 1) {

inv\_det = i;

break;

}

vector <vector <double> > res;

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

vector <double> help;

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

double number = 1;

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

number = -1;

vector <vector <double> > det;

for (int a = 0; a < N; a++) {

vector <double> row;

for (int b = 0; b < N; b++)

if (a != i && b != j)

row.push\_back(matrix[a][b]);

if (!row.empty())

det.push\_back(row);

}

int det\_dop = number \* detMatr(det, det.size(), -1);

while (det\_dop < 0)

det\_dop += field;

det\_dop %= field;

help.push\_back((det\_dop \* inv\_det) % field);

}

res.push\_back(help);

}

cout << "Обратная матрица: \n";

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

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

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

cout << endl;

}

}

void matrRelations() { //Матрицы

for (;;) {

cout << "\n1 - Операция сложения \n2 - Операция умножения \n3 - Операция транспонирования \n";

cout << "4 - Операция обращения \n5 - Выход \n";

int x;

cin >> x;

switch (x) {

case 1:

matrAdd();

break;

case 2:

matrMult();

break;

case 3:

matrTransposed();

break;

case 4:

matrReverse();

break;

case 5:

return;

default:

cout << "Incorrect. Try again! \n";

}

}

}

int main() {

setlocale(LC\_ALL, "ru");

for (;;) {

cout << "1 - Проверка свойств операции \n2 - Выполнение операций над бинарными отношениями \n3 - Выполнение операций над матрицами \n4 - Выход \n";

int x;

cin >> x;

switch (x) {

case 1:

checkFeatures();

cout << endl;

break;

case 2:

binRelations();

cout << endl;

break;

case 3:

matrRelations();

cout << endl;

break;

case 4:

return 0;

default:

cout << "Incorrect. Try again! \n";

}

}

}

ЗАКЛЮЧЕНИЕ

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